

 Fast campus × 

# Upstage AI Lab

LangChain Project | 2025. 11. 6 (목)

# 목차

- 01. 팀 소개**  
팀원 소개 / 협업 방식
- 02. 프로젝트 개요**  
목표 수립 / 기술 스택 & 아키텍처 설계
- 03. 프로젝트 수행 절차 및 방법**  
작품 소개 / 구현 전략 / 기능 리뷰
- 04. 회고**  
결과 / 인사이트 도출 / 향후 계획 / 느낀점

01

## 팀 소개

---

팀장/팀원 소개  
협업 방식

## [ 3조 ]



팀장

장윤정  
컴퓨터공학  
웹 서비스 개발



팀원

송인섭  
기계공학  
웹 서비스 개발



팀원

김명철  
산업공학  
Agent 개발



팀원

김상윤  
컴퓨터공학  
Agent 개발



팀원

정소현  
기업인사  
Agent 개발

# 프로젝트 협업 방식

## Langchain Project

### 1. 매일 오전 출석체크 후 회의 진행

- 작업 진행 상황 공유
- 이슈사항에 대해 학습한 내용 공유
- 협의를 통한 작업 방향 결정
- 필요 시 오후 추가 회의 진행

### 2. Slack 및 Github 작업 공유

- Github PR, merge 등 작업 내용을 Slack에 공유하고, 코드를 최신화하여 병합 시 충돌 발생 최소화
- 패키지 의존성 변경 시 공유하여 개발 환경 통일
- 환경설정 등 공유가 필요한 내용을 wiki에 작성

The screenshot displays a project management interface with two main sections:

- GitHub Pull Requests:** A list of merged pull requests from various contributors (qpwpep, yijiang06, 94KSY) across different branches (Slack, Webserver, feat). The interface includes filters for Author, Label, Projects, Milestones, Reviews, Assignee, and Sort.
- 회의록 (Meeting Notes) - Confluence Page:** A page titled "회의록" edited by yijiang 4 hours ago. It contains two sections for November 5th and November 4th, each listing tasks and sub-tasks. The sidebar on the right shows a navigation menu with "Pages 3" and a list of pages including "Home", "uv 사용방법", and "회의록".

**회의록 (Meeting Notes) - Confluence Page Content:**

- 2025-11-05**
  - 각자 작업 공유
    - 멀티턴 > 오래된 대화 요약 방식 적용
    - Slack 메시지 발송 tool 적용
    - 현재 진행 중인 작업 마무리 후 오후 회의 진행
    - 전체적인 기능 점검 및 테스트 시나리오 협의
- 2025-11-04**
  - 각자 작업 공유
    - Streamlit 파일 업로드 개발, 업로드 파일 기준으로 RAG tool 연동
    - 멀티턴 > 슬라이딩 윈도우 방식 적용, 오래된 대화 요약 방식 개발 진행
    - Slack 메시지 발송 tool 개발 진행

02

## 프로젝트 개요

---

목표 수립  
작품 소개 / 기술 스택

## 프로젝트 목표 수립

### Langchain Project | 목표 및 주요 작업

목표 수립

주제

#### Langchain 프로젝트 | Langchain을 이용한 App 개발 프로젝트

\* LangGraph 기반 AI Agent 서비스 개발

목표

개요

#### 목표

- \* LangGraph 및 Agent 구조를 함께 학습하며 성장
- \* 모든 팀원이 코드 개발과 PR에 참여하며 협업 역량 향상

#### 소개 및 배경 설명

- \* 공식 문서를 기반으로 기술 스택 가이드를 제공하여, 사용자가 직접 문서를 찾지 않아도 기본 답변을 받을 수 있는 챗봇

#### 주요 작업

- \* LangGraph를 활용하여 Agent 구현
- \* FastAPI, Streamlit을 사용하여 웹서비스 구현
- \* 도구 실행과 LLM 응답 구조를 확인하며 멀티턴 대화 구현

#### 기간

2025. 10. 24. ~ 2025. 11. 06

# 프로젝트 개요

## Langchain Project | 기술 스택 및 아키텍처 설계

1

### \* 개발 환경 구축

- \* pyenv를 통한 가상환경 구축 (python 3.12)

2

### \* 협업 전략

- \* git을 통한 협업
  - 각자 작업 브랜치를 만들어 main브랜치에 병합
  - github wiki를 통한 회의록 관리

3

### \* 기술 스택

- \* Agent
  - LangGraph
- \* Frontend
  - Streamlit
- \* Backend
  - Fast API

4

### \* 패키지

- \* uv를 활용하여 pyproject.toml 파일을 통해 패키지 관리
- \* 주요 패키지: langchain, langgraph, fastapi, streamlit

03

## 프로젝트 수행 절차 및 방법

---

작품 소개 / 구현 전략 / 기능 리뷰

# 프로젝트 수행 절차 및 방법

## Langchain Project | 작품 소개

### DocuMate: Document + Mate

- 공식 문서를 기반으로 학습을 돋는 AI 챗봇

! LLM에게 코드 요청은 익숙하지만, 기본 사용법을 놓치는

경우가 많음

! 여러 라이브러리 사용 시 일일히 공식 문서 페이지를 찾는

번거로움

! 지식 컷오프 때문에 예전 정보를 제공하는 경우가 있음

💡 여러 기술 스택 공식 문서 기반 답변을 한 곳에서 확인

💡 학습에 도움이 될 만한 챗봇 개발

**Slack 전송**

User ID (Uxxxxx)

Email (optional)

Channel ID (C/G/Dxxxxx, optional)

### DocuMate: 공식 문서를 기반으로 학습을 돋는 AI 챗봇

파이썬 오픈 소스 라이브러리의 활용법을 학습해보세요!

- 공식 문서 내용을 기반으로 정확한 정보를 얻으실 수 있습니다.  
(지원 문서: `python`, `git`, `LangChain`, `Matplotlib`, `NumPy`, `pandas`, `PyTorch`, `Hugging Face`, `FastAPI`, `BeautifulSoup`, `streamlit`, `gradio`, `scikit-learn`, `Pydantic`)
- 로컬 노트북 (AI 부트캠프 경진대회 `baseline code`) / 직접 파일을 업로드 하여 활용 사례를 확인하실 수 있습니다.
- 결과를 `txt` 파일로 저장하거나 슬랙에 공유하실 수 있습니다.

✓ 빠른 시작 예시

`pandas merge` 사용법 알려줘. 공식 문서 기준으로 설명해줘. `matplotlib`에서 `pie` 차트 옵션 정리해줘.  
이전 노트북에서 `matplotlib histplot`을 어떻게 썼는지 예제 코드 보여줘. 업로드한 `.ipynb` 안에 있는 `pandas concat` 예제를 찾아줘.  
`pandas concat` 기본 사용법을 설명하고, 내 노트북에서 실제로 사용한 예제도 함께 보여줘.

📎 파일 업로드

- `.py`, `.ipynb` 파일을 업로드 하여 해당 내용을 기반으로 서칭하기: 이 파일에서 쓰인 `pandas concat()` 함수 예제를 찾아줘  
(업로드 파일이 크면 핵심 코드/셀만 올리는 것이 더 빠릅니다.)

💾 결과 txt 파일로 저장 및 Slack 공유

- 결과를 `txt` 파일로 저장하기: 이 답변을 `txt`로 저장해줘 방금 결과를 파일로 저장해줘
- 결과를 Slack으로 보내기: 이 답변을 Slack으로 보내줘 이번 결과를 팀 채널에 공유해줘

👋 안녕하세요! 질문을 입력해주세요.

파일 업로드 (.py, .ipynb 등 챗봇에게 질문할 때 사용할 파일을 업로드하세요.)

Drag and drop file here  
Limit 200MB per file • IPYNB, PY

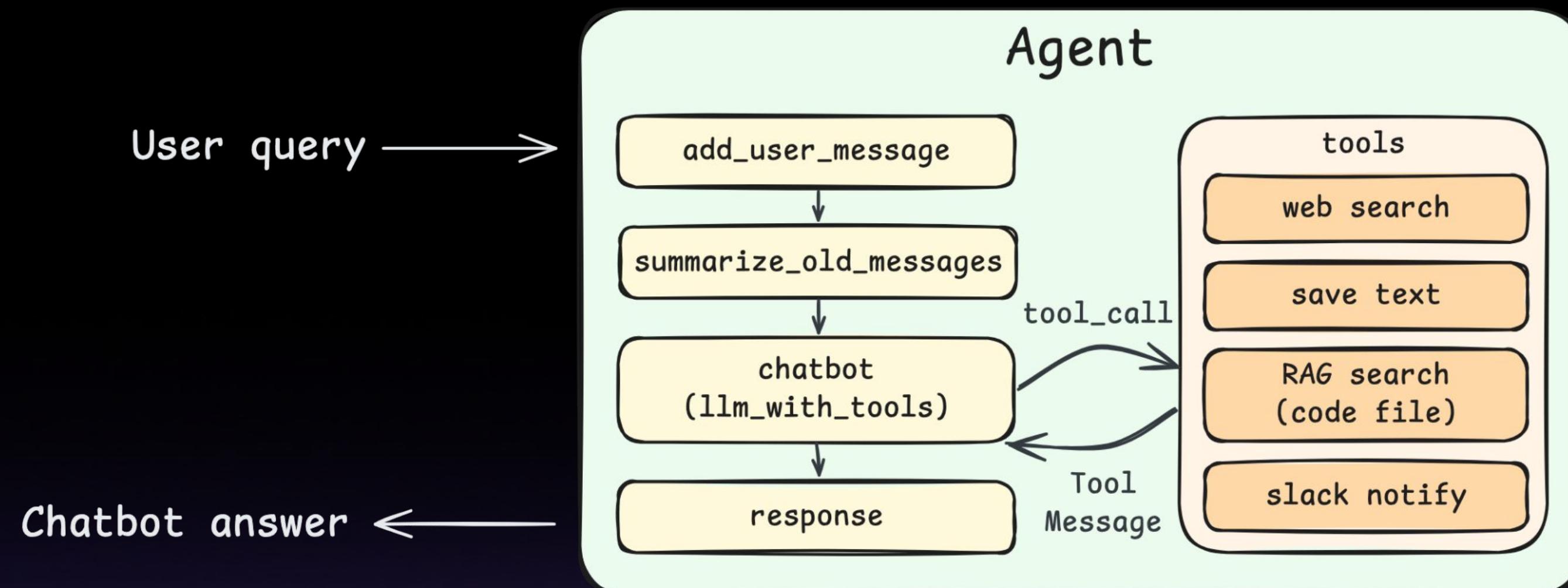
Browse files

여기에 질문을 입력하세요... ➤

# 프로젝트 수행 절차 및 방법

Langchain Project | 구현 전략

## Agent 구조



# 프로젝트 수행 절차 및 방법

## Langchain Project | 구현 전략

### 프롬프트 엔지니어링

#### \* 시스템 프롬프트

- \* 4개의 도구 중 적절한 도구를 선택하도록 사용 목적을 프롬프트에 작성
- \* 기술 스택 공식 문서를 참고한 답변일 경우,  
출처와 문서 링크를 출력하도록 포맷 정의
- \* 사용자 질문에 특정 키워드가 있는지  
체크하여 특정 도구 사용을 위한 패턴과  
일치할 경우, 도구 실행을 도와주는  
SystemMessage 추가

#### \* 도구 입력 구조화

- \* Pydantic BaseModel을 상속받아,  
도구에서 받을 입력 구조를 정의
- \* 정의된 구조를 통해 도구 호출 시  
올바른 입력이 전달되도록 처리

# 프로젝트 수행 절차 및 방법

## Langchain Project | 구현 전략

### 멀티턴 (Multi-turn)

1단계

#### \* 기초 멀티턴

- \* 한 세션 안의 대화 내용을 챗봇이 기억하게 하려고 이전 대화내용을 input에 포함해서 질문

2단계

#### \* 슬라이딩 윈도우

- \* 1단계의 문제점: 모든 대화 내용을 input값에 추가하다 보니 대화 내용이 길어질수록 토큰값이 기하급수적으로 늘어나게 됨
- \* 해결 방안: 슬라이딩 윈도우 기법을 사용하여 최근 6턴 (사용자의 질문 + AI의 응답 = 1턴) 의 내용을 input에 같이 넣어 최근 대화 내용을 기억하여 답변할 수 있도록 보완

3단계

#### \* 오래된 대화 요약

- \* 기능 개선: 개발 관련 질문은 이전 대화의 내용이 이어지는 경우가 많으므로 6턴 보다 더 오래된 내용은 4~5줄로 요약하여 input 값에 추가하여 오래된 내용도 포함할 수 있고 토큰의 사용량도 적절하게 조절할 수 있도록 개선함

User: 클래스 객체는 ?

[summary] merged (3 msgs → 4~5 lines)

[trim] messages for model input: 19 → 14

대화가 길어지면  
**summary**: 오래된 대화 내용 4~5줄로 요약  
**trim**: 최근 6턴만 input에 포함하여 messages수가 줄어드는 모습

# 프로젝트 수행 절차 및 방법

## Langchain Project | 기능 리뷰

### 웹 검색 (TavilySearch)

\* `include_domains`를 활용한 검색 범위 제한

- (1) 사용자가 'pandas 사용법을 알려줘.' 등의 명령 입력.
- (2) `prompts.py` 내의 `NEED_SEARCH_PATTERNS`가 패턴을 감지.
- (3) `node.py` 의 `SEARCH_HINT` 가 추가되어 LLM에게 힌트를 전달.
- (4) LLM이 `tavilysearch` 툴을 호출.  
-> '구버전/공식 문서가 아닌 문서'를 제외하기 위해,  
`include_domains`를 활용해 검색 범위를 제한.
- (5) 툴 실행 결과를 반환하여 응답 생성.

The screenshot shows the TavilySearch web application interface. At the top, there is a code editor window displaying Python code for defining search patterns and hints:

```

NEED_SEARCH_PATTERNS = [
    r"\b(latest|official|docs?|documentation|reference|api|syntax|parameter|manual)\b",
    r"(최신|공식|문서|레퍼런스|함수|매개변수|사용법|방법|API)"
]

SEARCH_HINT = "(이 질문은 공식/최신 문서 검색이 필요합니다. 'tavilysearch' 도구를 먼저 사용하세요.)"

```

Below the code editor is the application's user interface. It includes a sidebar with navigation links like 'Home', 'About', 'FAQ', 'Contact', and 'Logout'. The main content area displays a search result for the query 'pandas 사용법'. The result is a snippet from the Python documentation:

파이썬 문서: pandas

파이썬 문서는 pandas 라이브러리의 사용법과 API를 설명하는 문서입니다. pandas는 주로 데이터 분석과 처리에 사용되는 라이브러리입니다. pandas는 데이터 프레임, 시리즈, 판타지 등 다양한 데이터 구조를 제공하며, 다양한 통계적 기능과 시각화 도구를 포함합니다.

질문: pandas 사용법

답변: pandas는 주로 데이터 분석과 처리에 사용되는 라이브러리입니다. pandas는 데이터 프레임, 시리즈, 판타지 등 다양한 데이터 구조를 제공하며, 다양한 통계적 기능과 시각화 도구를 포함합니다.

底部 of the screenshot shows the source code for the `tavilysearch` tool, which uses the defined patterns and hints to filter search results:

```

tavilysearch = TavilySearch(
    max_results=3,
    include_domains=list(DEFAULT_DOCS.values()),
)

```

# 프로젝트 수행 절차 및 방법

## Langchain Project | 기능 리뷰

### 로컬 및 업로드 파일 검색 (RAG)

- \* LLM이 로컬 노트북이나 사용자가 업로드 한 파일에서 유사한 내용을 검색하고 응답하도록 설계 (1)
  - 로컬 지식(노트북) + 업로드 파일 인덱스 생성
    - (1) 변경 감지: manifest.json과 파일 목록/해시를 비교해 신규/수정/삭제를 구분
    - (2) 문서 로드: NotebookLoader로 셀 텍스트만 추출
    - (3) Chunking: RecursiveCharacterTextSplitter 사용해 문단 단위로 나눠 검색 성능/정확도 균형 확보
    - (4) 배치 임베딩: OpenAI Embeddings (text-embedding-3-small) 배치로 호출 (속도/토큰 최적화)
    - (5) 증분 인덱싱: add\_texts()로 새/수정 분만 반영, 삭제분 정리해 Chroma Vector DB에 저장
    - (6) 메타데이터 보강: 각 chunk에 source 기록 → 출처 표기에 사용
    - (7) manifest 갱신: 다음 빌드 시 빠르게 변경분만 반영

```

def extract_text_from_py(path: str) -> str:
    with open(path, "r", encoding="utf-8") as f:
        return f.read()

def extract_text_from_ipynb(path: str) -> str:
    import nbformat
    nb = nbformat.read(path, as_version=4)
    texts: List[str] = []
    for cell in nb.cells:
        if cell.get("cell_type") in {"code", "text"}:
            src = cell.get("source") or ""
            texts.append(src)

def _save_manifest(manifest: Dict[str, float]) -> None:
    INDEX_DIR.mkdir(parents=True, exist_ok=True)
    with open(MANIFEST_PATH, "w", encoding="utf-8") as f:
        json.dump(manifest, f, ensure_ascii=False)

def _load_ipynb_docs(file_paths: List[str]) -> List[Dict]:
    docs = []
    for p in file_paths:
        loader = NotebookLoader(p, include_outputs=True)
        docs.extend(loader.load())
    # normalize metadata
    for d in docs:
        d.metadata["source"] = d.metadata.get("source")
    return docs

def _split_docs(docs):
    splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=100)
    return splitter.split_documents(docs)

def _ensure_chroma(embeddings) -> Chroma:
    return Chroma(
        embedding_function=embeddings,
        persist_directory=str(INDEX_DIR),
        collection_name="notebooks",
    )
  
```



matplotlib boxchart 사용 예제 하나를 이전 노트북에서 찾아서 보여줘



이전 노트북에서 matplotlib의 boxplot 사용 예제가 발견되었습니다. 예제 코드 일부를 보여드리면 다음과 같습니다:

```

# 대표적인 연속형 변수인 “전용 면적” 변수 관련한 분포를 먼저 살펴보도록
fig = plt.figure(figsize=(7, 3))
sns.boxplot(data=concat_select, x='전용면적', color='lightgreen')
plt.title('전용면적 분포')
plt.xlabel('Area')
plt.show()
  
```

이 예제는 seaborn 라이브러리의 boxplot 을 사용하여 '전용면적' 컬럼의 분포를 시각화한 것으로 보입니다.

전체 예제는 노트북 `data\baseline_code-1.ipynb`에 있습니다.

[◆ 로컬 예제] `data\baseline_code-1.ipynb`

필요하면 더 자세한 내용을 불러와 보여드릴 수 있습니다.

# 프로젝트 수행 절차 및 방법

## Langchain Project | 기능 리뷰

### 로컬 및 업로드 파일 검색 (RAG)

\* LLM이 로컬 노트북이나 사용자가 업로드 한 파일에서 유사한 내용을 검색하고 응답에 반영하도록 설계 (2)

- (1) 사용자가 “이전 노트북에서 pandas merge 예제 찾아줘” 등의 명령 입력
- (2) prompts.py 내의 NEED\_RAG\_PATTERNS가 패턴을 감지
- (3) node.py 의 RAG\_HINT 가 추가되어 LLM에게 힌트를 전달  
→ “이 요청은 로컬 노트북/예제 기반 지식 검색이 필요합니다. ‘rag\_search’ 도구를 사용하세요.”
- (4) LLM이 rag\_search 를 호출
- (5) Chroma 벡터DB에서 관련 chunk를 검색 후 유사한 코드와 출처 반환
- (6) 둘 실행 결과를 응답에 반영 (공식 문서 기반 답변과 함께 혼합(hybrid) 형태로 인용 가능)

```
NEED_RAG_PATTERNS = [
    r"\b(example|sample|notebook|project|code|implementation|practice)\b",
    r"(이전|노트북|예제|코드|실습|프로젝트|데이터셋|baseline|결과)"
]

RAG_HINT = "(이 요청은 로컬 노트북/예제 기반 지식 검색이 필요합니다. 'rag_search' 도구를 사용하세요.)"

INDEX_PATH = "data/index"

def _load_chroma():
    emb = OpenAIEmbeddings(model="text-embedding-3-sma")
    return Chroma(
        embedding_function=emb,
        persist_directory=INDEX_PATH,
        collection_name="notebooks",
    )

def rag_search(query: str, k: int = 4) -> str:
    """Search local .ipynb notebooks and return relevant snippets.
    If not os.path.isdir(INDEX_PATH):
        return "RAG index not found. Please build it first."
    db = _load_chroma()
    docs = db.similarity_search(query, k=k)
    if not docs:
        return "No relevant passages found in local notebooks."
    lines = []
    for i, d in enumerate(Document(docs, 1)):
        src = d.metadata.get("source", "notebook")
        snippet = d.page_content.strip().replace("\n", " ")
        if len(snippet) > 500:
            snippet = snippet[:500] + " ..."
        lines.append(f"{i}. {snippet}\n[◆ 로컬 예제] {src}")
    return "\n".join(lines)

class RagArgs(BaseModel):
    query: str = Field(description="The user's information")
    k: int = Field(default=4, ge=1, le=10, description="Number of chunks to return.")

Ctrl+L to chat, Ctrl+K to generate
rag_search_tool = StructuredTool.from_function(
    name="rag_search",
    description=(
        "Search local .ipynb notebooks (vector index) and return relevant snippets with sources. "
        "Use this when the question is covered by our local documents."
    ),
    func=rag_search,
    args_schema=RagArgs,
)
```



matplotlib boxchart 사용 예제 하나를 이전 노트북에서 찾아서 보여줘



이전 노트북에서 matplotlib의 boxplot 사용 예제가 발견되었습니다. 예제 코드 일부를 보여드리면 다음과 같습니다:

```
# 대표적인 연속형 변수인 “전용 면적” 변수 관련한 분포를 먼저 살펴보도록
fig = plt.figure(figsize=(7, 3))
sns.boxplot(data=concat_select, x='전용면적', color='lightgreen')
plt.title('전용면적 분포')
plt.xlabel('Area')
plt.show()
```

이 예제는 seaborn 라이브러리의 boxplot 을 사용하여 '전용면적' 컬럼의 분포를 시각화한 것으로 보입니다.

전체 예제는 노트북 data\baseline\_code-1.ipynb 에 있습니다.

[◆ 로컬 예제] data\baseline\_code-1.ipynb

필요하면 더 자세한 내용을 불러와 보여드릴 수 있습니다.

# 프로젝트 수행 절차 및 방법

## Langchain Project | 기능 리뷰

### 파일 저장 (save text)

\* LLM이 답변 저장을 수행하고, 저장 결과를 응답에 반영하도록 설계

- (1) 사용자가 "이 답변을 저장해줘", "txt로 저장해줘" 등의 명령 입력
- (2) prompts.py 내의 NEED\_SAVE\_PATTERNS가 패턴을 감지
- (3) node.py 의 SAVE\_HINT 가 추가되어 LLM에게 힌트를 전달 (\*\*무한 loop 방지)  
→ “이 턴에서 save\_text 도구를 한 번만 호출하세요.”
- (4) LLM이 save\_text 툴을 호출  
툴 실행 후 저장 완료 메시지 반환  
(타임스탬프로 파일명 생성)  
→ “response\_20251105\_113022.txt”

```
NEED_SAVE_PATTERNS = [
    r"\b(save|export|write|txt)\b",
    r"(저장|내보내|텍스트|txt로|파일로)"
]

SAVE_HINT = "(사용자가 응답 저장을 요청했습니다. 최종 '응답 전문'을 content에 담아 'save_text' 도구를 한 번만 호출하세요.)"

def save_text_to_file(content: str, filename_prefix: str = "response") -> dict:
    try:
        """Save text content to a timestamped .txt file and return the path"""
        ts = datetime.now().strftime("%Y%m%d_%H%M%S")

        output_path = get_save_text_output_dir()
        os.makedirs(output_path, exist_ok=True)
        filename = f"{filename_prefix}_{ts}.txt"
        filepath = os.path.join(output_path, filename)
        with open(filepath, "w", encoding="utf-8") as f:
            f.write(content)

        return {
            "message": f"Saved output to {filename}",
            "file_path": filepath # 파일의 절대 경로
        }
    except Exception as e:
        return {"error": str(e)}

class SaveArgs(BaseModel):
    content: str = Field(description="The exact final response text to write")
    filename_prefix: Optional[str] = Field(
        default="response",
        description="Optional short prefix for the filename (no extension)."
    )

save_text_tool = StructuredTool.from_function(
    name="save_text",
    description=(
        "Save the given final response text into a timestamped .txt file in the current directory.  
Call this at most ONCE per user request. If you already saved, do not call again."
    ),
    func=save_text_to_file,
    args_schema=SaveArgs,
)
```

내용을 txt 파일로 저장해줘

이전 노트북에서 matplotlib의 boxplot 사용 예제가 발견되었습니다. 예제 코드 일부를 보여드리면 다음과 같습니다:

```
# 대표적인 연속형 변수인 “전용 면적” 변수 관련한 분포를 먼저 살펴보도록
fig = plt.figure(figsize=(7, 3))
sns.boxplot(data=concat_select, x='전용면적', color='lightgreen')
plt.title('전용면적 분포')
plt.xlabel('Area')
plt.show()
```

이 예제는 seaborn 라이브러리의 boxplot 을 사용하여 '전용면적' 컬럼의 분포를 시각화한 것으로 보입니다.

전체 예제는 노트북 `data\baseline_code-1.ipynb`에 있습니다.

[◆ 로컬 예제] `data\baseline_code-1.ipynb`

필요하면 더 자세한 내용을 불러와 보여드릴 수 있습니다.

해당 내용은 'matplotlib\_boxplot\_example\_20251105\_215704.txt' 파일로 저장되었습니다.

파일 저장 완료: `matplotlib_boxplot_example_20251105_215704.txt`

파일 다운로드 (`matplotlib_boxplot_example_20251105_215704.txt`)

# 프로젝트 수행 절차 및 방법

## Langchain Project | 기능 리뷰

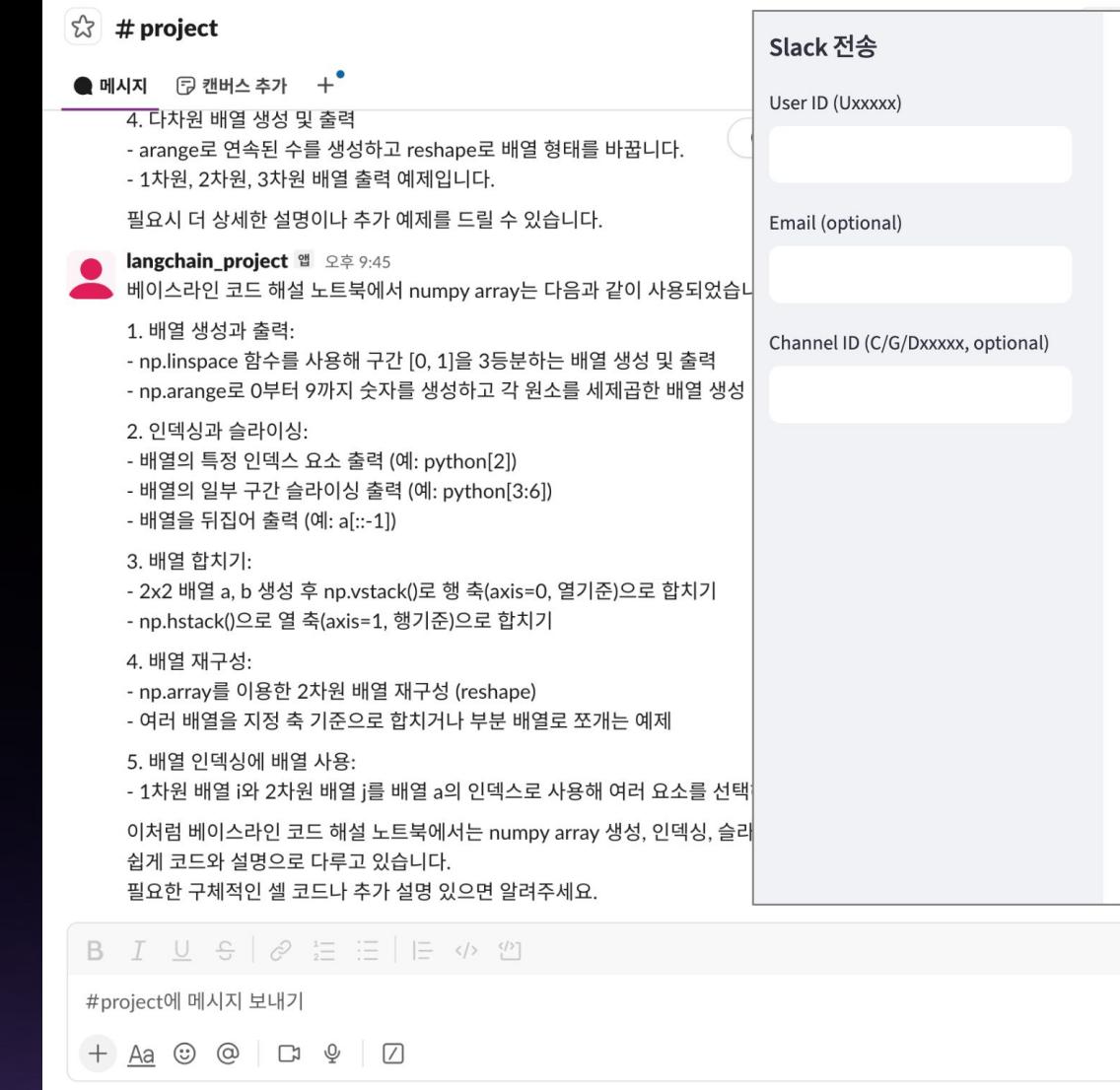
### Slack 연동 (slack notify)

#### \* 슬랙으로 답변을 전송하는 Tool 구현

- (1) 사용자가 streamlit의 사이드바에서 user\_id or user\_email or channel\_id를 입력.
- (2) '슬랙으로 보내줘.', '채널에 공유해줘.' 등의 메세지를 입력하면 LLM이 답변을 웹/터미널 뿐만 아니라 슬랙으로도 공유.

```
NEED_SLACK_PATTERNS = [
    r"(슬랙|slack).*보내", r"(DM|디엠).*보내", r"(채널|channel).*올려",
    r"(전송|공유).*(슬랙|slack|DM|디엠|채널)"
]

SLACK_HINT = "(사용자가 Slack 전송을 요청했습니다. 최종 답변을 'slack_notify' 도구로 보내세요. \
    가능하면 channel_id 또는 user_id/email 인자를 채워주세요.)"
```



### DocuMate: 공식 문서를 기반으로 학습을 돋는 AI 챗봇

파이썬 오픈 소스 라이브러리의 활용법을 학습해보세요!

- 공식 문서 내용을 기반으로 정확한 정보를 얻으실 수 있습니다.  
(지원 문서: python, git, LangChain, Matplotlib, NumPy, pandas, PyTorch, Hugging Face, FastAPI, BeautifulSoup, streamlit, gradio, scikit-learn, Pydantic)
- 로컬 노트북 (AI 부트캠프 경진대회 baseline code) / 직접 파일을 업로드 하여 활용 사례를 확인하실 수 있습니다.
- 결과를 txt 파일로 저장하거나 슬랙에 공유하실 수 있습니다.

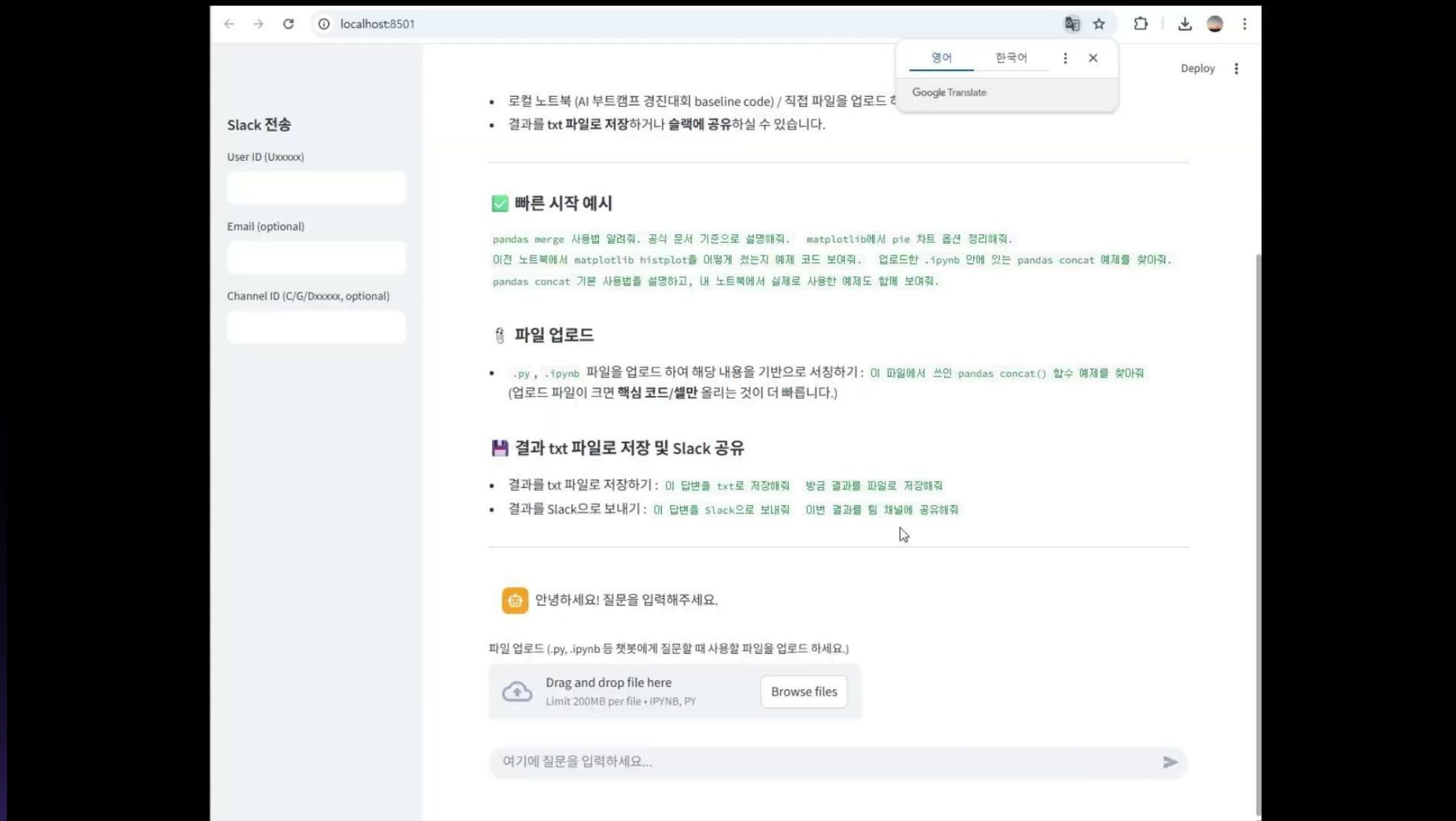
#### ✓ 빠른 시작 예시

pandas merge 사용법 알려줘. 공식 문서 기준으로 설명해줘. matplotlib에서 pie 차트 옵션 정리해줘.  
이전 노트북에서 matplotlib histplot을 어떻게 썼는지 예제 코드 보여줘. 업로드한 .ipynb 안에 있는 pandas concat 예제를 찾아줘.  
pandas concat 기본 사용법을 설명하고, 내 노트북에서 실제로 사용한 예제도 함께 보여줘.

# 프로젝트 수행 절차 및 방법

## Langchain Project | 프로젝트 시연 (Demo)

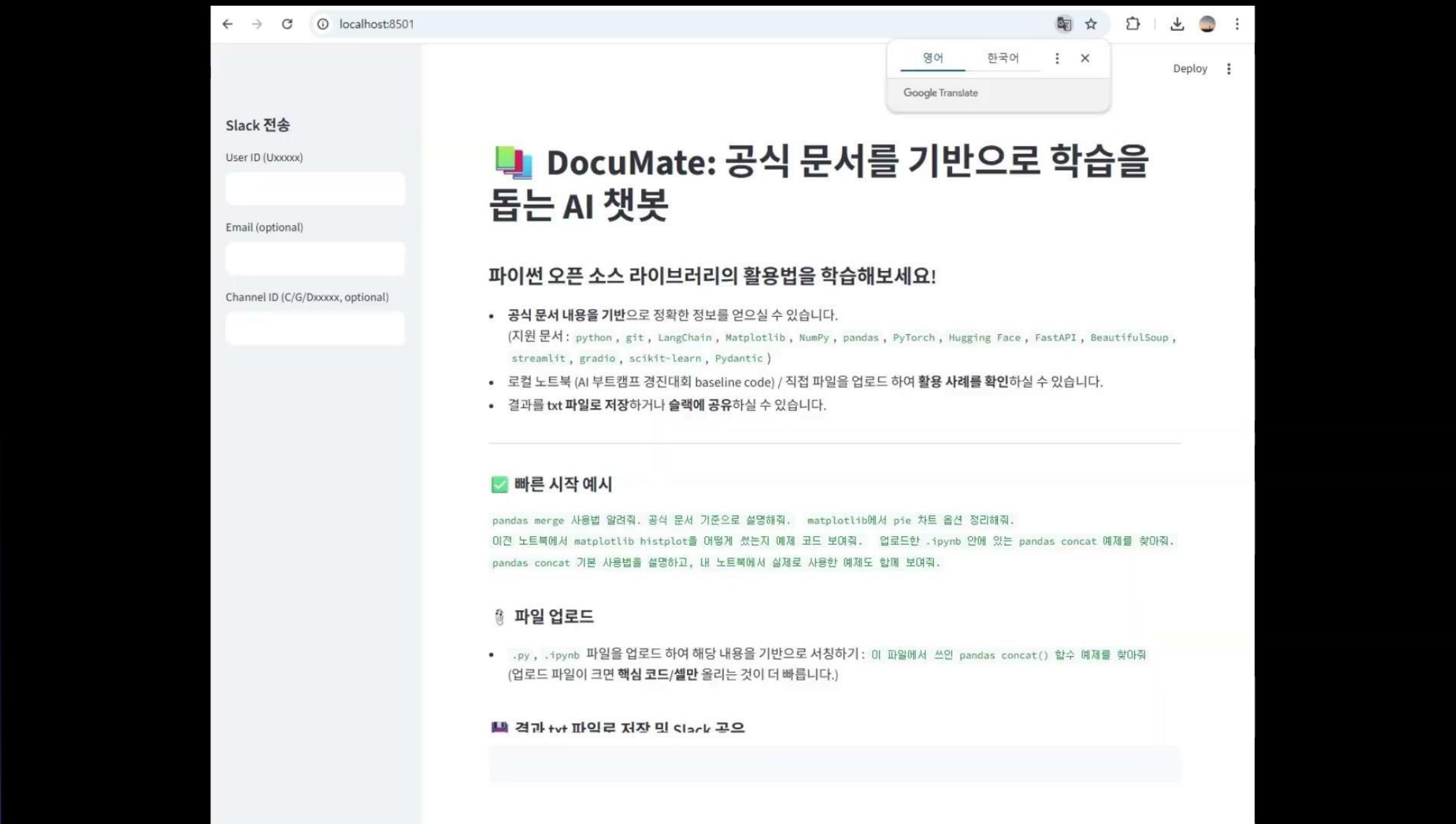
### 시연 1: 공식 문서 기반 + 로컬 노트북 RAG 서치 후 txt 파일로 답변 저장



# 프로젝트 수행 절차 및 방법

## Langchain Project | 프로젝트 시연 (Demo)

### 시연 2: 공식 문서 기반 + 사용자 업로드 노트북으로 RAG 서치 후 Slack 으로 답변 전송



04

## 회고

---

결과 / 인사이트 도출 / 향후 계획  
느낀점

# 프로젝트 회고

## :Langchain Project | 결과 및 향후 계획

1

### \* 최종 결과

- \* LangGraph 기반 Agent를 구현하여, 사용자 질의에 따라 동작하는 챗봇 구현

- \* FastAPI, Streamlit 을 연동하여, 챗봇 서비스 및 파일 업로드/다운로드 기능 구현



2

### \* 도전 과제 및 해결

- \* Langchain 및 LangGraph 구현  
- 문서와 실습 코드를 참고하여 필요한 기능을 직접 구현하며 구조를 이해하고 적용

- \* Agent와 FastAPI, Streamlit 연동  
- 객체 생성 책임을 명확하게 분리하고, 각 모듈간 연결 방식을 효율적으로 관리할 수 있도록 개선 시도



3

### \* 인사이트 도출

- \* LangGraph를 직접 설계하고 구현하며 노드 간 흐름 설계의 중요성을 체감

- \* 멀티턴 적용 시, 이전 대화 맥락을 어떻게 관리/갱신할지에 대한 다양한 방법을 학습

- \* 모듈화를 통해 각 모듈 간 데이터 인터페이스가 필요함을 경험



4

### \* 개선 방향

- \* Agent 응답 속도 개선

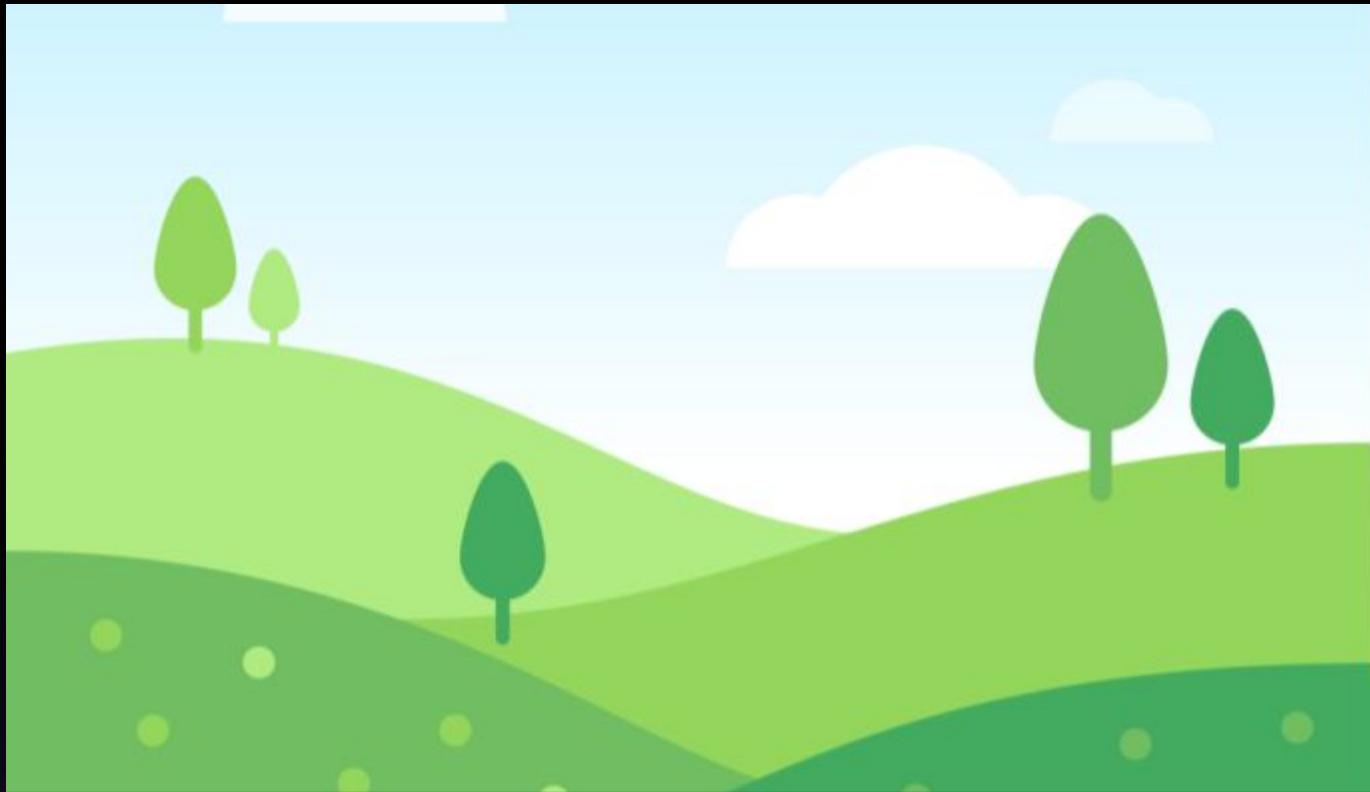
- \* 단순 동작 중심의 코드를 예외 처리 및 구조적 개선을 통해 코드 품질 향상

- \* txt 타입 텍스트 저장 기능을 마크다운 등 다른 타입으로 저장할 수 있도록 기능 확장



# 프로젝트 진행 소감

:Langchain Project | 느낀점



\* 장윤정 Agent를 직접 개발해 볼 수 있어서 좋았고, 팀원분들과 같이 작업하고 공유하면서 새로운 내용을 배울수 있어서 좋았습니다.

\* 송인섭 ChatBot, Agent, LangChain 등 주변에서 자주 들었던 용어와 개념들을 이해하는 시간이었고, 팀회의를 통해 협의해 가면서 구현되는 모습이 매우 흥미로웠다.

\* 김명철 재밌는 경험이었다. 실제 서비스는 아니었지만, 서비스를 발전시키는 과정과 새로운 기술을 배우는 과정을 경험하며 즐겁게 작업했다.

\* 김상윤 이번 부트캠프를 수강한 이유이기도 했고 지금까지의 프로젝트중 가장 재밌게 진행했던 것 같습니다.

\* 정소현 실제 사용될 법한 서비스를 구현해보면서 개념을 학습하고 응용을 시도해볼 수 있어 아주 좋은 경험이 되었습니다. 무엇보다 훌륭하게 이끌어주신 팀장님과 적극적으로 임해주신 팀원 분들께 감사 드립니다~!

Life-Changing Education

감사합니다.

---