프로젝트 개요

녹음된 음성파일에서 텟스트 축출후 법률 키워드 중심으로 판례정보 검색

프로젝트 개요 내용이 포함될수 있음.

필요한 부분이 있으면 살리고.

# 1.프로젝트 폴더 구조

모쥴 프로젝트 1차 5조 조원의 프로젝트 폴더는 다음과 같습니다.

자세한 설명은: C:/ProjectWork1 가 Root 가 됩니다

ProjectWork/

Documentation/

Design Specification/

Presentation/

temp/

Work1.ipynb

WorkRoot/

Apps/

Auth/

static/

templates/

Auth/

DocSum/

static/

templates/

DocSum/

FindRaw/

static/

templates/

FindRaw/

images/

S2T/

static/

templates/

S2T/

static/

css/

templates/

DataBase/

tests/

vEnv/

## ProjectWork1/Documentation/

### 문서관련 작업

## ProjectWork1/WorkRoot/

### Web App 관련 작업

WorkRoot/Apps/ => App 시작위치

## ProjectWork1/DataBase/

### DB 관련 작업

## ProjectWork1/vEnv/

### Python 가상환경

# 2. Git hub

1. **Windows 파일 탐색기 열기**: 먼저, Windows 파일 탐색기를 엽니다. 이는 보통 작업 표시줄에 있는 폴더 아이콘을 클릭하거나, 키보드에서 **Windows 키 + E**를 눌러 실행할 수 있습니다.
2. **'C:\' 폴더로 이동**: 파일 탐색기에서 'C:' 드라이브로 이동한 후.
3. **'Open Git Bash here' 실행**: 'C:\ ' 폴더에서 마우스 오른쪽 버튼을 클릭합니다. 이때 나타나는 컨텍스트 메뉴에서 'Open Git Bash here' (또는 유사한 명칭)를 찾아 클릭합니다. 이 옵션이 보이지 않는 경우, Git Bash가 제대로 설치되었는지 확인해야 합니다.
4. **Git Bash 사용**: 'Open Git Bash here'를 선택하면 Git Bash 터미널 창이 열립니다. 이 터미널 창은 'C:\ ' 폴더를 기본 디렉토리로 설정합니다. 이제 이 터미널에서 Git 명령어를 사용할 수 있습니다.

이 과정을 통해 특정 폴더에서 직접 Git Bash를 열고, Git 명령어를 사용하여 다양한 작업을 수행할 수 있습니다. 예를 들어, 이 디렉토리에서 새로운 Git 리포지토리를 초기화하거나 기존 리포지토리와의 작업을 진행할 수 있습니다.

# GitHub의 원격 저장소에 접근하여 최초 복제(clone)를 하는 과정은 Git의 **clone** 명령어를 사용하여 수행됩니다. 여기서는 **"https://github.com/MoonSongAi/ProjectWork1.git"**라는 원격 저장소를 예로 들어 설명하겠습니다.

### Git Clone 명령어 사용하기

1. **Git 설치 확인**: Git을 사용하기 전에 시스템에 Git이 설치되어 있고, 올바르게 설정되었는지 확인해야 합니다. Git이 설치되어 있는지 확인하려면, 터미널에서 다음 명령어를 실행합니다:

git --versionsion

이 명령어는 설치된 Git의 버전을 보여줍니다. Git이 설치되어 있지 않다면, Git 공식 웹사이트에서 다운로드하여 설치할 수 있습니다.

1. **터미널에서 Clone 명령어 실행**: 원하는 로컬 디렉토리로 이동한 후, 아래의 Git 명령어를 사용하여 원격 저장소를 복제합니다:

git clone -b master https://github.com/MoonSongAi/ProjectWork1.gitgit

이 명령어는 **"https://github.com/MoonSongAi/ProjectWork1.git"** 원격 저장소의 master 브랜치 내용을 현재 디렉토리에 **ProjectWork1**라는 이름의 폴더로 복제합니다.

1. **복제된 디렉토리 확인**: 복제가 완료되면, 생성된 **ProjectWork1** 폴더로 이동하여 내용을 확인합니다:

cd ProjectWork1

ls

### 추가 정보

* 복제는 원격 저장소의 모든 파일, 브랜치, 커밋 등을 로컬 컴퓨터로 가져오는 과정입니다.
* 이 작업을 수행하려면 인터넷 연결이 필요하며, 저장소의 크기에 따라 다운로드 시간이 달라질 수 있습니다.
* 복제된 로컬 저장소에서는 GitHub 원격 저장소의 변경 사항을 **pull** 명령어로 가져오거나 로컬 변경 사항을 **push** 명령어로 올릴 수 있습니다.

위의 단계를 통해 GitHub 원격 저장소를 로컬 컴퓨터로 최초 복제하는 방법을 알 수 있습니다.

GitHub의 원격 저장소 "https://github.com/MoonSongAi/ProjectWork1.git"에 Personal Access Token (PAT)을 사용하여 변경된 로컬 파일을 원격 저장소에 저장하는 절차는 다음과 같습니다:

### 1. 로컬 저장소 설정

먼저, 해당 GitHub 원격 저장소를 로컬 시스템에 클론해야 합니다. 이미 클론한 경우 이 단계를 건너뛸 수 있습니다.

bashCopy code

git clone –b master https://github.com/MoonSongAi/ProjectWork1.git

이 명령은 **ProjectWork1** 폴더에 원격 저장소의 내용을 복제합니다.

### 2. 변경 사항 작업

이제 로컬 저장소에서 필요한 파일 변경, 추가 또는 삭제 작업을 합니다.

### 3. 변경 사항 커밋

변경 사항을 스테이지에 추가하고 커밋합니다.

git add .

git commit -m "변경 사항에 대한 설명"

여기서 **git add .**는 모든 변경 사항을 스테이지에 추가합니다. **git commit -m "변경 사항에 대한 설명"**는 커밋 메시지와 함께 변경 사항을 커밋합니다.

### 4. GitHub에 Push

이제 변경 사항을 GitHub 원격 저장소에 push합니다.

git push origin master

이 명령은 로컬의 **master** 브랜치에서 변경 사항을 원격 저장소의 **master** 브랜치로 push합니다.

### 5. PAT 인증 사용

* 최초로 **push**를 시도할 때 GitHub 사용자 이름과 비밀번호를 입력하라는 요청을 받게 됩니다.
* 여기서 사용자 이름은 GitHub 계정의 사용자 이름을 입력하고, 비밀번호 대신에 이전에 생성한 **Personal Access Token**을 입력합니다.

### 주의사항

* 원격 저장소 URL이 HTTPS 형식이어야 PAT를 사용할 수 있습니다.
* PAT는 매우 민감한 정보이므로, 안전하게 관리하고 절대 공유하지 마세요.
* PAT를 사용하는 경우, 일반적으로 Git은 이 정보를 캐시하므로 다음 번 **push**에서 다시 입력할 필요가 없습니다.

이 과정을 통해 로컬에서의 변경 사항을 원격 저장소에 안전하게 업데이트할 수 있습니다.

# PAT 인증키는

## MoonSongAi@gmail.com

### ghp\_NordeBEQv7HAMc78Velgd6MgOXzGiX1fCxd1

### 예시: Git Push에서 PAT 사용하기

1. **터미널에서 Git Push 실행**
   * 로컬 저장소에서 터미널을 열고, 변경 사항을 원격 저장소에 push하려고 시도합니다:

bash code

git push origin master

1. **GitHub 사용자 이름 입력**
   * 터미널이 GitHub 사용자 이름을 물을 것입니다. 여기서는 예시로 'username'을 사용자 이름으로 가정합니다.
   * 터미널에 'username'을 입력하고 Enter 키를 누릅니다.
2. **PAT를 비밀번호로 사용**
   * 이제 터미널이 비밀번호를 묻습니다. 이 때, 비밀번호 대신에 이전에 생성한 Personal Access Token을 입력합니다.
   * 예를 들어, PAT가 **ghp\_xxxxxxxxxxxx**라면, 이 값을 비밀번호 필드에 입력하고 Enter 키를 누릅니다.

### 중요 사항

* 실제 토큰은 매우 긴 알파벳과 숫자의 조합으로 이루어져 있으며, 이는 예시일 뿐입니다.
* PAT는 매우 민감한 정보이므로, 입력 시 주변을 확인하고 안전하게 관리해야 합니다.
* 일부 Git 클라이언트나 GUI 도구에서는 이 과정이 조금 다를 수 있습니다.

이렇게 PAT를 사용하면 GitHub과의 통신 시 사용자 이름과 비밀번호 대신 PAT를 사용하여 인증할 수 있습니다.

# 위 처럼 사용자명과 비번을 요구 하는 프롬프가 발생 하지 않을떄

### git remote set-url origin <https://MoonSongAi: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@github.com/MoonSongAi/ProjectWork1.git>

원격 저장소의 파일을 로컬로 업데이트하는 과정은 주로 **git pull** 명령어를 사용하여 수행됩니다. 이 과정은 원격 저장소의 최신 변경 사항을 로컬 저장소로 가져오고, 로컬의 작업을 원격 저장소의 상태와 동기화하는 데 사용됩니다.

### 원격 저장소 파일을 로컬로 업데이트하는 과정:

1. **터미널 또는 명령 프롬프트 열기**:
   * 먼저, Git 명령어를 실행할 수 있는 터미널이나 명령 프롬프트를 엽니다.
2. **로컬 저장소로 이동**:
   * 원격 저장소와 연결된 로컬 저장소의 디렉토리로 이동합니다.
   * 예: **cd C:/ProjectWork1**
3. **현재 브랜치 확인**:
   * 현재 어떤 브랜치에 있는지 확인합니다. 일반적으로 업데이트하고자 하는 브랜치로 체크아웃되어 있어야 합니다.
   * **git branch** 명령어로 현재 브랜치를 확인할 수 있습니다.
4. **원격 저장소의 변경 사항 가져오기**:
   * **git pull** 명령어를 사용하여 원격 저장소의 최신 변경 사항을 로컬 저장소로 가져옵니다.
   * 명령어: **git pull origin branch-name**
   * 여기서 **branch-name** 은 업데이트하고자 하는 브랜치의 이름입니다.
   * 예를 들어, **master** 브랜치의 최신 변경 사항을 가져오고자 한다면, **git pull origin master** 명령어를 사용합니다.
5. **변경 사항 확인**:
   * **pull** 명령어 실행 후, 로컬 저장소에 원격 저장소의 최신 변경 사항이 반영되었는지 확인합니다.

### 주의사항:

* **git pull**은 원격 저장소의 변경 사항을 가져와 현재 로컬 브랜치와 병합합니다. 이 과정에서 충돌이 발생할 수 있으므로, 충돌이 발생하면 해결 후 커밋해야 합니다.
* 원격 저장소의 URL이나 브랜치 이름이 정확한지 확인하세요.
* 로컬에서 작업 중인 변경 사항이 있다면, **pull**을 하기 전에 커밋하거나 stash하는 것이 좋습니다.

이러한 단계를 통해 원격 저장소의 최신 변경 사항을 로컬 저장소로 가져올 수 있습니다.

# 3. 가상환경

Win + R => powershell

Windows PowerShell 에서 실행 합니다.

PS C:\> PowerShell Set-ExecutionPolicy RemoteSigned CurrentUser

PS C:\> cd .\ProjectWork1\

PS C:\ProjectWork1> python -m venv vEnv

"C:\ProjectWork1\vEnv\Scripts\activate" 로 Activate 시킬수 있음

Visual Studio Code (VSCode)의 터미널에서 Python 가상 환경을 만드는 과정은 다음과 같습니다. 이 과정은 VSCode가 이미 설치되어 있고, Python 확장이 설치되어 있다고 가정합니다.

### 1. VSCode 열기 및 프로젝트 폴더 선택

- VSCode를 열고, 작업하고자 하는 프로젝트 폴더를 엽니다.

- 폴더를 열기 위해 "파일" > "폴더 열기..."를 선택하거나, 시작 화면에서 "폴더 열기"를 클릭합니다.

### 2. 터미널 열기

- VSCode에서 내장된 터미널을 엽니다.

- "보기" 메뉴에서 "터미널"을 선택하거나, `Ctrl` + `` ` `` (백틱 키) 단축키를 사용합니다.

### 3. 가상 환경 생성

- 터미널에서 Python 가상 환경을 생성합니다. 예를 들어, `myenv`라는 이름으로 가상 환경을 만들고자 한다면, 다음 명령어를 사용합니다:

```bash

python -m venv myenv

```

- 이 명령은 현재 디렉토리에 `myenv`라는 이름의 폴더를 만들고 가상 환경을 설정합니다.

### 4. 가상 환경 활성화

- 생성된 가상 환경을 활성화합니다. 운영 체제에 따라 다음과 같이 활성화 명령어가 다릅니다:

- \*\*Windows\*\*:

```bash

.\myenv\Scripts\activate

```

-

- 가상 환경이 활성화되면, 터미널 프롬프트 앞에 가상 환경의 이름이 표시됩니다 (예: `(myenv)`).

### 5. 필요한 패키지 설치

- 가상 환경이 활성화된 상태에서 필요한 Python 패키지를 설치합니다. 예를 들어, `requests` 패키지를 설치하려면 다음 명령어를 사용합니다:

```bash

pip install requests

```

### 6. 작업 후 가상 환경 비활성화

- 작업을 마치고 가상 환경을 비활성화하려면 다음 명령어를 입력합니다:

```bash

deactivate

```

### 추가 팁

- \*\*가상 환경 폴더\*\*: 가상 환경은 프로젝트의 루트 디렉토리에 생성하는 것이 일반적입니다. 이렇게 하면 프로젝트와 관련된 모든 설정이 한 곳에 유지되어 관리가 용이합니다.

- \*\*Python 버전\*\*: 시스템에 여러 버전의 Python이 설치된 경우, `python` 대신 `python3` 또는 특정 버전을 지정할 수 있습니다 (예: `python3.8`).

- \*\*VSCode Python 환경 설정\*\*: VSCode에서는 작업 중인 가상 환경을 자동으로 감지하고 선택할 수 있습니다. 이를 위해 오른쪽 하단의 상태 표시줄에서 Python 인터프리터 버전 옆에 있는 드롭다운 메뉴를 클릭하고 원하는 환경을 선택합니다.

이러한 단계를 통해 VSCode의 터미널에서 Python 가상 환경을 쉽게 설정하고 관리할 수 있습니다.