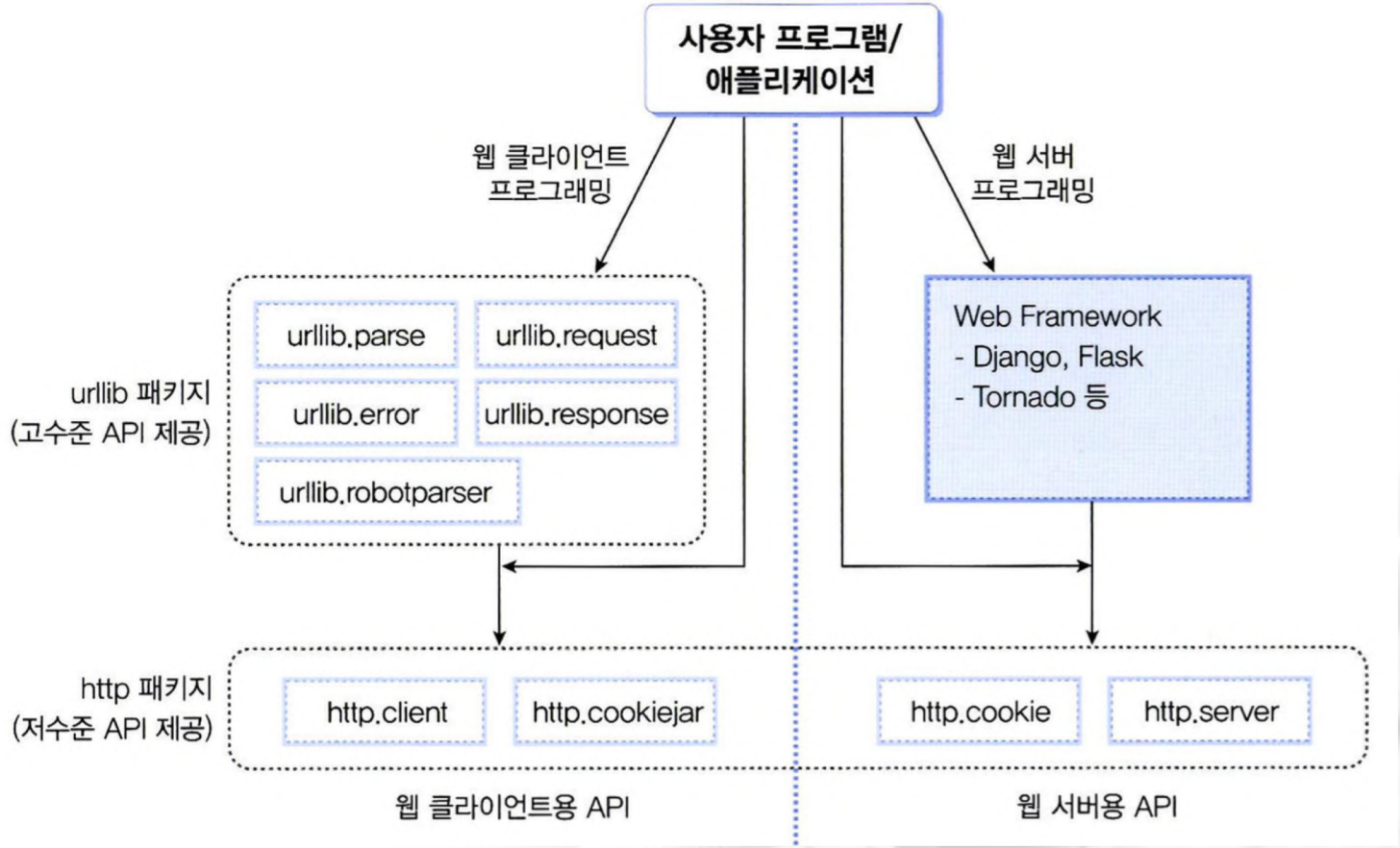
Django

Django

❖ Web 표준 라이브러리



Django

- Django
  - ✓ Django는 python의 대표적인 Web Application Framework으로서 Framework 자체가 python 으로 개발됨
  - ✓ Django는 Open Source Project로 공식 사이트인 http://www.djangoproject.com 에서 각종 최신 정보를 제공
  - ✓ Django 1.11 부터는 Python 3 와 Python 2.7에서 모두 사용할 수 있지만 공식 웹사이트에선 Python 3를 사용할 것을 권장
  - ✓ Django의 이전 버전들을 보면 각 버전 마다 호환되는 Python 버전들이 따로 있으므로 Django의 버전마다 호환되는 Python 버전을 설치해서 사용
  - ✓ Django는 다른 프레임워크들에 비해 자유도가 낮음
  - ✓ Django 설치

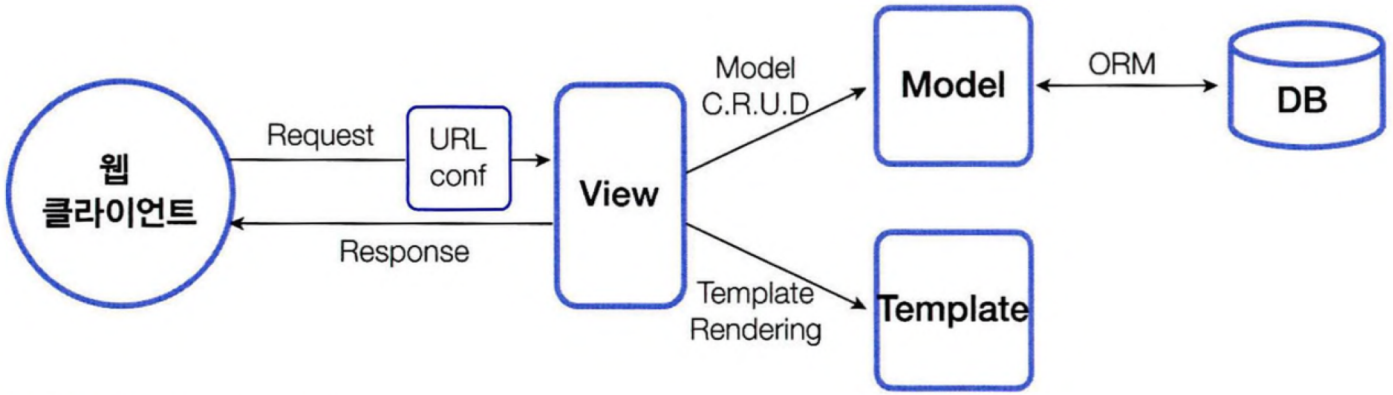
    pip install Django

Django

- Django
  - ✓ 개발 방식 - MTV
    - 웹 프로그래밍 영역을 3가지 개념으로 나눠서 개발하는 방식
    - 데이터를 정의하는 Model 그리고 사용자가 보게 될 화면의 모습을 정의하는 Template 과 애플리케이션의 제어 흐름 및 처리 로직을 정의하는 View로 구분해서 개발
    - 역할 별로 나눠서 개발을 진행하면 모델, 템플릿, 뷰 모듈 간에 독립성을 유지할 수 있고 소프트웨어 개발의 중요한 원칙인 느슨한 결합(Loose Coupling) 설계의 원칙에도 부합하며 Front End 개발자, Back End 개발자, DB 설계자 간에 협업이 쉬워지며 유지보수가 편리
    - Django에서 프로젝트를 생성하기 위해 startproject 및 startapp 명령을 실행하면 자동으로 프로젝트 뼈대에 해당하는 디렉토리와 파일들을 생성해주는데 모델은 models.py 파일에 작성하고 템플릿은 templates 디렉토리 하위의 *.html 파일에 그리고 뷰는 views.py 파일에 작성하도록 처음부터 뼈대를 만들어 줌
    - 애플리케이션을 MTV 방식으로 개발할 수 있도록 골격이 만들어지고 파일 이름도 Django에서 지어주는데 이 또한 Django의 특징으로 모든 애플리케이션 개발에 반드시 필요한 파일들을 Django가 알아서 생성해주고 개발자가 해당 내용을 채워넣기만 하면 되기 때문에 개발자가 어떤 파일들을 만들어야 할지 고민할 필요가 없음

Django

- Django
  - ✓ 개발 방식 - MTV



- 클라이언트로부터 요청을 받으면 URLconf를 이용하여 URL을 분석
- URL 분석 결과를 통해 해당 URL에 대한 처리를 담당할 뷰를 결정
- 뷰는 자신의 로직을 실행하면서 데이터베이스 처리가 필요하면 모델을 통해 처리하고 그 결과를 리턴
- 뷰는 자신의 로직 처리가 끝나면 템플릿을 사용하여 클라이언트에 전송할 HTML 파일을 생성
- 뷰는 최종 결과로 HTML 파일을 클라이언트에게 보내 응답

Django

- Django
  - ✓ Model
    - 모델이란 사용될 데이터에 대한 정의를 담고 있는 Django의 클래스
    - Django는 ORM 기법을 사용하여 애플리케이션에서 사용할 데이터베이스를 클래스로 매핑해서 코딩하는데 하나의 모델 클래스는 하나의 테이블에 매핑되고 모델 클래스의 속성은 테이블의 컬럼에 매핑
    - ORM 기법을 사용하여 테이블을 클래스로 매핑하면 애플리케이션에서는 데이터베이스에 대한 액세스를 SQL 없이도 인스턴스를 다루는 것처럼 할 수 있어서 편리
    - ORM 기법을 사용하면 SQLite3, MySQL, PostgreSQL 등 데이터베이스 엔진을 변경하더라도 ORM 기법을 이용하는 API는 변경할 필요가 없기 때문에 필요에 따라 데이터베이스 엔진을 훨씬 쉽게 변경할 수 있음
    - ORM(Object-Relational Mapping)은 객체와 관계형 데이터베이스를 연결해주는 역할을 수행하는데 데이터베이스 프로그래밍을 할 때 직접 SQL 언어를 사용해 작업을 수행할 수 있는데 이 경우 개발자는 SQL 및 데이터베이스에 접근하기 위한 드라이버 API 등에 대해 잘 알고 있어야 하고 각 데이터베이스의 SQL을 숙지해야 하는데 ORM에서는 SQL 작업 대신에 인스턴스를 사용해 데이터 작업을 수행할 수 있기 때문에 인스턴스를 대상으로 필요한 작업을 실행하면 ORM이 자동으로 적절한 SQL 구문이나 데이터베이스 API를 호출해서 처리함

Django

- Django
  - ✓ URLconf – URL 정의
    - 클라이언트로부터 요청을 받으면 Django는 가장 먼저 요청에 들어있는 URL을 분석
    - 요청에 들어있는 URL이 urls.py 파일에 정의된 URL 패턴과 매칭되는지를 확인
    - URL을 정의하기 위해서는 urls.py 파일에 URL 과 처리 함수(View 라고 부름)를 매핑하는 python 코드를 작성하면 되는데 이러한 URL/뷰 매핑을 URLconf 라고 함

Django

- Django
  - ✓ View – Logic 정의
    - Django는 웹 요청에 있는 URL을 분석하고 그 결과로 해당 URL에 매핑된 뷰를 호출
    - 뷰는 웹 요청을 받아서 데이터베이스 접속 등 해당 애플리케이션의 로직에 맞는 처리를 하고 그 결과 데이터를 HTML로 변환하기 위하여 템플릿 처리를 한 후에 최종 HTML로 된 응답 데이터를 웹 클라이언트로 반환하는 역할을 수행
    - Django에서의 뷰는 함수 또는 클래스의 메서드로 작성되며 웹 요청을 받고 응답을 리턴
    - 응답은 HTML 데이터일 수도 있고 리다이렉션 명령일 수도 있고 404 에러 메시지일 수도 있음
    - 다양한 형태의 응답 데이터를 만들어내기 위한 로직을 뷰에 작성하는 것
    - 뷰는 보통 views.py 파일에 작성하지만 다른 파일에 작성해도 무방

Django

- Django
  - ✓ Template – 화면 UI 정의
    - Django가 클라이언트에게 반환하는 최종 응답 중 하나는 HTML 텍스트
    - 개발자가 응답에 사용할 *.html 파일을 작성하면 Django는 이를 해석해서 최종 HTML 텍스트 응답을 생성하고 이를 클라이언트에게 전송
    - 클라이언트(보통 웹 브라우저)는 응답으로 받은 HTML 텍스트를 해석해서 우리가 보는 웹 브라우저 화면에 이를 출력
    - 개발자가 작성하는 *.html 파일을 템플릿이라 부르며 여기에 UI 모습을 작성
    - Django는 자체 템플릿 엔진을 갖고 있기 때문에 디자이너도 쉽게 이해할 수 있는 문법을 제공하고 화면의 디자인을 변경할 일이 생기면 디자이너는 프로그램 로직에 상관없이 문법에 맞게 템플릿만 수정하면 되므로 디자이너와 개발자 간에 협업이 편리
    - Django에서 제공하는 템플릿은 템플릿 태그/필터 기능을 사용하여 python 코드를 직접 사용할 수 있기 때문에 더욱 강력하고 확장하기 쉬운 구조로 되어 있음
    - 템플릿 파일은 *.html 확장자를 가지며 Django의 템플릿 시스템 문법에 맞게 작성해야 하는데 유의할 점은 템플릿 파일을 적절한 디렉토리에 위치시켜야 한다는 것인데 Django에서 템플릿 파일을 찾는 방식을 이해하고 있어야 하며 Django는 그에 맞는 위치에 템플릿 파일이 위치해야 템플릿 파일을 찾을 수 있음
    - Django에서 템플릿 파일을 찾을 때는 settings.py 파일에 설정된 항목 중 TEMPLATES 및 INSTALLED_APPS에서 지정된 앱의 디렉토리에서 검색하며 여러 개의 디렉토리를 지정할 수 있는데 지정된 순서대로 디렉토리를 검색하여 템플릿 파일을 찾음

Django

- Django
  - ✓ 캐시 시스템
  - ✓ 다국어 지원
  - ✓ 풍부한 개발 환경
  - ✓ 소스 변경 사항 자동 반영

Django

- Django
  - ✓ MTV 작업
    - 프로젝트 뼈대 만들기: 프로젝트 및 앱 개발에 필요한 디렉토리 와 파일 생성
    - 모델 코딩: 테이블 관련 사항을 개발(models.py, admin.py 파일)
    - URLconf 코딩: URL 및 뷰 매핑 관계를 정의(urls.py 파일)
    - 뷰 코딩: 애플리케이션 로직 개발(views.py 파일)
    - 템플릿 코딩: 화면 UI 개발(templates/ 디렉토리 하위의 *.html 파일들)

Django

- ❖ 프로젝트 생성 및 실행
  - ✓ 프로젝트 생성
    - 가상 환경 생성

      python3 -m venv myvenv
      - ◆ 명령을 수행하고 나면 mysite 디렉토리 와 myvenv 디렉토리가 생성됨
    - 가상 환경 활성화

      MAC: source myvenv/bin/activate

      Windows: myvenv₩Scripts₩activate
    - Django 설치

      pip install django
    - 프로젝트 생성: mysite

      django-admin startproject mysite .
      - ◆ 명령을 수행하고 나면 mysite 디렉토리가 생성됨
      - ◆ mysite 프로젝트는 프로젝트와 관련된 디렉토리/파일을 모으는 역할만 담당하므로 이름을 변경해도 됨

Django

- ❖ 프로젝트 생성 및 실행
  - ✓ Django App
    - Django에서 사용하는 python 패키지
    - Django App 패키지는 그 안에 자신의 모델(model), 뷰(view), 템플릿(template), URL 매핑 등을 독자적으로 가지고 있으며 일반적으로 하나의 Django 프로젝트는 하나 이상의 Django App으로 구성되어 있음
    - 규모가 큰 Django 프로젝트는 여러 개의 Django App들을 모듈화하여 구성하는데 모듈화 된 App들로 구성하면 개발 및 유지 보수가 효율적이기 때문
    - Django App을 생성
      - manage.py startapp App이름 – 루트 디렉토리에서 수행

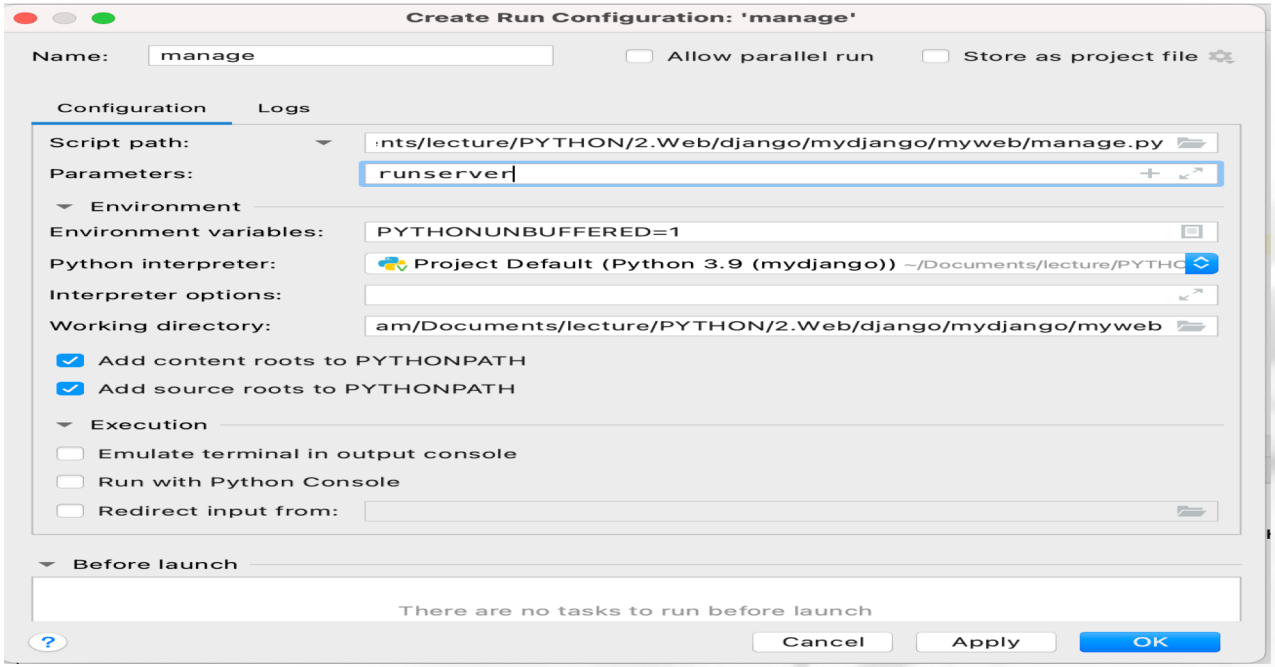
        python manage.py startapp myweb

Django

- ❖ 프로젝트 생성 및 실행
  - ✓ 터미널에서 실행
    - ● 테스트 용으로 실행

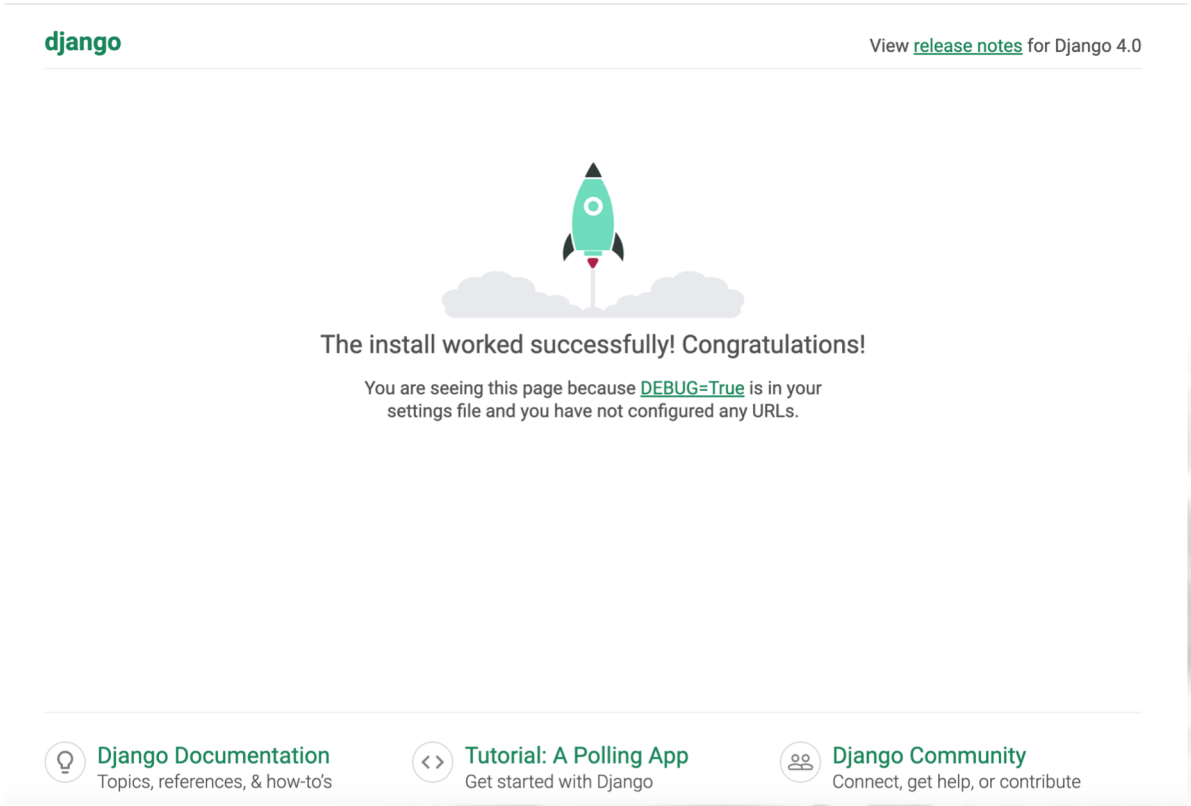
      python manage.py runserver IP주소:포트번호

      - ◆ 포트 번호를 생략하면 8000 이며 IP 주소를 생략하면 127.0.0.1 으로 실행
  - ✓ PyCharm에서 실행
    - ● [Run] – [Run] 메뉴를 클릭하고 Parameter에 runserver 추가한 후 실행



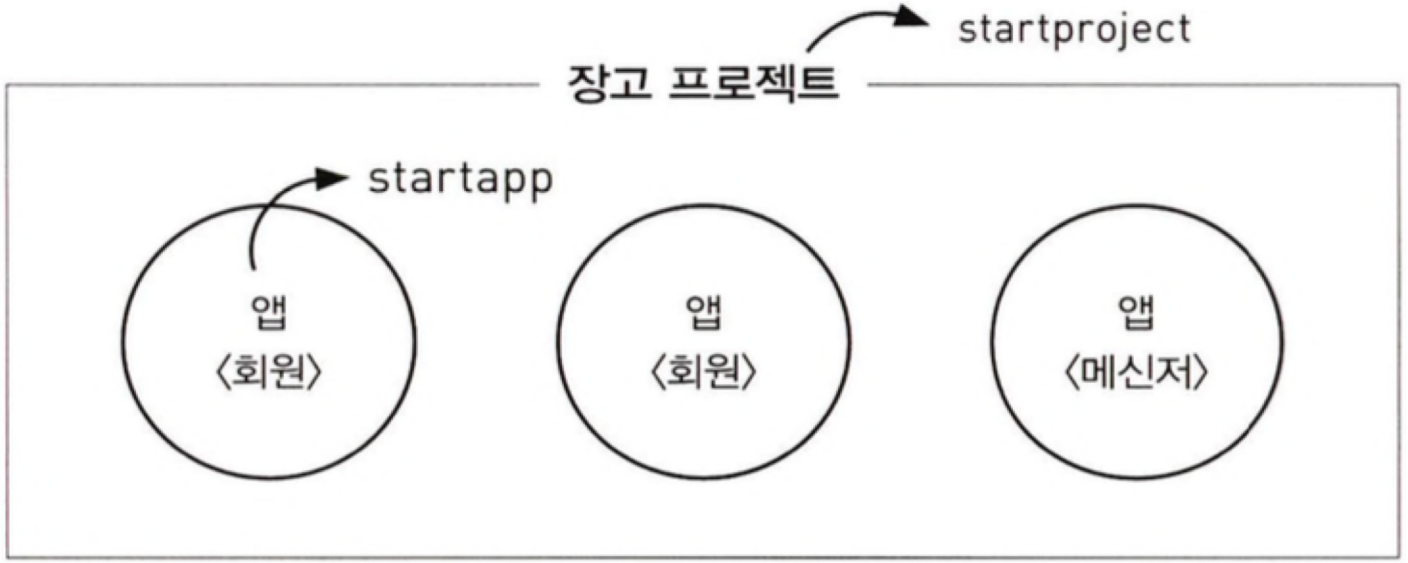
Django

- ❖ 프로젝트 생성 및 실행
  - ✓ 사이트 접속
    - ● 브라우저에서 http://127.0.0.1:8000 으로 접속



Django

- ❖ 프로젝트 구조 및 설정
  - ✓ 프로젝트 와 앱



Django

- ❖ 프로젝트 구조 및 설정
  - ✓ settings.py
    - ● 프로젝트 설정 파일

      from pathlib import Path

      BASE_DIR = Path(__file__).resolve().parent.parent

      - ◆ pathlib 모듈은 다른 운영 체제에 적합한 의미 체계를 가진 파일 시스템 경로를 나타내는 클래스를 제공
      - ◆ 코드가 실행되는 플랫폼의 구상 경로를 인스턴스화 하는 것
      - ◆ Path(__file__)은 현재 파일의 구상 경로를 인스턴스화 하고 resolve() 메서드를 통해 절대경로를 반환
      - ◆ 부모 경로로 2번 옮겨가면 django 프로젝트가 생성된 위치가 BASE_DIR 변수로 인스턴스화 된 것

Django

- ❖ 프로젝트 구조 및 설정
  - ✓ settings.py
    - ● 프로젝트 설정 파일

      SECRET_KEY = "django-insecure-=2kj2hzkzdd=d9lp&ut6*0qpw#s&zb22cj6&@aj%pg^#7a^-l9"

      - ◆ SECRET_KEY는 특정한 장고 설치를 위한 키로 암호화 서명을 제공하는데 사용되기도 하고 유니크한 값으로 설정됨
      - ◆ 프로젝트를 생성할 때 자동으로 SECRET_KEY가 생성됨
      - ◆ 키를 사용할 때 텍스트 또는 바이트라고 가정하면 안되는데 사용할 땐 꼭 force_str() 또는 force_byte()해서 변환해야 하며 SECRET_KEY가 없으면 프로젝트는 실행되지 않음
      - ◆ SECRET_KEY는 다음 경우에 사용
        - ▪ django.contrib.sessions.backends.cache 이외에 다른 백엔드 세션을 사용하거나 default인 get_session_auth_hash() 메서드로 SHA-256 해싱할 때의 모든 세션
        - ▪ CookieStorage나 FallbackStorage를 사용할 때의 모든 메세지
        - ▪ 다른 키가 제공되지 않았을 때 암호화 서명

Django

- ❖ 프로젝트 구조 및 설정
  - ✓ settings.py
    - ● 프로젝트 설정 파일

      DEBUG = True

      - ◆ 앱의 실행 모드 설정
      - ◆ 보안 위험을 초래하거나 성능을 저하시킬 수 있는 디버깅 정보를 프로덕션 환경에 의도치 않게 포함하는 것을 방지하는 데 도움을 줌

```
DEBUG = True


if DEBUG:
    # DEBUG Mode 일 때 수행할 코드
    print("Debugging information enabled")
else:
    # PRODUCTUON Mode 일 때 수행할 코드
    print("Debugging information disabled")
```

Django

- ❖ 프로젝트 구조 및 설정
  - ✓ settings.py
    - ● 프로젝트 설정 파일

      ALLOWED_HOSTS = [ ]

      - ◆ 운영 모드인 경우는 ALLOWD_HOSTS에 반드시 서버의 IP 나 도메인을 지정해야 하고 개발 모드인 경우에는 값을 지정하지 않아도 ['localhost', '127.0.0.1']로 간주
      - ◆ Django의 runserver를 기동할 서버의 IP가 127.0.0.1 뿐이라면 [ ]로 해도 되지만 운영 모드로 사용할 때는 아래와 같이 지정
        - ▪ ALLOWED_HOSTS = ['서버의 IP', 'localhost', '127.0.0.1']
      - ◆ 모든 아이피에서 실행 가능 하도록하고자 할 때는 [' * ']

Django

- 프로젝트 구조 및 설정
  - ✓ settings.py
    - 프로젝트 설정 파일

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "myweb",
]
```

      - ◆ 사용할 Application을 등록하는 부분으로 Application을 추가하면 여기에 등록을 해야만 Application 내의 코드를 읽어냄

Django

- ❖ 프로젝트 구조 및 설정
  - ✓ settings.py
    - ● 프로젝트 설정 파일

```
MIDDLEWARE = [
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
]
```

      - ◆ 미들웨어를 설정하는 부분으로 클라이언트의 요청을 처리하기 전이나 처리한 후에 수행할 동작을 등록
      - ◆ 현재는 보안, 세션 사용, www 추가, Csrf, 로그인 정보 저장, 클릭 재킹 관련된 설정이 이미 작성되어 있음

Django

- ❖ 프로젝트 구조 및 설정
  - ✓ settings.py
    - ● 프로젝트 설정 파일

      ROOT_URLCONF = "config.urls"
      - ◆ Root url을 구성하는 string

Django

- 프로젝트 구조 및 설정
  - ✓ settings.py
    - 프로젝트 설정 파일

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

      - ◆ 템플릿 파일이 위치한 디렉토리를 설정하는 곳으로 일반적으로 DIRS 만 수정해서 사용

Django

- 프로젝트 구조 및 설정
  - ✓ settings.py
    - 프로젝트 설정 파일

      WSGI_APPLICATION = "mysite.wsgi.application"

      - 장고가 사용할 wsgi(Web Server Gateway Interface - 파이썬 웹 응용 프로그램을 위한 것으로 웹서버와 파이썬으로 작성된 웹 응용 프로그램 간의 표준 인터페이스) 애플리케이션의 경로

Django

- ❖ 프로젝트 구조 및 설정
  - ✓ settings.py
    - ● 프로젝트 설정 파일

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

      - ◆ 데이터베이스 설정
      - ◆ 기본은 sqlite3를 사용하도록 설정되어 있음

Django

- ❖ 프로젝트 구조 및 설정
  - ✓ settings.py
    - ● 프로젝트 설정 파일

```
AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME":
"django.contrib.auth.password_validation.UserAttributeSimilarityValidator",
    },
    {
        "NAME":
"django.contrib.auth.password_validation.MinimumLengthValidator",
    },
    {
        "NAME":
"django.contrib.auth.password_validation.CommonPasswordValidator",
    },
    {
        "NAME":
"django.contrib.auth.password_validation.NumericPasswordValidator",
    },
]
```

◆ 패스워드의 보안 강도를 체크하는 validatar 리스트

Django

- ❖ 프로젝트 구조 및 설정
  - ✓ settings.py
    - ● 프로젝트 설정 파일

```
LANGUAGE_CODE = "en-us"

#TIME_ZONE = 'UTC'
TIME_ZONE = 'Asia/Seoul'

USE_I18N = True

USE_TZ = True
```

      - ▪ 언어 코드, 타임존, 장고의 번역 시스템을 활성화 여부, 내부적으로 날짜/시간 사용 여부를 설정

Django

- ❖ 프로젝트 구조 및 설정
  - ✓ settings.py
    - ● 프로젝트 설정 파일

      STATIC_URL = '/static/'

      STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]

      - ◆ 정적 파일에 관한 설정으로 STATICFILES_DIRS 항목을 이용하면 프로젝트 정적 파일이 위치한 디렉토리를 의미하는데 수동으로 직접 지정 가능

Django

- ❖ 프로젝트 구조 및 설정
  - ✓ settings.py
    - ● 프로젝트 설정 파일

      DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"

      - ◆ AutoField는 Django ORM에서 자동적으로 생성되는 Auto increment Primary key
      - ◆ 기본값이 AutoField로 int 크기를 갖게 되는데 거의 매번 bigint 사이즈 사용을 위해 Model을 정의할 때 bigint로 정의 하기 모델마다 별도 선언을 해주었는데 이제는 settings.py에 DEFAULT_AUTO_FIELD 설정을 하면 전역적으로 설정이 반영됨
      - ◆ 이미 진행하고 있는 프로젝트에는 해당 기능을 넣기는 애매할 수 있으나 처음 시작할 때에는 해당 옵션을 포함하는 것을 추천

Django

- 프로젝트 구조 및 설정
  - ✓ settings.py
    - 프로젝트 설정 파일
      - ◆ 파일 업로드를 하고자 하는 경우 설정 추가

```
MEDIAURL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

Django

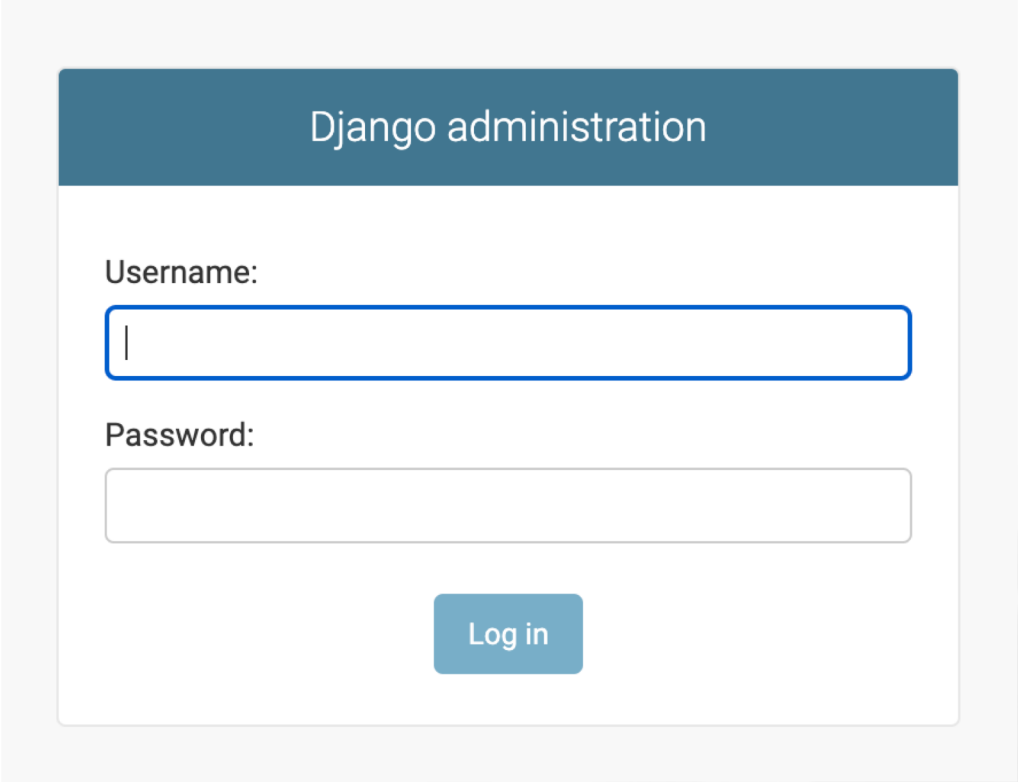
- ❖ 프로젝트 구조 및 설정
  - ✓ urls.py
    - 프로젝트의 URL을 등록하는 파일

```
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path("admin/", admin.site.urls),
]
```

    - 하나의 파일에 정의할 수도 있고 여러 개의 파일에 정의하는 것도 가능
    - project 전체 URL을 정의하는 프로젝트 URL 과 App 마다 정의하는 App URL 이렇게 2 계층으로 나누어서 설정하는 것도 가능
    - 여러 개로 구현하는 이유는 모듈을 계층적으로 구성하므로 변경도 쉽고 확장도 용이해 지기 때문이며 이미 개발해 놓은 앱을 다른 프로젝트에서 사용하는 경우에도 urls.py 파일을 수정없이 재활용할 수 있는 장점이 생기며 URL 패턴 별로 이름을 지정할 수 있고 패턴 그룹에 대해 namespace을 지정할 수 도 있는데 이는 reverse() 함수나 {% url %} 템플릿 태그를 사용해 소스에 URL을 하드 코딩하지 않아도 필요한 URL을 추출할 수 있음

Django

- ❖ 관리자 생성 및 접속
  - ✓ 관리자 사이트 접속
    - ● 브라우저에서 http://127.0.0.1:8000/amin 으로 접속



Django

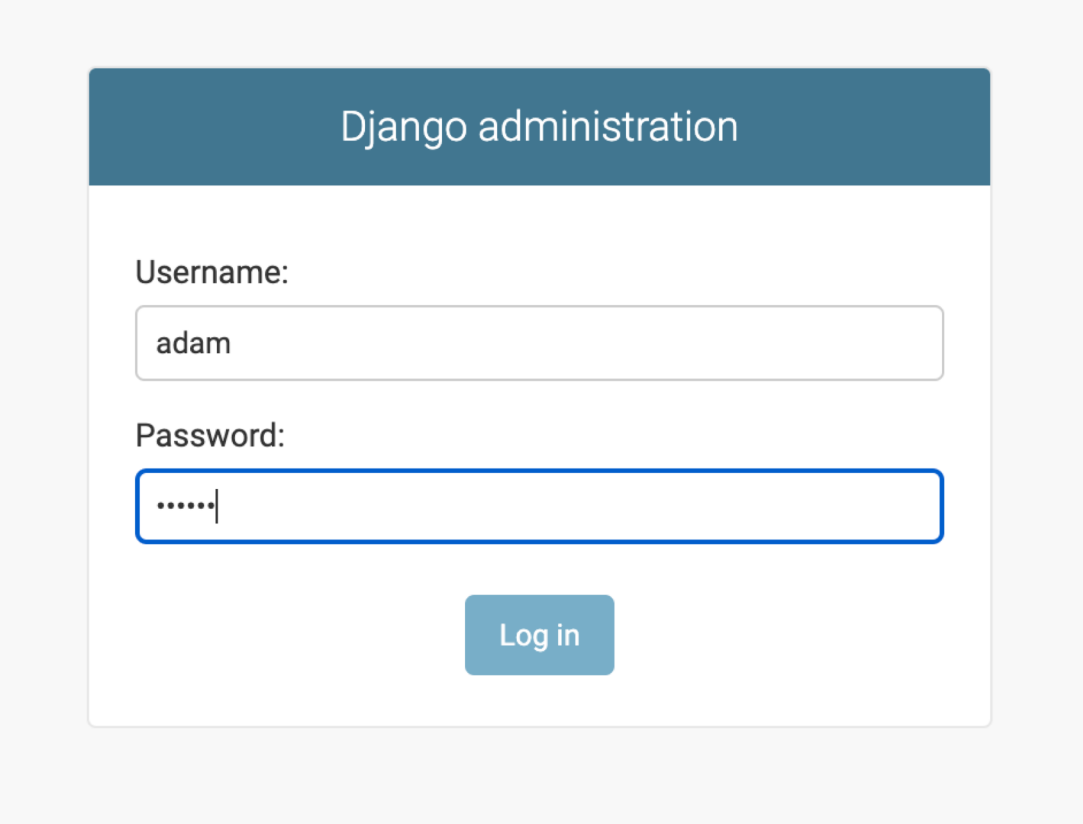
- ❖ 관리자 생성 및 접속
  - ✓ 기본 테이블 생성
    - 데이터베이스에 변경 사항이 있을 때 이를 반영하는 python manage.py migrate 명령을 터미널에서 실행
    - Django는 모든 웹 프로젝트 개발 시 반드시 사용자 와 그룹 테이블 등이 필요하다는 가정 하에 설계되었기 때문에 테이블을 전혀 만들지 않았더라도 Django가 미리 정의해 둔 사용자 및 그룹 테이블 등 을 만들어주기 위해서 프로젝트 개발 시작 시점에 이 명령을 실행하는데 명령을 실행하면 migrate 명령에 대한 로그가 보이고 실행 결과로 SQLite3 데이터베이스 파일인 db.sqlite3 파일이 생성됨

Django

- ❖ 관리자 생성 및 접속
  - ✓ 관리자 계정 생성
    - Username 과 Password를 넣어야 하는데 아직 생성되지 않음
    - 관리자 생성
      - (venv) (base) adam@ADAMui-MacBookPro mysite % python manage.py makemigrations
      - (venv) (base) adam@ADAMui-MacBookPro mysite % python manage.py migrate
      - (venv) (base) adam@ADAMui-MacBookPro mysite % python manage.py createsuperuser
      - Username (leave blank to use 'adam'): adam
      - Email address: ggangpae1@gmail.com
      - Password: ******
      - Password (again): *****
      - Bypass password validation and create user anyway? [y/N]: y
      - Superuser created successfully.

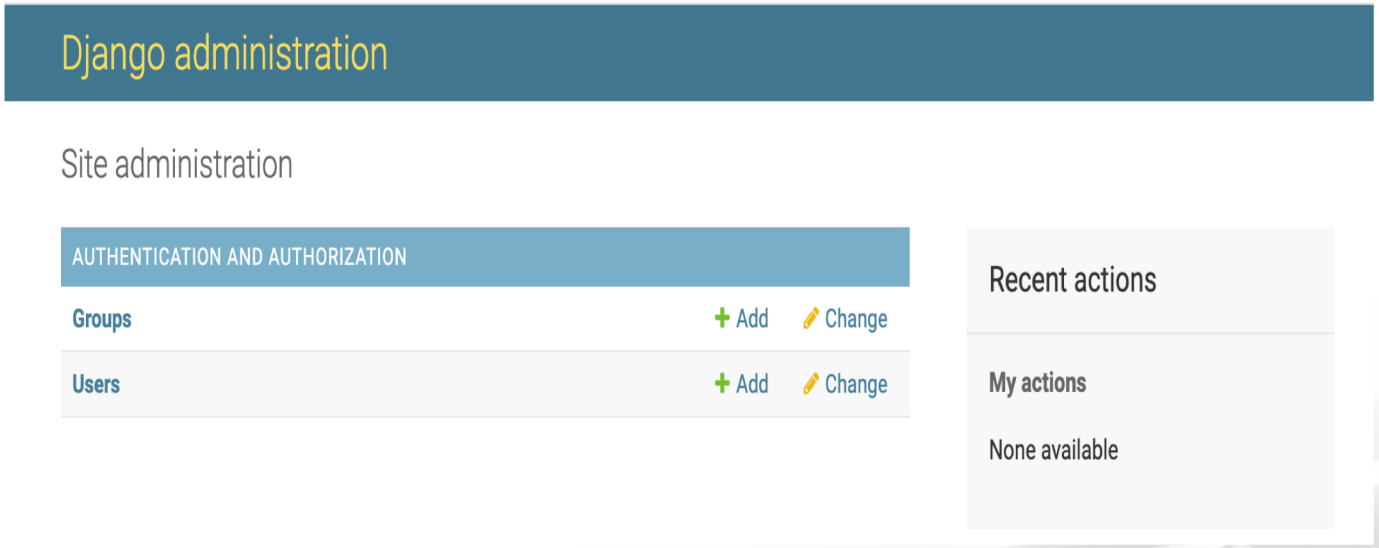
Django

- ❖ 관리자 생성 및 접속
  - ✓ 관리자 사이트 접속
    - ● 브라우저에서 http://127.0.0.1:8000/amin 으로 접속하고 설정한 아이디 와 비번 입력



Django

- ❖ 관리자 생성 및 접속
  - ✓ 관리자 사이트 접속
    - ● 관리자 페이지



Django

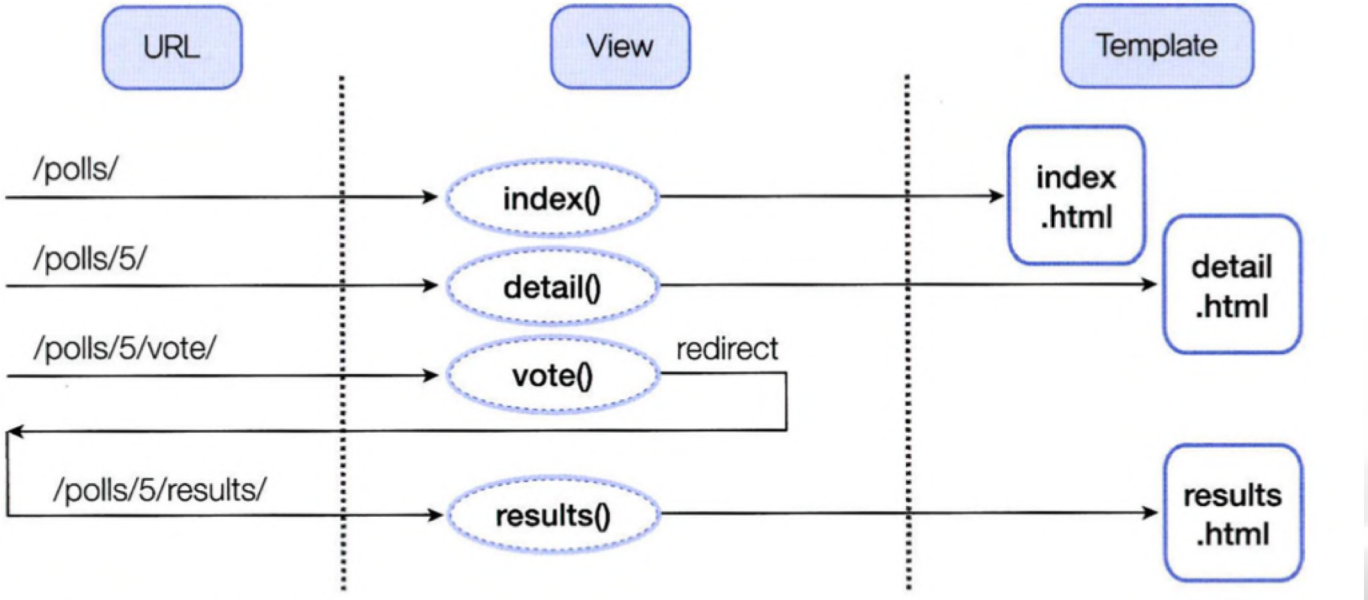
- ❖ 관리자 생성 및 접속
  - ✓ 관리자 사이트 접속
    - ● 관리자 페이지
      - ◆ Django에서 만들어준 Users와 Groups 테이블이 생성된 것을 확인할 수 있음
      - ◆ Admin 사이트에서 Users와 Groups 테이블을 포함하여 테이블에 대한 데이터의 입력, 변경, 삭제 등의 작업을 할 수 있음
      - ◆ Admin 화면에서 기본적으로 Users와 Groups 테이블이 보이는 것은 이미 settings.py 파일에 django.contrib.auth 애플리케이션이 등록되어 있기 때문인데 Django에서 기본으로 제공하는 auth 앱에 Users와 Groups 테이블이 미리 정의되어 있는 것

Django

- View 설정
  - ✓ views.py
    - views.py는 뷰 로직을 코딩하는 가장 중요한 파일
    - 간단한 로직이면 몇 줄만 코딩하면 되지만 프로젝트 개발 범위가 커짐에 따라 로직도 점점 복잡해지고 views.py 파일의 코드 양이 많아지므로 가독성과 유지보수 편리, 재활용 등을 고려
    - Django에서는 뷰 로직을 함수로 코딩할 것인지 클래스로 코딩할 것인지에 따라 함수형 뷰(Function-based view) 와 클래스형 뷰(Class-based view)로 구분
    - 개발자가 편한 방식으로 코딩하면 되므로 보통은 하나의 프로젝트에 둘 다 사용하는 경우가 많지만 클래스형 뷰를 사용하는 것이 Django가 제공하는 제네릭 뷰를 사용할 수 있고 재활용 및 확장성 측면에서 유리
    - Django는 다른 프레임워크에 비해 뷰 작성에 편리한 기능을 많이 제공하고 있는데 대표적인 것이 단축 함수 및 클래스 형 제네릭 뷰인데 쉽고 빠르게 웹 프로그래밍을 할 수 있도록 도와주는 대표적인 기능

Django

- View 설정
  - ✓ 처리 흐름



Django

- View 설정
  - ✓ URLconf
    - urls.py 파일에 작성
      - URL/뷰 매핑을 정의하는 방식은 항상 동일하며 URL 패턴 매칭은 위에서 아래로 진행하므로 정의하는 순서에 유의
      - 여러 개의 파일에 나누어 작성하는 것도 가능
    - path 함수
      - URL/뷰 매핑을 정의하는 함수로 route, view 2개의 필수 인자와 kwargs, name 2개의 선택 인자를 받아서 처리
      - route: URL 패턴을 표현하는 문자열
      - view: URL 스트링이 매칭되면 호출되는 뷰 함수로 HttpRequest 객체와 URL 스트링에서 추출된 항목이 뷰 함수의 인자로 전달됨
      - kwargs: URL 스트링에서 추출된 항목 외에 추가적인 인자를 뷰 함수에 전달할 때 python 의 dict 타입으로 인자를 정의
      - name: 각 URL 패턴 별로 이름을 붙여줌

Django

- View 설정
  - ✓ URLconf – 요청 처리
    - mysite/urls.py 파일을 수정

```
from django.contrib import admin
from django.urls import path
from myweb import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index)
]
```

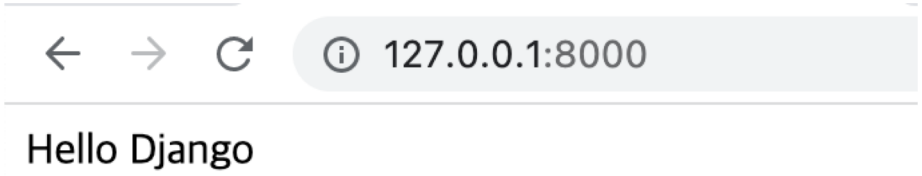
Django

- View 설정
  - ✓ URLconf – 요청 처리
    - 뷰 함수와 템플릿은 서로에게 영향을 미치기 때문에 보통 같이 작업
    - ui 화면을 생각하면서 로직을 풀어나가는 것이 쉽기 때문에 뷰보다는 템플릿을 먼저 코딩하는 경우가 많음
    - myweb/views.py 파일을 수정

```
from django.shortcuts import render
from django.http import HttpResponse

# Create your views here.
def index(request):
    return HttpResponse("Hello Django")
```

Django

- View 설정
  - ✓ 클라이언트가 전송한 데이터 읽기
    - 결과 확인: 브라우저를 실행하고 localhost:8000을 입력 한 후 확인



Django

- View 설정
  - ✓ View
    - Django에서의 뷰(View)는 다른 MVC Framework에서의 Controller 와 비슷한 역할
    - View는 클라이언트의 데이터를 받아서 적절한 처리를 수행한 후 웹 페이지 결과를 만들도록 컨트롤하는 역할
    - View들은 Django App 안의 views.py 라는 파일에 정의하게 되는데 각 함수가 하나의 View를 정의
    - View는 HTTP Request를 입력 파라미터로 받아들이고 HTTP Response를 리턴

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("<h1>Hello, World!</h1>")
```

      - ◆ 하나의 View 함수를 표현한 것인 이 함수는 입력으로 항상 request(요청)를 받아들이고 response(응답)를 리턴
      - ◆ 간단한 HTML Text를 포함한 HttpResponse() 객체를 리턴
      - ◆ Django 에서는 좀 더 복잡한 HTML을 처리하기 위해 뷰 템플릿(Template)을 사용

Django

- Template
  - ✓ 개요
    - Django에서의 템플릿(Template)은 MVC Framework에서의 View와 비슷한 역할
    - 템플릿(Template)은 View로부터 전달된 데이터를 템플릿에 적용하여 Dynamic 한 웹 페이지를 만드는데 사용
    - Template은 HTML 파일로 Django App 디렉토리 밑에 templates 라는 서브 디렉토리를 만들고 그 안에 생성
    - Django 개발 가이드라인은 App디렉토리/templates/App명/템플릿이름 처럼 각 App 디렉토리 밑에 templates 서브 디렉토리를 만들고 다시 그 안에 App명을 사용하여 서브 디렉토리를 만든 후 템플릿 파일을 그 안에 넣기를 권장

      (예: /home/templates/home/index.html)
    - 복수의 App들이 동일한 이름의 템플릿을 가진 경우 View에서 잘못된 템플릿을 가져올 수 있기 때문인데 App1에 create.html이 있고 App2에 동일한 create.html 템플릿이 있는 경우 App2의 View에서 create.html를 지정하면 App1의 create.html을 사용하게 되는데 템플릿을 찾을 때 자신의 App 내의 템플릿을 먼저 찾는 것이 아니라 전체 App들의 템플릿 디렉토리들을 처음부터 순서대로 찾기 때문
    - View에서 App2/create.html 과 같이 템플릿 명을 지정하면 이런 혼동은 없어짐
    - 템플릿은 순수하게 HTML로만 쓰여진 Static HTML 파일 일 수 있지만 View로부터 데이터를 전달받아 HTML 템플릿 안에 그 데이터를 동적으로 치환해서 사용

Django

- Template
  - ✓ 개요
    - 템플릿 디렉토리는 프로젝트 템플릿 디렉토리 와 애플리케이션 템플릿 디렉토리로 구분해서 사용하는데 프로젝트 템플릿 디렉토리는 TEMPLATES 설정의 DIRS 항목에 지정된 디렉토리 이고 애플리케이션 템플릿 디렉토리는 각 애플리케이션 디렉토리마다 존재하는 templates/ 디렉토리
    - 프로젝트 템플릿 디렉토리에는 base.html 등 전체 프로젝트의 룩앤필(Look And Feel)에 관련된 파일들을 모아두고 각 애플리케이션에서 사용하는 템플릿 파일들은 앱 템플릿 디렉토리에 위치시킴
    - Django에서 템플릿 파일들을 찾는 순서도 알아두어야 하는데 별도로 변경하지 않는다면 프로젝트 템플릿 디렉토리를 먼저 검색하고 그 다음 애플리케이션 템플릿 디렉토리를 검색하는데 애플리케이션 템플릿 디렉토리도 각 애플리케이션 마다 하나씩 여러 개가 존재하는데 INSTALED_APPS 설정 항목에 등록된 순서대로 검색

Django

- Template
  - ✓ View에서의 Template 처리
    - render는 django.shortcuts 패키지에 있는 render 함수 이용
    - 파라미터
      - 첫번째 파라미터는 request
      - 두번째 파라미터는 템플릿 이름
      - 세번째 파라미터는 Optional 인데 View 에서 템플릿에 전달한 데이터를 Dictionary로 전달하는데 Dictionary의 Key는 템플릿에서 사용할 키(or 변수명)이고 Value는 전달하는 데이터의 내용을 저장

Django

- Template
  - ✓ View에서의 Template 처리
    - myweb/views.py 파일의 index 함수 수정

```
from django.shortcuts import render
from django.http import HttpResponse

def index(request):
    msg = 'My Message'
    return render(request, 'index.html', {'message': msg})
```

Django

- Template
  - ✓ View에서의 Template 처리
    - myweb 디렉토리에 templates 디렉토리를 생성
    - templates 디렉토리 안에 index.html 파일을 생성하고 작성

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <h1>{{message}}</h1>
</body>
</html>
```

      - ◆ body 태그 안에 message를 보면 {{ }} 으로 둘러싸인 것을 볼 수 있는데 Django 의 템플릿에서 {{ 변수명 }} 은 해당 변수의 값을 그 자리에 치환하라는 의미
      - ◆ Django Template은 또한 View로 부터 전달된 다양한 데이터들을 템플릿에 편리하게 넣을 수 있도록 여러 템플릿 태그({% 탬플릿태그 %} 와 같은 형태)들을 제공

Django

- Template
  - ✓ Django 템플릿 언어
    - Django 템플릿에서 사용하는 특별한 태그 및 문법을 Django 템플릿 언어(Django Template Language)라 부르는데 템플릿 언어는 크게 템플릿 변수, 템플릿 태그, 템플릿 필터, 코멘트 등으로 나눌 수 있음
    - 템플릿 변수: {{ 와 }} 으로 둘러 싸여 있는 변수로서 그 변수의 값이 해당 위치에 치환되는데 변수에는 기본 자료형의 데이터를 갖는 변수 혹은 객체의 속성 등을 넣을 수 있음

```
<h4>
        Name : {{ name }}
        Type : {{ vip.key }}
</h4>
```

Django

- Template
  - ✓ Django 템플릿 언어
    - 템플릿 태그는 {% 와 %} 으로 둘러 싸여 있는데 이 태그 안에는 if, for 루프 같은 Flow Control 문장에서부터 웹 컨트롤 처럼 내부 처리 결과를 직접 덤프하는 등 여러 가지를 작성할 수 있음
    - if 와 for 태그를 사용한 예

```
{% if count > 0 %}
    Data Count = {{ count }}
{% else %}
    No Data
{% endif %}

{% for item in dataList %}
  <li>{{ item.name }}</li>
{% endfor %}
```

Django

- Template
  - ✓ Django 템플릿 언어
    - 템플릿 필터: 변수의 값을 특정한 포맷으로 변형하는 기능을 하는데 날짜를 특정 날짜 포맷으로 변경하거나 문자열을 대소문자로 변경하는 등의 작업을 할 수 있음

      날짜 포맷 지정

      {{ createDate|date:"Y-m-d" }}

      소문자로 변경

      {{ lastName|lower }}
    - 코멘트: 템플릿에서 코멘트를 넣는 방법은 크게 2가지인데 한 라인에 코멘트를 적용할 때는 코멘트를 {# 과 #} 으로 둘러싸면 되고 복수 라인 문장을 코멘트할 경우는 문장들을 {% comment %} 태그와 {% endcomment %}로 둘러싸면 됨

      ```
      {# 1 라인 코멘트 #}
      {% comment %}
            <p>
               불필요한 블럭
            </p>
      {% endcomment %}
      ```

Django

- Template
  - ✓ Django 템플릿 언어
    - HTML Escape: HTML 내용 중에 <, >, ', ", & 등과 같은 문자들이 있을 경우 이를 그 문자에 상응하는 HTML Entity로 변환해 주어야 하는데, Django 템플릿에서 이러한 작업을 자동으로 처리해 주기 위해 {% autoescape on %} 템플릿 태그나 escape 라는 필터를 사용
    - content 라는 변수에 인용부호가 들어 있다고 했을 때 아래와 같이 autoescape 태그나 escape 필터를 사용해서 자동으로 변환하게 할 수 있는데 만약 이러한 변환을 하지 않으면 HTML이 중간에 깨지게 됨

```
{% autoescape on %}      # autoescape 태그
    {{ content }}
{% endautoescape %}


{{ content|escape }}     # escape 필터
```

    - HTML escape 혹은 HTML 인코딩 기능을 사용하지 않고, <, >, ', ", & 이 들어간 문자열을 HTML에서 사용하고자 한다면 각 문자를 HTML Entity로 미리 변환해 주어야 함

Django

- 클라이언트가 전송한 데이터 읽기
  - ✓ URL에 포함된 데이터 처리
    - urls.py 파일에 요청 경로 와 요청 처리 함수 매핑

      path('요청경로/<자료형:변수이름", 요청 처리 함수)
    - urls.py 파일에 요청 처리 함수 작성

      def 요청처리함수(request, 변수이름):

      처리 내용

Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ URL에 포함된 데이터 처리
    - ● 프로젝트의 urls.py 파일의 urlpattens에 URL 처리 내용 추가

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index),
    path('get/<str:itemid>', views.getItem)
]
```

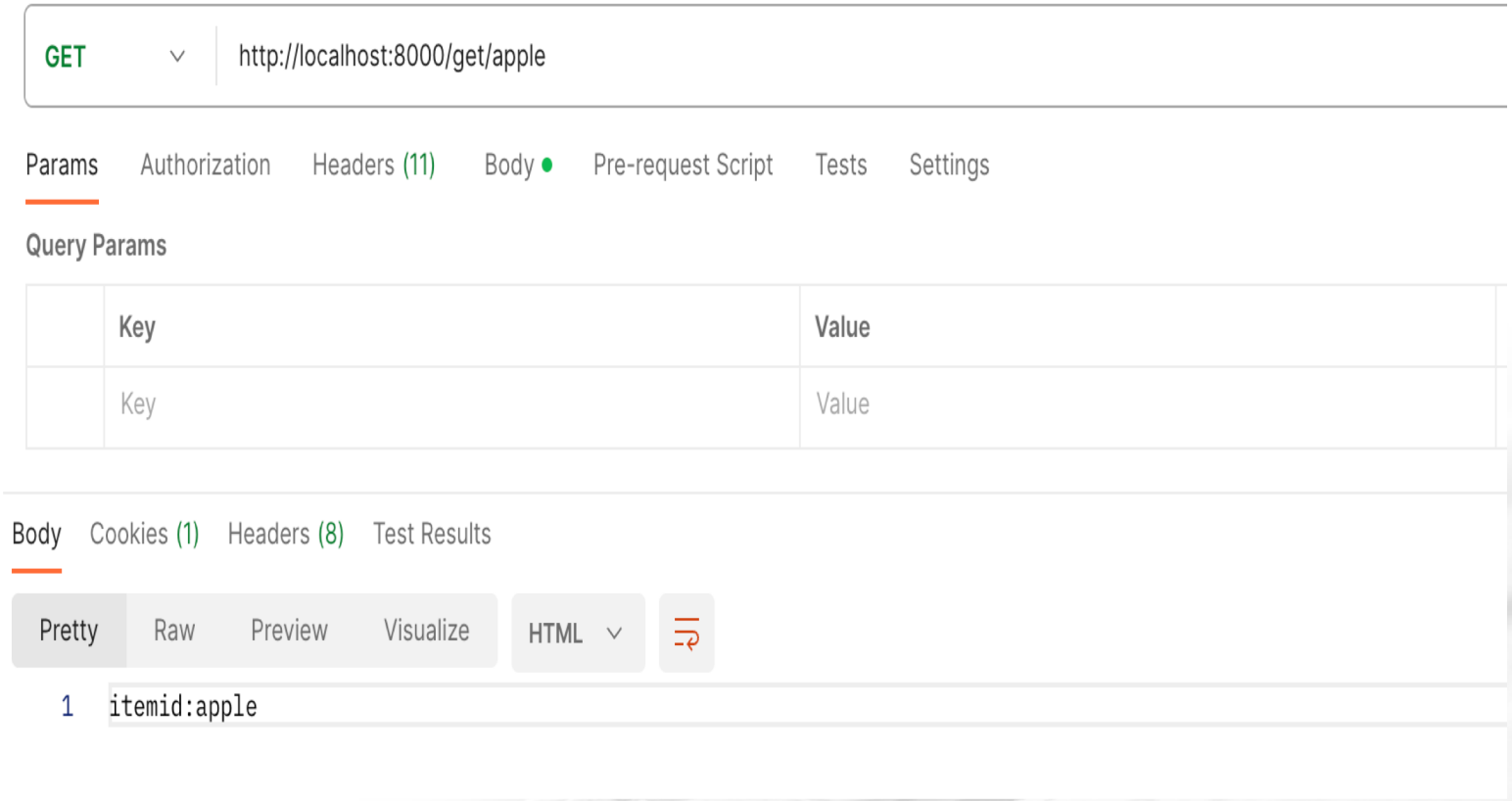
Django

- 클라이언트가 전송한 데이터 읽기
  - ✓ URL에 포함된 데이터 처리
    - ● 애플리케이션의 views.py 파일에 요청 처리 함수 추가

```
def getItem(request, itemid):
    result = itemid
    return HttpResponse("itemid:" + itemid)
```

Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ URL에 포함된 데이터 처리
    - 결과 확인



Django

- 클라이언트가 전송한 데이터 읽기
  - ✓ GET 방식의 Query String 처리
    - GET 방식에서의 데이터 전송
      - 쿼리 스트링은 URL + ?+ 데이터이름=값 형태로 표현
      - 여러 개가 존재하는 경우 값 뒤에 &를 추가하고 데이터이름 = 값의 형태로 표현
      - form에서 전송 방식을 get으로 설정한 경우도 URL은 동일한 형태로 만들어 짐
      - 데이터가 URL에 표시되므로 보안에 주의해야 하며 사이즈가 큰 데이터(file 등)는 전송할 수 없음
      - 서버에서 읽을 때는 request.GET[" 데이터이름"] 의 형태로 읽을 수 있음

Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ GET 방식의 Query String 처리
    - ● 프로젝트의 urls.py 파일의 urlpattens에 URL 처리 내용 추가

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index),
    path('get/<str:itemid>', views.getItem),
    path('querystring', views.queryString),
]
```

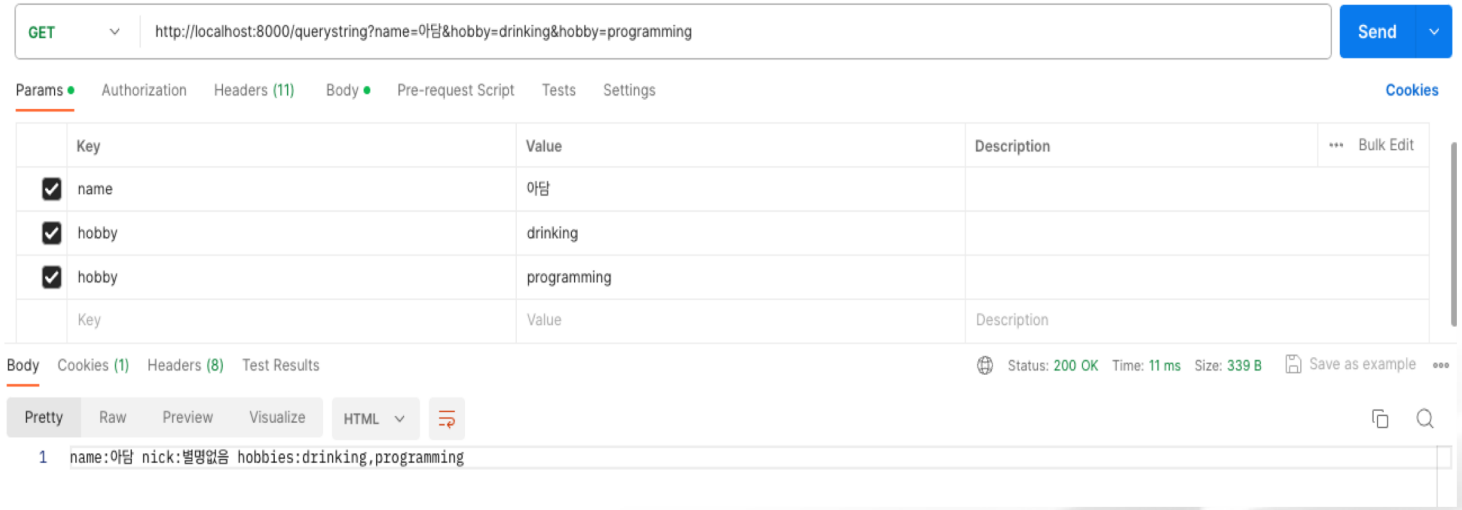
Django

- 클라이언트가 전송한 데이터 읽기
  - ✓ GET 방식의 Query String 처리
    - 애플리케이션의 views.py 파일에 요청 처리 함수 추가

```
def queryString(request):
    name = request.GET.get("name")

    #데이터가 Optional 인 경우
    #nick = request.GET["nick"]
    nick = request.GET.get("nick")
    if not nick:
        nick = "별명없음"

    hobbies = request.GET.getlist("hobby")

    return HttpResponse("name:" + name + " nick:" + nick + " hobbies:" +
','.join(hobbies))
```

Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ GET 방식의 Query String 처리
    - 확인



Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ POST 방식의 데이터 처리
    - Request Body 처리
      - ◆ 서버에서 읽을 때는 json 모듈을 이용해서 처리
      - ◆ json.loads(request.body)
    - Form Data 처리
      - ◆ request.POST를 이용해서 처리

Django

- 클라이언트가 전송한 데이터 읽기
  - ✓ POST 방식의 데이터 처리
    - 프로젝트의 urls.py 파일의 urlpattens에 URL 처리 내용 추가

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index),
    path('get/<str:itemid>', views.getItem),
    path('querystring', views.queryString),
    path('requestbody', views.requestBody),
    path('formdata', views.formData),
]
```

Django

- 클라이언트가 전송한 데이터 읽기
  - ✓ POST 방식의 데이터 처리
    - 애플리케이션의 views.py 파일에 요청 처리 함수 추가

```
import json
def requestBody(request):
    user = json.loads(request.body)
    return HttpResponse("name:" + user["name"] + " nick:" + user["nick"])

def formData(request):
    name = request.POST.get("name")
    return HttpResponse("name:" + name)
```

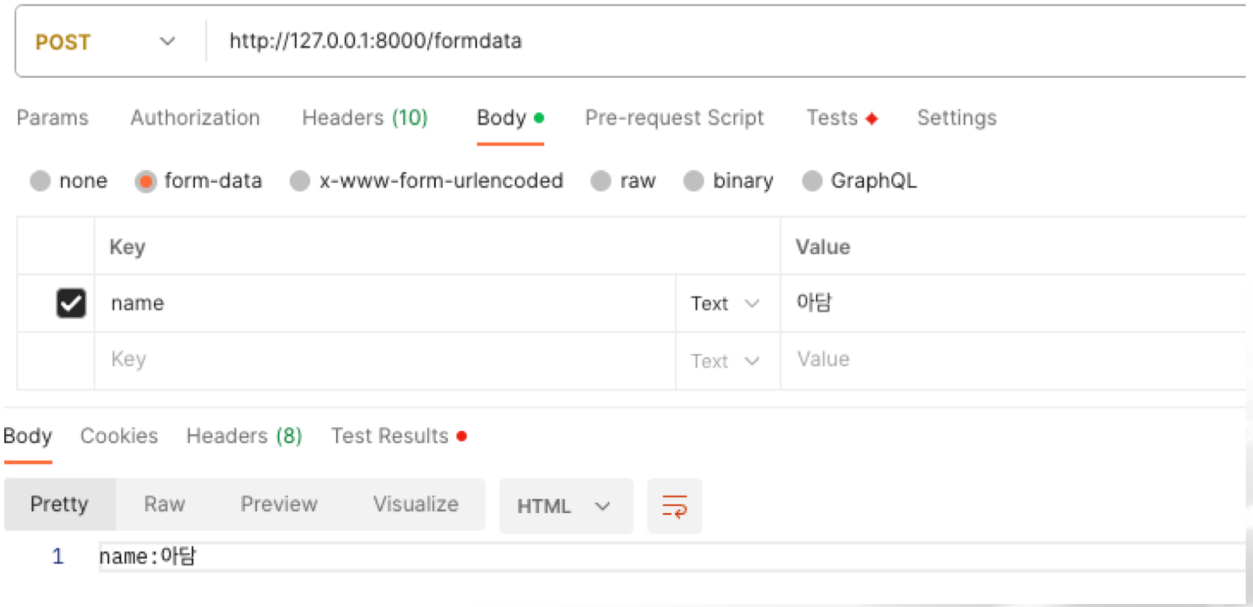
Django

- 클라이언트가 전송한 데이터 읽기
  - ✓ POST 방식의 데이터 처리
    - 테스트를 위해서 settings.py에서 CSRF 사용을 위한 미들웨어를 주석 처리

```
MIDDLEWARE = [
    "django.middleware.security.SecurityMiddleware",
    "django.contrib.sessions.middleware.SessionMiddleware",
    "django.middleware.common.CommonMiddleware",
    #"django.middleware.csrf.CsrfViewMiddleware",
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
]
```

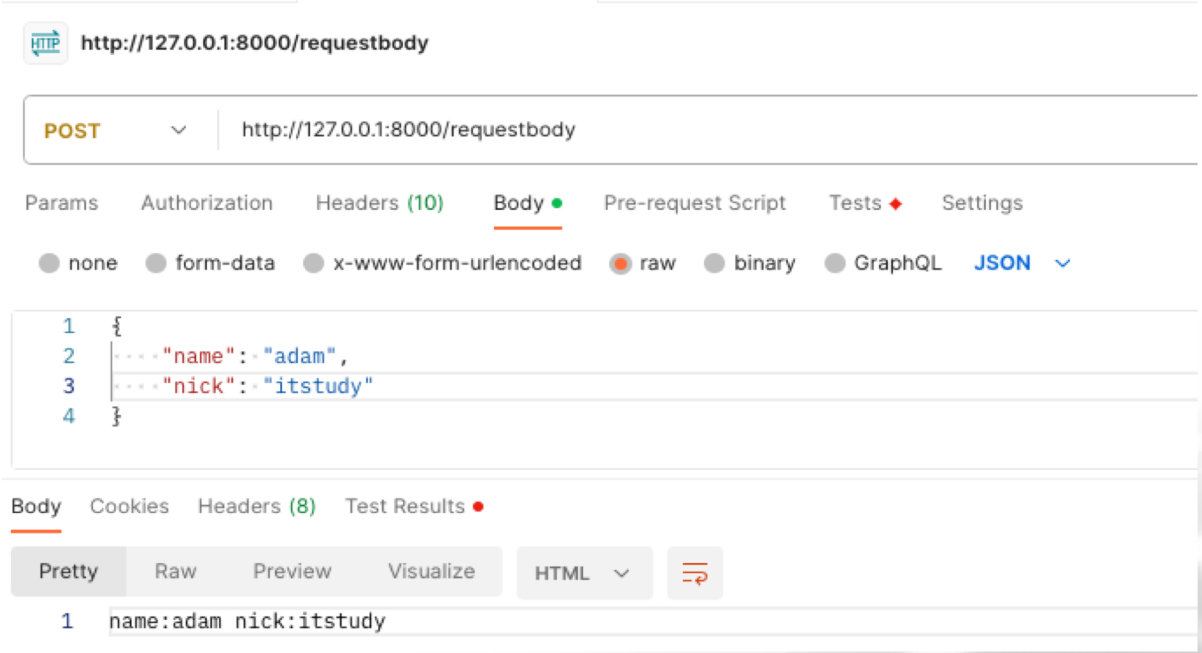
Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ POST 방식의 데이터 처리
    - 확인



Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ POST 방식의 데이터 처리
    - 확인



Django

- 클라이언트가 전송한 데이터 읽기
  - ✓ File Upload
    - 프로젝트의 urls.py 파일의 urlpattens에 URL 처리 내용 추가

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index),
    path('get/<str:itemid>', views.getItem),
    path('querystring', views.queryString),
    path('requestbody', views.requestBody),
    path('formdata', views.formData),
    path('fileupload', views.simple_upload),
]
```

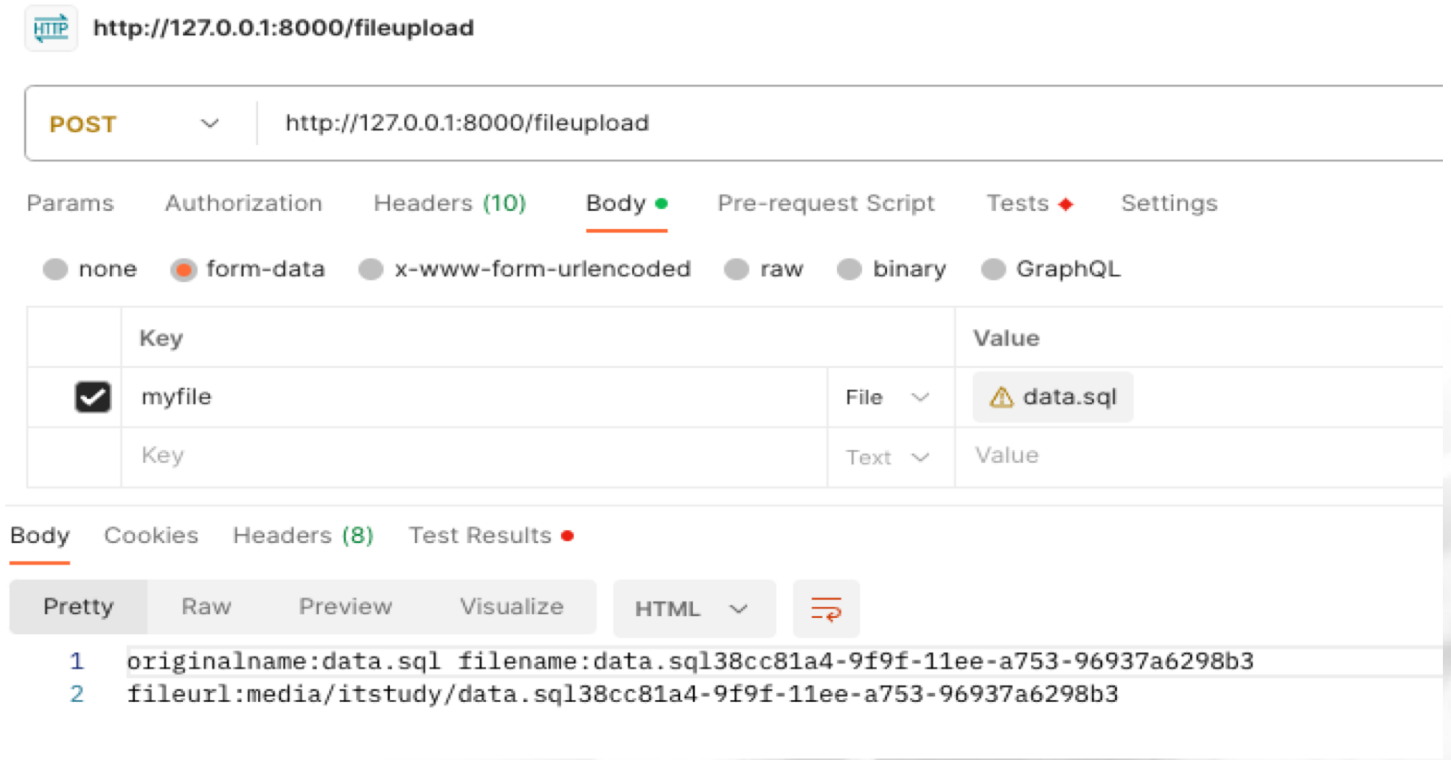
Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ File Upload
    - ● 애플리케이션의 views.py 파일에 요청 처리 함수 추가

```
from django.core.files.storage import FileSystemStorage
import uuid

def simple_upload(request):
    try:
        if request.method == 'POST' and request.FILES['myfile']:
            myfile = request.FILES['myfile']
            #파일을 업로드 할 디렉토리 설정
            fs = FileSystemStorage(location='media/itstudy',
                                   base_url='media/itstudy')
            originalname = myfile.name
            # FileSystemStorage.save(file_name, file_content)
            filename = fs.save(originalname + str(uuid.uuid1()), myfile)
            uploaded_file_url = fs.url(filename)
            return HttpResponse("originalname:" + originalname + " filename:" +
filename +
                                " fileurl:" + uploaded_file_url)
    except:
        return HttpResponse("업로드 한 파일이 없습니다.")
```

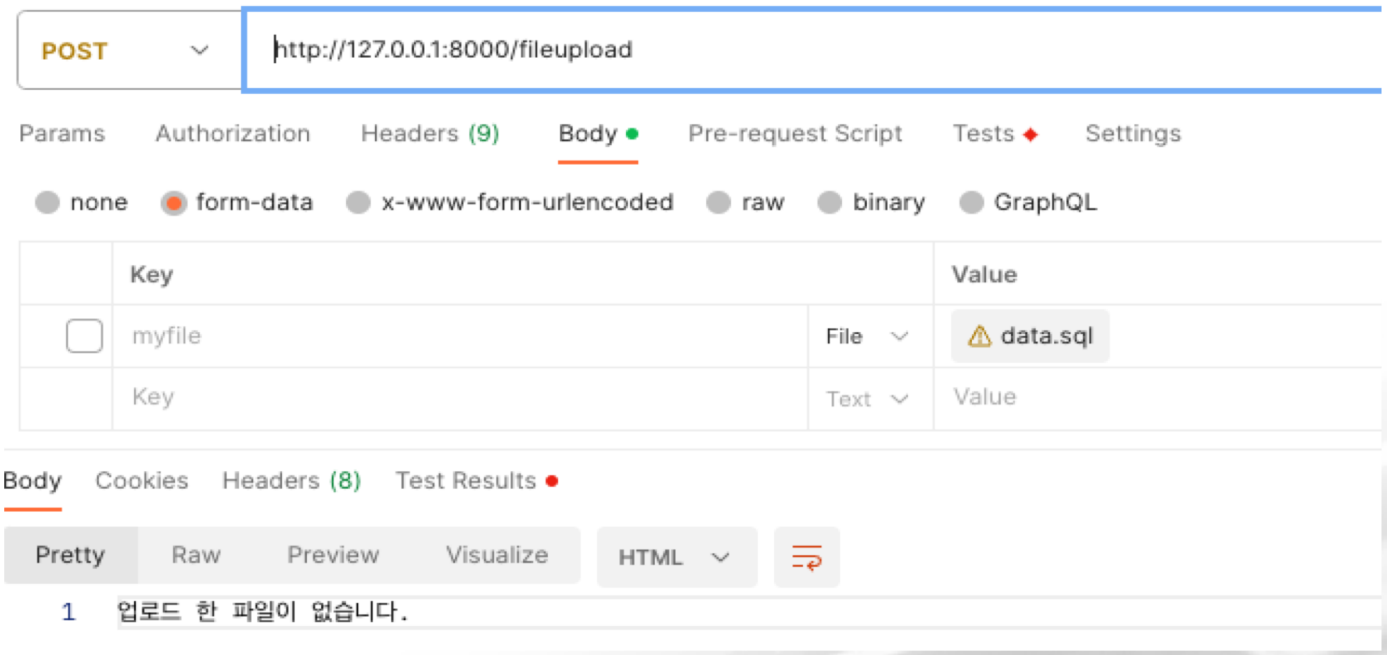
Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ File Upload
    - 확인



Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ File Upload
    - 확인

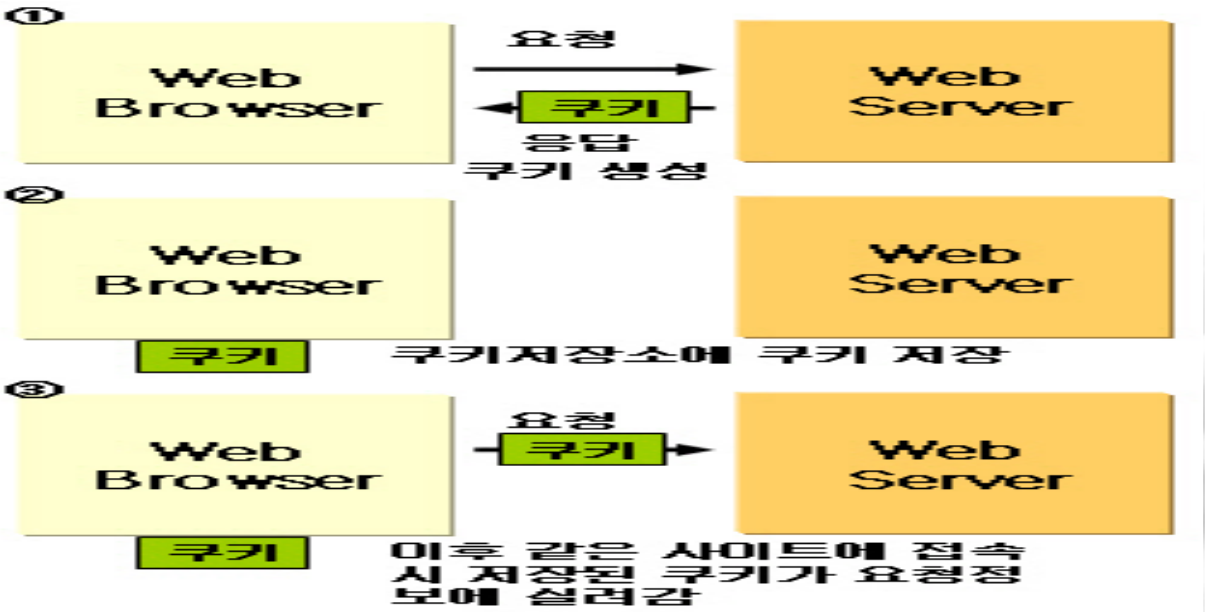


Django

- 클라이언트가 전송한 데이터 읽기
  - ✓ HTTP 프로토콜은 상태가 없음
    - 이전에 무엇을 했고 지금 무엇을 했는지에 대한 정보를 갖고 있지 않음
    - 웹 브라우저(클라이언트)의 요청에 대한 응답을 하고 나면 해당 클라이언트와의 연결을 지속하지 않음
    - 웹 서버 측에 웹 브라우저의 정보를 저장할 수 있음
    - 웹 브라우저의 요청에 포함되어 있는 웹 브라우저의 정보와 서버에 저장되어 있는 각각의 웹 브라우저에 대한 정보를 비교해서 동일한 웹 브라우저로부터 온 요청을 판단

Django

- 클라이언트가 전송한 데이터 읽기
  - ✓ Cookie
    - Cookie 기술은 웹 서버가 웹 브라우저로 데이터를 보냈다가 웹 서버 쪽으로 다시 되돌려 받는 방법을 사용
    - 웹 컴포넌트는 웹 브라우저로 **HTML** 문서를 보낼 때 데이터를 함께 보내며 웹 브라우저는 그 데이터를 저장해 두었다가 다른 웹 컴포넌트를 호출할 때 **URL**과 함께 웹 서버로 전송



Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ Cookie
    - 쿠키 만들기

      set_cookie(name, value, max_age = None)
      - ◆ 쿠키를 만들기 위해서는 2개의 필수 인수와 1개의 선택적 인수가 필요
      - ◆ name: 쿠키의 이름(필수)
      - ◆ value: 쿠키에 저장하려는 값(필수)
      - ◆ max_age: 쿠키의 경과 시간(초)으로 지정하지 않으면 브라우저가 닫힐 때까지 쿠키가 존재(선택)
    - 쿠키 읽기

      request.COOKIE[name]
      - ◆ 쿠키는 dict와 같은 속성을 지니고 있는데 object
      - ◆ 쿠키 데이터를 읽으면 문자열로 리턴

Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ Cookie
    - ● 프로젝트의 urls.py 파일의 urlpattens에 URL 처리 내용 추가

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.index),
    path('get/<str:itemid>', views.getItem),
    path('querystring', views.queryString),
    path('requestbody', views.requestBody),
    path('formdata', views.formData),
    path('fileupload', views.simple_upload),
    path('cookie', views.cookieUse),
]
```

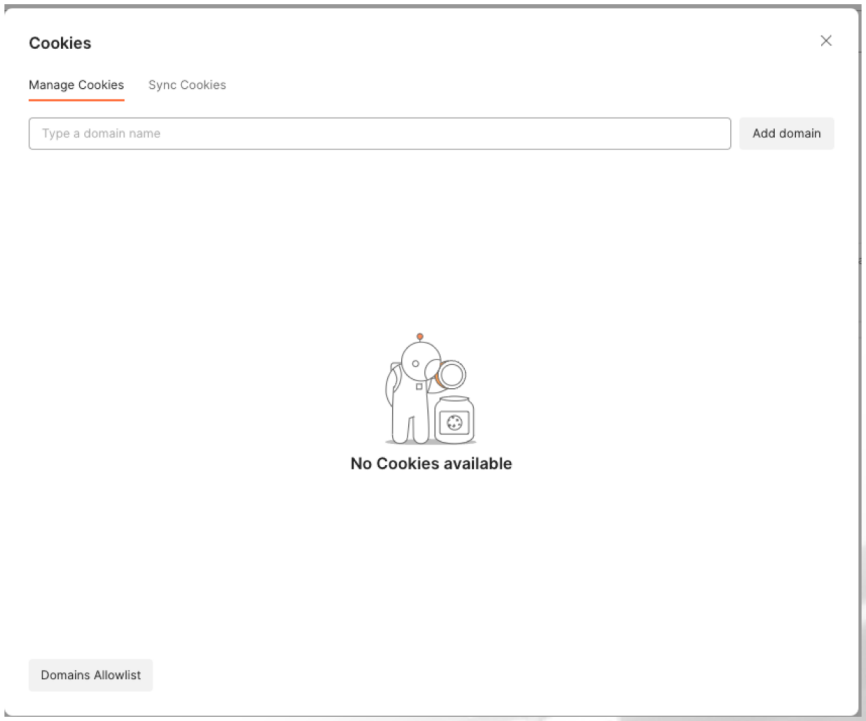
Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ Cookie
    - ● 애플리케이션의 views.py 파일에 요청 처리 함수 추가

```
def cookieUse(request):
    if request.COOKIES.get('username') is not None:
        username = request.COOKIES.get('username')
        return HttpResponse("name:" + username)
    else:
        response = HttpResponse()
        response.set_cookie("username", "itstudy")
        return response
```

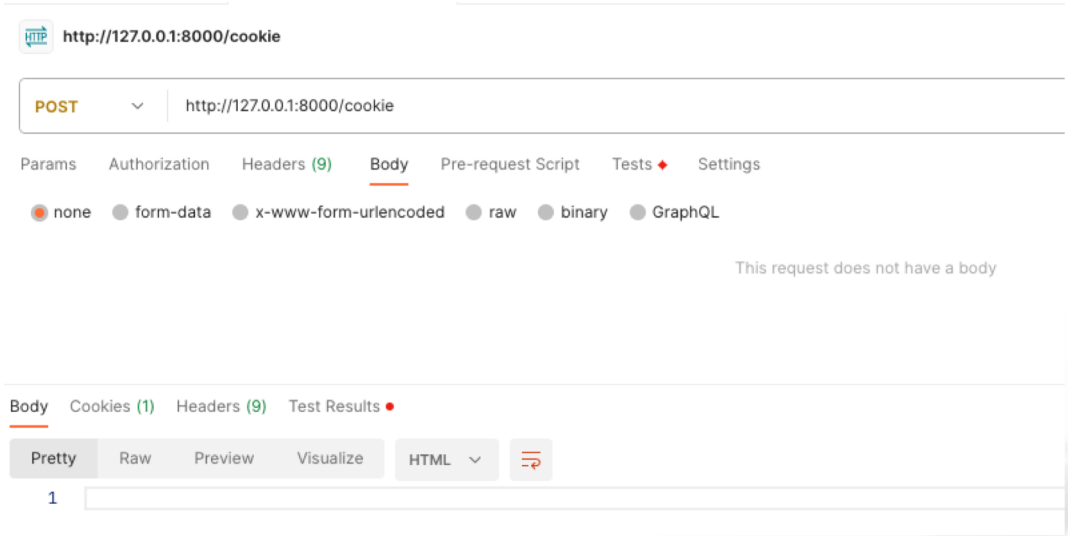
Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ Cookie
    - ● 확인



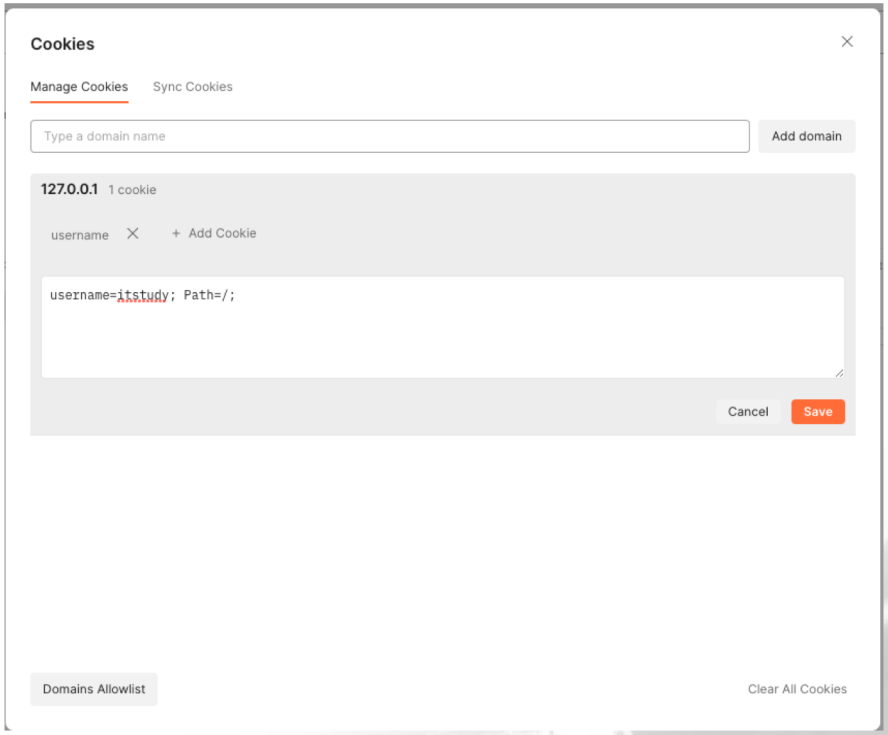
Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ Cookie
    - ● 확인



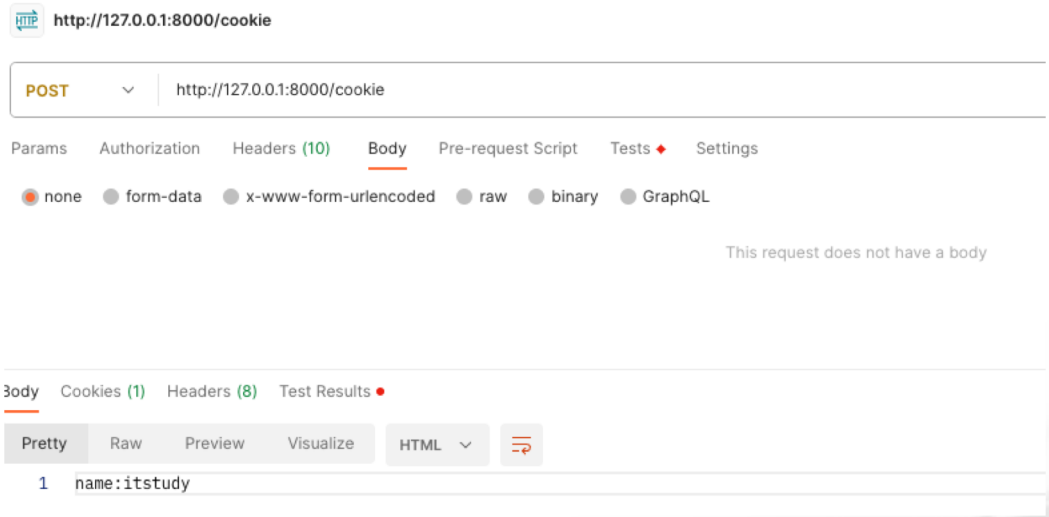
Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ Cookie
    - ● 확인



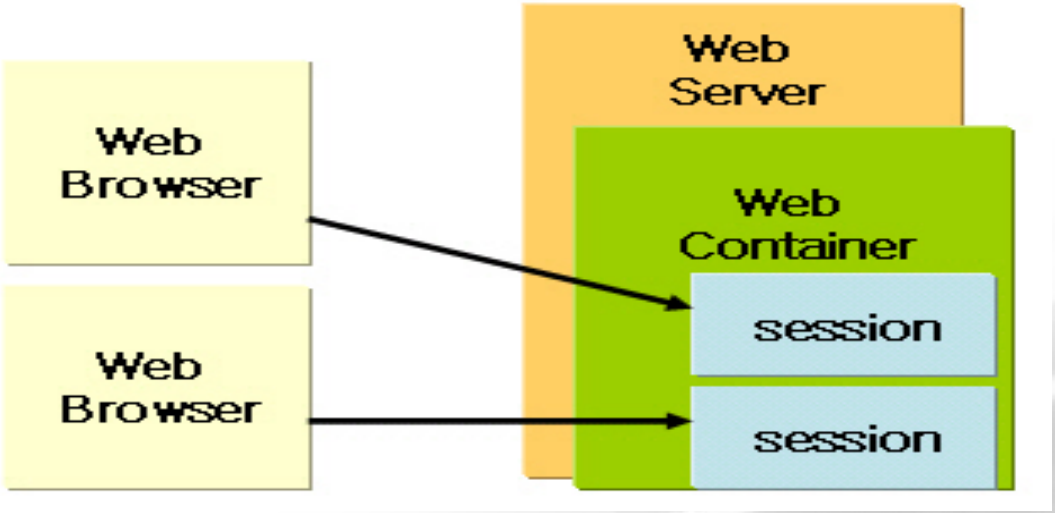
Django

- ❖ 클라이언트가 전송한 데이터 읽기
  - ✓ Cookie
    - ● 확인



Django

- 클라이언트가 전송한 데이터 읽기
  - ✓ Session
    - Session 기술은 웹 브라우저를 거치지 않고 웹 서버에 있는 데이터 영역을 통해 데이터를 저장하는 방법
    - 첫 번째 웹 컴포넌트는 웹 서버 쪽에 데이터를 저장해 놓고 그 데이터를 읽기 위해 필요한 Session 아이디만 웹 브라우저로 보내면 웹 브라우저는 아이디를 저장해 두었다가 두 번째 웹 컴포넌트를 호출할 때 웹 서버로 보내며 그 아이디를 이용하면 저장된 데C



Django

- 클라이언트가 전송한 데이터 읽기
  - ✓ Session
    - request 객체에 session 이라는 이름으로 생성됨
    - 디셔너리를 사용하는 형태로 사용하는 것이 가능

Database 연동

- Django 모델(Model)
  - ✓ 개요
    - Django에서 Model은 데이터 서비스를 제공하는 Layer
    - Model은 각 App안에 기본적으로 생성되는 models.py 모듈 안에 정의
    - models.py 모듈 안에 하나 이상의 모델 클래스를 정의할 수 있으며 하나의 모델 클래스는 데이터베이스에서 하나의 테이블에 해당
    - Django 모델은 django.db.models.Model 의 파생 클래스이며 모델 클래스의 각 속성(Attribute)은 테이블의 필드에 해당
    - Primary Key가 지정되지 않으면 DB 테이블 생성시 자동으로 id 가 생성됨
    - 모델 클래스는 필드를 정의하기 위해 인스턴스 변수가 아닌 클래스 변수를 사용하는데 이는 그 변수가 테이블 필드의 내용을 갖는 것이 아니라 테이블의 컬럼 메타 데이터를 정의하는 것이기 때문
    - 필드를 정의하는 각각의 클래스 변수는 models.CharField(), models,IntegerField(), models.DateTimeField(), models.TextField() 등의 각 필드 타입에 맞는 Field 클래스 객체를 생성하여 할당
    - Field 클래스는 여러 종류가 있는데 생성자 호출 시 필요한 옵션들을 지정할 수 있음
    - Field 클래스마다 반드시 지정해야 주어야 하는 옵션이 있을 수 있는데 예를 들어 CharField (와 그 서브클래스들)은 필드의 최대 길이를 나타내는 max_length를 항상 지정해 주어야 함

Database 연동

- Django 모델(Model)
  - ✓ 필드 타입
    - CharField: 제한된 문자열 필드 타입로 최대 길이를 max_length 옵션에 지정해야 하고 문자열의 특별한 용도에 따라 CharField의 파생클래스로 이메일 주소를 체크를 하는 EmailField, IP 주소를 체크를 하는 GenericIPAddressField, 콤마로 정수를 분리한 CommaSeparatedIntegerField, 특정 디렉토리의 파일 패스를 표현하는 FilePathField, URL을 표현하는 URLField 등이 있음
    - TextField: 대용량 문자열을 갖는 필드
    - IntegerField: 32 비트 정수형 필드로 정수 사이즈에 따라 BigIntegerField, SmallIntegerField 을 사용할 수 있음
    - BooleanField: true/false 필드로 Null 을 허용하기 위해서는 NullBooleanField를 사용
    - DateTimeField: 날짜와 시간을 갖는 필드로 날짜만 가질 경우는 DateField, 시간만 가질 경우는 TimeField를 사용
    - DecimalField: 소숫점을 갖는 decimal 필드
    - BinaryField: 바이너리 데이터를 저장하는 필드
    - FileField: 파일 업로드 필드
    - ImageField: FileField의 파생클래스로서 이미지 파일인지 체크
    - UUIDField: UUID를 저장하는 필드
    - ForeignKey: 외래키 설정

Database 연동

- Django 모델(Model)
  - ✓ 필드 옵션
    - null(Field.null): null=True 이면, Empty 값을 DB에 NULL로 저장하고 DB에서 Null이 허용 - models.IntegerField(null=True)
    - blank(Field.blank): blank=False 이면 필드가 Required 필드가 되고 Blank=True 이면 Optional 필드 - models.DateTimeField(blank=True)
    - primary_key(Field.primary_key): 해당 필드가 Primary Key임을 표시 - models.CharField(max_length=10, primary_key=True)
    - unique (Field.unique): 해당 필드가 테이블에서 Unique함을 표시하는데 해당 컬럼에 대해 Unique Index를 생성 - models.IntegerField(unique=True)
    - default(Field.default):필드의 디폴트 값을 지정 - models.CharField(max_length=2, default="WA")
    - db_column (Field.db_column): 컬럼명은 디폴트로 필드명을 사용하는데 만약 다르게 쓸 경우 지정

Database 연동

- ❖ Django 모델(Model)
  - ✓ 예시
    - ● 테이블
      - ● Owner: 자동차 주인 정보
      - ● Brand: 자동차를 생산하는 생산업체 정보
      - ● Car_Model: Brand들이 출시한 차량 정보
      - ● Car: 차주들이 소유하고 있는 차량의 정보

Database 연동

- Django 모델(Model)
  - ✓ 예시
    - models.py

```
from enum import unique
from django.db import models

class Owner(models.Model):
    name = models.CharField(max_length=128, unique=True)
    email = models.EmailField(unique=True)
    phone = models.CharField(max_length=128)

class Brand(models.Model):
    brand_name = models.CharField(max_length=128, unique=True)

class Car_Model(models.Model):
    brand = models.ForeignKey(Brand, on_delete=models.CASCADE)
    model_name = models.CharField(max_length=128)

class Car(models.Model):
    car_number = models.CharField(max_length=128)
    owner = models.ForeignKey(Owner, on_delete=models.CASCADE)
    car_model = models.ForeignKey(Car_Model, on_delete=models.CASCADE)
```

Database 연동

- DB 설정
  - ✓ Django 에서 사용하는 DB 에 대한 정보는 Django 프로젝트의 settings.py 파일에 설정
  - ✓ Django 프로젝트가 생성되면 기본적으로 설정되어 있는 셋팅은 데이터베이스는 디폴트로 sqlite3 를 사용하고 파일명은 /db.sqlite3 로 지정되어 있음

```
# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Database 연동

- DB 설정
  - ✓ 사용하고자 하는 데이터베이스에 따라 패키지 설치
  - ✓ DATABASES 에는 반드시 "default"가 설정되어야 하며 뒤에 다른 DB 설정도 추가할 수 있음
  - ✓ Django 프레임워크은 현재 다음과 같은 DB 엔진을 지원하고 있는데 DB 설정의 "ENGINE" 키에 해당 DB 엔진 값을 지정할 수 있음
    - django.db.backends.postgresql
    - django.db.backends.mysql
    - django.db.backends.sqlite3
    - django.db.backends.oracle

Database 연동

- DB 설정
  - ✓ MySQL 사용
    - mysqlclient를 설치: pip install mysqlclient
      - Mac OS에서 설치가 되지 않는 경우

```
$ brew install mysql
$ brew install openssl
$ brew install pkg-config
$ pip install mysqlclient
```

    - MySQL에 대한 연결 정보를 담은 DB 설정 – 자신의 Database 로 설정 변경:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'adam',
        'USER': 'root',
        'PASSWORD': 'wnddkd',
        'HOST': '127.0.0.1',
        'PORT': ''
    }
}
```

Database 연동

- DB Migration
  - ✓ Django에서 Model 클래스를 생성하고 난 후 해당 모델에 상응하는 테이블을 데이터베이스에서 생성할 수 있음
  - ✓ Python 모델 클래스의 수정 및 생성을 DB에 적용하는 과정을 Migration이라 부름
  - ✓ Django가 기본적으로 제공하는 ORM(Object-Relational Mapping) 서비스를 통해 진행
  - ✓ Django 모델 클래스로부터 테이블을 생성하기 위해서는 크게 Migration을 준비하는 과정과 이를 적용하는 과정으로 나뉘는데 구체적으로는 다음과 같은 절차를 따름
    - 모델 클래스로부터 테이블 스키마를 생성 혹은 수정하기 위하여 아래 명령을 실행
    - 명령이 실행되면 해당 Django App 안에 migrations 라는 서브 디렉토리를 만들고 테이블 생성 및 수정을 위한 python 마이크레이션 파일들을 생성

      python manage.py makemigrations
    - 모델 클래스로부터 실제 DB에 테이블을 생성하거나 수정하기 위해 아래 명령을 실행

      python manage.py migrate
    - Migration에 의해 생성되는 테이블은 App명_ModelClass명 의 형식으로 생성

Database 연동

- MySQL CRUD
  - ✓ MariaDB 에 접속해서 샘플 데이터 작성 – 테이블 이름은 자신의 App_item 으로 작성

```
use itstudy;

CREATE TABLE myweb_item(
        itemid int,
        itemname VARCHAR(20),
        price int,
        description VARCHAR(50),
        pictureurl VARCHAR(20),
        PRIMARY KEY (itemid)
);

insert into myweb_item values(1, 'Lemon', 500, 'Vitamin-A', 'lemon.jpg');
insert into myweb_item values(2, 'Orange', 1500, 'Vitamin-B', 'orange.jpg');
insert into myweb_item values(3, 'Kiwi', 2000, 'Vitamin-C', 'kiwi.jpg');
insert into myweb_item values(4, 'Grape', 1000, 'Vitamin-D', 'grape.jpg');
insert into myweb_item values(5, 'Strawberry', 2000, 'Vitamin-E', 'strawberry.jpg');
insert into myweb_item values(6, 'Mandarin', 300, 'Vitamin-F', 'mandarin.jpg');

commit;

select * from myweb_item;
```

Database 연동

- MySQL CRUD
  - ✓ myweb/models.py 파일에 모델 작성

```
from django.db import models

class Item(models.Model):
    itemid = models.CharField(max_length=50,primary_key=True)
    itemname=models.CharField(max_length=50)
    price=models.IntegerField()
    description=models.CharField(max_length=50)
    pictureurl = models.CharField(max_length=50)
```

Database 연동

- MySQL CRUD
  - ✓ CRUD 작업
    - ● 데이터 삽입
      - ◆ 데이터를 삽입하기 위해서는 먼저 테이블에 해당하는 모델(Model Class)로부터 객체를 생성하고 그 객체를 가지고 save() 메서드를 호출

Database 연동

- MySQL CRUD
  - ✓ CRUD 작업
    - SELECT
      - Django 모델 클래스에 대해 objects 라는 Manager(django.db.models.Manager) 객체를 자동으로 추가
      - Manager를 통해 특정 데이터를 필터링 할 수도 있고 정렬할 수도 있으며 기타 여러 기능들을 사용할 수 있음
      - 데이터를 읽어오기 위해서는 모델 클래스.objects 를 사용
      - Feedback 이라는 모델의 경우 Feedback.objects
      - all(): 테이블 데이터를 전부 가져오기 위해서는 Feedback.objects.all() 과 같이 all() 메서드를 사용
      - Feedback 테이블의 모든 데이터의 id와 name 컬럼을 읽기

```
for f in Feedback.objects.all():
    s += str(f.id) + ' : ' + f.name + '₩n'
```

Database 연동

- MySQL CRUD
  - ✓ CRUD 작업
    - SELECT
      - get(): 하나의 Row만을 가져오기 위해서는 get() 메서드를 사용하는데 예를 들어 아래는 Primary Key (일반적으로 id 컬럼)가 1인 row를 가져옴

```
row = Feedback.objects.get(pk=1)
print(row.name)
```

      - filter(): 특정 조건에 맞는 Row들을 가져오기 위해서는 filter() 메서드를 사용하는데 예를 들어 아래는 name 필드가 Kim 인 데이터만 가져옴

```
rows = Feedback.objects.filter(name='Kim')
```

      - exclude(): 특정 조건을 제외한 나머지 Row들을 가져오기 위해서는 exclude() 메서드를 사용하는데 예를 들어 아래는 name 필드가 Kim이 아닌 데이터만 가져옴

```
rows = Feedback.objects.exclude(name='Kim')
```

      - count(): 데이터의 갯수(row 수)를 세기 위해 count() 메서드를 사용

```
n = Feedback.objects.count()
```

Database 연동

- MySQL CRUD
  - ✓ CRUD 작업
    - SELECT
      - order_by(): 데이터를 키에 따라 정렬하기 위해 order_by() 메서드를 사용하는데 order_by() 안에는 정렬 키를 나열할 수 있으며 앞에 -가 붙으면 내림차순으로 아래는 id를 기준으로 오름차순, createDate로 내림차순으로 정렬

        ```
        rows = Feedback.objects.order_by('id', '-createData')
        ```

      - distinct(): 중복된 값은 하나로만 표시하기 위해 distinct() 메서드를 사용하는데 아래는 name필드가 중복되는 경우 한번만 표시

        ```
        rows = Feedback.objects.distinct('name')
        ```

      - first(): 데이터 들 중 첫번째 row만을 리턴하는데 아래는 name필드로 정렬했을 때 첫번째 row를 리턴

        ```
        rows = Feedback.objects.order_by('name').first()
        ```

      - last(): 데이터들 중 마지막 row를 리턴하는데 아래는 name필드로 정렬했을 때 마지막 row 리턴

        ```
        rows = Feedback.objects.order_by('name').last()
        ```

Database 연동

- MySQL CRUD
  - ✓ CRUD 작업
    - 수정
      - UPDATE: 데이터를 수정하기 위해서는 먼저 수정할 Row 객체를 얻은 후 변경할 필드들을 수정
      - 마지막에 save() 메서드를 호출되면, SQL의 UPDATE이 실행되어 테이블에 데이터가 갱신
      - 아래는 id가 1인 Feedback 객체에 이름을 변경하는 코드

```
fb = Feedback.objects.get(pk=1)
fb.name = 'Park'
fb.save()
```

    - 삭제
      - 데이터를 삭제하기 위해서는 먼저 삭제할 Row 객체를 얻은 후 delete() 메서드를 호출
      - 아래는 id가 2인 Feedback 객체를 삭제하는 코드

```
fb = Feedback.objects.get(pk=2)
fb.delete()
```

Database 연동

- MySQL CRUD
  - ✓ 테이블의 데이터 전부 출력

상품 목록 화면

상품ID	상품명	가격
1	Lemon	500원
2	Orange	1500원
3	Kiwi	2000원
4	Grape	1000원
5	Strawberry	2000원
6	Mandarin	300원

Database 연동

- MySQL CRUD
  - ✓ 테이블의 데이터 전부 출력
    - myweb/views.py 파일에 index 메서드 수정

```
from django.shortcuts import render
from myweb.models import Item

def index(request):
    print(Item)

    data = Item.objects.all()
    return render(request, 'index.html', {'data': data})
```

Database 연동

- MySQL CRUD
  - ✓ 테이블의 데이터 전부 출력
    - myweb 디렉토리에 static 디렉토리를 생성하고 그 안에 css 디렉토리를 생성한 후 style.css 파일을 생성하고 작성

```
div.body{
        overflow-y: auto;
    scrollbar-face-color: #C9BFED;
    scrollbar-shadow-color: #EDEDED;
        margin-top: 50px;
        margin-bottom: 50px;
}

tr.header{
    background: #C9BFED;
}

tr.record{
    background: #EDEDED;
}
```

Database 연동

- MySQL CRUD
  - ✓ 테이블의 데이터 전부 출력
    - settings.py 파일에 추가

```
import os
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'static'),
)
```

Database 연동

- MySQL CRUD
  - ✓ 테이블의 데이터 전부 출력
    - index.html 파일 수정

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
<link rel="stylesheet" href="{% static 'css/style.css' %}">
    <title>상품 목록</title>
</head>
<body>
   <div align="center" class="body">
                    <h2>상품 목록 화면</h2>
                    <table border="1">
                                <tr class="header">
                                            <th align="center" width="80">상품
ID</th>
                                            <th align="center" width="320">상품명
</th>
                                            <th align="center" width="100">가격
</th>
                                </tr>
```

Database 연동

- MySQL CRUD
  - ✓ 테이블의 데이터 전부 출력
    - index.html 파일 수정

```
                                {% for item in data %}
                                    <tr class="record">
                                        <td
align="center">{{item.itemid}}</td>
                                        <td
align="left">{{item.itemname}} </td>
                                        <td align="right">{{item.price}}
원</td>
                                    </tr>
                                {% endfor %}
                            </table>
                        </div>
</body>
</html>
```

Database 연동

- MySQL CRUD
  - ✓ 상세 보기

Database 연동

- MySQL CRUD
  - ✓ 상세 보기
    - index.html 파일에서 제목 출력 부분 수정

```
<td align="left"><a href="./detail/{{item.itemid}}">{{item.itemname}}</a> </td>
```

Database 연동

- MySQL CRUD
  - ✓ 상세 보기
    - urls.py 파일에 요청 처리 코드를 추가

```
urlpatterns = [
    path("admin/", admin.site.urls),
    path('', views.index),
    path('detail/<int:itemid>', views.detail),
]
```

Database 연동

- MySQL CRUD
  - ✓ 상세 보기
    - views.py 파일에 detail 요청을 처리해주는 함수를 추가

```
def detail(request, itemid):
    item = Item.objects.get(itemid=itemid)
    return render(request, 'detail.html', {'item': item})
```

Database 연동

- MySQL CRUD
  - ✓ 상세 보기
    - static 디렉토리에 img 디렉토리를 복사

Database 연동

- MySQL CRUD
  - ✓ 상세 보기
    - templates 디렉토리에 detail.html 파일을 작성

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>상품 상세 보기</title>
</head>
```

Database 연동

- MySQL CRUD
  - ✓ 상세 보기
    - templates 디렉토리에 detail.html 파일을 작성

```
<body>
    <div align="center" class="body">
            <h2>상품 상세</h2>
            <table>
                <tr>
                    <td><img src="{% get_static_prefix %}img/{{item.pictureurl}}"
alt=""></td>
                    <td align="center">
                        <table>
                            <tr height="50">
                                <td width="80">상품명</td>
                                <td width="160">{{item.itemname}}</td>
                            </tr>
                            <tr height="50">
                                <td width="80">가격</td>
                                <td width="160">{{item.price}}원</td>
                            </tr>
```

Database 연동

- MySQL CRUD
  - ✓ 상세 보기
    - templates 디렉토리에 detail.html 파일을 작성

```
                        <tr height="50">
                            <td width="80">비고</td>
                            <td width="160">{{item.description}}</td>
                        </tr>
                        <tr>
                            <td colspan="2" align="center" width="240"><a
href="../">■
                            목록으로 돌아가기</a></td>
                        </tr>
                    </table>
                </td>
            </tr>
        </table>
    </div>
</body>
</html>
```

Database 연동

- MySQL CRUD
  - ✓ 데이터 삽입
    - settings.py 파일에 파일 업로드 경로를 설정

      MEDIA_URL = '/media/'

      MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
    - 프로젝트에 media 디렉토리 생성

Database 연동

- MySQL CRUD
  - ✓ 데이터 삽입
    - models.py 파일에서 파일 업로드가 가능하도록 모델 수정

```
from django.db import models
class Item(models.Model):
    itemid = models.CharField(max_length=50,primary_key=True)
    itemname=models.CharField(max_length=50)
    price=models.IntegerField()
    description=models.CharField(max_length=50)
    pictureurl = models.ImageField(upload_to='images/',blank=True, null=True)
```

    - 데이터베이스에서 테이블 삭제

```
drop table myweb_item ;
```

    - 모델 변경 내용을 적용하기 위해서 터미널에서 명령 수행

```
pip install Pillow
python manage.py makemigrations
python manage.py migrate
```

Database 연동

- MySQL CRUD
  - ✓ 데이터 삽입
    - urls.py 파일에 삽입 요청 처리 와 media 파일의 경로 설정을 추가

```
from django.urls import path
from myweb import views
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('', views.index),
    path('detail/<int:itemid>', views.detail),
    path('insert', views.insert),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Database 연동

- MySQL CRUD
  - ✓ 데이터 삽입
    - index.html 파일에 삽입 링크를 추가

```
<p><a href="./insert">데이터 삽입</a></p>
```

Database 연동

- MySQL CRUD
  - ✓ 데이터 삽입
    - 데이터 삽입 화면을 위한 insert.html 파일을 templates 디렉토리에 생성하고 작성

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>데이터 삽입</title>
</head>
<body>
	<form method="post" enctype="multipart/form-data">{% csrf_token %}
		아이템 이름<input type="text" name="itemname" /><br/>
		아이템 가격<input type="text" name="price" /><br/>
		설명<input type="text" name="description" /><br/>
		이미지<input type="file" name="pictureurl"
accept="image/*"/><br />
		<input type="submit" value="삽입">
	</form>
</body>
</html>
```

Database 연동

- MySQL CRUD
  - ✓ 데이터 삽입
    - views.py 파일에 데이터 삽입 처리를 위한 코드를 작성

```
from django.db.models import Max
from django.shortcuts import redirect

def insert(request):
    if request.method == 'GET':
        return render(request, 'insert.html')
    else:
        obj = Item.objects.aggregate(itemid=Max('itemid'))
        if obj['itemid'] == None:
            obj['itemid'] = 0
        itemid = int(obj['itemid']) + 1
        itemname = request.POST['itemname']
        price = request.POST['price']
        description = request.POST['description']
```

Database 연동

- MySQL CRUD
  - ✓ 데이터 삽입
    - views.py 파일에 데이터 삽입 처리를 위한 코드를 작성

```
for img in request.FILES.getlist('pictureurl'):
    item = Item()
    item.itemid = itemid
    item.itemname = itemname
    item.price = price
    item.description = description
    item.pictureurl = img

    item.save()
return redirect('/')
```

Database 연동

- MySQL CRUD
  - ✓ 데이터 삽입
    - detail.html 파일에 이미지를 출력하는 부분을 수정

```
<td><img src="{{item.pictureurl.url}}" width='300' alt=""></td>
```

JSON 출력

- REST API
  - ✓ views.py 파일에 데이터베이스에서 읽어온 데이터를 Dictionary로 변경하는 함수를 추가

```
def itemToDictionary(item):
    output = {}
    output["itemid"] = item.itemid
    output["itemname"] = item.itemname
    output["price"] = item.price
    output["description"] = item.description
    output["pictureurl"] = str(item.pictureurl)
    return output
```

JSON 출력

- REST API
  - ✓ views.py 파일에 json 요청이 오면 처리하는 함수를 추가

```
from django.http import HttpResponse, JsonResponse

def itemJson(request):

    items = Item.objects.all()
    tempItems = []

    for i in range(len(items)):
        tempItems.append(itemToDictionary(items[i])) # Converting `QuerySet` to a
Python Dictionary

    data = {
        "result":True,
        "items": tempItems
    }
    return JsonResponse(data);
```

JSON 출력

- REST API
  - ✓ urls.py 파일에 json 요청이 오면 호출되는 함수를 지정

```
urlpatterns = [
    path('', views.index),
    path('detail/<int:itemid>', views.detail),
    path('insert', views.insert),
    path('itemjson', views.itemJson),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

JSON 출력

- REST API
  - ✓ 확인