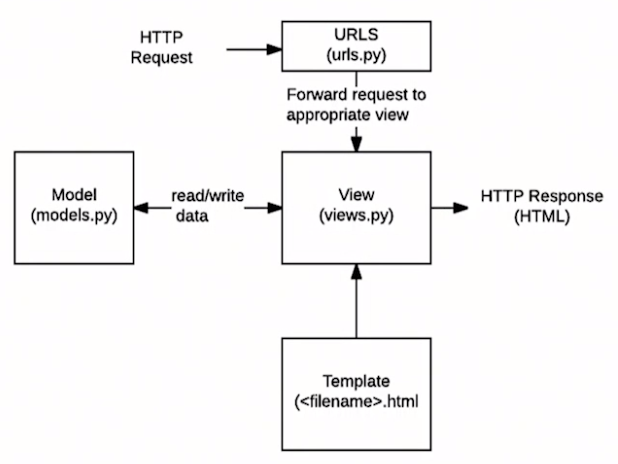
**0907 Django 웹 프레임 워크**

**(3/26 Django 프레임워크 및 HW/WS 해설)**

◆ **Django**

* **웹 프레임 워크**
  + 웹 개발에 필요한 소스코드, 라이브러리의 집합
* **학습 이유**
  + 파이썬 기반으로 작성된 오픈소스 웹 어플리케이션 프레임워크
  + 다양한 기업에서 사용 중
* **역사**
  + 2005년에 0.90 버전으로 시작 현재는 3.1, 3.2LTS 까지 발전하였음
* **동작방식**
  + MTV(model, Template, View) 패턴으로 분류
  + **MVC와 대응** 
    - Model = Model (데이터 관리)
    - View = Template (화면, 인터페이스)
    - Controller = View (상호 동작, 중간 관리)
  + 요청 접수: URLS, urls.py
  + View: url받고 model과 template에 일 시킨후 html 반환, Views.py
  + Template: html파일을 View에 보냄, 파일명.py
  + Model: 데이터를 읽고 씀, models.py



* **패키지 설치**
  + pip install django== 버전
  + django-admin startproject 프로젝트명
  + python manage.py startapp 앱명
  + python manage.py runserver 포트번호
* **구조**
  + 프로젝트 설정폴더와 여러개의 app 폴더들로 구성된 프로젝트
  + **공통**
    - \_\_init\_\_.py
      * 폴더를 하나의 파이썬 모듈로 인식하게 만듦
  + **프로젝트 설정 폴더**
    - 프로젝트 명과 같은 이름의 폴더
    - 프로젝트의 기본 설정 파일 들이 있음
    - urls.py
      * urls.py에 app들의 urls.py의 메서드들을 추가해줌
        + url 뒤에는 항상 / 써야함, ‘index/’
      * 해당 url로 들어오면 어떤 Views에서 처리하겠다 지정
      * path(‘/api’, include(“api.urls’))
      * api로 요청오면 api.urls로 보냄
      * api라는 app의 urls.py를 추가하겠다는 뜻
    - settings.py에 만든 앱을 추가
      * installed\_apps에 ‘app’, 처럼 추가
  + **app 폴더**
    - app, 실제 기능을 작성하는 파일들 위치
    - python manage.py startapp {이름} 으로 생성
      * 생성하자마자 settings.py의 installed\_apps에 추가할 것
    - Views.py
      * 함수
        + 정의하는 경우 항상 첫번째 인자를 request로 정의
        + return의 경우에도 항상 첫번째로 request 정의
        + return render(request, html파일명, 파일에서 쓸 변수)
        + 변수의 타입은 dictionary로 생각하면 됨
      * 각 폴더의 views.py 에 메서드를 만들어서 원하는 데이터 or html을 반환하도록 설정
      * urls.py에 url과 해당 url을 처리할 파일명.메서드명을 지정
        + 예) return render(request, ‘index.html’) -> def index(html)
        + urls.py에서 (‘index’, views.index) 일때 index를 호출하면 views.py의 index 메서드 호출 -> 결과 받아옴
    - templates 폴더
      * 반환할 html파일을 작성하는 장소
      * 항상 templates라는 이름의 폴더로 만들어야함 -> 규약
* **작업 순서 (일반 MTV)**
  1. 시스템 설정폴더 - urls.py에 url 추가
  2. 해당하는 app의 views.py에 실행될 함수 추가
  3. 함수에서 리턴할 html을 template폴더에 작성

**0907 Django MTV**

**(3/27 MTV 아키텍처 HW/WS 해설, 3/30 View 활용)**

* **공통 구조 설명**
  + \_\_init\_\_py
    - 각 폴더를 파이썬 모듈로 인식하게 해주는 파일
    - 내용은 없지만 지우면 안됨
  + 프로젝트 구조를 탐색하기위해 Octotree를 이용하면 편함
    - 깃랩에서 편하게 볼 수 있음
* **시스템 설정 폴더 구조 설명**
  + **Settings.py**
    - DEBUG : True이면 개발모드, 에러페이지를 쉽게 볼 수 있음
    - ALLOWED\_HOST: 접근을 허가할 호스트를 지정
    - INSTALLED\_APPS: 설치된 앱 + 만든 앱을 추가
    - Templates
      * APP\_DIRS: True
        + 등록된 app(Installed\_apps)의 디렉토리에 있는 template 폴더를 템플릿 폴더로 활용
    - databases 설정
    - AUTH\_PASSWORD\_VALIDATORS : 패스워드 설정
    - 언어, 타임존, 인코딩 타입등 설정 내용 있음
      * DB에는 UTC로 저장, 표현/작성시에는 설정한 타임존 기준
  + **urls.py**
    - 프로젝트에서 url을 매핑하는곳
      * include를 사용하여 app별 urls에 넘기는 방법 주로 사용
        + path("api/", include("api.urls"))
    - url이 들어왔을때 어느것을 실행할 지 지정 가능
    - path(‘hi/<str:name>’, 파일명) 처럼 url에 값을 넣는것도 가능
      * 메서드에서 name으로 받음
        + 매개변수로 받을때도 반드시 이름이 동일해야함
        + request의 경우 바꿔도 동작하지만 하지말 것
      * path(‘add/<int:a>/<int:b>’, views.add) 처럼 여러개 가능
  + wsgi.py: 배포 작업을 할때 연관됨, Web Server Gateway Interface
  + **manage.py**: 명령어를 실행할 수 있는 내용들, 수정하지 말것
* **app 폴더 구조 설명** 
  + apps.py: 이게 어떤 앱이다 라고 설정해 놓은 파일이라고 생각하면 됨
  + **views.py**
    - url과 매핑되는 메서드 들을 작성
    - model과 template에 작업지시를 내리고 결과물을 return
    - Django shortcut functions 검색해서 함수 참조
  + urls.py
    - urlpatterns에 app내의 모든 url 작성
      * 시스템 폴더의 urls.py와 동일한 역할
    - from . import views 로 해당 views.py 전체를 가져와서 사용
    - url을 ‘’ 으로 하면 추가 url없이 바로 실행됨
  + Templates
    - 리턴할 html 파일들이 위치하는 장소
    - 실행될때 각 app의 templates폴더가 합쳐져서 탐색됨 -> 다른 app 템플릿 리턴
      * 탐색 순서는 installed\_apps 에 적힌 순서대로 탐색 하기 때문
      * 폴더 아래에 앱 이름으로 폴더를 만들어서 충돌 안나도록 처리

**0908 Django model**

**(3/31 model 기초)**

* **model**
  + 데이터를 다루는 역할 + DB 통신
  + class 단위로 지정(ORM) -> JPA와 비슷
    - class 이름(models.Model)
    - class 내부에 변수명과 타입 지정(속성 정의)
  + **타입**
    - IntegerField(), CharField(), TextField() 등
    - ForeignKey(대상 class, on\_delete+models.CASCADE) -> 외래키
    - DateTimeFiled(auto\_now\_add=True) -> 처음에만 현재 시간을 넣음
      * 업데이트 시에 가져와서 저장해놔야함
      * 안하면 업데이트할때 null이 들어가게 됨
      * auto\_now=True 의 경우 값이 들어갈때마다 저장됨 -> 업데이트 시간에 사용
  + **옵션**
    - primary\_key=True -> 대상 변수를 pk로 사용
    - max\_length -> 최대 길이
    - null=True -> null 가능
    - on\_delete-models.CASCADE -> 참조값이 사라지면 같이 지움
  + models.py 작성후 마이그레이션 작업 필요
  + **Migration 흐름**
    - 1. **Model 생성/수정/삭제 등**
      2. **migration 파일 생성** 
         1. python manage.py makemigrations 실행
         2. 마이그레이션 파일은 모델의 변경사항을 관리
         3. migrations 폴더를 만들고 폴더 밑의 파일에 설정한 파일내용 확인 가능
         4. model의 변경사항을 기록하고 DB에 반영하기 위한 코드들로 구성됨
      3. **migrate를 통한 DB에 적용** 
         1. python.py manage.py migrate
         2. 마이그레이션 파일을 DB에 반영
      4. **왜 이런작업을 거치는가?** 
         1. migration 파일은 DB 버전관리를 위한 git이라고 생각하면 됨
         2. DB스키마를 위한 버전관리 시스템
* **CLI 환경에서 DB확인** 
  + pip install django-extensions, ipython
    - CLI환경에서 편하게 볼 수 있도록 도와줌
    - pip install후에 installed\_apps에 extensions만 추가할 것
  + python manage.py shell\_plus
    - 쉘로 들어갈 수 있음
    - 모델명.objects.all() -> 쿼리셋 확인 가능, 전체 몇개 있는지
    - 모델명.objects.all()[인덱스].변수명 -> 해당 변수의 값 확인 가능
* **CRUD**
  + **생성** 
    - 변수 = 모델() -> 변수.변수명 = 값 으로 객체를 생성 후 값 할당
    - 변수.save() 로 DB에 저장
    - pk값은 자동으로 1씩 증가되면서 저장됨
  + **조회**
    - 객체.object.all() -> 전체 조회
      * 쿼리셋 리턴 = 유사 리스트, object 들을 보여줌
      * 객체(숫자) = 숫자는 해당 객체의 id를 나타냄
    - 객체.object.get(조건) -> id=1 등, 보통 pk로 조회
  + **수정**
    - 객체를 조회 -> 값을 변경 -> save
  + **삭제**
    - 객체 조회 -> 객체.delete()
    - 삭제후 삭제된 객체 반환
* **게시판 구현** 
  + **공통작업**

1. 미리 model 작성하여 DB에 반영
2. urls에 url 매핑
3. view에서 사용할 모델 import 및 해당 작업 수행
   * **게시글 목록**
4. 객체.objects.all()로 객체의 모든 내용 가져옴
5. 값을 serializer를 해서 return
6. 프론트에서 배열에서 각 객체 -> 변수들에 접근하여 사용
   * **작성**
7. 받아온 내용을 request에서 꺼내서 새로운 객체에 저장
   1. request.GET.get(‘변수명’) 으로 값을 꺼낼 수 있음
8. 객체.save()로 DB에 저장
   * **조회**
9. url에 <int:pk> 처럼 id값이 전달되도록 추가
10. views.py의 메서드의 메개변수에 (pk) 추가
11. 객체.objects.get(id=pk) 로 해당 글의 정보를 가져옴
    1. pk=pk로 하여도 동작함
    2. id가 pk 이므로 파이썬 내부에서는 동일한 값으로 접근
12. serializer를 이용하여 변환 후 return Response(객체)

**0908 Django REST API**

**(5/11 Django REST API)**

* **Django REST API**
  + MVT에서 MV만 사용
    - 요청에 따라 JSON 파일을 프론트에 넘김
* **프로젝트 구조**
  + **models.py**
    - 데이터를 읽고 쓰기위해서 DB 스키마 정의
    - class 이름(models.Model) 형태로 정의
    - class 내부에 DB스키마로 쓸 변수명 = 변수타입 작성
  + **serializers.py**
    - 만든 model을 nested한 JSON 파일로 변환하기 위하여 작성
    - class 변수Serializer(serializers.ModelSerializer) 로 정의
    - 내부에 class Meta
      * 사용할 모델과 그 모델의 필드(변수) 지정
      * 변수의 경우 ‘\_\_all\_\_’로 모든 필드를 지정할 수 있음
        + 모델에서 필요한 필드만을 지정할 수 있음
    - 상황에따라 사용하는 변수가 다를 수 있으므로 필요한 만큼 class 생성
  + **views.py**
    - 만든 모델과 serializers import
    - 메서드 정의
      * 조회, 삭제시에 값이 들어오는경우 매개변수에 request와 같이 써줌
      * model을 불러온뒤 serializer에 넣어 변환
        + serializer 사용시에 데이터가 여러개이면 many=True 옵션 필요

없으면 에러 발생 -> 디폴트는 1개로만 생각

쿼리셋임을 알려주는 용도

* + - * Response(객체.data) 등을 통해 return
        + from rest\_framework.response import Response 해서 사용
    - 메서드 별 **요청 구분**
      * 메서드 위에 @api\_view([‘GET’]) 처럼 작성
        + get 요청만 받아주겠다
        + @require\_GET, @require\_http\_methods([‘GET’]) 과 동일
      * 안쓰면 동작 안함
    - 조회
      * 객체 조회시에 get\_object\_or\_404(객체, pk=변수)
        + 있으면 객체 없으면 404 반환
* **요청구분 순서**
  1. 시스템 폴더의 urls에서 해당 app으로 요청
  2. app의 urls.py에서 url에 따라 views.py의 메서드 호출
  3. 해당 메서드가 request에서 파일을 가져옴
  4. serializer에 파일을 넣어 JSON으로 변환
  5. 결과물 return
* **API 테스트**
  + URL 대신 포스트맨 사용하여 확인
  + URL을 입력하여 요청방식을 보내면 테스트 결과 확인 가능
  + 쿠키로 인증을 하는 경우 동작 못함
    - Postman Interceptor를 사용
    - 로컬에서 하는경우 필요 없을 수 있음
  + Postman Interceptor 설정
    1. postman 오른쪽 위의 접시모양 클릭 -> cookies
    2. capture cookies on -> install interceptor bridge 클릭
    3. 생긴 domains 창에 url을 입력 (root url을 넣음)
       1. 히스토리가 막 바뀜
    4. Django 서버 재부팅