Ask Django

배포 준비 (공통) - DJANGO_SETTINGS_MODULE 분기

DJANGO_SETTINGS_MODULE 이란?

- 다양한 프로젝트 설정 (DB설정/캐시설정/앱설정/Storage설정 등)을 담는 파이썬 소스 파일
- 실행환경 별로 필요한 프로젝트 설정이 다를 수 있습니다.
- 장고는 DJANGO_SETTINGS_MODULE 환경변수를 통해, 로딩할 settings 소스파일의 경로를 참조
  - "askdjango.settings" : 프로젝트 초기 생성 시의 settings 경로
  - "askdjango.settings.prod" : 뒤에서 생성할 settings 경로
  - "askdjango.settings.prod_azure"
  - "askdjango.settings.prod_aws"
  - "askdjango.settings.prod_heroku"
- django/conf/global_settings.py #src 베이스에 특정 설정을 **재정의**

DJANGO_SETTINGS_MODULE 지정하기

- 장고 애플리케이션에서 기본 진입점
  - manage.py #src
  - 프로젝트/wsgi.py #src
- DJANGO_SETTINGS_MODULE 환경변수 **디폴트 설정** 코드
  - 환경변수를 제공하지 않으면, 아래 디폴트 설정값이 사용됩니다.

```
# 예시
import os
os.environ.setdefault('DJANGO_SETTINGS_MODULE', '프로젝트.settings')
```

setdefault은 어떻게 동작하나요?

- os.environ을 통해 현재 파이썬 실행 중에 셋업된 환경변수 목록에 접근할 수 있습니다.
- dict과 같은 인터페이스를 제공해줍니다.
- dict에서의 setdefault와 같은 동작을 합니다.

```
# 본 코드는
names = {}
if 'tom' not in names:
    names['tom'] = 10

# 아래 코드와 같은 동작을 합니다.
names = {}
names.setdefault('tom', 10)

# 즉, 아래 코드는 DJANGO_SETTINGS_MODULE 환경변수가 세팅되어있지 않을 때,
# DJANGO_SETTINGS_MODULE 환경변수를 "프로젝트.settings" 로 세팅하겠다는 뜻합니다.
os.environ.setdefault('DJANGO_SETTINGS_MODULE', '프로젝트.settings')
```

DJANGO SETTINGS MODULE 지정하는 3가지 방법

환경변수 세팅하기

- **서비스 배포 시에도 꼭 사용하는 방법**
- (주의) 배포방법마다 환경변수를 세팅하는 방법이 다릅니다.
- 방법1) 직접 환경변수 세팅하고, manage.py 명령 실행하기
  - 현재 쉘 세션이 살아있는 동안에만 환경변수가 유효합니다.

```
쉘입력> export DJANGO_SETTINGS_MODULE=askdjango.settings.prod
쉘입력> python3 manage.py runserver
```

- 방법2) 직접 환경변수 세팅하면서, manage.py 명령 실행하기
  - 본 명령에 한해서 환경변수 세팅

```
쉘입력> DJANGO_SETTINGS_MODULE=askdjango.settings.prod python3 manage.py runserver
```

DJANGO SETTINGS MODULE 지정하는 3가지 방법

manage.py 실행시에 --settings 인자 지정하기

- 방법3) manage.py 에서는 --settings 인자를 통해 지정이 가능
  - 미지정시 manage.py 소스코드에 지정된 디폴트 경로가 지정

쉘입력> `python3 manage.py runserver --settings=askdjango.settings.prod`

장고 애플리케이션, 실행환경 별 SETTINGS

- 개발용_일반
- 개발용_서비스v2
- 배포용_일반
- 배포용_Heroku
- 그 외 여러 환경

settings 파일 생성 예시 - 방법 1 (so so)

- manage.py
- 프로젝트/settings_common.py
- 프로젝트/settings_dev.py
- 프로젝트/settings_prod.py
- 프로젝트/settings*prod*heroku.py

settings 파일 생성 예시 - 방법 2 (추천)

- manage.py
- 프로젝트/settings/common.py
- 프로젝트/settings/dev.py
- 프로젝트/settings/prod.py
- 프로젝트/settings/prod_heroku.py

settings 필수 설정 예시

askdjango/settings/common.py

- BASE_DIR : 장고 프로젝트 ROOT 경로를 계산
  - settings.py 의 위치를 기준으로 계산하는데, 본 파일의 위치가 1 depth 깊어지므로 이를 조정

```
from os.path import abspath, dirname, join
BASE_DIR = dirname(dirname(dirname(os.path.abspath(__file__))))
```

- INSTALLED_APPS #doc
  - django-debug-toolbar 앱은 개발전용이므로, settings/dev.py 로 옮깁니다.

```
INSTALLED_APPS = [...]  # 'debug_toolbar' 제거
MIDDLEWARE = [...]      # 'debug_toolbar.middleware.DebugToolbarMiddleware' 제거
```

- ALLOWED_HOSTS #doc : 접근을 허용할 서비스 도메인만을 지정 (보안 설정)
  - ['*'] 설정 : 모든 도메인 허용 (비추천)
  - ['nomade.kr', 'festi.kr'] 설정 : 서비스 도메인을 꼭 지정해주세요.
    - 'www.nomade.kr' 과 'nomade.kr'은 서로 다른 도메인입니다.
  - 지정하지 않은 도메인으로 접근 시에, 400 Bad Request 응답
    - `request.get_host()` 호출 시에 체크
    - common MIDDLEWARE에서 request.get_host() 호출

```
ALLOWED_HOSTS = ['사용하시는서비스도메인']    # 허용할 도메인 서버 주소
```

- MEDIA 관련 설정
  - MEDIA_URL #doc : 유저가 업로드한 파일을 서빙할 때, prefix url
    - 끝이 필히 / 로 끝나야 합니다.
    - ex) 현재 서버에서 서빙할 경우 : "/media/"
    - ex) 다른 서버에서 서빙할 경우 : "https://도메인.com/media/"
  - MEDIA_ROOT #doc : 유저가 업로드한 파일을 저장하는 ROOT 경로 (필히 절대 경로로 지정)

```
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, '..', 'media')  # 유저가 업로드하는 파일이 저장될 경로
```

- STATIC 관련 설정
  - STATICFILES_DIRS #doc : 프로젝트 전반적으로 사용될 static 파일의 디렉토리 경로를 다수 지정
    - "앱디렉토리별/static/" 경로는 별도 설정이 필요없습니다.
  - STATIC_URL #doc : 개발리소스로서의 정적파일을 서빙할 때, prefix url
    - 끝이 필히 / 로 끝나야 합니다.
    - ex) 현재 서버에서 서빙할 경우 : "/static/"
    - ex) 다른 서버에서 서빙할 경우 : "https://도메인.com/static/"
  - STATIC_ROOT #doc : python3 manage.py collectstatic 명령을 수행할 때, 여러 디렉토리에 나눠 저장된 파일들을 복사할 목적 디렉토리 경로 (필히 절대경로로 지정)

```
# Filesystem Finder 설정
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'askdjango', 'static'),
]

STATIC_URL = '/static/'

# 여러 디렉토리에 나눠진 static파일을 지정 경로로 복사
#  $ python3 manage.py collectstatic 명령
STATIC_ROOT = os.path.join(BASE_DIR, '..', 'staticfiles')
```

settings 필수 설정 예시

askdjango/settings/dev.py

- django-debug-toolbar는 개발환경에서만 쓸 것입니다.

```
from .common import *

INSTALLED_APPS += ['debug_toolbar']
MIDDLEWARE += ['debug_toolbar.middleware.DebugToolbarMiddleware']
INTERNAL_IPS = ['127.0.0.1']
```

settings 필수 설정 예시

askdjango/settings/prod.py

- DEBUG : 실서비스에서는 필히 **False**로 지정하셔야 합니다.
  - 그래야, ORM 쿼리 실행내역에 메모리에 누적되지 않으며, => 메모리에 누적되다보면 어느 순간 시스템 메모리가 부족한 순간이 오겠죠? => 서비스 불능상태
  - 예외 발생 시 예외내역이 유저에게 노출되지 않습니다. => 자세한 디버깅 정보에 데이터베이스 계정정보가 포함되어있을 수 있어요. :(
- DATABASES : 실제 서비스에서 쓸 데이터베이스 설정

```
DEBUG = False

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'HOST': '데이터베이스_호스트',
        'NAME': '데이터베이스_DB명',
        'USER': '데이터베이스_유저',
        'PASSWORD': '데이터베이스_암호',
    },
}
```

- 참고 : 실 서비스 시에 AWS S3에 업로드할려면, django-storages #doc의 S3BotoStorage #doc 를 써보세요.

Tip: 실서비스 settings 설정 팁

계정정보를 settings 소스파일에 넣어두시기 껄꺼로우시죠? (특히나 소스코드 Public 프로젝트 !!!)

환경변수를 통한 설정을 하실 수 있습니다.

```
# askdjango/settings/prod.py
import os

DEBUG = False

DATABASES = {
    'default': {
        'ENGINE': os.environ.get('DATABASE_ENGINE', 'django.db.backends.postgresql'),    # ENGINE
        'HOST': os.environ.get('DATABASE_HOST', None),          # 데이터베이스_호스트
        'NAME': os.environ.get('DATABASE_NAME', None),          # 데이터베이스_DB명
        'USER': os.environ.get('DATABASE_USER', None),          # 데이터베이스_유저
        'PASSWORD': os.environ.get('DATABASE_PASSWORD', None), # 데이터베이스_암호
    },
}
```

디폴트 settings 경로 수정

manage.py

- 개발단계에서 많이 사용되기에, **settings.dev** 로 디폴트 지정
- 물론 --settings 옵션이나, DJANGO_SETTINGS_MODULE 환경변수를 통해 재지정 가능

```
#!/usr/bin/env python
import os
import sys

if __name__ == "__main__":
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "프로젝트.settings.dev")
    # 나머지 생략
```

프로젝트/wsgi.py

- 실서비스에서 사용되는 진입점이기에, **settings.prod** 로 지정
- 물론 DJANGO_SETTINGS_MODULE 환경변수를 통해 재지정 가능

```
import os
from django.core.wsgi import get_wsgi_application

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "프로젝트.settings.prod")
application = get_wsgi_application()
```

Tip: 서버에서는 settings.prod를 쓰도록 했는데

manage.py에서는 settings.dev가 지정되어있을 경우

같은 서버에서 명령을 내린다고 해서, 무조건 같은 settings를 쓰는 것이 아닙니다.

```
# 아래 명령은 settings.dev가 사용되어, 실제DB에 슈퍼유저가 생성되는 것이 아닌
# settings.dev에 명시된 DB(ex - sqlite3 DB)에 유저가 생성될 것입니다.
쉘입력> python3 manage.py createsuperuser

# 아래와 같이 settings를 직접 지정하시거나
쉘입력> python3 manage.py runserver --settings=askdjango.settings.prod

# 쉘에서 미리 환경변수를 세팅해두는 것도 좋습니다.
# 쉘설정파일 bash(~/.bashrc) 나 zsh(~/.zshrc)에 넣어두는 것도 좋습니다.
쉘입력> export DJANGO_SETTINGS_MODULE=askdjango.settings.prod
쉘입력> python3 manage.py runserver
```

"ImproperlyConfigured: The SECRET_KEY setting must not be empty." 예외가 발생한다면?

- 엉뚱한 settings가 지정되어, SECRET_KEY 설정이 정의되어 않은 채로, 장고 프로젝트가 시작했을 때, 발생
- 대개 settings 위치를 변경하면서 manage.py, 프로젝트/wsgi.py의 DJANGO_SETTINGS_MODULE 디폴트 위치를 변경하지 않았을 경우입니다.
- 프로젝트가 실행은 되어도, 엉뚱하게 동작한다면, settings파일 지정을 확인해보세요.

장고 소스코드 공통 배포셋업

끝

이제 각 서비스 별로, 셋업과정