

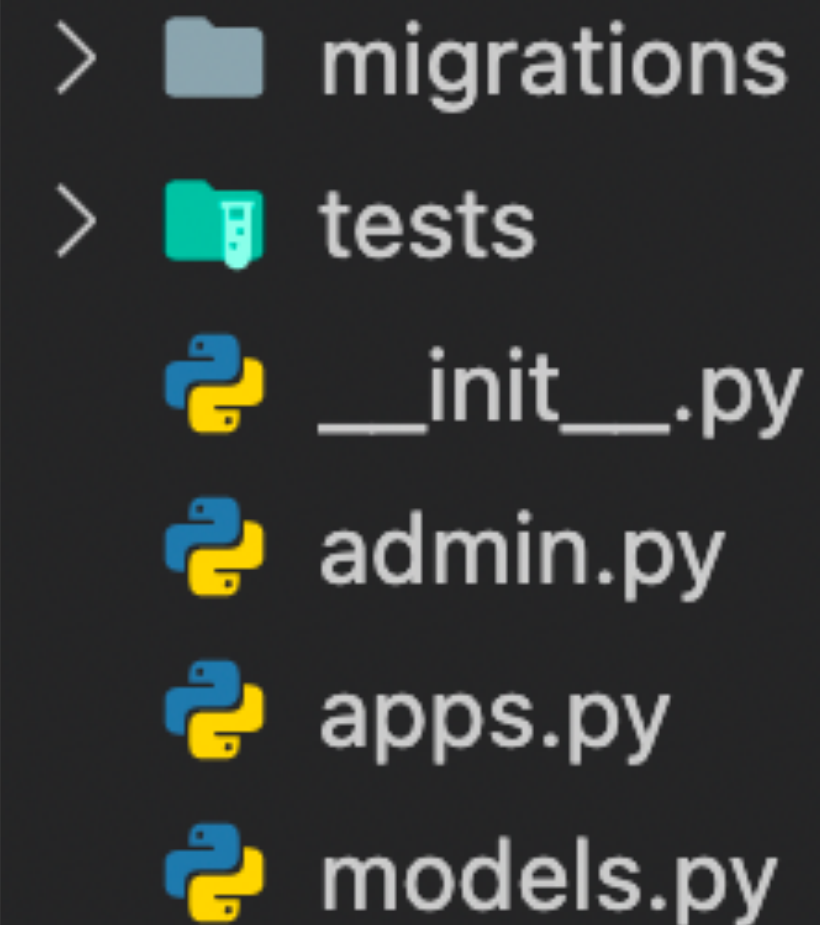
Django 알아두면 좋은 것들

모델 생성하기

개요

1. User 모델 추가 (DB 구현)
2. 관련 url, view 추가
3. 가능하다면, 테스트 코드 추가 및 실행

생성한 프로젝트 앱 하위에 오른쪽과 같은 폴더 구조를 가지도록 작성
파이썬의 경우, `__init__.py` 가 있어야 파일을 찾을 수 있기 때문에
tests 폴더, migrations에도 각각 생성,
admin.py 어드민 페이지에서 확인하려면 추가한다. (꼭 필요하지는 않다.)
apps.py에서는 url에 연결 및 settings에 등록을 해야하기 때문에 반드시 필요하다.



```
> migrations
> tests
__init__.py
admin.py
apps.py
models.py
```

```
# apps.py
```

```
from django.apps import AppConfig
```

```
class UsersConfig(AppConfig):  
    default_auto_field = "django.db.models.BigAutoField"  
    name = "apps.model.users"
```

apps.py 에 다음과 같이 작성한다. 이렇게 하고, settings에 등록한다.

```
# settings.py
```

```
INSTALLED_APPS = [  
    ...,  
    "apps.model.users",  
]
```

그 뒤 오른쪽과 같이 models.py 를 작성한다.

```
# models.py
```

```
class User(AbstractBaseUser, PermissionsMixin):  
    """유저 모델입니다.  
    Notes:  
        유저 이메일, 비밀번호로 구성되고, 유저 식별 외 개인정보를 담지 않습니다.  
    """
```

```
    email = models.EmailField(  
        "이메일",  
        max_length=256,  
        unique=True,  
    )  
    name = models.CharField(  
        "이름",  
        max_length=256,  
        default="",  
        blank=True,  
    )  
    introduction = models.TextField(  
        "소개",  
        max_length=2048,  
        default="",  
        null=True,  
    )  
    is_staff = models.BooleanField(  
        "스태프 권한",  
        default=False,  
    )  
    activate = models.BooleanField("활성화 여부", default=True)
```

```
    objects = BasicUserManager()  
    USERNAME_FIELD = "email"  
    REQUIRED_FIELDS = [  
        "name",  
        "password",  
    ]
```

```
class Meta:  
    db_table = "users"  
    verbose_name = "user"  
    verbose_name_plural = "users"  
    swappable = "AUTH_USER_MODEL"
```

models.py

```
class User(AbstractBaseUser, PermissionsMixin):
    """유저 모델입니다.
    Notes:
        유저 이메일, 패스워드로 구성되고, 유저 식별 외 개인정보를 담지 않습니다.
    """

    email = models.EmailField(
        "이메일",
        max_length=256,
        unique=True,
    )
    name = models.CharField(
        "이름",
        max_length=256,
        default="",
        blank=True,
    )
    introduction = models.TextField(
        "소개",
        max_length=2048,
        default="",
        null=True,
    )
    is_staff = models.BooleanField(
        "스태프 권한",
        default=False,
    )
    activate = models.BooleanField("활성화 여부", default=True)

    objects = BasicUserManager()
    USERNAME_FIELD = "email"
    REQUIRED_FIELDS = [
        "name",
        "password",
    ]

    class Meta:
        db_table = "users"
        verbose_name = "user"
        verbose_name_plural = "users"
        swappable = "AUTH_USER_MODEL"
```

User 모델을 보면, AbstractBaseUser 에서 상속받는 것을 확인 할 수 있다.

AbstarctUser 에서도 상속을 받을 수 있는데, 각각 차이점이 있으니 주의하자. 그 중 하나는, AbstractBaseUser 를 이용하는 경우 objects = Manager 설정, USERNAME_FIELD, REQUIRED_FIELDS 설정을 해야한다는 점이다.

```
class BasicUserManager(UserManager):
    def create_user(self, email, name, password):
        if not email:
            raise ValueError(("Users must have an email address"))
        validate_password(password)
        user = self.model(email=self.normalize_email(email), name=name)
        user.set_password(password)
        user.save(using=self._db)
        return user

    def create_superuser(self, email, name, password):
        if not email:
            raise ValueError(("Users must have an email address"))
        validate_password(password)
        pass
```

Manager 도 추가한 다음

`python manage.py makemigrations`

`python manage.py migrate`

로 반영

CRUD

- 모델까지 전부 설정되었다면, Django 에서 API 를 구성할 때 View 파일을 작성한다.
- View 를 작성하는 방식으로 Function Based(FBV), Class Based(CBV) 두가지 방식이 있는데, Class Based 방식으로 사용하는 걸 추천한다. (FBV 는 메소드마다 분기처리 해 줘야함.)
- CRUD에서 가장 중요한 것은 요청한 사용자의 권한에 따라 분기처리를 하는 것.

Function Based

- Method 마다 분기처리
- 함수 하나에 모든 메소드 로직을 적다보니 코드가 길어짐 -> 유지보수 힘들(테스트 코드도 공통된 함수에 적용됨)
- 퍼미션 등 데코레이터로 선언

```
@api_view(['POST', 'GET'])
@permission_classes([IsAuthenticated])
def index(request):
    if request.method == 'GET':
        print('GET 요청이다')
    elif request.method == 'POST':
        print('POST 요청이다')
    else :
        print('다른 메소드 요청이다')
```

Class Based

- 클래스 별로 라우트와 연결 (Method 별로 하위 함수로 작성하면 됨)
- 기존 View가 있어서 상속 후 필요한 것만 설정
- 유지보수가 편하다.

```
class CurrentUserView(APIView):  
    authentication_classes = [JWTAuthentication]  
    permission_classes = [IsAuthenticated]  
  
    def get(self, request):  
        if request.user is not None:  
            serializer = CurrentUserSerializer(request.user)  
            return Response({serializer}, status.HTTP_200_OK)  
        else:  
            return Response({"error_msg": "비 로그인 상태입니다."})
```


API 에서는...

class based view

```
class CurrentUserView(APIView):
    authentication_classes = [JWTAuthentication] ← 로그인, 로그아웃 인증 여부
    permission_classes = [IsAuthenticated] ← 유저에게 권한이 있는지

    def get(self, request):
        if request.user is not None:
            serializer = CurrentUserSerializer(request.user)
            return Response({serializer}, status.HTTP_200_OK)
        else:
            return Response({"error_msg": "비 로그인 상태입니다."})
```

authentication_classes 의 경우

BasicAuthentication : 유저 로그인 시 request.user 로 판단 가능

SessionAuthentication : 쿠키에 세션 정보 저장, 요청 시 쿠키 값을 이용해 유저 구분

JWTAuthentication : 쿠키에 JWT 토큰 저장, 요청 시 JWT 를 이용해 유저 구분

permission_classes 의 경우

IsAuthenticated : 로그인 한 사용자만

IsAuthenticatedOrReadOnly : ReadOnly 또는 로그인한 사용자만

AllowAny : 아무나 이용가능

기능에 맞게 인증 여부, 권한 여부를 각각 설정해 줄 수 있다.

풀스택에서는...

풀스택 에서는 다음과 같이 구현하였다. 각각 로그인, 회원가입, 로그아웃이다.
세션 방식을 기본적으로 제공하는 auth 를 이용한다.
각각 성공한 로직이라면 home 화면으로 이동한다.
그렇지 않으면, 다시 기존 화면을 렌더링 한다.
원래 {} 에 에러 메시지도 같이 보내주어야 한다.

```
class LoginPage(View):
    def loginpage(request):
        if request.method == "POST":
            username = request.POST["username"]
            password = request.POST["password"]
            user = auth.authenticate(username=username, password=password)
            if user is not None:
                auth.login(request, user)
                return redirect("home")
            else:
                return render(request, "login.html", {})
        else:
            return render(request, "login.html", {})

    def signuppage(request):
        if request.method == "POST":
            if request.POST["password1"] == request.POST["password2"]:
                User.objects.create_user(
                    username=request.POST["username"],
                    password=request.POST["password1"],
                )
                return redirect("home")
            return render(request, "signup.html", {})
        return render(request, "signup.html", {})

    def logout(request):
        try:
            auth.logout(request)
        except KeyError:
            pass
        return redirect("home")
```

template 에서...

rest api 가 아닌 풀스택으로 작성하는 경우 template을 이용해서 사용자의 값을 받는다.
이때 {% csrf_token %} 을 작성해주어야 한다. 이를 추가함으로 CSRF 공격을 막는다.

```
<form method="POST" action="{% url 'signup' %}">
  {% csrf_token %}
  <input type="text" name="username">
  <input type="password" name="password">
  <input type="submit" value="제출">
</form>
```

urls.py

```
path("signup", UserSignUpView(), name="signup"),
```

우리가 회원가입, 로그인 시 form 을 이용해서 요청을 보내기 때문에 꼭 해당 코드를 작성해주자.
위 코드에서 제출 버튼을 누르면, signup 라는 url로 post 요청이 간다. (url에서 name으로 지정)
이때 input 태그의 name으로 적은 것이 파라미터로 가기 때문에 view에서 처리할 때 값을 일치 시켜줘야 한다.

CSRF 란?

Cross Site Request Forgery 로 사이트 사이의 요청을 이용해서 공격하는 기법이다.
주로, 비슷한 사이트에 접속하게 해서 쿠키를 빼돌리고, 해당 쿠키를 이용해 요청을 보내는 방식이다.

template 에서...

유저의 인증 여부 is_authenticated 에 따라서 분기 처리가 가능하다

```
{% if user.is_authenticated %}
  <span>👤 {{ user.username }} / </span>
  <a href="/accounts/logout" style="text-decoration: none;"> 🔒 Logout </a>
{% else %}
  <a href="/accounts/login" style="text-decoration: none;"> 🔑 Login  </a><span> / </span>
  <a href="/accounts/signup" style="text-decoration: none;"> 📝 Signup </a>
{% endif %}
```

Django 보안 설정

기본적으로 제공하는 보안

- XSS
- CSRF
- SQL INJECTION
- Click Jacking
- SSL/HTTPS

마지막으로 api 를 구성하다보면 cors 이슈를 접하게 되는데,
django-cors-headers 라이브러리를 이용하는 걸 추천한다.