[파이썬반] dj-rest-auth 회원 라이브러리 커스터마이징

Django 회원 라이브러리 커스터마이징

- 장고의 회원 관련 라이브러리인 `dj-rest-auth` 및 `allauth` 를 사용할 때 유저 필드 커스터마이징 하는 방법에 대한 문서입니다.
- 라이브러리를 사용하는 동시에 `nickname` 등의 추가 필드를 사용할 때는 필수적으로 알아야 합니다.

준비사항

- 필요 라이브러리 설치

```
$ pip install django==4.2.6 djangorestframework dj-rest-auth django-allauth
```

- accounts 앱 생성 및 등록

```
INSTALLED_APPS = [
    # APP
    'accounts',

    # DRF
    'rest_framework',
    'rest_framework.authtoken',

    # REST_AUTH
    'dj_rest_auth',
    'allauth',
    'allauth.account',

        # social login 필요 시 추가
      # 'django.contrib.sites',
      # 'allauth.socialaccount',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

# social login 필요 시 추가
# SITE_ID = 1
```

- models.py 에 User 작성
  - nickname 필드를 추가합니다.

```
from django.db import models
from django.contrib.auth.models import AbstractUser


class User(AbstractUser):
    nickname = models.CharField(max_length=100)
```

- settings.py 설정

```
# DRF auth settings
# Token 인증을 기본으로 사용하도록 설정
REST_FRAMEWORK = {
```

```
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
    ]
}

# 사용자 수정
AUTH_USER_MODEL = 'accounts.User'
```

dj-rest-auth 의 기본 로직

- `rest-auth` 의 기본 url 설정

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

urlpatterns = [
    ...
    path('dj-rest-auth/', include('dj_rest_auth.urls')),
    path('dj-rest-auth/registration/', include('dj_rest_auth.registration.urls')),
]
```

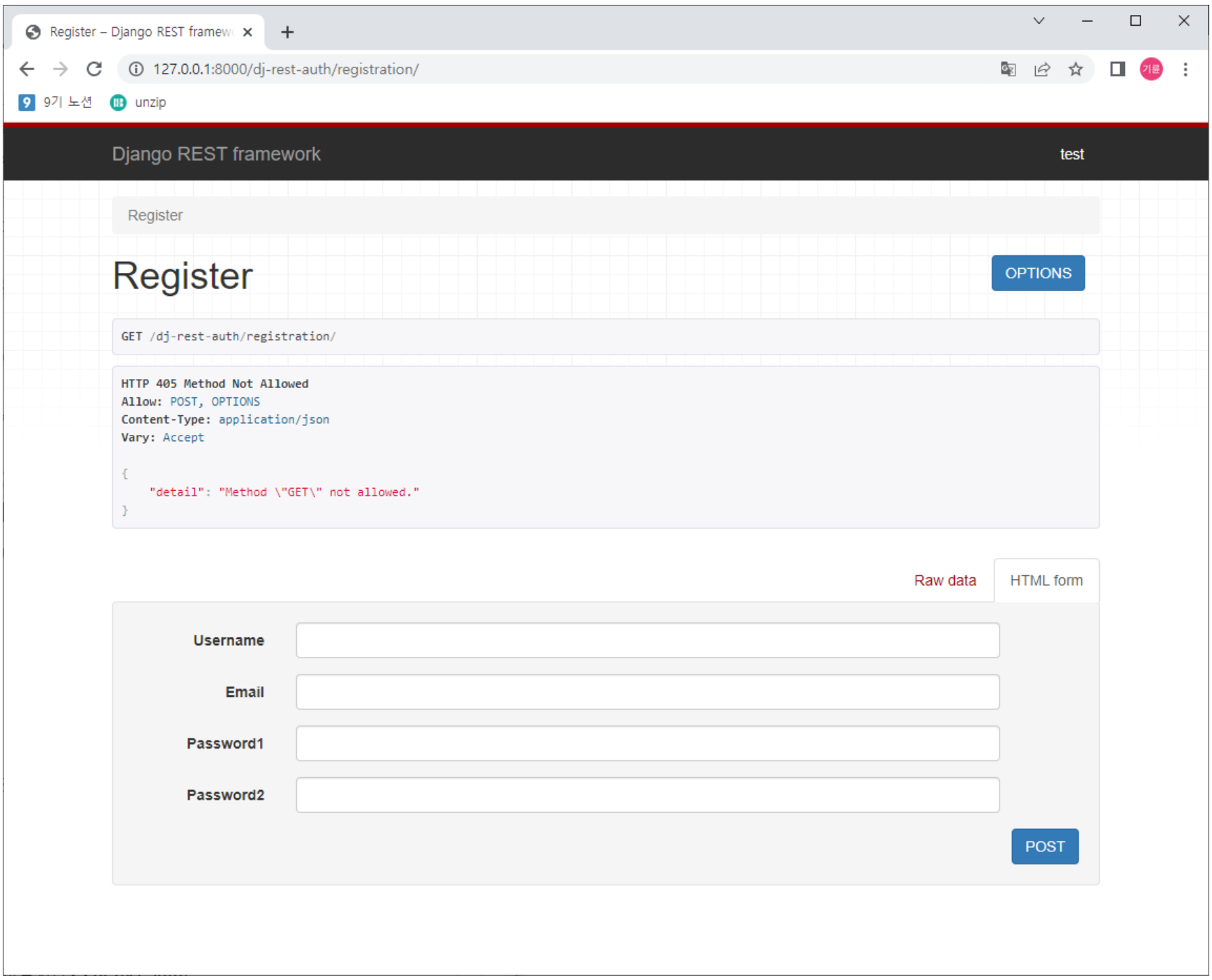
- `http://127.0.0.1:8000/dj-rest-auth/` 결과

- dj-rest-auth 관련 기능은 다음과 같습니다.
  - 참고사이트 - dj-rest-auth docs

1. `dj-rest-auth/ ^password/reset/$ [name='rest_password_reset']`
   - 패스워드 초기화 (이메일로 전송)
2. `dj-rest-auth/ ^password/reset/confirm/$ [name='rest_password_reset_confirm']`
   - 패스워드 초기화 (이메일 확인 후 초기화 페이지)
3. `dj-rest-auth/ ^login/$ [name='rest_login']`
   - 로그인
4. `dj-rest-auth/ ^logout/$ [name='rest_logout']`
   - 로그아웃
5. `dj-rest-auth/ ^user/$ [name='rest_user_details']`
   - 유저 정보 반환
6. `dj-rest-auth/ ^password/change/$ [name='rest_password_change']`
   - 패스워드 변경
7. `dj-rest-auth/registration/`
   - 회원가입

- 프로젝트에서 사용할 주요 기능은 `3. 로그인`, `4. 로그아웃`, `5. 유저 정보 반환`, `7. 회원가입` 4가지입니다.

- 회원가입
  - `http://127.0.0.1:8000/dj-rest-auth/registration/` 접속 시
  - 다음과 같이 회원가입 시 입력 받을 필드들이 출력됩니다.
  - 즉, REST 요청 시 아래 필드들에 대한 정보를 보내야 합니다.



models.py 에서 User 커스터마이징만 작성하면, 해당 필드들이 출력되지 않습니다.

위의 필드들을 입력 받도록 하는 기준은 github 코드를 참고해야 합니다.

rest-auth github

- 코드를 직접 살펴보자! ( )
  - github 의 코드 정의 따라가기 기능을 이용하면 훨씬 쉽게 추적이 가능합니다.
- github 의 `dj_rest_auth/registeration/` 부분이 회원가입과 관련이 된 부분입니다.
- urls.py 의 `path('', RegisterView.as_view(), name='rest_register')` 를 확인하면, `RegisterView` 로 연결된 것으로 확인됩니다.

```
# dj-rest-auth/views.py

class RegisterView(CreateAPIView):
    serializer_class = RegisterSerializer
    permission_classes = register_permission_classes()
    token_model = TokenModel
    throttle_scope = 'dj_rest_auth'

        ...
```

- Django 의 CreateAPIView 에서 serializer_class 로 지정한 `RegisterSerializer` 가 **회원가입 시 사용되는 serializer** 입니다.
- `RegisterSerializer` 코드

```
# dj-rest-auth/dj_rest_auth/registration/app_settings.py
DEFAULTS = {
        ...

    'REGISTER_SERIALIZER': 'dj_rest_auth.registration.serializers.RegisterSerializer',

    ...
}
```

- github 에서 따라가기를 해보면 위와 같이 `app_settings.py` 의 코드를 볼 수 있습니다.
  - `RegisterSerializer`: 회원가입 시 사용하는 Serialzier 로 `dj_rest_auth.registration.serializers.RegisterSerializer` 를 사용하는 것을 볼 수 있습니다.
  - 경로를 따라가면 아래 코드를 확인할 수 있습니다.
- `RegisterSerializer` 코드

```
class RegisterSerializer(serializers.Serializer):
    username = serializers.CharField(
        max_length=get_username_max_length(),
        min_length=allauth_account_settings.USERNAME_MIN_LENGTH,
        required=allauth_account_settings.USERNAME_REQUIRED,
    )
    email = serializers.EmailField(required=allauth_account_settings.EMAIL_REQUIRED)
    password1 = serializers.CharField(write_only=True)
    password2 = serializers.CharField(write_only=True)

    def validate_username(self, username):
        username = get_adapter().clean_username(username)
        return username

    def validate_email(self, email):
        email = get_adapter().clean_email(email)
        if allauth_account_settings.UNIQUE_EMAIL:
            if email and EmailAddress.objects.is_verified(email):
                raise serializers.ValidationError(
                    _('A user is already registered with this e-mail address.'),
                )
        return email

    def validate_password1(self, password):
        return get_adapter().clean_password(password)

    def validate(self, data):
        if data['password1'] != data['password2']:
            raise serializers.ValidationError(_("The two password fields didn't match."))
        return data

    def custom_signup(self, request, user):
        pass

    def get_cleaned_data(self):
        return {
            'username': self.validated_data.get('username', ''),
            'password1': self.validated_data.get('password1', ''),
            'email': self.validated_data.get('email', ''),
        }

    def save(self, request):
        adapter = get_adapter()
        user = adapter.new_user(request)
        self.cleaned_data = self.get_cleaned_data()
        user = adapter.save_user(request, user, self, commit=False)
        if "password1" in self.cleaned_data:
            try:
                adapter.clean_password(self.cleaned_data['password1'], user=user)
            except DjangoValidationError as exc:
                raise serializers.ValidationError(
                    detail=serializers.as_serializer_error(exc)
                )
        user.save()
        self.custom_signup(request, user)
```

```
        setup_user_email(request, user, [])
        return user
```

- 기본적으로 `username`, `email`, `password1`, `password2` 4가지 필드가 설정되어 있습니다.
- 또한, 저장 시 사용되는 `save()` 함수와 `get_cleaned_data()` 함수를 보면, 기본적으로 설정된 필드만 저장 시 사용 가능 하다는 것을 확인할 수 있습니다.

따라서, Model 만 재정의 해서는 rest-auth 에서 회원가입 시 재정의 한 필드를 사용할 수 없습니다.

RegisterSerializer 재정의

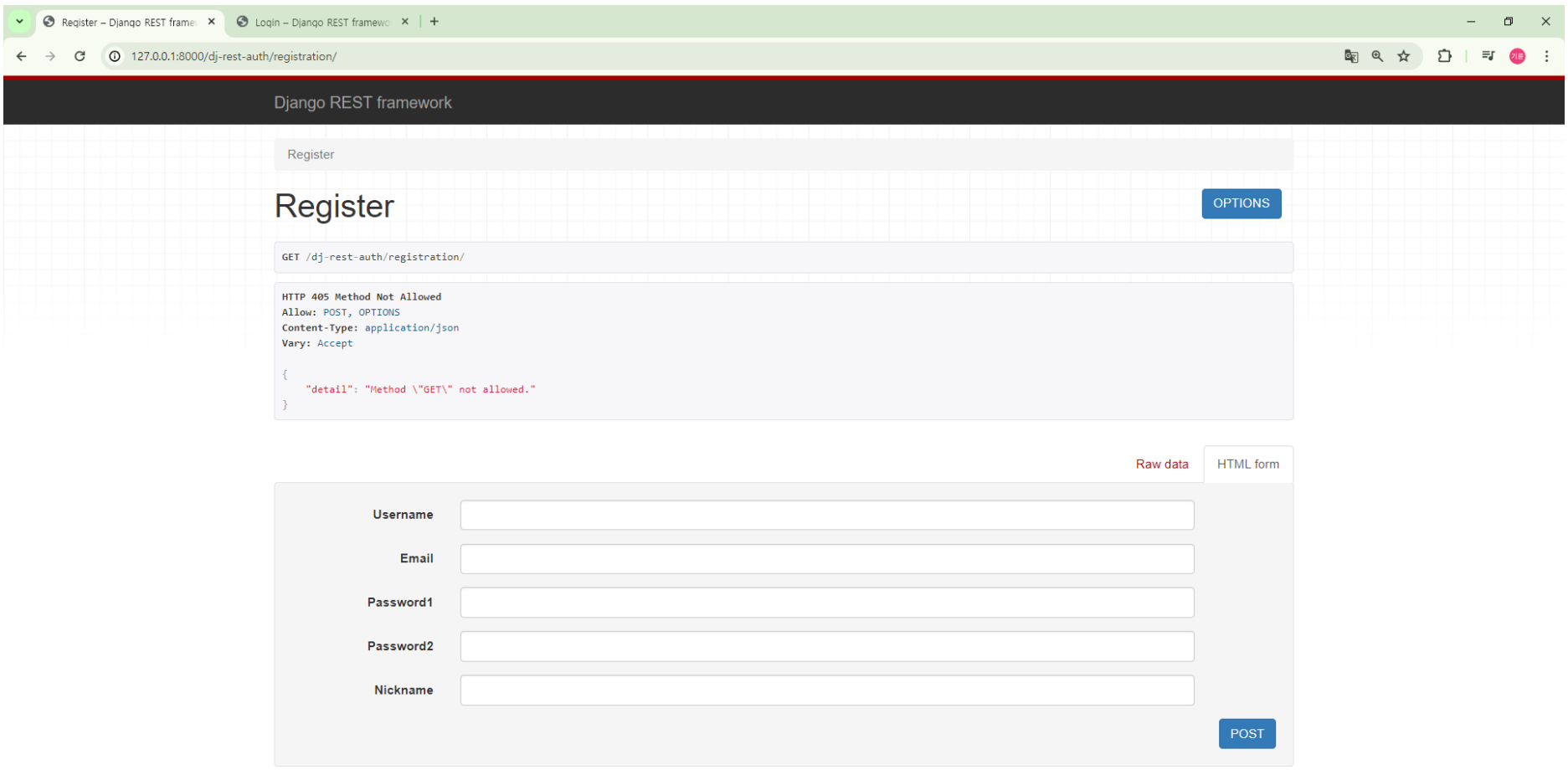
- accounts/serializer.py 에 다음과 같이 회원가입 시 사용할 serializer 를 재정의합니다.
  - 유저 기본 정보 + `nickname` 필드를 추가
  - `dj_rest_auth.registration.serializers.RegisterSerializer` 를 상속받아 필요한 부분을 추가해줍니다.

```
# accounts/serializers.py
from rest_framework import serializers
from dj_rest_auth.registration.serializers import RegisterSerializer
from .models import User

class CustomRegisterSerializer(RegisterSerializer):
    # 필요한 필드들을 추가합니다.
    nickname = serializers.CharField(
      required=False,
      allow_blank=True,
      max_length=255
    )

    # 해당 필드도 저장 시 함께 사용하도록 설정합니다.
    def get_cleaned_data(self):
        return {
            'username': self.validated_data.get('username', ''),
            'password1': self.validated_data.get('password1', ''),
            # nickname 필드 추가
            'nickname': self.validated_data.get('nickname', ''),
        }
```

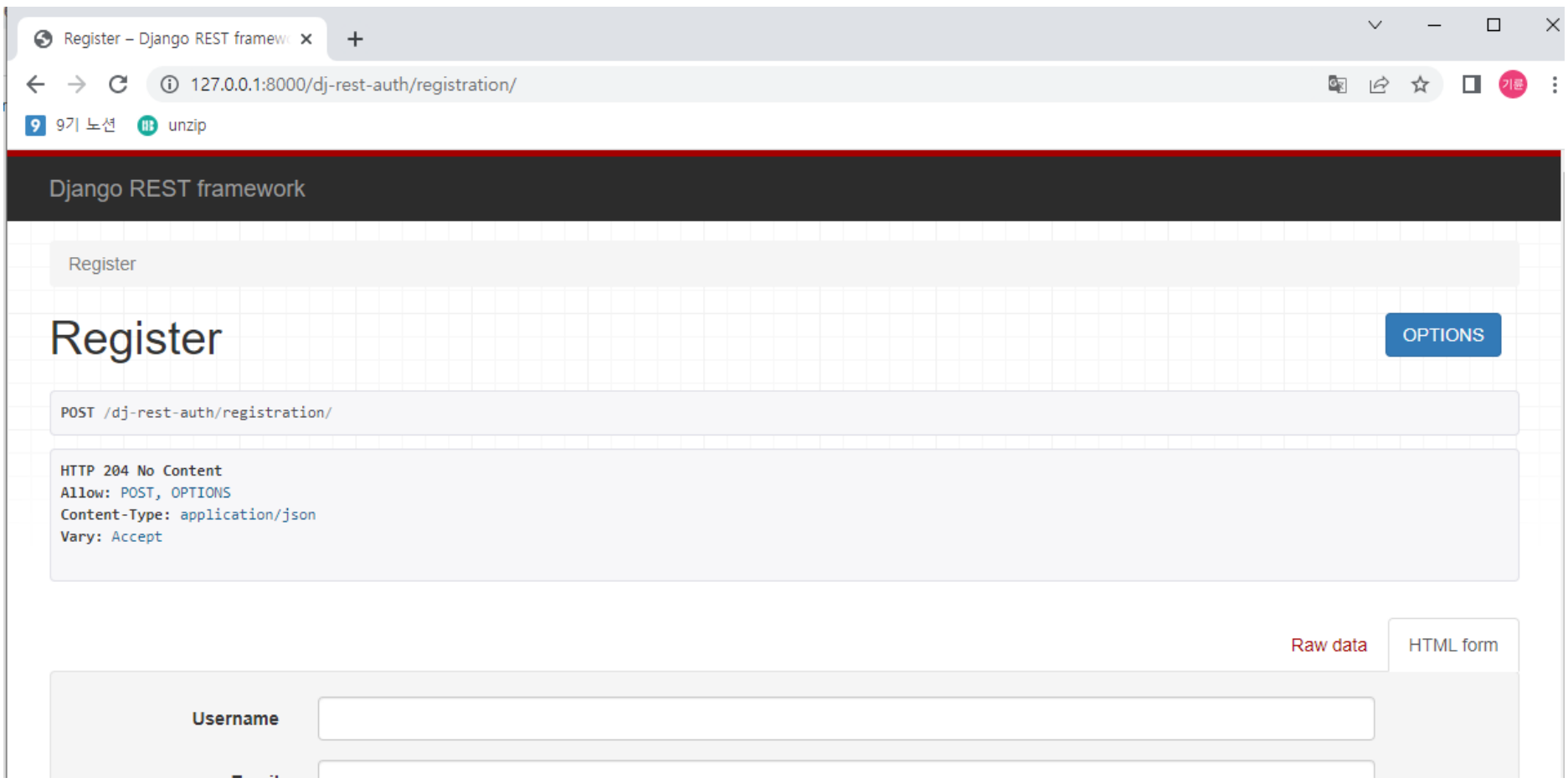
- settings.py 에 rest-auth 가 회원 가입 시 위에서 구현한 serializer 를 호출하도록 설정합니다.
  - dj-rest-auth 설정은 아래 공식 문서를 참고합니다.
  - https://dj-rest-auth.readthedocs.io/en/latest/configuration.html

```
# REST-AUTH 회원가입 기본 Serailizer 재정의
REST_AUTH = {
    'REGISTER_SERIALIZER': 'accounts.serializers.CustomRegisterSerializer',
}
```

- 결과
  - nickname 필드가 정상적으로 추가되었습니다.

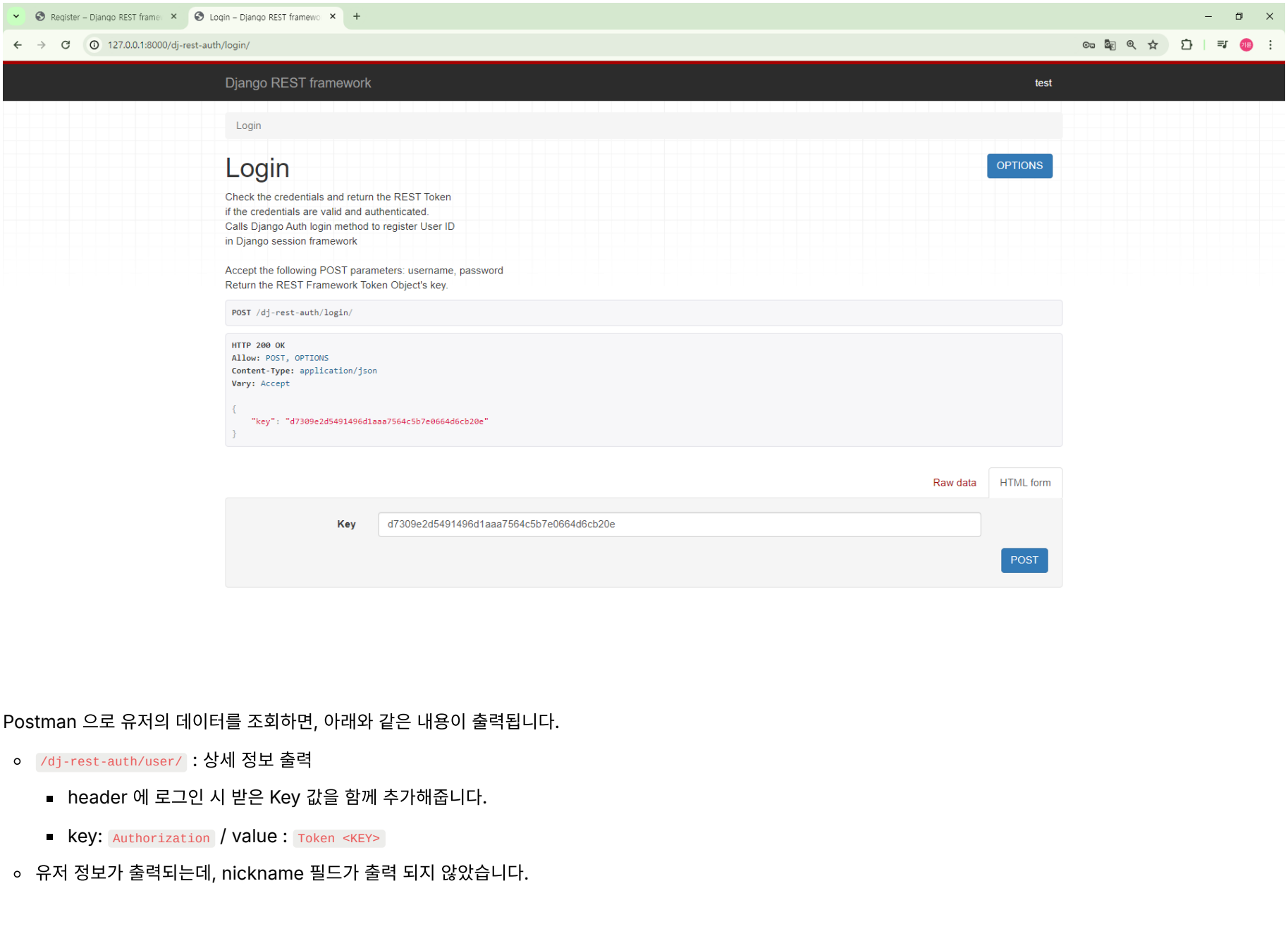


- 모든 데이터를 넣고 회원가입을 시도하면 가입이 진행됩니다.
- 이 때, 다음과 같이 `204 STATUS` 를 반환합니다.

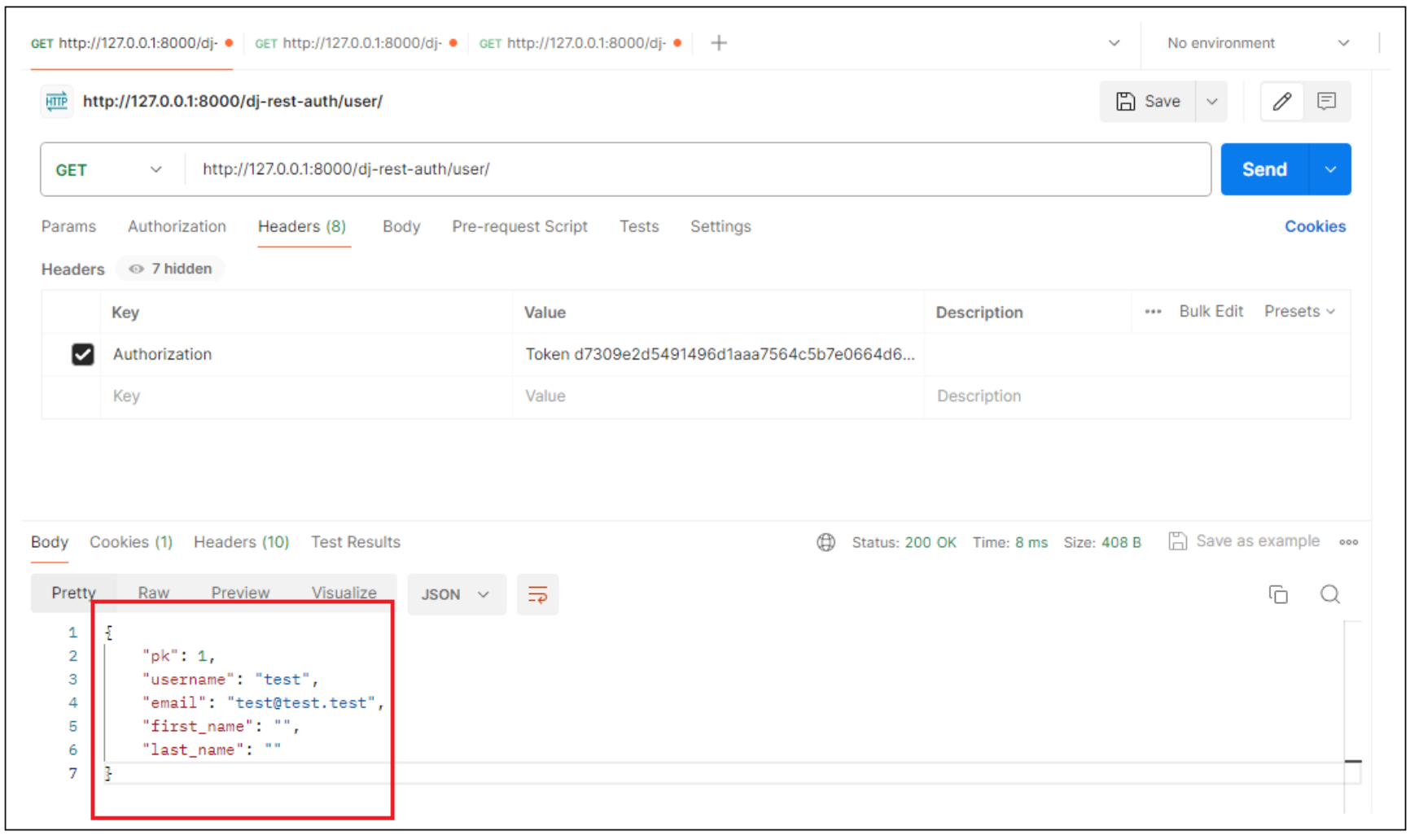


- 이는 dj_rest_auth 가 사용하는 `RegisterView` 를 재정의하여 하여 해결 할 수 있으나, 진도 외의 내용이므로 자동 로그인이 아닌 회원 가입 후 추가적으로 로그인 과정을 해주도록 합니다.

- 모든 필드 저장이 잘 되어 있는 지, 로그인 후 유저 상세 정보를 확인해보겠습니다.
- `/dj-rest-auth/login/` 페이지로 이동하여 로그인을 합니다.
  - 로그인 성공 시 아래와 같이 key 를 반환해 줍니다.



- Postman 으로 유저의 데이터를 조회하면, 아래와 같은 내용이 출력됩니다.
  - `/dj-rest-auth/user/` : 상세 정보 출력
    - header 에 로그인 시 받은 Key 값을 함께 추가해줍니다.
    - key: `Authorization` / value : `Token <KEY>`
  - 유저 정보가 출력되는데, nickname 필드가 출력 되지 않았습니다.



dj-rest-auth 의 설정을 다시 확인해보면 사용자의 정보를 돌려줄 때 사용하는 설정은 다음과 같습니다.

```
REST_AUTH = {
        ...
    'USER_DETAILS_SERIALIZER': 'dj_rest_auth.serializers.UserDetailsSerializer',
    ...
}
```

- UserDetailsSerializer 코드는 아래와 같습니다.

```
# dj-rest-auth/dj_rest_auth/serializers.py
class UserDetailsSerializer(serializers.ModelSerializer):
    """
    User model w/o password
    """

    @staticmethod
    def validate_username(username):
        if 'allauth.account' not in settings.INSTALLED_APPS:
            # We don't need to call the all-auth
            # username validator unless its installed
            return username

        from allauth.account.adapter import get_adapter
        username = get_adapter().clean_username(username)
        return username

    class Meta:
        extra_fields = []
        # see https://github.com/iMerica/dj-rest-auth/issues/181
        # UserModel.XYZ causing attribute error while importing other
        # classes from `serializers.py`. So, we need to check whether the auth model has
        # the attribute or not
        if hasattr(UserModel, 'USERNAME_FIELD'):
            extra_fields.append(UserModel.USERNAME_FIELD)
        if hasattr(UserModel, 'EMAIL_FIELD'):
            extra_fields.append(UserModel.EMAIL_FIELD)
        if hasattr(UserModel, 'first_name'):
            extra_fields.append('first_name')
        if hasattr(UserModel, 'last_name'):
            extra_fields.append('last_name')
        model = UserModel
        fields = ('pk', *extra_fields)
        read_only_fields = ('email',)
```

- Meta 쪽의 fields 를 보면, pk 값을 포함한 extra_fields 를 사용합니다.
  - 이 때, extra_fields 에 username, email, first_name, last_name 4 가지 종류만 설정되어 있습니다.
  - nickname 필드도 추가하여, 우리만의 UserDetailsSerializer 를 작성합니다.

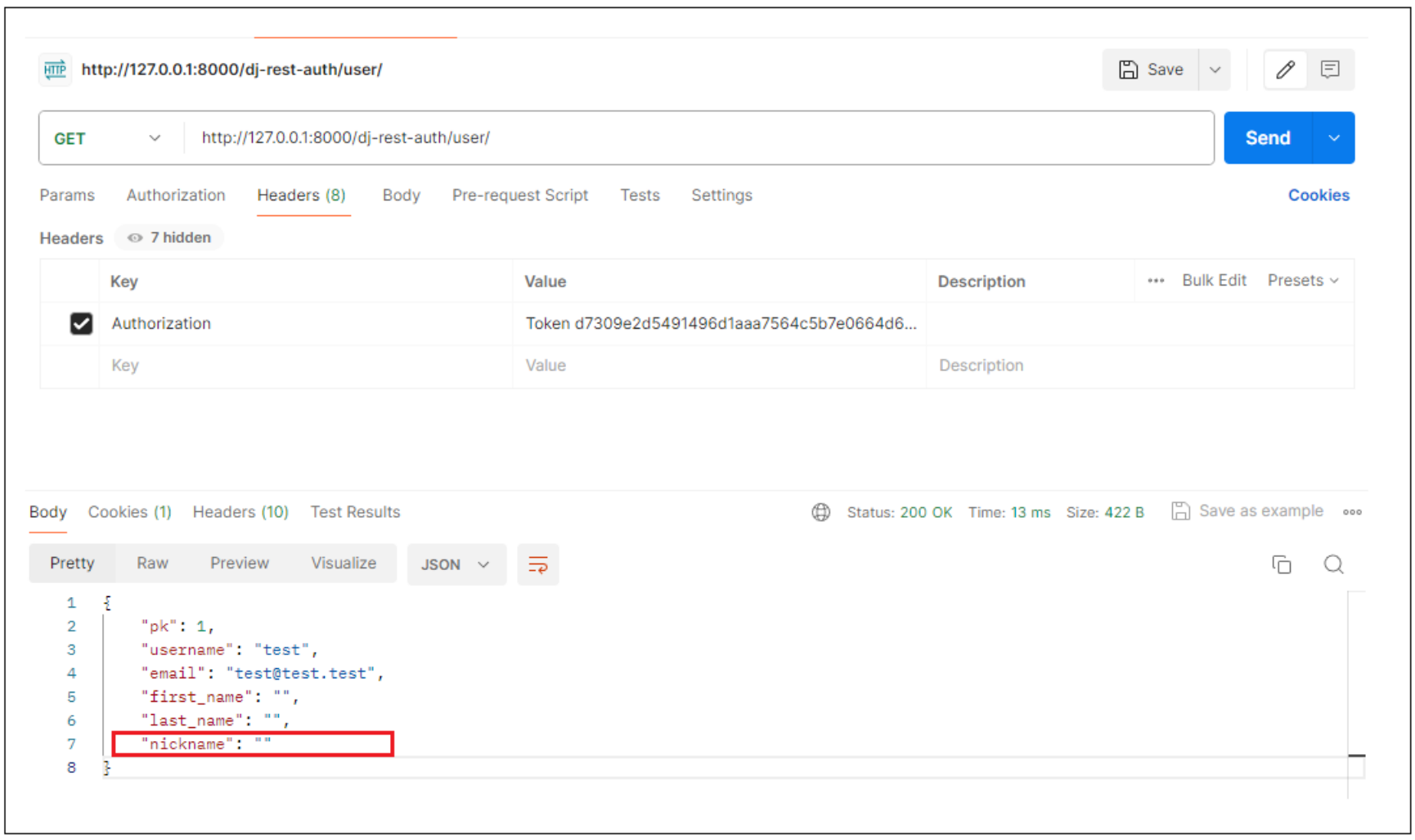
```
# accounts/serailizers.py
from django.contrib.auth import get_user_model
UserModel = get_user_model()

class CustomUserDetailsSerializer(UserDetailsSerializer):
    class Meta:
        extra_fields = []
        # see https://github.com/iMerica/dj-rest-auth/issues/181
        # UserModel.XYZ causing attribute error while importing other
        # classes from `serializers.py`. So, we need to check whether the auth model has
        # the attribute or not
        if hasattr(UserModel, 'USERNAME_FIELD'):
            extra_fields.append(UserModel.USERNAME_FIELD)
        if hasattr(UserModel, 'EMAIL_FIELD'):
            extra_fields.append(UserModel.EMAIL_FIELD)
        if hasattr(UserModel, 'first_name'):
            extra_fields.append('first_name')
        if hasattr(UserModel, 'last_name'):
            extra_fields.append('last_name')
        if hasattr(UserModel, 'nickname'):
            extra_fields.append('nickname')
        model = UserModel
        fields = ('pk', *extra_fields)
        read_only_fields = ('email',)
```

- 정의한 CustomUserDetailsSerializer 를 사용하도록 REST_AUTH 설정을 수정해줍니다.

```
# settings.py
# REST-AUTH 회원가입 기본 Serailizer 재정의
```

```
REST_AUTH = {
    'REGISTER_SERIALIZER': 'accounts.serializers.CustomRegisterSerializer',
    'USER_DETAILS_SERIALIZER': 'accounts.serializers.CustomUserDetailsSerializer',
}
```

- POSTMAN 으로 다시 한 번 User 정보를 확인하면 다음과 같은 정보가 출력됩니다.
  - 아직도 nickname 필드가 정상적으로 저장되지 않았습니다!!



이유는 아래와 같습니다.

allauth 의 adapter

- RegisterSerializer 의 save 함수를 다시 확인해보면 아래와 같습니다.

```
# dj_rest_auth/registration/serializers.py
class RegisterSerializer(serializers.Serializer):
        ...

        def save(self, request):
        adapter = get_adapter()
        user = adapter.new_user(request)
        self.cleaned_data = self.get_cleaned_data()
        user = adapter.save_user(request, user, self, commit=False)
        if "password1" in self.cleaned_data:
            try:
                adapter.clean_password(self.cleaned_data['password1'], user=user)
            except DjangoValidationError as exc:
                raise serializers.ValidationError(
                    detail=serializers.as_serializer_error(exc)
                )
        user.save()
        self.custom_signup(request, user)
        setup_user_email(request, user, [])
        return user
```

- `adater = get_adapter()` 코드를 확인할 수 있습니다.

    - `get_adapter` 코드를 따라가면 `allauth` 의 코드가 나옵니다.
        - `from allauth.account.adapter import get_adapter` 로 가져오고 있습니다.
        - dj-rest-auth 에서 allauth 를 사용하는 것을 확인할 수 있습니다.
        - allauth github

```
# django-allauth/allauth/account/adapter.py

def get_adapter(request=None):
    return import_attribute(app_settings.ADAPTER)(request)
```

- `app_settings.ADAPTER` 를 github 를 이용하여 따라가보면, 다음 코드를 볼 수 있습니다.

```
# django-allauth/allauth/account/app_settings.py

@property
def ADAPTER(self):
    return self._setting("ADAPTER", "allauth.account.adapter.DefaultAccountAdapter")
```

- 기본 값으로 `DefaultAccountAdapter` 를 사용한다는 것을 알 수 있습니다.
- `DefaultAccountAdapter` 코드

```
# django-allauth/allauth/account/adapter.py

class DefaultAccountAdapter(object):
        ...

        def save_user(self, request, user, form, commit=True):
        """
        Saves a new `User` instance using information provided in the
        signup form.
        """
        from .utils import user_email, user_field, user_username

        data = form.cleaned_data
        first_name = data.get("first_name")
        last_name = data.get("last_name")
        email = data.get("email")
        username = data.get("username")
        user_email(user, email)
        user_username(user, username)
        if first_name:
            user_field(user, "first_name", first_name)
        if last_name:
            user_field(user, "last_name", last_name)
        if "password1" in data:
            user.set_password(data["password1"])
        else:
            user.set_unusable_password()
        self.populate_username(request, user)
        if commit:
            # Ability not to commit makes it easier to derive from
            # this adapter by adding
            user.save()
        return user
```

- 위 코드를 보면 저장 시 사용하는 `save_user` 함수에서도 필드들이 고정이 되어있습니다.
- 여기서 **Adapter 또한 커스터마이징을 해야한다는 것을 알 수 있습니다.**

Adapter 커스터마이징

- 기본적으로 사용하는 `DefaultAccountAdapter` 를 상속받아, 우리가 수정이 필요한 `save_user` 함수만 오버라이딩 하면 쉽게 사용할 수 있습니다.

```
# accounts/models.py

from allauth.account.adapter import DefaultAccountAdapter

class CustomAccountAdapter(DefaultAccountAdapter):
    def save_user(self, request, user, form, commit=True):
```

```
        """
        Saves a new `User` instance using information provided in the
        signup form.
        """
        from allauth.account.utils import user_email, user_field, user_username

        data = form.cleaned_data
        first_name = data.get("first_name")
        last_name = data.get("last_name")
        email = data.get("email")
        username = data.get("username")
        # nickname 필드를 추가
        nickname = data.get("nickname")

        user_email(user, email)
        user_username(user, username)
        if first_name:
            user_field(user, "first_name", first_name)
        if last_name:
            user_field(user, "last_name", last_name)
        if nickname:
            user_field(user, "nickname", nickname)
        if "password1" in data:
            user.set_password(data["password1"])
        else:
            user.set_unusable_password()
        self.populate_username(request, user)
        if commit:
            # Ability not to commit makes it easier to derive from
            # this adapter by adding
            user.save()
        return user
```

- 필요 시, 기존 코드를 참고하여 추가적인 내용들을 작성합니다.

- 회원 가입 시 사용하는 `allauth.account.adapter.DefaultAccountAdapter` 를 위에서 구현한 adapter 로 설정해주어야 합니다.
  - settings.py 에 아래 내용을 추가한다.

```
ACCOUNT_ADAPTER  = 'accounts.models.CustomAccountAdapter'
```

- 이제 다시 한 번 확인해보겠습니다.
  - 신규 유저 회원가입 → 로그인 → 해당 User Token 으로 상세정보 조회
  - 아래와 같이 정상적으로 저장되고 출력됩니다!

http://127.0.0.1:8000/dj-rest-auth/user/

Save

GET http://127.0.0.1:8000/dj-rest-auth/user/ Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Headers 7 hidden

Key	Value	Description	Bulk Edit Presets
Authorization	Token 4ef7e0cadc22a4c07f9ec89c91cda7186159f...		
Key	Value	Description	

Body Cookies (1) Headers (10) Test Results Status: 200 OK Time: 6 ms Size: 430 B Save as example

Pretty Raw Preview Visualize JSON


```
{
    "pk": 2,
    "username": "test1",
    "email": "test1@test.test",
    "first_name": "",
    "last_name": "",
    "nickname": "test1"
}
```