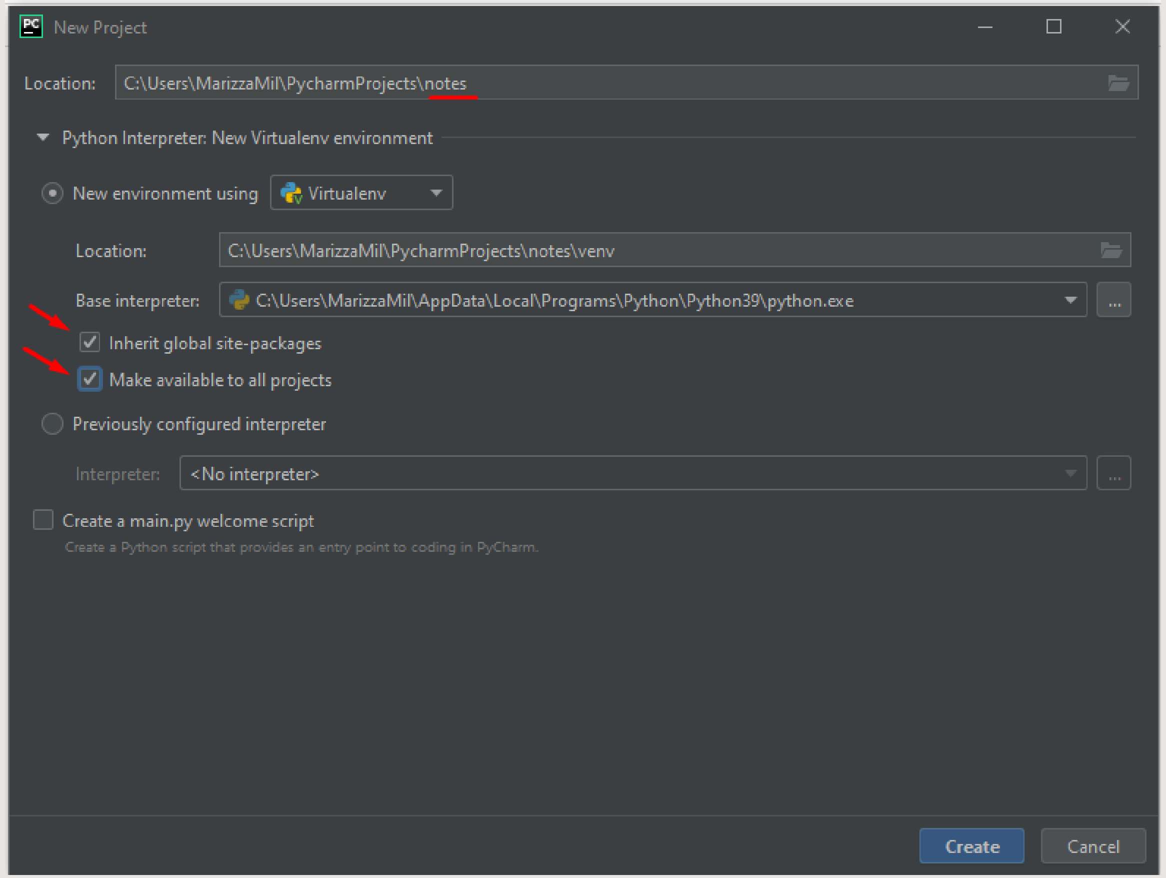
JWT Auth with DjangoREST API

Token based authentication allows the server and the frontend (whether that be web, native mobile or something else) to be decoupled and reside on different domains. ~~JSON Web Tokens~~ (JWT) is a popular implementation of token based authentication, and in this article we'll use it to authenticate users in an API for notes built with ~~Django REST Framework~~.

We'll set up user registration and authentication, and we will define the notes model. We will also ensure that the current logged in user is set as the owner of a note when it is created, and that a user can perform read and write operations only to his own notes.

Adding JSON Web Token Authentication

- Create project in PyCharm



New Project

Location: C:\Users\MarizzaMil\PycharmProjects\notes

Python Interpreter: New Virtualenv environment

● New environment using Virtualenv

Location: C:\Users\MarizzaMil\PycharmProjects\notes\venv

Base interpreter: C:\Users\MarizzaMil\AppData\Local\Programs\Python\Python39\python.exe

☑ Inherit global site-packages

☑ Make available to all projects

○ Previously configured interpreter

Interpreter: <No interpreter>

□ Create a main.py welcome script

Create a Python script that provides an entry point to coding in PyCharm.

Adding JSON Web Token Authentication

- Create project in PyCharm
- Install Django, DjangoREST and django-cors-headers:

```
$ pip install django django-rest-framework django-cors-headers
```

- We install basic Django, as well as Django REST Framework, which will assist us in implementing the API. We also install django-cors-headers, which makes it possible to access our endpoints from browsers on other domains than the Django server.
- Next we create a new Django project:

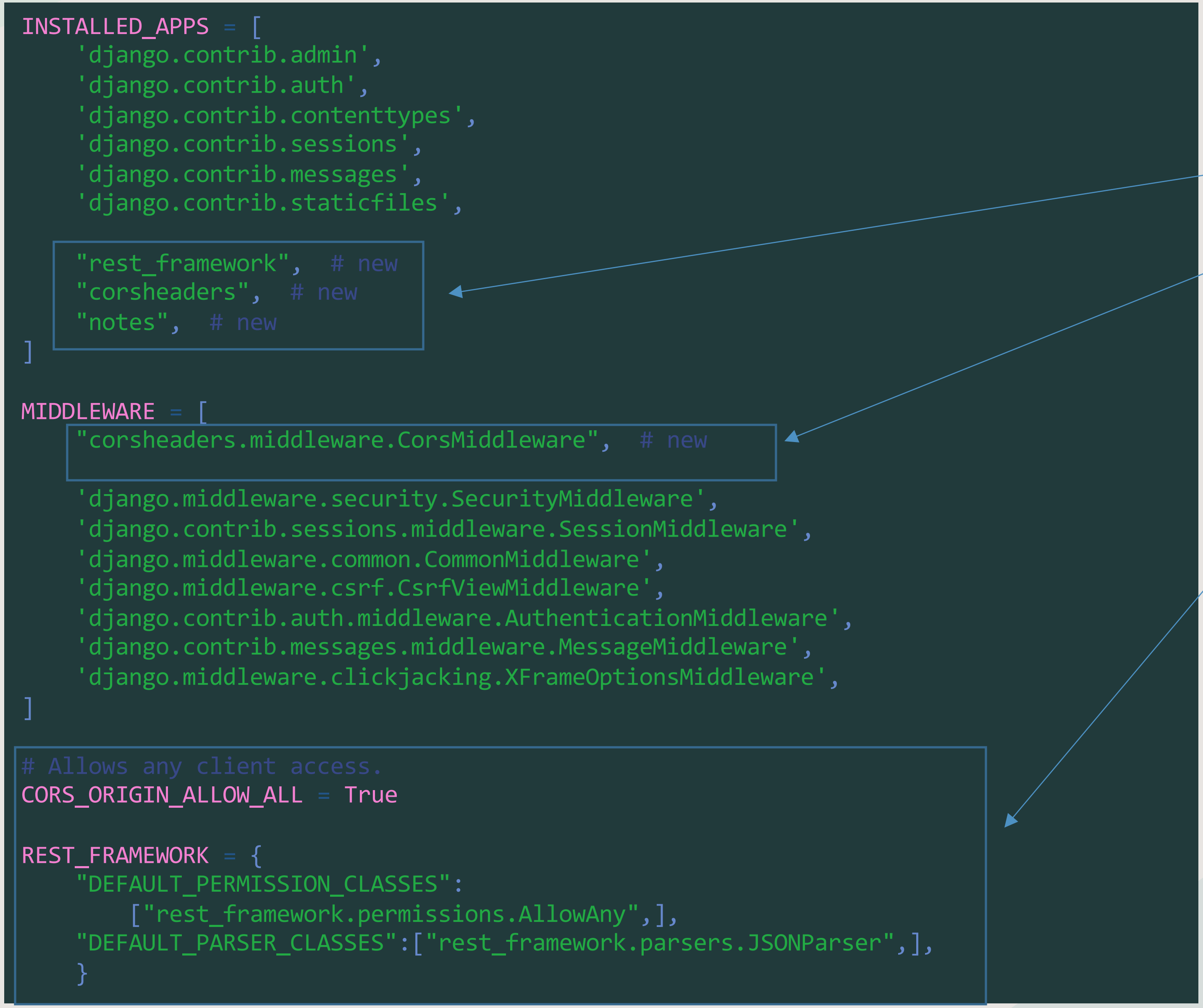
```
$ django-admin startproject project .
$ cd project/
```

- The dot at the end of the command will create the project inside the project we're in, the notes directory.
- Next we create a notes app for our project and run migrations:

```
$ python manage.py startapp notes
$ python manage.py migrate
```

Adding JSON Web Token Authentication

- We'll add the installed apps and REST Framework settings in project/settings.py:



```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    "rest_framework",  # new
    "corsheaders",  # new
    "notes",  # new
]

MIDDLEWARE = [
    "corsheaders.middleware.CorsMiddleware",  # new

    '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',
]

# Allows any client access.
CORS_ORIGIN_ALLOW_ALL = True

REST_FRAMEWORK = {
    "DEFAULT_PERMISSION_CLASSES":
        ["rest_framework.permissions.AllowAny",],
    "DEFAULT_PARSER_CLASSES":["rest_framework.parsers.JSONParser",],
    }
```

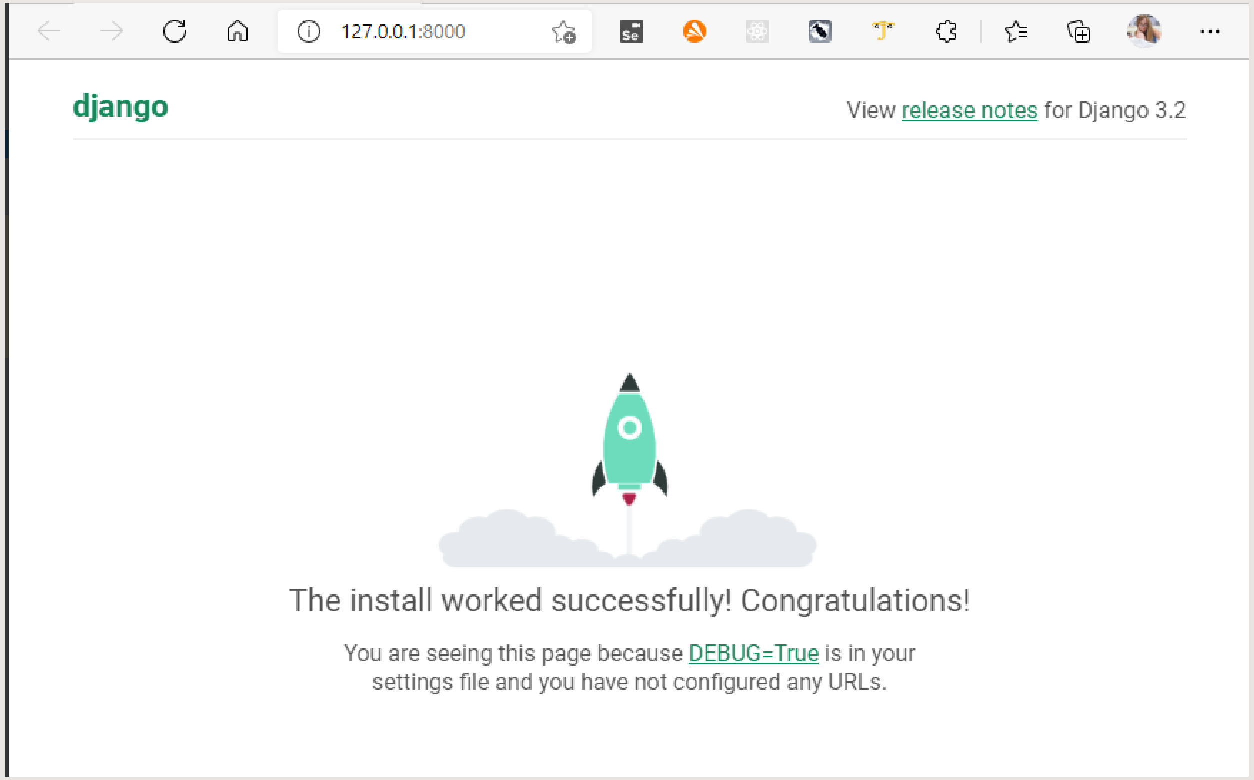
Add
code

Adding JSON Web Token Authentication

- To check that everything's working, let's run the server:

```
$ python manage.py runserver
```

- Visit http://localhost:8000
- You should see the Django welcome page.



django

View release notes for Django 3.2

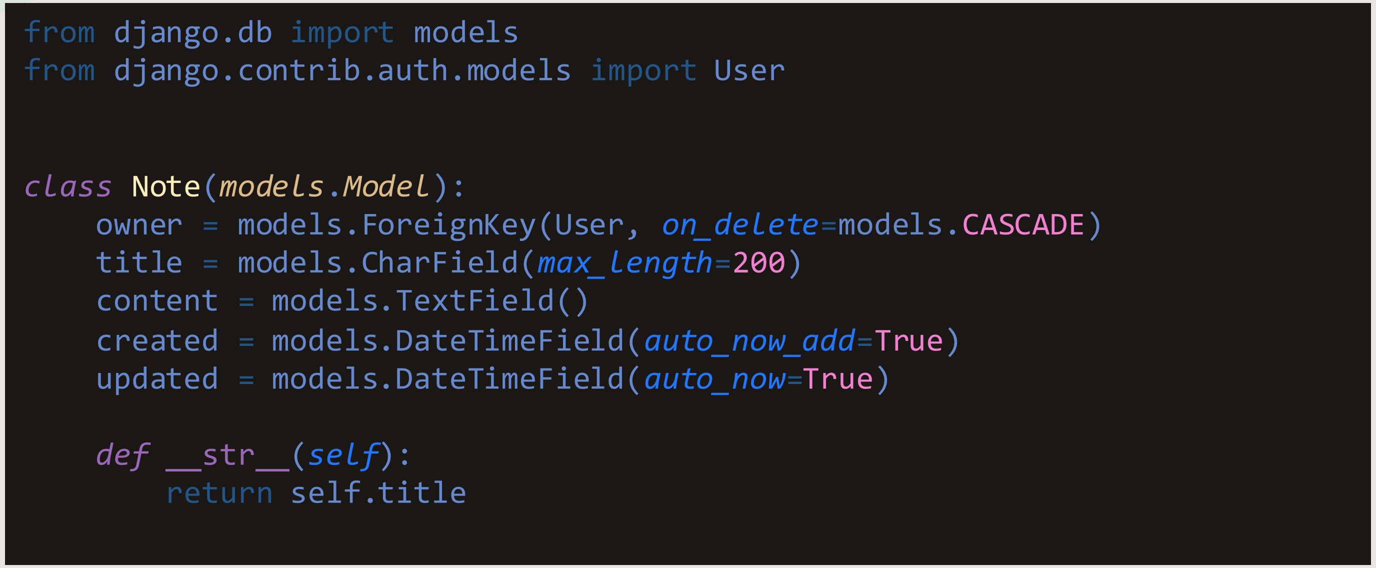
The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

But we still have no API and no data to serve in the API. Let's make a notes model next.

Create a Notes Model and a Superuser

In notes/models.py add the following:

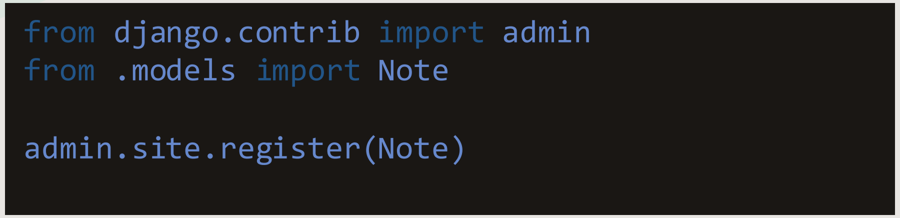


```python
from django.db import models
from django.contrib.auth.models import User


class Note(models.Model):
    owner = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    content = models.TextField()
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.title
```

We create a Note model with five attributes. The owner is a User object, which we import from Django's auth module. Later, when we have added authentication, we will ensure that the owner is set to the currently logged in user. We also have title and content for the note, a shorter char field and a longer text field, respectively. Lastly we add a created and updated field.

Create a Notes Model and a Superuser

Also register the model with the admin site so that it will appear there. notes/admin.py:



```
from django.contrib import admin
from .models import Note

admin.site.register(Note)
```

Since we've created a new model we need to create and run migrations to sync it with the database:

```
$ python manage.py makemigrations
$ python manage.py migrate
```

Let's also create a superuser so we can inspect and create notes objects in the admin panel. (You can just select the default username and leave email blank when prompted.)

```
$ python manage.py createsuperuser
```

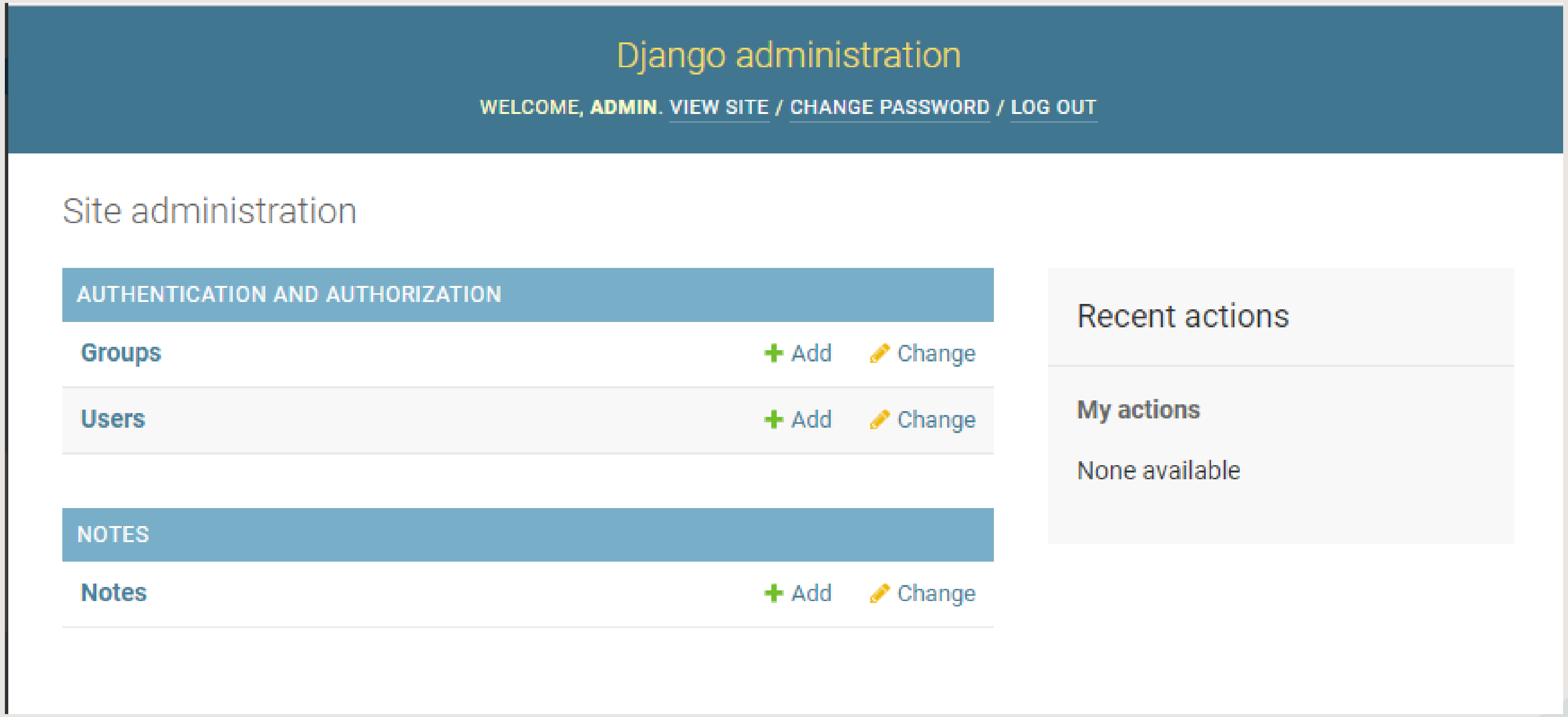
Let's start the server again.

```
$ python manage.py runserver
```

Create a Notes Model and a Superuser

Then visit http://localhost:8000/admin/ and log in with your superuser's credentials.

You should see something like this:



Django administration

WELCOME, **ADMIN**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

NOTES		
Notes	+ Add	Change

Recent actions

My actions

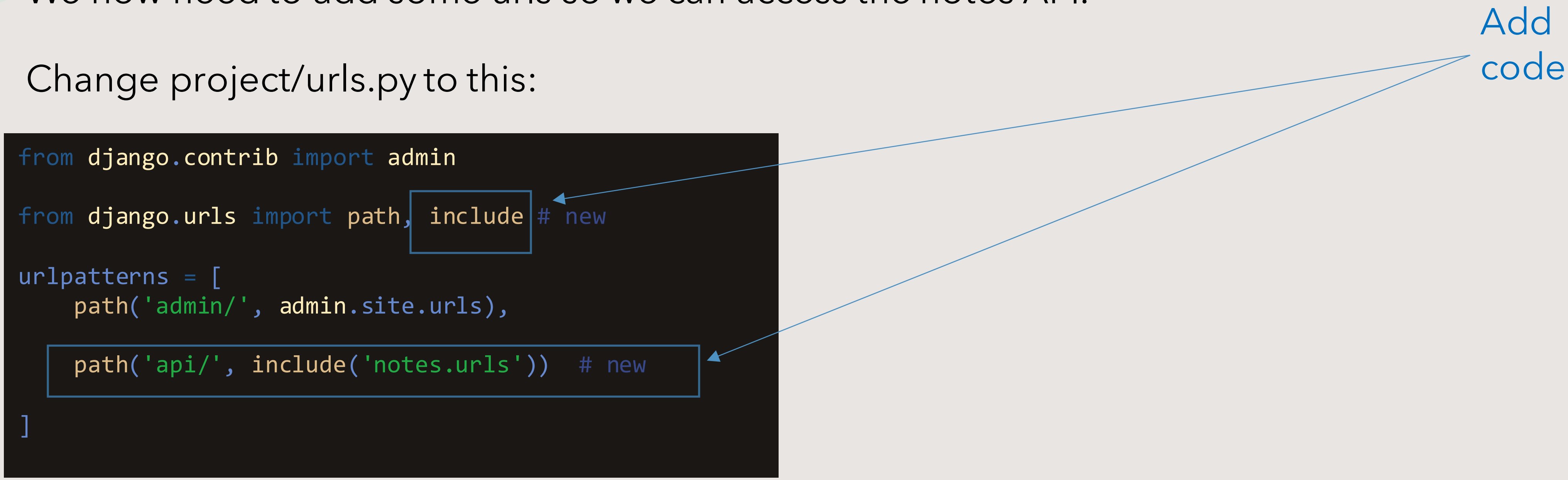
None available

Add a couple of notes so we have some data to work with.

Serve the Notes data as an API

We now need to add some urls so we can access the notes API.

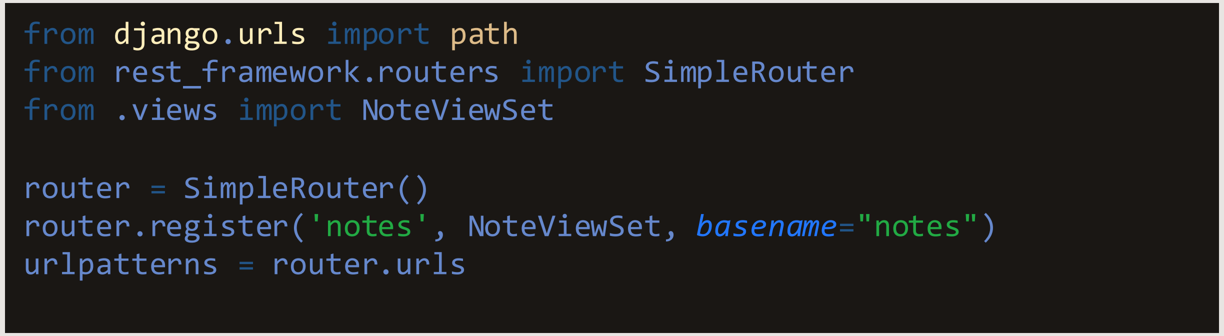
Change project/urls.py to this:



```
from django.contrib import admin

from django.urls import path, include # new

urlpatterns = [
    path('admin/', admin.site.urls),

    path('api/', include('notes.urls'))  # new

]
```

Add code

Next create the file notes/urls.py and add the following:



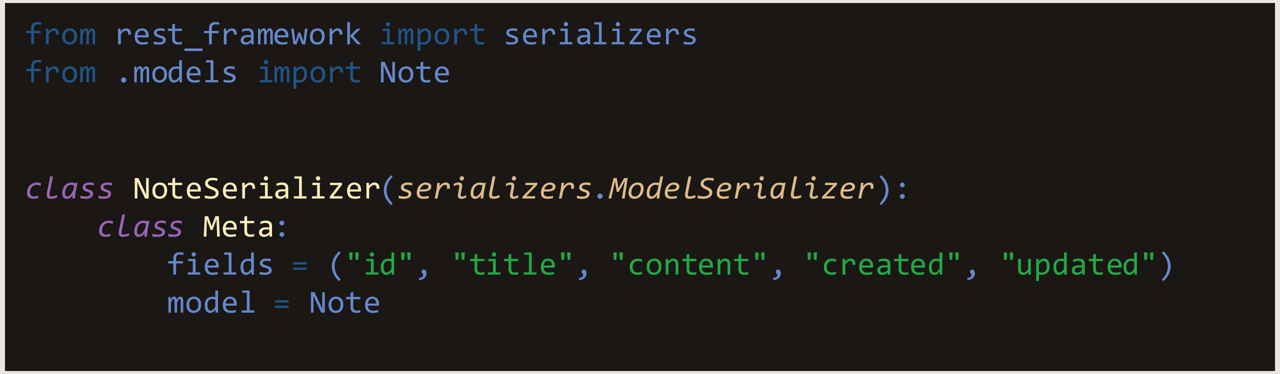
```
from django.urls import path
from rest_framework.routers import SimpleRouter
from .views import NoteViewSet

router = SimpleRouter()
router.register('notes', NoteViewSet, basename="notes")
urlpatterns = router.urls
```

Serve the Notes data as an API

We’re using rest_framework’s SimpleRouter to automatically create the routes for us. Also notice that we import NoteViewSet, which we haven’t created yet. We’ll do that soon, but first we need to create a serializer for our model.

Serializers come with rest_framework and provide a way to translate our model’s data back and forth to a format that is suitable for our API, in our case JSON. Serializers can also perform validations as well as let us specify which fields to include in the data we exchange with the rest of the world.

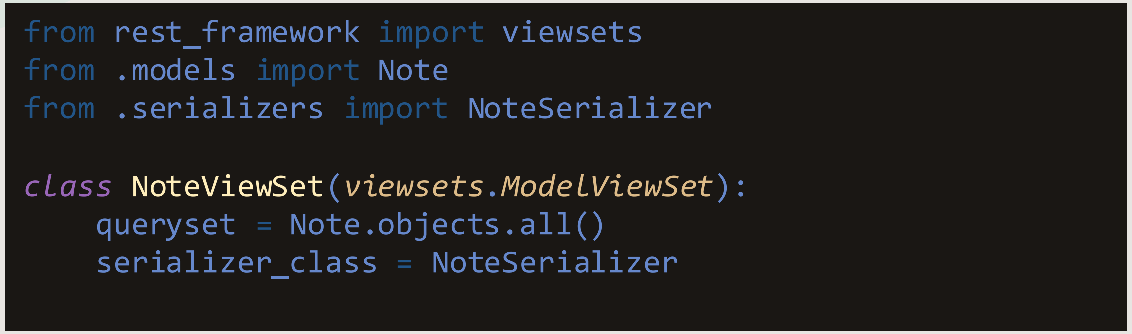
Create the file notes/serializers.py:



```python
from rest_framework import serializers
from .models import Note


class NoteSerializer(serializers.ModelSerializer):
    class Meta:
        fields = ("id", "title", "content", "created", "updated")
        model = Note
```

Serve the Notes data as an API

Next we'll create the view in notes/views.py:



```
from rest_framework import viewsets
from .models import Note
from .serializers import NoteSerializer

class NoteViewSet(viewsets.ModelViewSet):
    queryset = Note.objects.all()
    serializer_class = NoteSerializer
```

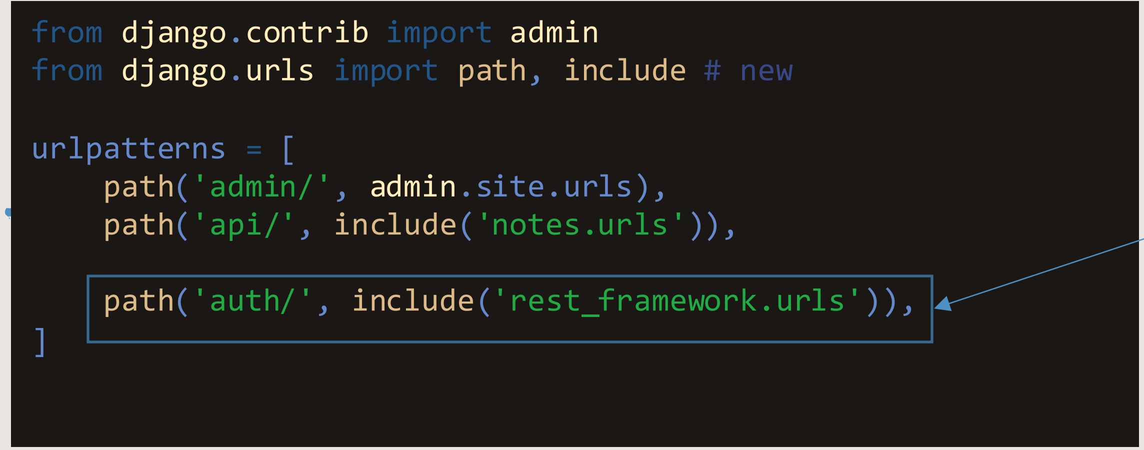
Ensure that your are logged in with your superuser, and go to http://localhost:8000/api/notes/

You should now be able to see the notes you have created, as well as create new notes.

Authentication and Permissions

For us to explore authentication and permission, create a new user in the admin panel. (All you need to add is username and password.)

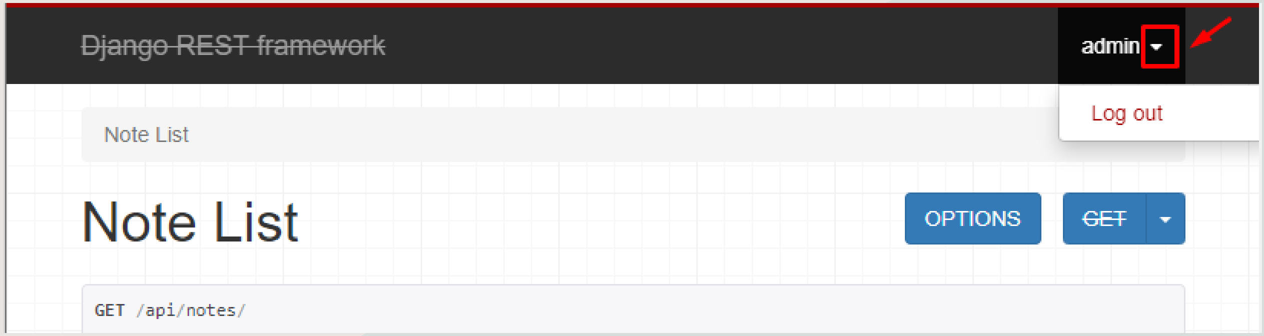
Then add the following line to the urlpatterns list in project/urls.py



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

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('notes.urls')),

    path('auth/', include('rest_framework.urls')),
]
```

Add code

This ensures that we have a path for logging in with django_rest. You'll now see that there's a down arrow by the user name in the upper right corner:



```
GET /api/notes/
```

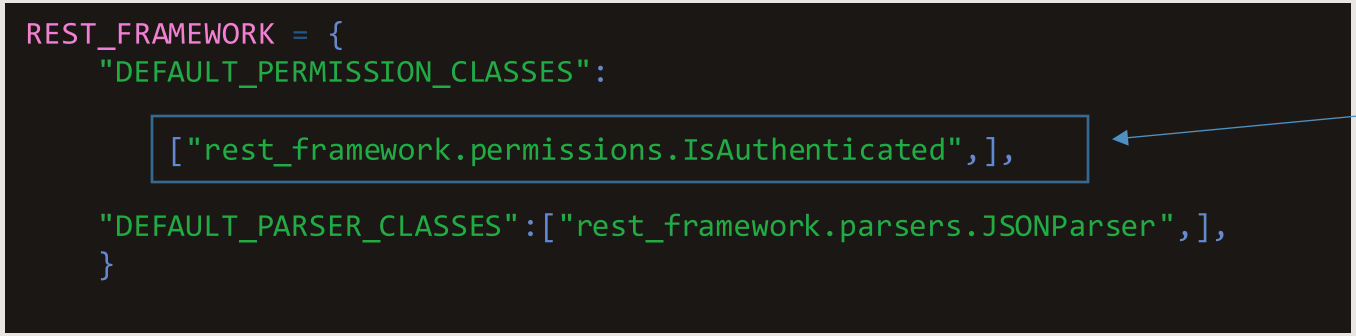
If you click on it you can log out and switch between users.

Authentication and Permissions

One obvious problem now is that every user can see every other user's notes. Even worse, users who are not logged in can see and create notes. (You can verify this by logging on and then going to http://localhost:8000/api/notes/)

Let's fix that.

The reason even users who are not logged in have access to notes, is that we have configured rest_framework with AllowAny. Go into your project/settings.py file and make the following change:



```
REST_FRAMEWORK = {
    "DEFAULT_PERMISSION_CLASSES":

        ["rest_framework.permissions.IsAuthenticated",],

    "DEFAULT_PARSER_CLASSES":["rest_framework.parsers.JSONParser",],
    }
```

Change code

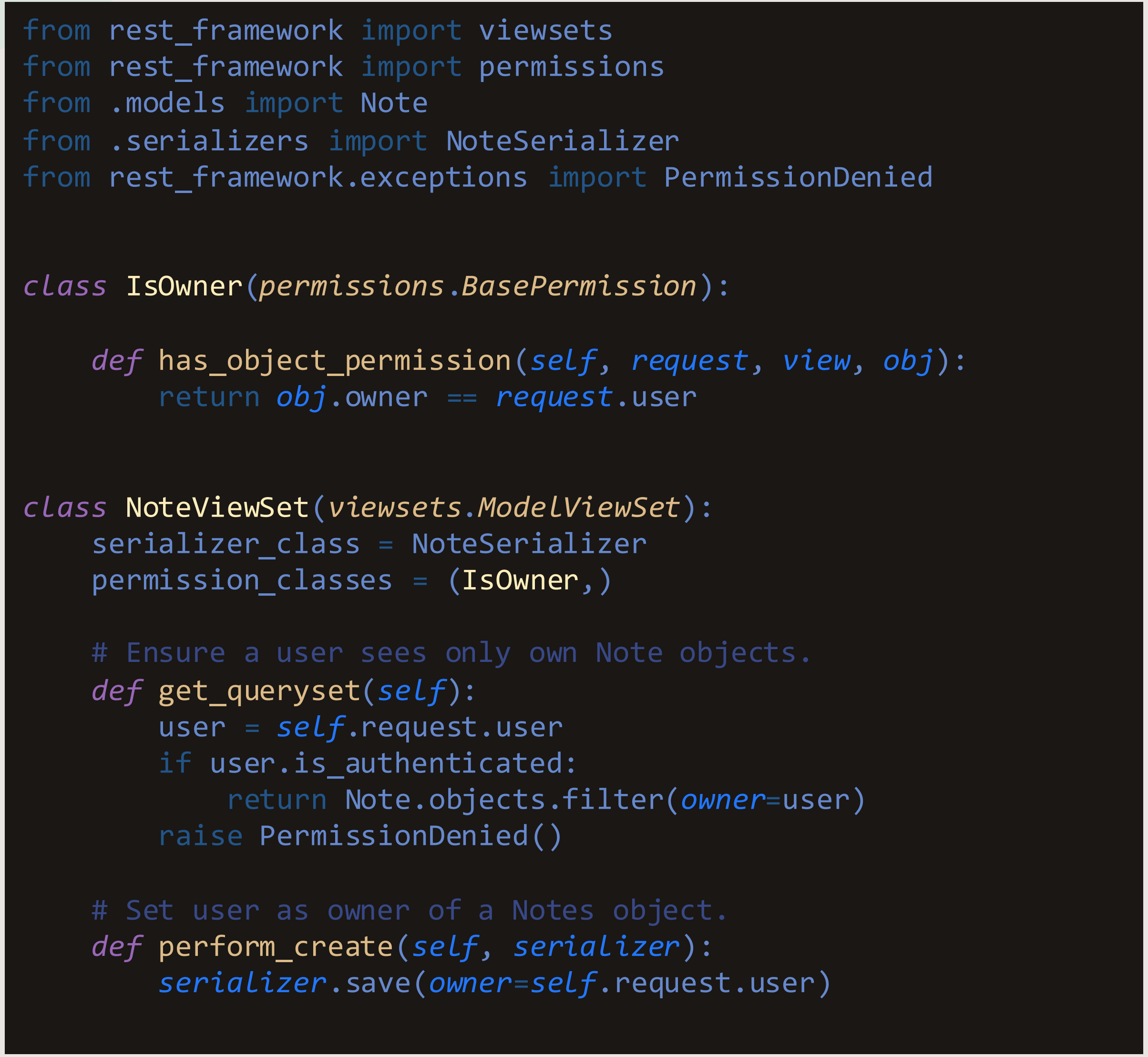
Then try to access http://localhost:8000/api/notes/

You should no longer see any notes, but instead receive the following message: "detail": "Authentication credentials were not provided."

Log in again, and you will see the notes. How can we fix the problem with one logged in user being able to see every other user's notes?

Authentication and Permissions

Make the following changes in notes/views.py:



```python
from rest_framework import viewsets
from rest_framework import permissions
from .models import Note
from .serializers import NoteSerializer
from rest_framework.exceptions import PermissionDenied


class IsOwner(permissions.BasePermission):

    def has_object_permission(self, request, view, obj):
        return obj.owner == request.user


class NoteViewSet(viewsets.ModelViewSet):
    serializer_class = NoteSerializer
    permission_classes = (IsOwner,)

    # Ensure a user sees only own Note objects.
    def get_queryset(self):
        user = self.request.user
        if user.is_authenticated:
            return Note.objects.filter(owner=user)
        raise PermissionDenied()

    # Set user as owner of a Notes object.
    def perform_create(self, serializer):
        serializer.save(owner=self.request.user)
```

We remove the queryset property from the class and instead override the get_queryset method, where we filter the notes objects to ensure we only return those who belong to the current user.

Furtermore we add an IsOwner permission check. This ensures that a user can only modify (update/delete) his own objects.

Lastly we override the perform_create method so that when a new note object is created the owner is always set to the current user.

If you visit http://localhost:8000/api/notes/ you should only see the notes belonging to the current logged in user. If you try to make a new note and set the owner to be another user than the one you are logged in as, the note will be created, but the user will be the current logged in user regardless.

Authentication and Permissions

We have now implemented authentication and permissions, but django_rest is still using session_authentication.

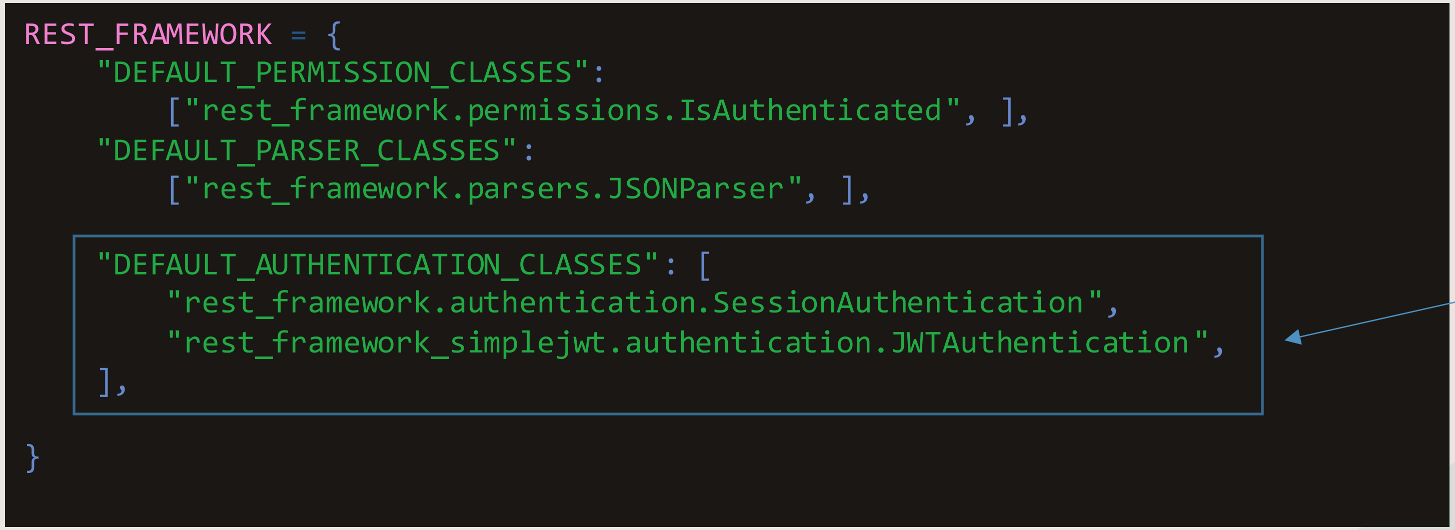
What we want is for any client, web or otherwise, to be able to register a user, login and logout and be authenticated from anywhere. To do that we will use JWT.

Adding JSON Web Token Authentication

To implement token authentication with JWT, we will use a library, Simple JWT:

```
$ pip install djangorestframework_simplejwt
```

Then we need to add it to our list of authentication classes in our mysite/settings.py file:

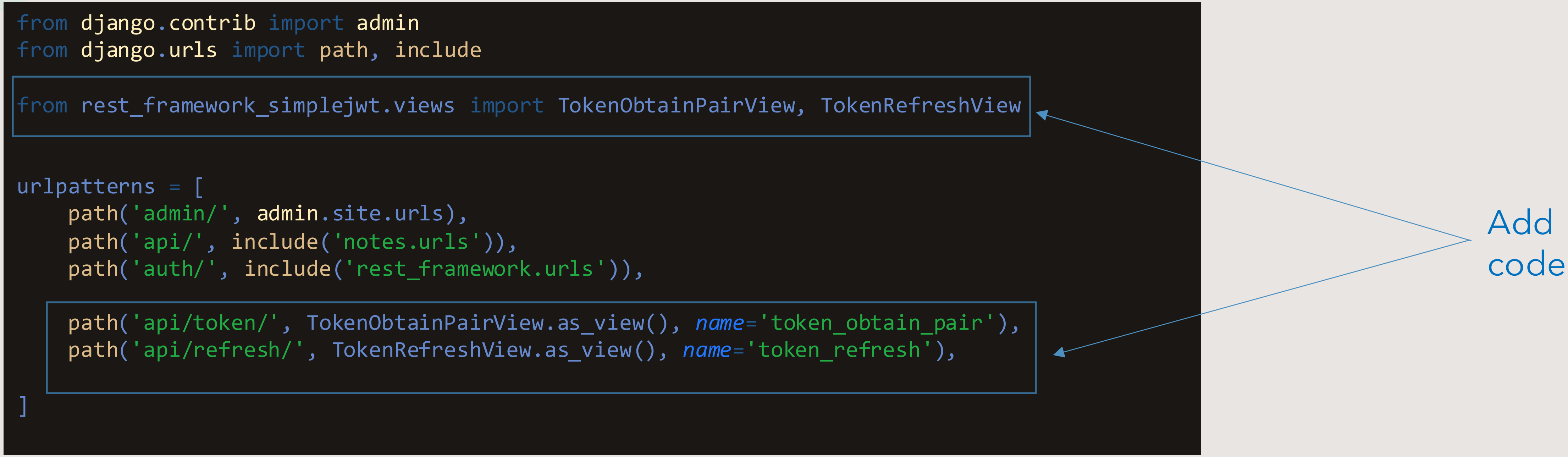


```
REST_FRAMEWORK = {
    "DEFAULT_PERMISSION_CLASSES":
        ["rest_framework.permissions.IsAuthenticated", ],
    "DEFAULT_PARSER_CLASSES":
        ["rest_framework.parsers.JSONParser", ],

    "DEFAULT_AUTHENTICATION_CLASSES": [
        "rest_framework.authentication.SessionAuthentication",
        "rest_framework_simplejwt.authentication.JWTAuthentication",
    ],

}
```

Add code

Adding JSON Web Token Authentication

And add to new endpoints to our mysite/urls.py file:



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

from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView


urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('notes.urls')),
    path('auth/', include('rest_framework.urls')),

    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/refresh/', TokenRefreshView.as_view(), name='token_refresh'),

]
```

Add code

These new endpoints provide what we need for user log in. The route for login would be /api/token/. The /api/refresh/ route is used to get a new token before the old expires.

And we actually don't need an endpoint for logging out, since the server doesn't maintain any state. To log out we can simply delete the token on the client. The token will expire "on it's own" (the time can be set using Simple JWT settings).

But what we currently miss is a way to register a user and get a JWT token back.

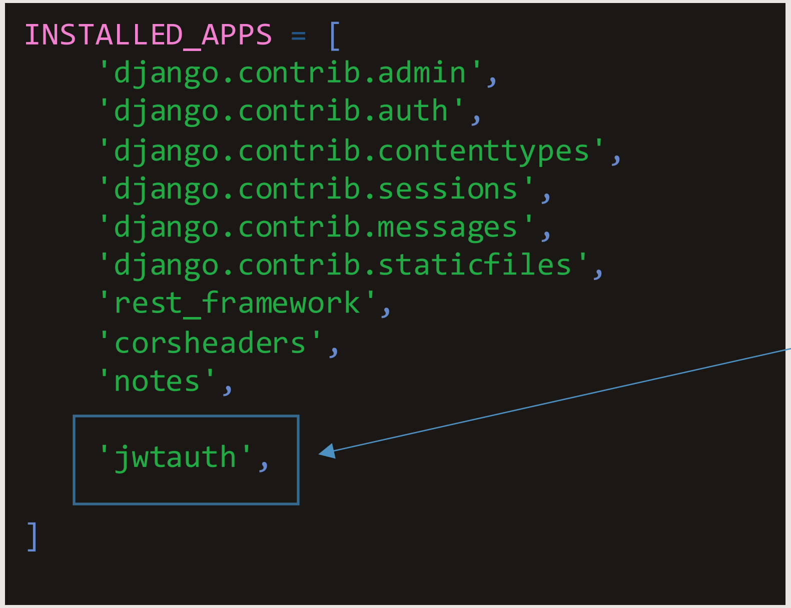
Adding JSON Web Token Authentication

User Sign Up

We need a new endpoint where users can sign up. This doesn’t belong in the Notes app, since this is another domain, authentication. So we’ll start by creating a new app, jwtauth:

```
$ python manage.py startapp jwtauth
```

Add app in project/settings.py:

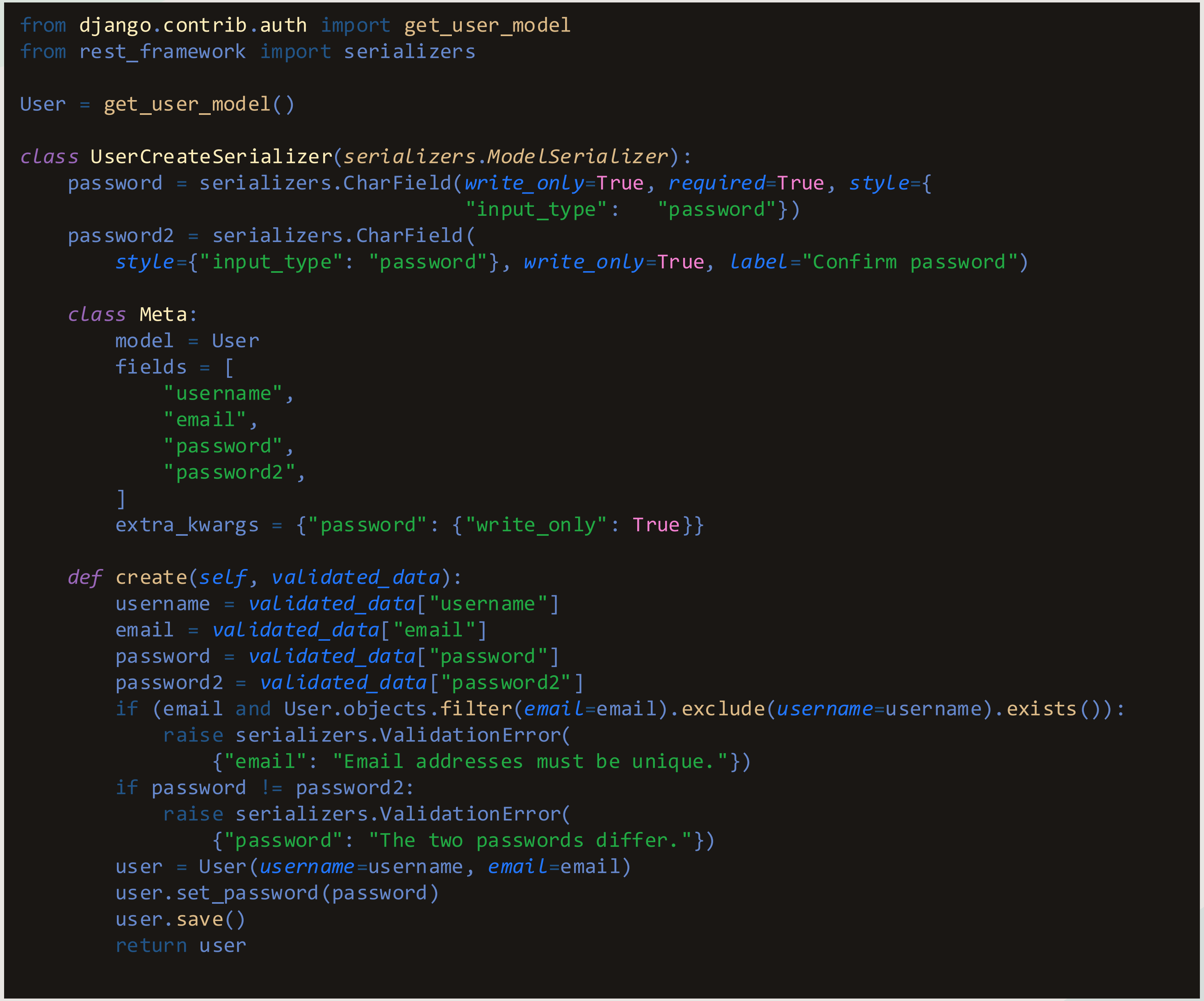


```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'corsheaders',
    'notes',

    'jwtauth',

]
```

Add code

Adding JSON Web Token Authentication

Next we'll need a serializer for the User object. Add jwtauth/serializers.py:



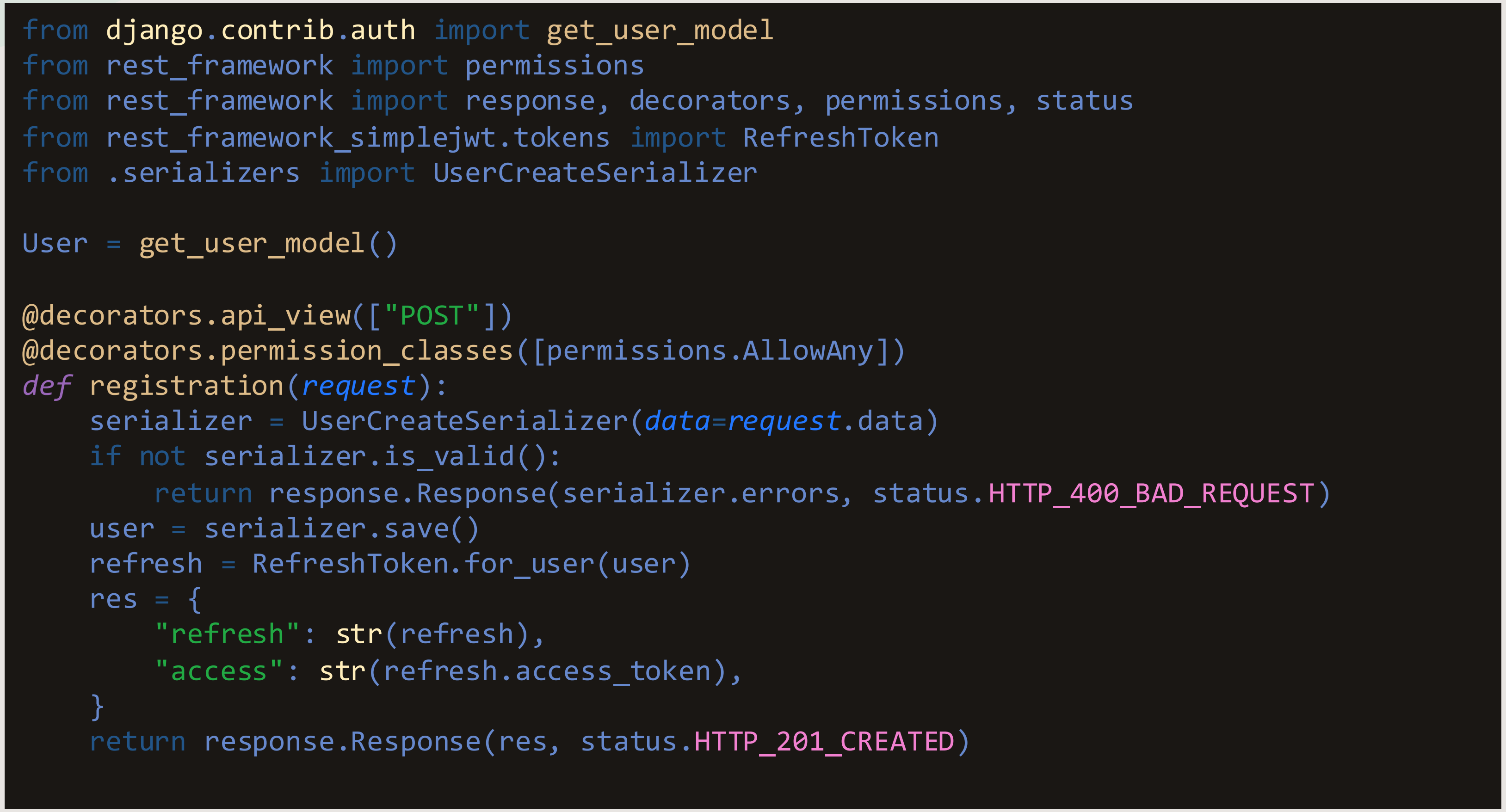
```python
from django.contrib.auth import get_user_model
from rest_framework import serializers

User = get_user_model()

class UserCreateSerializer(serializers.ModelSerializer):
    password = serializers.CharField(write_only=True, required=True, style={
                                     "input_type":   "password"})
    password2 = serializers.CharField(
        style={"input_type": "password"}, write_only=True, label="Confirm password")

    class Meta:
        model = User
        fields = [
            "username",
            "email",
            "password",
            "password2",
        ]
        extra_kwargs = {"password": {"write_only": True}}

    def create(self, validated_data):
        username = validated_data["username"]
        email = validated_data["email"]
        password = validated_data["password"]
        password2 = validated_data["password2"]
        if (email and User.objects.filter(email=email).exclude(username=username).exists()):
            raise serializers.ValidationError(
                {"email": "Email addresses must be unique."})
        if password != password2:
            raise serializers.ValidationError(
                {"password": "The two passwords differ."})
        user = User(username=username, email=email)
        user.set_password(password)
        user.save()
        return user
```

We'll be using Django's built in User model, which we get by calling the get_user_model() function. It's good practice to do it this way instead of importing the User directly, since it will ensure that we get the currently active User model even if we have customized it.

We also override the create() method and check that the confirmation password is identical to the password, and that no other user has the same email address.

Adding JSON Web Token Authentication

Next we'll add a view in jwtauth/views.py:



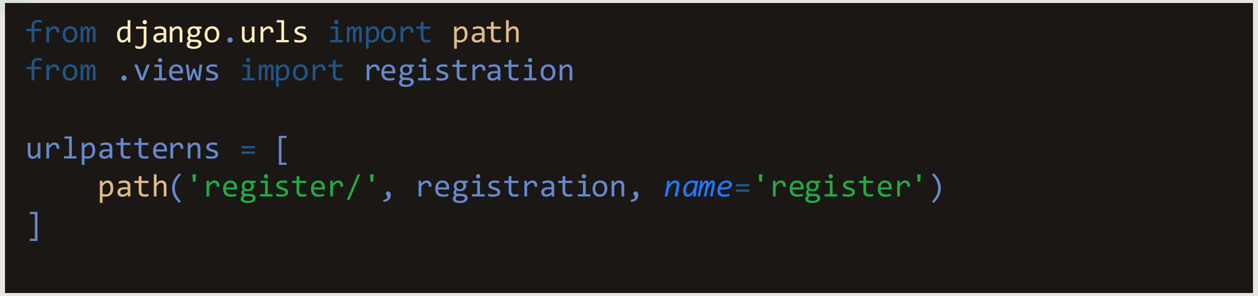
```python
from django.contrib.auth import get_user_model
from rest_framework import permissions
from rest_framework import response, decorators, permissions, status
from rest_framework_simplejwt.tokens import RefreshToken
from .serializers import UserCreateSerializer

User = get_user_model()

@decorators.api_view(["POST"])
@decorators.permission_classes([permissions.AllowAny])
def registration(request):
    serializer = UserCreateSerializer(data=request.data)
    if not serializer.is_valid():
        return response.Response(serializer.errors, status.HTTP_400_BAD_REQUEST)
    user = serializer.save()
    refresh = RefreshToken.for_user(user)
    res = {
        "refresh": str(refresh),
        "access": str(refresh.access_token),
    }
    return response.Response(res, status.HTTP_201_CREATED)
```

This is the first time we use a function based view and not a class based view. We choose a function based view here since it only responds to the POST http verb, and we use decorators to ensure this, as well as make and exception from the permissions defined in the settings.py file to allow anyone access to just this endpoint.

We do a check to see if the serializer has validated the data we got, and if not return it's error object. If everything is fine we save the serializer, which returns the newly created user object. We can then obtain a JWT token for this user and return it.

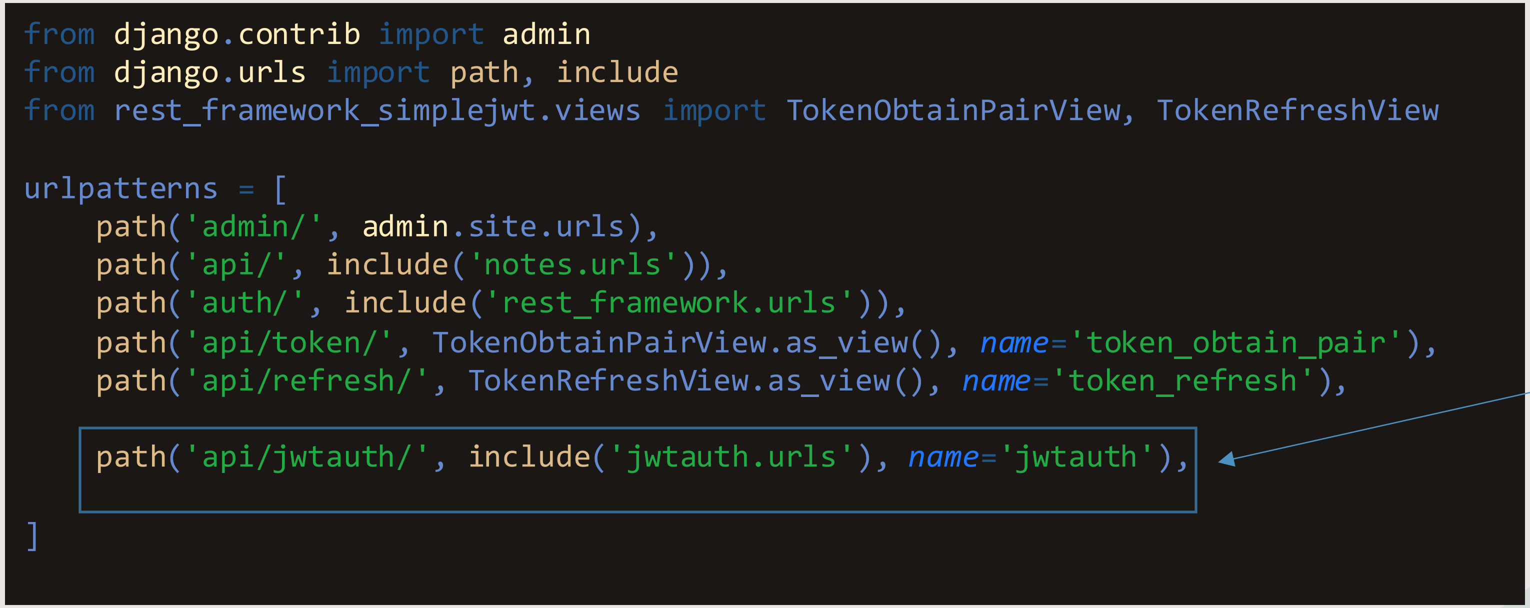
Adding JSON Web Token Authentication

We need to create a urls file add the view to it, jwtauth/urls.py:



```
from django.urls import path
from .views import registration

urlpatterns = [
    path('register/', registration, name='register')
]
```

And lastly include the jwtauth urls in project/urls.py:

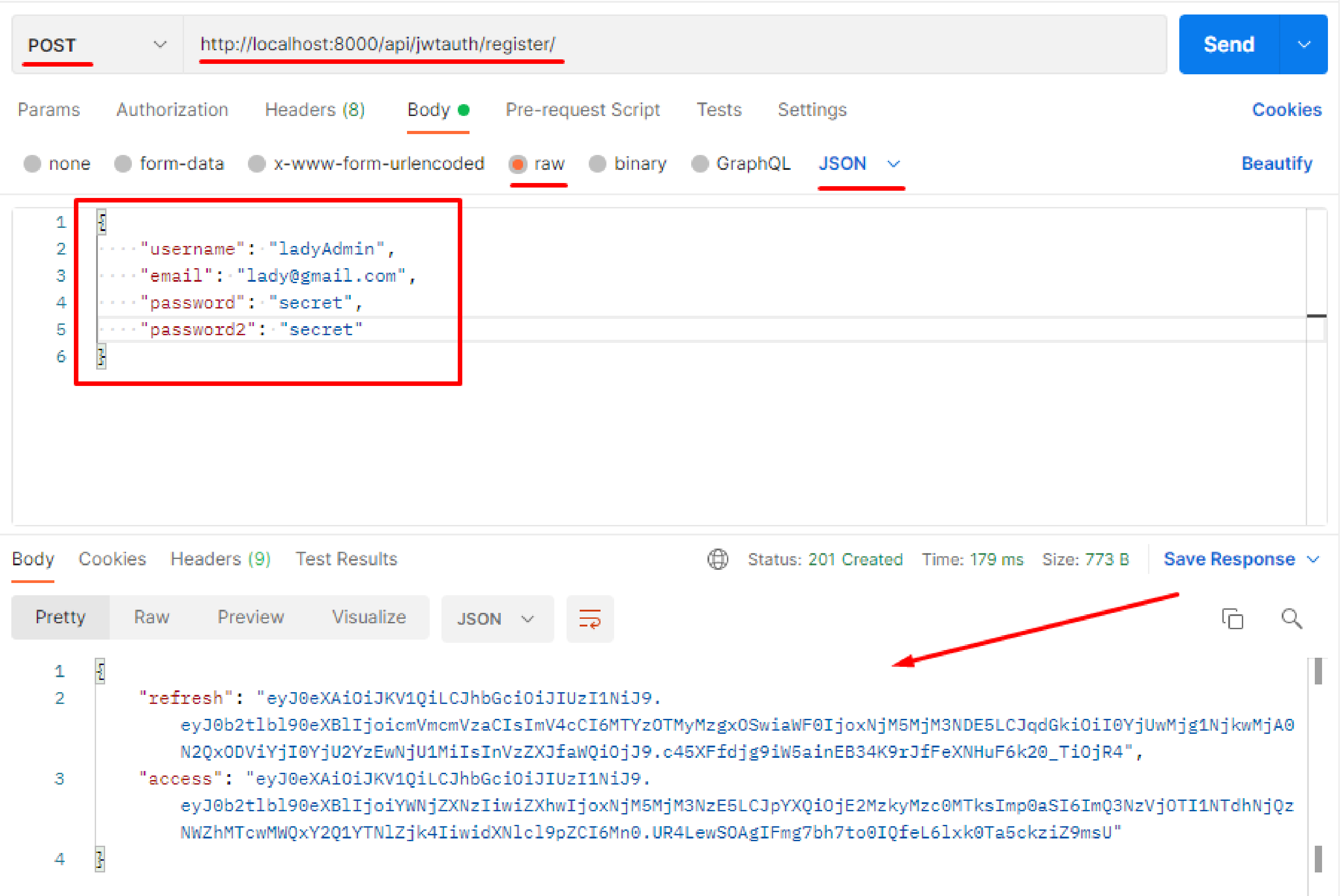


```
from django.contrib import admin
from django.urls import path, include
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('notes.urls')),
    path('auth/', include('rest_framework.urls')),
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/refresh/', TokenRefreshView.as_view(), name='token_refresh'),

    path('api/jwtauth/', include('jwtauth.urls'), name='jwtauth'),

]
```

Add code

Adding JSON Web Token Authentication

And that's it. We should now have a new endpoint at http://localhost:8000/api/jwtauth/register/
We can test it in Postman:



```
{
    "username": "ladyAdmin",
    "email": "lady@gmail.com",
    "password": "secret",
    "password2": "secret"
}
```

```
{
    "refresh": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
        eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4cCI6MTYzOTMyMzgxOSwiaWF0IjoxNjM5MjM3NDE5LCJqdGkiOiI0YjUwMjg1NjkwMjA0
        N2QxODViYjI0YjU2YzEwNjU1MiIsInVzZXJfaWQiOjJ9.c45XFfdjg9iW5ainEB34K9rJfFeXNHuF6k20_TiOjR4",
    "access": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
        eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjM5MjM3NzE5LCJpYXQiOjE2MzkyMzc0MTksImp0aSI6ImQ3NzVjOTI1NTdhNjQz
        NWZhMTcwMWQxY2Q1YTNlZjk4IiwidXNlcl9pZCI6Mn0.UR4LewSOAgIFmg7bh7to0IQfeL6lxk0Ta5ckziZ9msU"
}
```

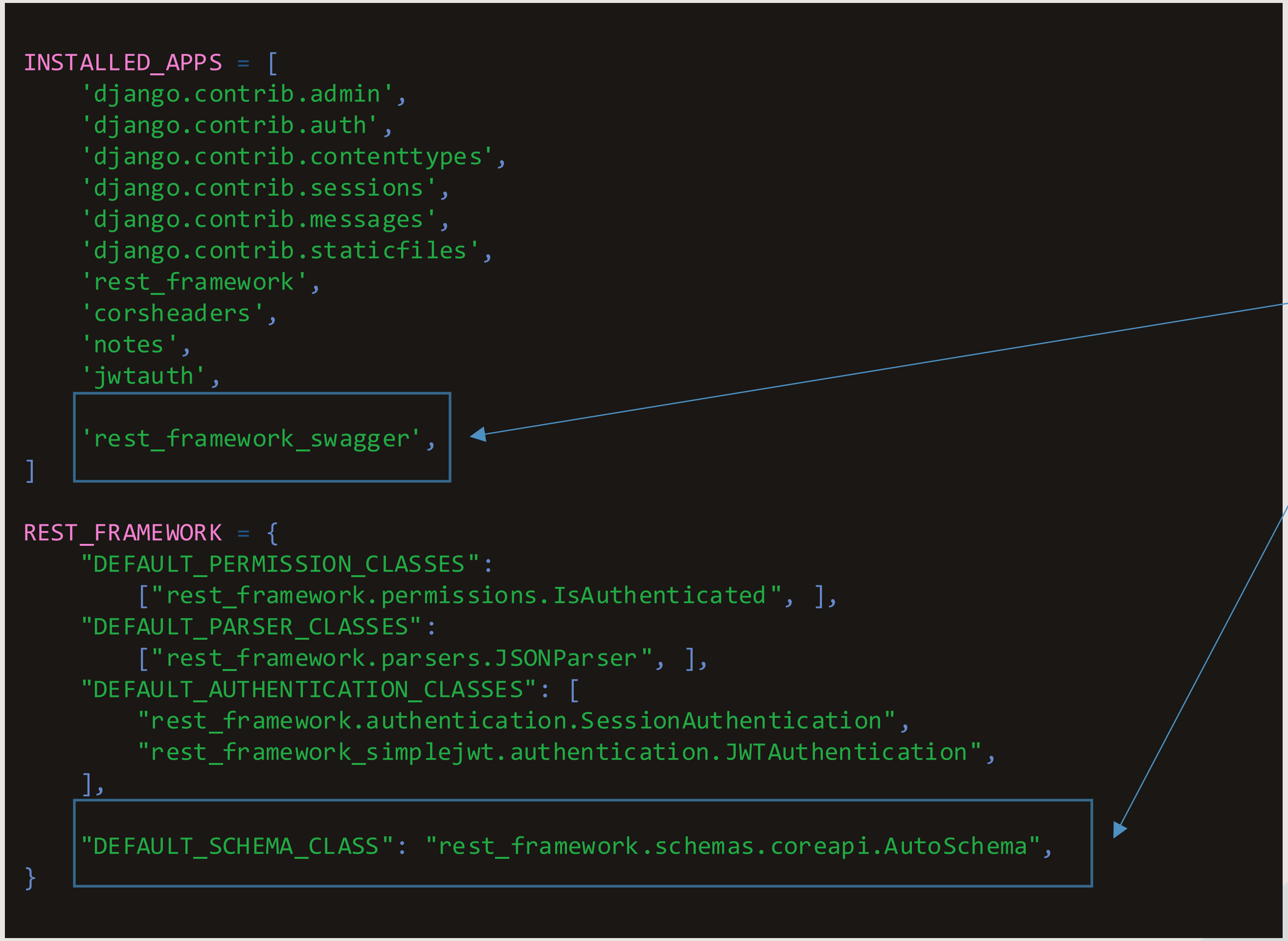
If the validations pass it should return an object with refresh and access tokens.

Adding Swagger Docs

The last thing we'll do is add Swagger docs so consumers of the API can see what endpoints are available.

```
$ pip install django-rest-swagger
```

Add it to your INSTALLED_APPS list in project/settings.py. Also include the new DEFAULT_SCHEMA_CLASS in REST_FRAMEWORK settings:



```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'corsheaders',
    'notes',
    'jwtauth',

    'rest_framework_swagger',
]

REST_FRAMEWORK = {
    "DEFAULT_PERMISSION_CLASSES":
        ["rest_framework.permissions.IsAuthenticated", ],
    "DEFAULT_PARSER_CLASSES":
        ["rest_framework.parsers.JSONParser", ],
    "DEFAULT_AUTHENTICATION_CLASSES": [
        "rest_framework.authentication.SessionAuthentication",
        "rest_framework_simplejwt.authentication.JWTAuthentication",
    ],

    "DEFAULT_SCHEMA_CLASS": "rest_framework.schemas.coreapi.AutoSchema",
}
```

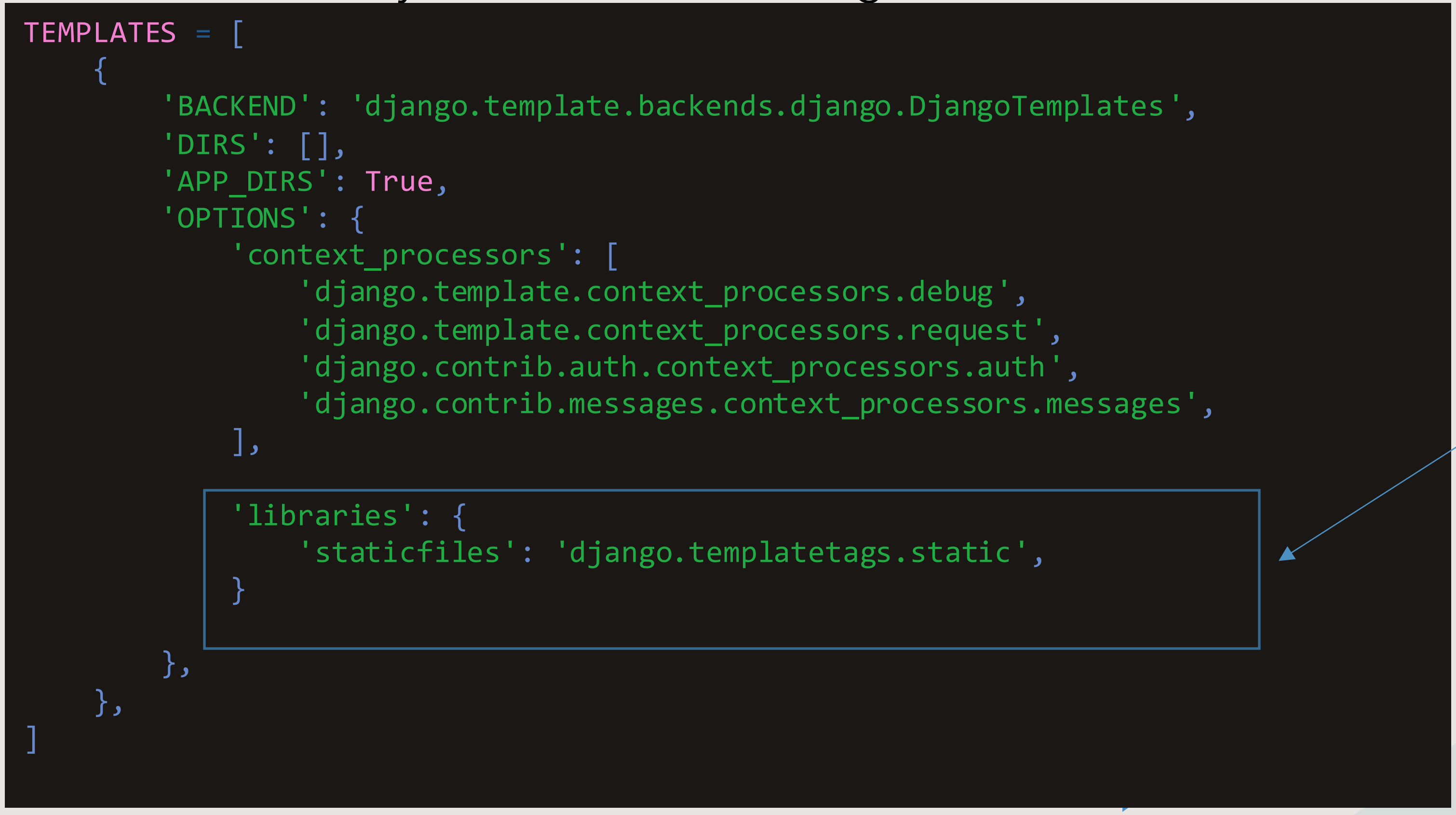
Add code

Adding Swagger Docs

Register staticfiles to tag library

staticfiles has been change to **static**

You can register with the fallowing code in your setting.py
Add this code in your TEMPALTE settings

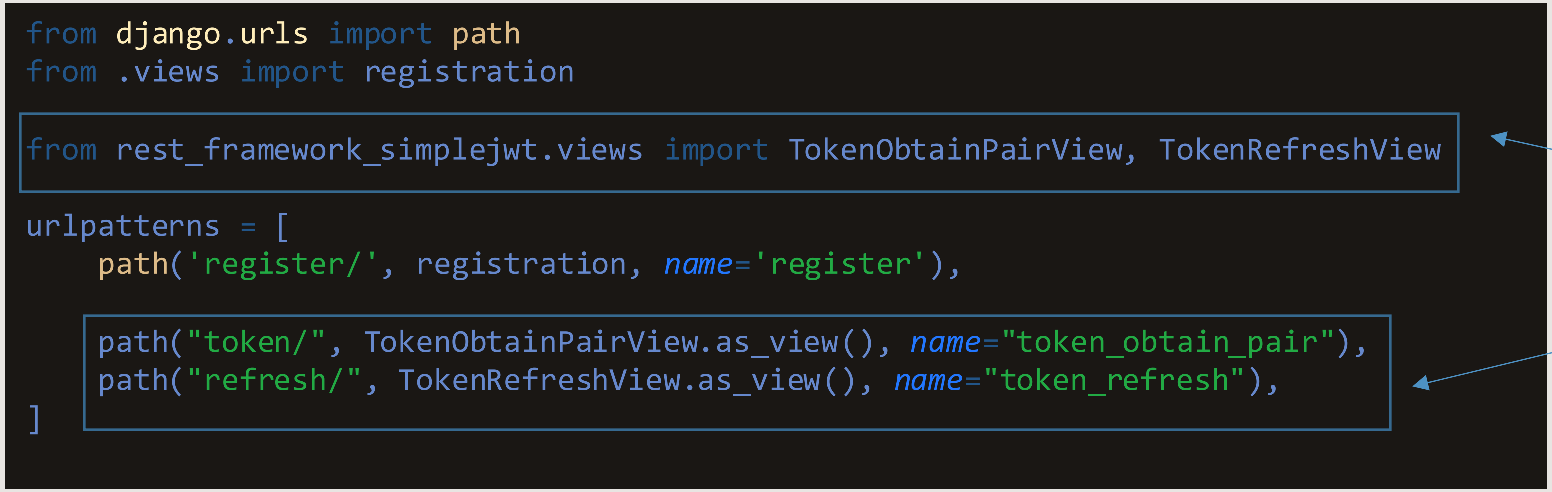


```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',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

            'libraries': {
                'staticfiles': 'django.templatetags.static',
            }

        },
    },
]
```

Add
code

Adding Swagger Docs

Then put the token and refresh enpoints in jwtauth/urls.py:



```
from django.urls import path
from .views import registration

from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView

urlpatterns = [
    path('register/', registration, name='register'),

    path("token/", TokenObtainPairView.as_view(), name="token_obtain_pair"),
    path("refresh/", TokenRefreshView.as_view(), name="token_refresh"),
]
```

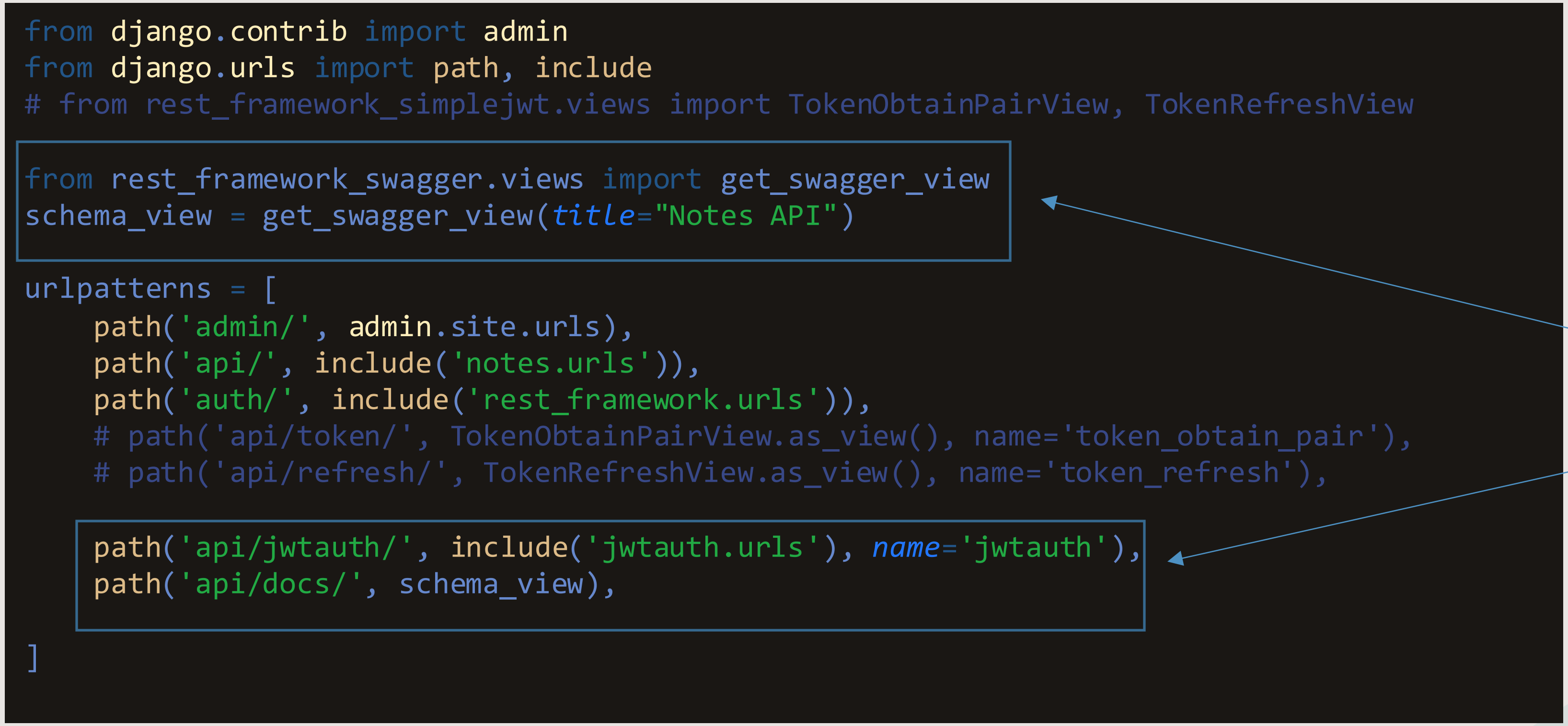
Add code

Go to http://localhost:8000/api/docs/ to see the full list of APIs. You should also see that the token endpoints now belong to the /jwtauth grouping.

And that's it! You now have a fully functionall notes API that can be consumed by a client from anywhere.

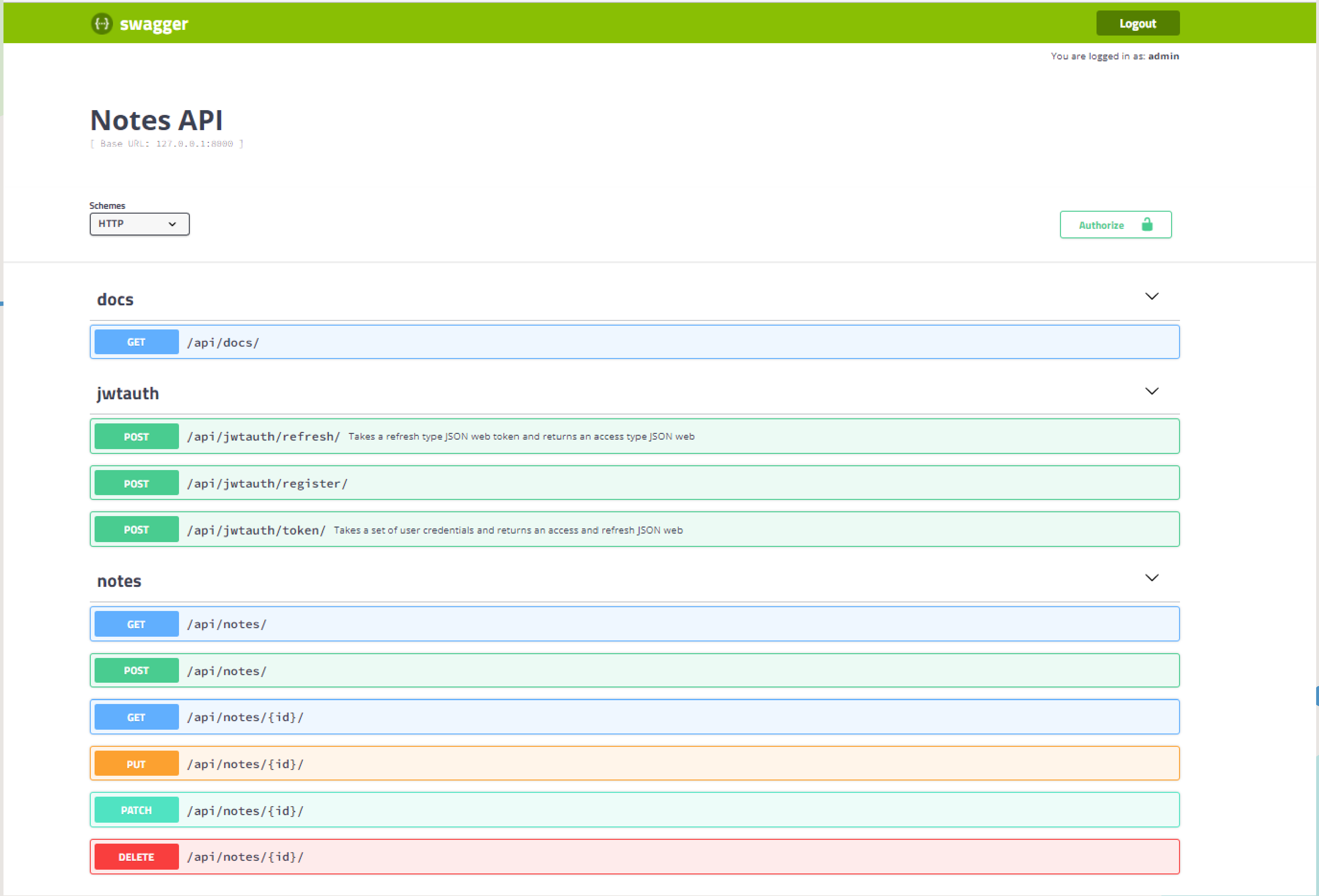
Adding Swagger Docs

And include it in your project/urls.py file. To finish up we'll also do a little bit of refactoring. Refresh and token are separate endpoints now. Since they have to do with authentication it would be better if they were in the jwtauth app. Let's move them there, as well as add a url for the docs.



```
from django.contrib import admin
from django.urls import path, include
# from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView

from rest_framework_swagger.views import get_swagger_view
schema_view = get_swagger_view(title="Notes API")

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('notes.urls')),
    path('auth/', include('rest_framework.urls')),
    # path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    # path('api/refresh/', TokenRefreshView.as_view(), name='token_refresh'),

    path('api/jwtauth/', include('jwtauth.urls'), name='jwtauth'),
    path('api/docs/', schema_view),

]
```

Add code



You are logged in as: **admin**

Notes API

[Base URL: 127.0.0.1:8000]

docs

GET /api/docs/

jwtauth

POST /api/jwtauth/refresh/ Takes a refresh type JSON web token and returns an access type JSON web

POST /api/jwtauth/register/

POST /api/jwtauth/token/ Takes a set of user credentials and returns an access and refresh JSON web

notes

GET /api/notes/

POST /api/notes/

GET /api/notes/{id}/

PUT /api/notes/{id}/

PATCH /api/notes/{id}/

DELETE /api/notes/{id}/