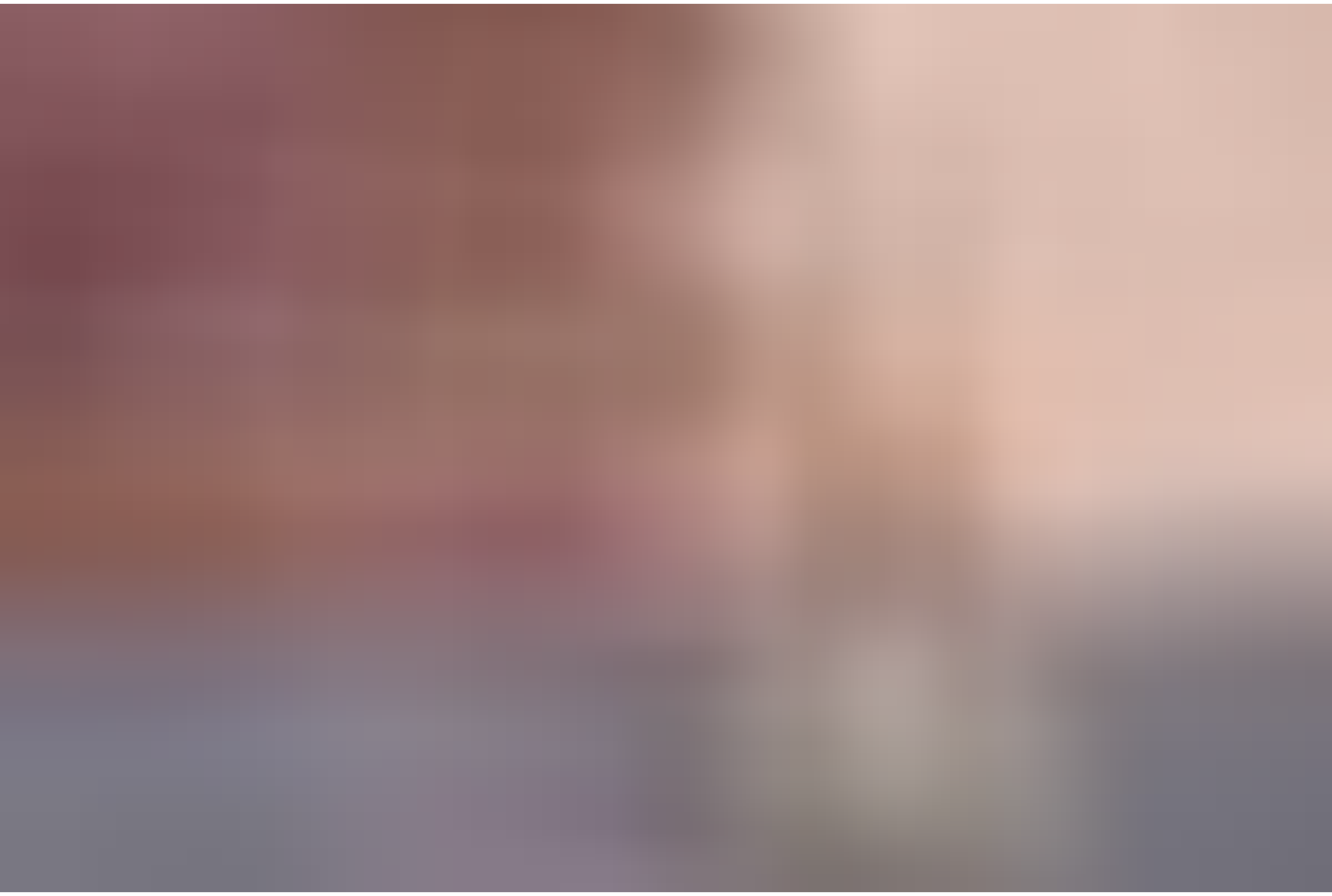
Aug 31, 2017 · 4 min read

Full stack Django: Quick start with JWT auth and React/Redux (Part I)



This article shows how to quick-start with SPA applications development using Django and React/Redux.

It’s hard to imagine a Django application without user authentication and authorization. So we start with a simple application that can authenticate a user and perform an authorized request to an API.

In addition, you will see how to setup redux-router, local state persistence with redux-persist and write a simple UI with reactstrap (bootstrap 4 binding for the React)

In the first part of the tutorial we will create a simple Django backend with JWT token authentication. Second part shows how to setup React/Redux frontend application and the third part shows how to implement JWT refresh token workflow.

At the end, you will have tiny, ready to extend application with Django back-end and React/Redux front-end.

To work over first part of the tutorial, you need to have Python 3.6 installed (under Ubuntu ensure that you have `python3.6-dev` and `python3.6-venv` packages)

What is JWT

JSON Web token is the stateless authorization method. Server-generated tokens may be kept on the client only. A JWT token itself could contain an additional payload inside. It could be a username, email or user permissions signed by a server side key. That became very handy when you going to split your Django monolith into different servers without shared users database or even perform authenticated requests to backends implemented in other technologies.

You can read more about JWT internals in the 5 Easy Steps to Understand JSON Web Tokens (JWT) article

JWT authorization endpoint can offer two token types—a short-living Access Token and a long-living Refresh Token. Short-living Access Token could be used to perform API calls over different services, whereas the long-living Refresh Token suited to obtain new Access Token when previous going to be expired.

Long-living tokens allows you to ask the user for his username and password only one time once he authenticates for the first time, and the user will stay logged for a while

Constant refresh of short living access tokens allows keeping in sync token payload (remember the user name or permissions)

Another reason to have two kinds of tokens is the ability to block a user from receiving new Access tokens.

The server side

The servers side project is pretty straightforward. Start with new virtual environment and install required Django packages

```
$ mkdir backend/ && cd backend/


$ python3.6 -m venv env
$ source env/bin/activate
$ pip install coreapi django djangorestframework \
      djangorestframework-simplejwt
$ pip freeze > requirements.txt

$ django-admin startproject config .
```

We installed coreapi package, to allow to auto generate an API schema describes what resources are available, what their URLs are, how they are represented and what operations they support.

And the django-rest-framework-simplejwt package that implements JWT authorization and authentication with Refresh and Access tokens.

Let’s edit `config/settings.py` to enable it

```
INSTALLED_APPS = [
    ...
    'rest_framework',
]

# Rest Framework

REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
    ),
    'DEFAULT_AUTHENTICATION_CLASSES': (

'rest_framework_simplejwt.authentication.JWTAuthentication'
,

'rest_framework.authentication.SessionAuthentication',
    ),
}
```

We added `rest_framework` into `INSTALLED_APPS` settings, protect by default all API resource with `IsAuthenticate` guard, and enable `JWTAuthentication` alongside with `SessionAuthentication`. We going to use standard django session authentication to get access to the protected schema view.

Let's edit `config/urls.py` to enable authentication endpoints

```
from django.conf.urls import url, include
from django.views import generic
from rest_framework.schemas import get_schema_view
from rest_framework_simplejwt.views import (
    TokenObtainPairView,
    TokenRefreshView,
)

urlpatterns = [
    url(r'^$', generic.RedirectView.as_view(
         url='/api/', permanent=False)),

    url(r'^api/$', get_schema_view()),
    url(r'^api/auth/', include(
        'rest_framework.urls',
namespace='rest_framework')),
    url(r'^api/auth/token/obtain/$',
TokenObtainPairView.as_view()),
    url(r'^api/auth/token/refresh/$',
TokenRefreshView.as_view()),
]
```

We've got enabled a schema view, session authentication URLs from the django-rest-framework, and `TokenObtainPairView` with `TokenRefreshView` for JWT authentication.

At the last step, let's make a simple `echo` API endpoint, to test calls from our front-end. when we would be authorized. For the demo project, we could add this directly into `config/urls.py`

```
from rest_framework import views, serializers, status
from rest_framework.response import Response
```

```
class MessageSerializer(serializers.Serializer):
    message = serializers.CharField()


class EchoView(views.APIView):
    def post(self, request, *args, **kwargs):
        serializer = MessageSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        return Response(
            serializer.data,
status=status.HTTP_201_CREATED)


urlpatterns = [
    ...
    url(r'^api/echo/$', EchoView.as_view())
]
```

Now we can run the server

```
$ ./manage.py migrate
$ ./manage.py createsuperuser
$ ./manage.py runserver
```

Open `http://127.0.0.1:8000` login and see auto-generated schema of our API.

Go to `http://127.0.0.1:8000/api/auth/token/obtain/` enter credentials, and see that we got new Access and Refresh tokens in response

And on the `http://127.0.0.1:8000/api/echo` we could check our echo endpoint.

So, we a ready to create a front-end. Let's go to the Part II