**DJANGO - RESTFRAWEWORK + REACT 16.13.1** <https://blog.logrocket.com/creating-an-app-with-react-and-django/>

Here’s the list of things you need to have set up in your machine before you can follow this article:

* [Python 3](https://www.python.org/download/releases/3.0/) (if you’re using Linux, chances are that it’s already installed. Run *python3 -V* command to check)
* [Pip](https://pypi.org/project/pip/) (the default Python package installer)
* [NodeJS](https://nodejs.org/en/)(in a version 6 or plus) and [npm](https://www.npmjs.com/) (5.2+)

Instalaremos react con create-app y lo conectaremos con la api de django mediante axios.

|  |  |
| --- | --- |
| **Instalar env python y activarlo** |  |
| python3 -m venv venv . ./venv/bin<Scripts>/activate | Create an enviroment for python, si no usar python en lugar de python3 |

|  |  |
| --- | --- |
| **Instalar dependencia** |  |
| -pip install django djangorestframework djangorestframework-simplejwt django-cors-headers  **pip install docutils  pip install** -U drf-yasg  pip install jsonfield  pip install django-readonly-field | -Framework, DRF, Cors -Documentacion Django -Documentacion DRF  from jsonfield import JSONField from django.db import models  class Question(models.Model): question\_text = JSONField(max\_length=200) pub\_date = models.DateTimeField('date published') |
| pip install axios @material-ui/core @material-ui/icons material-table --save | Frontend react, materialui |

|  |  |
| --- | --- |
| **Crear proyecto y modulos (apps)** |  |
| django-admin startproject <name> | Create our project |
| django-admin startapp <name>  (python manage.py startapp students) | Create an app. A project can have one or more apps, because Django is modular, we can export an app to another project.    Note: Project and app folder must be in the same level of folder. If don’t then change it. |

|  |  |
| --- | --- |
| **Activar apps en nuestro proyecto** |  |
| INSTALLED\_APPS = [  ...  'rest\_framework',  'corsheaders', #Conectar Diferentes servicios  '<myAppName>.apps.<myAppName>Config',  'rest\_framework.authtoken',   'django.contrib.admindocs', #Documentacion  'drf\_yasg' #Documentacion Rest Framework  ] | Edit our Settings project |
| MIDDLEWARE = [  ....  'corsheaders.middleware.CorsMiddleware',  'django.middleware.common.CommonMiddleware', ] | add the following into the MIDDLEWARE array:  They correspond to a filter that’ll intercept all of our application’s requests and apply CORS logic to them. |
| CORS\_ORIGIN\_ALLOW\_ALL = True | However, since we’re working full localhost, we’ll disable the CORS feature by adding the following to the same file: |

|  |  |
| --- | --- |
| **Crear modelo** |  |
| <myAppName>/models.py  ###PUT YOUR CODE MODEL### | Create our model |

|  |  |
| --- | --- |
| **Crear migracion** |  |
| python manage.py makemigrations | Make a migration.    f you want to reverse all migrations applied for an app, use the name zero:    $ python manage.py migrate <nameModel> zero |
| python manage.py migrate | Migrate it |
| Optional: python manage.py makemigrations --empty --name <nameModel> <nameModel> | Create a data migration file, this new file it’s a reference to the initial migration, for production reference. |
| Optional: django\_react\_proj/<nameApp>/migrations/ new file created  # Generated by Django 3.0.7 on 2020-06-03 23:00  from django.db import migrations  def create\_data(apps, schema\_editor):      Student = apps.get\_model('students', 'Student')      Student(name="Joe Silver", email="joe@email.com", document="22342342", phone="00000000").save()  class Migration(migrations.Migration):      dependencies = [          ('students', '0001\_initial'),      ]      operations = [          migrations.RunPython(create\_data),      ] | Modify to create a new object in our database. |
| Optional: python manage.py migrate | Migrate it with the new object. |

|  |  |
| --- | --- |
| **Test** |  |
| Optional: pip install coverage | TEST |
| Optional: coverage run --source='.' manage.py test | RUN IN EVERY CHANGE IN YOUR CODE |
| Optional: coverage html | Generate the report |
| Optional: coverage report | See the report in command line |

|  |  |  |
| --- | --- | --- |
| **Create a serializer for DRF** |  |  |
| <nameApp>/serializers.py | Transforming an object into another data format to save it to a file or send it through the network. Models are our base to create the serializers. Serializers are necessary when use API rest. | <https://www.django-rest-framework.org/api-guide/relations/> |

|  |  |
| --- | --- |
| **Mapear Urls** |  |
| <Proyeco>/urls.py  urlpatterns = [  path('admin/', admin.site.urls), path('api/v1/', include('<appName>.urls')),  ]    <appName>/urls.py  urlpatterns = [    path('nombreUrl', views.<VistaConClaseGenerica>.as\_view() ), #Use Capital letters in Adjunto because browser converter to Capital in automatic    re\_path(r'^nombreUrl/$', views.UsuarioList.as\_view() ), #permite pasar como parametro el id    re\_path(r'^nombreUrl/(?P<pk>[0-9]+)$', views.UsuarioDetails.as\_view() ),    ] | Url mapping the app.    Note that each of them is connected to a view function (to be created), so this is the place where we route our requests.  The first endpoint will handle both creations (POST) and listing (GET).  The second one will remove (DELETE) or update (PUT) the data of a single student. Simple, right? |

|  |  |  |
| --- | --- | --- |
| Vistas |  |  |
| students/views.py    # Funciones    from rest\_framework.response import Response from rest\_framework.decorators import api\_view from rest\_framework import status  from .models import Student from .serializers import \*  @api\_view(['GET', 'POST']) def students\_list(request):  if request.method == 'GET':  data = Student.objects.all()  serializer = StudentSerializer(data, context={'request': request}, many=True)  return Response(serializer.data)  elif request.method == 'POST':  serializer = StudentSerializer(data=request.data)  if serializer.is\_valid():  serializer.save()  return Response(status=status.HTTP\_201\_CREATED)    return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)  @api\_view(['PUT', 'DELETE']) def students\_detail(request, pk):  try:  student = Student.objects.get(pk=pk)  except Student.DoesNotExist:  return Response(status=status.HTTP\_404\_NOT\_FOUND)  if request.method == 'PUT':  serializer = StudentSerializer(student, data=request.data,context={'request': request})  if serializer.is\_valid():  serializer.save()  return Response(status=status.HTTP\_204\_NO\_CONTENT)  return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)  elif request.method == 'DELETE':  student.delete()  return Response(status=status.HTTP\_204\_NO\_CONTENT) | **There are function views, class based views, and generic <API> view**    The first method, students\_list, is handling both GET and POST operations over the root endpoint of our API.  This means every time we make a request over <http://localhost:8000/api/students> with GET and POST HTTP verbs, we’ll execute this method.    Then, we pass the result to our serializer, which will take care of the converting process before we return it as a response.  For the POST method, note that we’re first calling the is\_valid() method on the serializer to ensure that the data received is conformed with our model.  Otherwise the serializer will throw an exception here. If all is fine, we save it to the datastore.  The next PUT and DELETE operations are pretty much the same, changing only the HTTP verbs and the responses.        #CLASES GENERICAS CON MIXINS (MISMA FUNCIONALIDAD QUE EL CODIGO DE A LADO)  class UsuarioMixin(object):      queryset = Usuario.objects.all()      serializer\_class = UsuarioSerializer  class UsuarioList(UsuarioMixin, ListCreateAPIView):      pass  class UsuarioDetails(UsuarioMixin, RetrieveUpdateDestroyAPIView):      pass | <https://docs.djangoproject.com/en/1.10/ref/class-based-views/> |

|  |  |
| --- | --- |
| **Ejecutar** |  |
| python manage.py runserver | Run |
| REST\_FRAMEWORK = {  'DEFAULT\_RENDERER\_CLASSES': (  'rest\_framework.renderers.JSONRenderer',  ) } | Edit our settings project to disable the browseable API in production |

|  |  |
| --- | --- |
| **Crear usuario con la auth de Django** |  |
| python3 manage.py createsuperuser    Nota: Si se hace desde gitBash usar:  winpty python manage.py createsuperuser  *From <*[*https://stackoverflow.com/questions/32532900/not-able-to-create-super-user-with-django-manage-py*](https://stackoverflow.com/questions/32532900/not-able-to-create-super-user-with-django-manage-py)*>*    Datos: rootescom@gmail.com, alumno123 | Una vez el comando termine un nuevo administrador será agregado a la base de datos.   Hace referencia a la tabla: auth\_user (id, password, last\_login, is\_superuser, username, first\_name, email, is\_staff, is\_active, date\_joined,last\_name) |

|  |  |
| --- | --- |
| Configurar Cors |  |
| Instaled apps:  'corsheaders', #enable corsheades  ] | Settings.py |
| Middleware:  'corsheaders.middleware.CorsMiddleware',  ] | " |
| CORS\_ORIGIN\_WHITELIST = (      '<http://localhost:3000>',  )  CORS\_ALLOW\_CREDENTIALS = True | " |

|  |  |
| --- | --- |
| **JSON Web Token (JWT)** |  |
| Pip install djangorestframework-**simplejwt** | Install |
| REST\_FRAMEWORK = {      'DEFAULT\_PERMISSION\_CLASSES': (          'rest\_framework.permissions.IsAuthenticated',      ),      'DEFAULT\_AUTHENTICATION\_CLASSES': (          'rest\_framework\_jwt.authentication.JSONWebTokenAuthentication',          'rest\_framework.authentication.SessionAuthentication',          'rest\_framework.authentication.BasicAuthentication',      ),  }    JWT\_AUTH = {      'JWT\_RESPONSE\_PAYLOAD\_HANDLER': 'PAW\_SP\_ACE.utils.my\_jwt\_response\_handler'  } | Settings.py |
| from rest\_framework\_simplejwt import views as jwt\_views  urlpatterns = [  path('token/obtain/', jwt\_views.TokenObtainPairView.as\_view(), name='token\_create'),  # override sjwt stock token      path('token/refresh/', jwt\_views.TokenRefreshView.as\_view(), name='token\_refresh'),    ] | Urls.py |

|  |  |
| --- | --- |
| **Crear front-end con React** |  |
| npx create-react-app <appName> | Create the react app. Dentro del projecto, para que se cree como otro modulo de Django |

|  |  |
| --- | --- |
| **Instalar dependencias de React** |  |
| Carpeta app react: npm install axios --save | Install axios, react no incluye ninguna libreria para hacer peticiones HTTP, por eso se puede usar axios en lugar de fetch, ya que es mas optimizado. Para peticiones asincronas.    --save hace que se registre en nuestro package las dependencias. |
| npm install @material-ui/core --save | Se intentara instalar material ui en su lugar. |
| npm install @material-ui/icons --save |  |
| npm install material-table --save |  |
| Npm install react-router-dom |  |

|  |  |
| --- | --- |
| **Estructura React** |  |
| Index.js | Importaremos y pondremos los componentes que usaremos |
| Services | Crearemos una carpeta para registrar nuestros servicios, mediante axios haremos las peticiones CRUD a nuestra API |
| Componentes | Crearemos una carpeta para registrar nuestros componentes. Para ello se creara un archivo y se instanciara el servicio respectivo de dicho componente. |
| Route | Permite relacionar las rutas de nuestra aplicacion |

|  |  |
| --- | --- |
| **Agregando documentación de la *app* al sitio de administración**  Veamos cómo agregar documentación de tu *app.*  *From <*[*https://blog.nearsoftjobs.com/los-secretos-de-la-documentaci%C3%B3n-autom%C3%A1tica-de-django-e9feaecf70c2*](https://blog.nearsoftjobs.com/los-secretos-de-la-documentaci%C3%B3n-autom%C3%A1tica-de-django-e9feaecf70c2)*>* |  |
| **Pip install docutils** | docutils , para documentar django, modelos… |
| INSTALLED\_APPS agrega django.contrib.admindocs . | Settings.py |
| path('admin/doc/', include('django.contrib.admindocs.urls')) | Urls.py |
| pip install -U drf-yasg    *From <*[*https://drf-yasg.readthedocs.io/en/stable/readme.html#installation*](https://drf-yasg.readthedocs.io/en/stable/readme.html#installation)*>*  pip install -U drf-yasg[validation]    *From <*[*https://drf-yasg.readthedocs.io/en/stable/readme.html#installation*](https://drf-yasg.readthedocs.io/en/stable/readme.html#installation)*>* | [**drf-yasg**](https://www.django-rest-framework.org/topics/documenting-your-api/#drf-yasg-yet-another-swagger-generator)  **para documentar Django rest framework**  <https://drf-yasg.readthedocs.io/en/stable/> |
| INSTALLED\_APPS = [  ...  'drf\_yasg',  ... ] | Settings.py |
| ... from rest\_framework import permissions from drf\_yasg.views import get\_schema\_view from drf\_yasg import openapi  ...  schema\_view = get\_schema\_view(  openapi.Info(  title="Snippets API",  default\_version='v1',  description="Test description",  terms\_of\_service="https://www.google.com/policies/terms/",  contact=openapi.Contact(email="contact@snippets.local"),  license=openapi.License(name="BSD License"),  ),  public=True,  permission\_classes=(permissions.AllowAny,), )  urlpatterns = [  url(r'^swagger(?P<format>\.json|\.yaml)$', schema\_view.without\_ui(cache\_timeout=0), name='schema-json'),  url(r'^swagger/$', schema\_view.with\_ui('swagger', cache\_timeout=0), name='schema-swagger-ui'),  url(r'^redoc/$', schema\_view.with\_ui('redoc', cache\_timeout=0), name='schema-redoc'),  ... ]    *From <*[*https://drf-yasg.readthedocs.io/en/stable/readme.html#installation*](https://drf-yasg.readthedocs.io/en/stable/readme.html#installation)*>* | Url.py  This exposes 4 endpoints:   * + A JSON view of your API specification at /swagger.json   + A YAML view of your API specification at /swagger.yaml   + A swagger-ui view of your API specification at /swagger/   + A ReDoc view of your API specification at /redoc/     *From <*[*https://drf-yasg.readthedocs.io/en/stable/readme.html#installation*](https://drf-yasg.readthedocs.io/en/stable/readme.html#installation)*>*     * + full support for nested Serializers and Schemas   + response schemas and descriptions   + model definitions compatible with codegen tools   + customization hooks at all points in the spec generation process   + JSON and YAML format for spec   + bundles latest version of [swagger-ui](https://github.com/swagger-api/swagger-ui) and [redoc](https://github.com/Rebilly/ReDoc) for viewing the generated documentation   + schema view is cacheable out of the box   + generated Swagger schema can be automatically validated by [swagger-spec-validator](https://github.com/Yelp/swagger_spec_validator)   + supports Django REST Framework API versioning with URLPathVersioning and NamespaceVersioning; other DRF or custom versioning schemes are not currently supported |
| Features   * + full support for nested Serializers and Schemas   + response schemas and descriptions   + model definitions compatible with codegen tools   + customization hooks at all points in the spec generation process   + JSON and YAML format for spec   + bundles latest version of [swagger-ui](https://github.com/swagger-api/swagger-ui) and [redoc](https://github.com/Rebilly/ReDoc) for viewing the generated documentation   + schema view is cacheable out of the box   + generated Swagger schema can be automatically validated by [swagger-spec-validator](https://github.com/Yelp/swagger_spec_validator)   + supports Django REST Framework API versioning with URLPathVersioning and NamespaceVersioning; other DRF or custom versioning schemes are not currently supported     *From <*[*https://github.com/axnsan12/drf-yasg*](https://github.com/axnsan12/drf-yasg)*>* |  |
| Pip freeze | Ver que se ha instalado y las versiones. |