Django Rest API- From Udemy Beginner Class

### By Mark Winterbottom

Table of Contents

[By Mark Winterbottom 1](#_Toc14941208)

[Technologies included In This Course: 3](#_Toc14941209)

[DRF: 3](#_Toc14941210)

[Vagrant: 3](#_Toc14941211)

[»Why Vagrant? 4](#_Toc14941212)

[Virtual Box: 4](#_Toc14941213)

[ModHeader 5](#_Toc14941214)

[Teacher’s Cheat Sheet for This Course 5](#_Toc14941215)

[Build a Backend REST API with Python & Django - Beginner 5](#_Toc14941216)

[Git 6](#_Toc14941217)

[SSH Key Management 6](#_Toc14941218)

[Virtual Environments 7](#_Toc14941219)

[Django Management Commands 7](#_Toc14941220)

[Vagrant 7](#_Toc14941221)

[Terminal / GitBash Commands 8](#_Toc14941222)

[Finding Your Home DIR on MAC OS 8](#_Toc14941223)

[Create Git Project 9](#_Toc14941224)

[Create Readme file 9](#_Toc14941225)

[Create .gitignore file 10](#_Toc14941226)

[Create a License File 13](#_Toc14941227)

[Git add and git commit 14](#_Toc14941228)

[Push Could to github 14](#_Toc14941229)

[Create a Public/Private key pair 14](#_Toc14941230)

[Store SSH Public Key (Never give away your private key) 16](#_Toc14941231)

[Create a new repo for this course on Github 16](#_Toc14941232)

[Do a git add origin to add the new repo to your local git repo through the CMD 16](#_Toc14941233)

[Then do a git push orgin master 17](#_Toc14941234)

[Create Local Dev Server 17](#_Toc14941235)

[Configure a Vagrantfile 17](#_Toc14941236)

[Configure Vagrant Box 18](#_Toc14941237)

[Running and connecting to our dev server 19](#_Toc14941238)

[Connect to dev box 19](#_Toc14941239)

[Exit vargrant dev server: 20](#_Toc14941240)

[Section 5: Creating our Django Application 20](#_Toc14941241)

[Create python virtual env on vagrant server 20](#_Toc14941242)

[Activate the virtual server 20](#_Toc14941243)

[Close virtual env 20](#_Toc14941244)

[Install required Python Packages for our project 21](#_Toc14941245)

[Verify what got installed: 22](#_Toc14941246)

[Create New Django Project 22](#_Toc14941247)

[Create a Django application 22](#_Toc14941248)

[Enable our app in Django settings.py file 23](#_Toc14941249)

[Start The server 23](#_Toc14941250)

[Models: 24](#_Toc14941251)

[Substituting a custom User model¶ 25](#_Toc14941252)

[Using a custom user model when starting a project¶ 26](#_Toc14941253)

[Create customer User Manager 27](#_Toc14941254)

[Set Custom User Model in settings.py 27](#_Toc14941255)

## Technologies included In This Course:

* **Django Rest Framework-** Django REST framework is a powerful and flexible toolkit for building Web APIs
* **Vagrant** - Vagrant is a tool for building and managing virtual machine environments in a single workflow
* **Virtual Box** - VirtualBox is a powerful x86 and AMD64/Intel64 [virtualization](https://www.virtualbox.org/wiki/Virtualization) product for enterprise as well as home use
* **Mod Header**- It is a Chrome Extension that modifies request and response headers
* Atom- IDE (need to use separate Command Window)
* Git – Source Code
* GitHub

## DRF:

Django REST framework is a powerful and flexible toolkit for building Web APIs.

Some reasons you might want to use REST framework:

* The [Web browsable API](https://restframework.herokuapp.com/) is a huge usability win for your developers.
* [Authentication policies](https://www.django-rest-framework.org/api-guide/authentication/) including packages for [OAuth1a](https://www.django-rest-framework.org/api-guide/authentication/#django-rest-framework-oauth) and [OAuth2](https://www.django-rest-framework.org/api-guide/authentication/#django-oauth-toolkit).
* [Serialization](https://www.django-rest-framework.org/api-guide/serializers/) that supports both [ORM](https://www.django-rest-framework.org/api-guide/serializers#modelserializer) and [non-ORM](https://www.django-rest-framework.org/api-guide/serializers#serializers) data sources.
* Customizable all the way down - just use [regular function-based views](https://www.django-rest-framework.org/api-guide/views#function-based-views) if you don't need the [more](https://www.django-rest-framework.org/api-guide/generic-views/) [powerful](https://www.django-rest-framework.org/api-guide/viewsets/) [features](https://www.django-rest-framework.org/api-guide/routers/).
* [Extensive documentation](https://www.django-rest-framework.org/), and [great community support](https://groups.google.com/forum/?fromgroups#!forum/django-rest-framework).
* Used and trusted by internationally recognised companies including [Mozilla](https://www.mozilla.org/en-US/about/), [Red Hat](https://www.redhat.com/), [Heroku](https://www.heroku.com/), and [Eventbrite](https://www.eventbrite.co.uk/about/).

## Vagrant:

Vagrant is a tool for building and managing virtual machine environments in a single workflow. With an easy-to-use workflow and focus on automation, Vagrant lowers development environment setup time, increases production parity, and makes the "works on my machine" excuse a relic of the past.

If you are already familiar with the basics of Vagrant, the [documentation](https://www.vagrantup.com/docs/index.html) provides a better reference build for all available features and internals.

## **[»](https://www.vagrantup.com/intro/index.html" \l "why-vagrant-)Why Vagrant?**

Vagrant provides easy to configure, reproducible, and portable work environments built on top of industry-standard technology and controlled by a single consistent workflow to help maximize the productivity and flexibility of you and your team.

To achieve its magic, Vagrant stands on the shoulders of giants. Machines are provisioned on top of VirtualBox, VMware, AWS, or [any other provider](https://www.vagrantup.com/docs/providers/). Then, industry-standard [provisioning tools](https://www.vagrantup.com/docs/provisioning/) such as shell scripts, Chef, or Puppet, can automatically install and configure software on the virtual machine.

## Virtual Box:

When we describe VirtualBox as a "virtualization" product, we refer to "full virtualization", that is, the particular kind of virtualization that allows an *unmodified* operating system with all of its installed software to run in a special environment, on top of your existing operating system. This environment, called a "virtual machine", is created by the virtualization software by intercepting access to certain hardware components and certain features. The physical computer is then usually called the "host", while the virtual machine is often called a "guest". Most of the guest code runs unmodified, directly on the host computer, and the guest operating system "thinks" it's running on real machine.

This approach, often called "native virtualization", is different from mere emulation. With that approach, as performed by programs such as [BOCHS](http://bochs.sourceforge.net/), guest code is not allowed to run directly on the host. Instead, every single machine instruction is translated ("emulated"). While emulators theoretically allow running code written for one type of hardware on completely different hardware (say, running 64-bit code on 32-bit hardware), they are typically quite slow. Virtualizers such as VirtualBox, on the other hand, can achieve near-native performance for the guest code, but can only run guest code that was written for the same target hardware (such as 32-bit Linux on a 32-bit Windows host).

VirtualBox is also different from so-called "paravirtualization" solutions such as [Xen](http://www.xen.org/), which require that the guest operating system be modified.

There are several scenarios that make virtualization attractive:

* **Operating system support.** With a virtualizer such as VirtualBox, one can run software written for one operating system on another (say, Windows software on Linux) without having to reboot.
* **Infrastructure consolidation.** Since the full performance of today's computers is rarely needed full-time, instead of running many such physical computers, one can "pack" many virtual machines onto a few powerful hosts and balance the loads between them. This can save a lot of hardware costs: e.g. by consolidating many servers into a few. VirtualBox is unique on the virtualization market in that it also allows for consolidating clients onto just a few RDP servers, with full client USB support, while "thin clients" only need to display RDP data.
* **Testing and disaster recovery.** Especially with the use of snapshots?, one can mess with a computing environment by running it as a virtual machine. If something goes wrong, one can easily switch back to a previous snapshot and avoid the need of frequent backups and restores.

## ModHeader

Modify request and response headers

Add and modify HTTP request headers and response headers.

\*\* Features \*\*

- Add/modify/remove request headers and response headers

- Enable header modification based on URL/resource type

- Add comments to header

- Multiple different profiles

- Sorting headers by name, value, or comments

- Append value to existing request or response header

- Export and import header

- Clone profile

- Cloud backup

- Tab locking!

# Teacher’s Cheat Sheet for This Course

# Build a Backend REST API with Python & Django - Beginner

This is the supplementary cheat sheet document for our course: [Build a Backend REST API with Python & Django - Beginner](https://www.udemy.com/django-python/?couponCode=2019UPDATECHEATSHEET)

* [Git](https://github.com/LondonAppDev/build-a-backend-api-python-drf-beginner-cheat-sheet/blob/master/README.md#git)
* [SSH Key Management](https://github.com/LondonAppDev/build-a-backend-api-python-drf-beginner-cheat-sheet/blob/master/README.md#ssh-key-management)
* [Virtual Environments](https://github.com/LondonAppDev/build-a-backend-api-python-drf-beginner-cheat-sheet/blob/master/README.md#virtual-environments)
* [Django Management Commands](https://github.com/LondonAppDev/build-a-backend-api-python-drf-beginner-cheat-sheet/blob/master/README.md#django-management-commands)
* [Vagrant](https://github.com/LondonAppDev/build-a-backend-api-python-drf-beginner-cheat-sheet/blob/master/README.md#vagrant)
* [Terminal / GitBash Commands](https://github.com/LondonAppDev/build-a-backend-api-python-drf-beginner-cheat-sheet/blob/master/README.md#terminal-gitbash-commands)

## Git

Use the below Git commands in the Windows Command Prompt or macOS Terminal.

**Configure default email and name**

Note: This only needs to be done the first time you use Git on your machine

git config --global user.email "your@email.com"

git config --global user.name "Your Name"

**Initialise a new Git repository**

git init

**Commit changes to Git**

git add .

git commit -am "Commit message"

**Set Git remote**

Note: This only needs to be done once, the details are provided by GitHub after creating a new project

git remote add origin <URL TO PROJECT>

git push -u origin master

**Push changes to GitHub**

git push origin

## SSH Key Management

The below commands are used to manage SSH keys on your local development machine.

**Checking for existing SSH key**

ls ~/.ssh/

**Print contents of public key**

cat ~/.ssh/id\_rsa.pub

**Generate new SSH key on your local machine**

ssh-keygen -t rsa -b 4096 -C "EMAIL ADDRESS"

## Virtual Environments

The below commands are used for managing Virtual Environments using Python3-env. Use these commands when connected to your Vagrant server.

**Create new environment**

python -m venv ~/env

**Activate virtual environment**

source ~/env/bin/activate

**De-activate virtual environment**

deactivate

**Install requirements from requirements.txt**

Note: Virtual environment must be activated

pip install -r requirements.txt

## Django Management Commands

**Create new Django project**

django-admin.py startproject profiles\_project .

**Create new Django app**

python manage.py startapp profiles\_api

**Start Django development server**

python manage.py runserver 0.0.0.0:8000

**Create database migrations file**

python manage.py makemigrations

**Run migrations**

python manage.py migrate

**Create new superuser**

python manage.py createsuperuser

## Vagrant

These commands are used for managing Vagrant using the GitBash or Terminal windows.

**Initialise Vagrant on project**

vagrant init ubuntu/bionic64

**Start Vagrant box**

vagrant up

**Connect to Vagrant box**

vagrant ssh

**Disconnect from Vagrant box**

Note: This command is a standard linux command for ending an SSH session

exit

**Stop Vagrant box**

vagrant halt

**Remove Vagrant box**

vagrant destroy

**Update Vagrant box image**

Note: you must rebuild the image after updating

vagrant box update

## Terminal / GitBash Commands

Change directory

cd /directory\_name

Change to parent directory

cd ..

### Finding Your Home DIR on MAC OS

From File structure:

Go-🡪 “Go To Folder” and in the search field enter Tilda “~”

This will take you to your home dir for me it is my username

ATOM

Setup project

File 🡪 “ Add Project Folder” navigate to our newly created folder that we created “profiles-rest-api”

## Create Git Project

Claudias-iMac:profiles-rest-api claudiaacerra$ **git init**

\*\*\*MUST RESTART ATOM TO DETECT THAT IT IS NOW A GIT REPO\*\*\*

Now this icon will spear next to the project name



## Create Readme file

Create file called readme.md (md stands for markdown)

# is the syntax for a heading



## Create .gitignore file

These are files we will exclude from our git repo

HERE it is for a python and vagrant project. There is a link below on how to do this right in the .gitignore

# Created by https://www.gitignore.io/api/python,vagrant

# Edit at https://www.gitignore.io/?templates=python,vagrant

### Python ###

# Byte-compiled / optimized / DLL files

\_\_pycache\_\_/

\*.py[cod]

\*$py.class

# C extensions

\*.so

# Distribution / packaging

.Python

build/

develop-eggs/

dist/

downloads/

eggs/

.eggs/

lib/

lib64/

parts/

sdist/

var/

wheels/

pip-wheel-metadata/

share/python-wheels/

\*.egg-info/

.installed.cfg

\*.egg

MANIFEST

# PyInstaller

# Usually these files are written by a python script from a template

# before PyInstaller builds the exe, so as to inject date/other infos into it.

\*.manifest

\*.spec

# Installer logs

pip-log.txt

pip-delete-this-directory.txt

# Unit test / coverage reports

htmlcov/

.tox/

.nox/

.coverage

.coverage.\*

.cache

nosetests.xml

coverage.xml

\*.cover

.hypothesis/

.pytest\_cache/

# Translations

\*.mo

\*.pot

# Django stuff:

\*.log

local\_settings.py

db.sqlite3

# Flask stuff:

instance/

.webassets-cache

# Scrapy stuff:

.scrapy

# Sphinx documentation

docs/\_build/

# PyBuilder

target/

# Jupyter Notebook

.ipynb\_checkpoints

# IPython

profile\_default/

ipython\_config.py

# pyenv

.python-version

# celery beat schedule file

celerybeat-schedule

# SageMath parsed files

\*.sage.py

# Environments

.env

.venv

env/

venv/

ENV/

env.bak/

venv.bak/

# Spyder project settings

.spyderproject

.spyproject

# Rope project settings

.ropeproject

# mkdocs documentation

/site

# mypy

.mypy\_cache/

.dmypy.json

dmypy.json

# Pyre type checker

.pyre/

### Python Patch ###

.venv/

### Vagrant ###

# General

.vagrant/

# Log files (if you are creating logs in debug mode, uncomment this)

# \*.logs

### Vagrant Patch ###

\*.box

# End of https://www.gitignore.io/api/python,vagrant

eb-package.zip

static/

@theyogicoderRI

## Create a License File

Protect yourself so no one can sue you.

MIT License

Copyright (c) 2019 YogiCoder

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files (the "Software"), to deal

in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.

## Git add and git commit

Claudias-iMac:profiles-rest-api claudiaacerra$ **git add .**

Claudias-iMac:profiles-rest-api claudiaacerra$ **git commit -am "First** commit license, readme and .gitignore"

[master (root-commit) 847efd2] First commit license, readme and .gitignore

3 files changed, 162 insertions(+)

create mode 100644 .gitignore

create mode 100644 LICENSE

create mode 100644 readme.md

## Push Could to github

Do not commit any secret keys to github

### Create a Public/Private key pair

Claudias-iMac:profiles-rest-api claudiaacerra$ ls ~/.ssh

ls: /Users/claudiaacerra/.ssh: No such file or directory

Claudias-iMac:profiles-rest-api claudiaacerra$ **ssh-keygen -t rsa -b** 4096 -C "claudia.acerra@gmail.com"

Generating public/private rsa key pair.

Enter file in which to save the key (/Users/claudiaacerra/.ssh/id\_rsa):

Created directory '/Users/claudiaacerra/.ssh'.

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

**Your identification has been saved in** /Users/claudiaacerra/.ssh/id\_rsa.

**Your public key has been saved in** /Users/claudiaacerra/.ssh/id\_rsa.pub.

**The key fingerprint is:**

SHA256:8MIOnrYL1HbymicbxTLz8VRssdfMXr5WtJq84kWaCGE claudia.acerra@gmail.com

**The key's randomart image is:**

+---[RSA 4096]----+

| . |

| . o + |

| .E = . + o|

| . o.o.o . . +.|

| . O B.S ...o|

| . o & =. . = o o|

| . = + .. o = o |

| o.=. .. o |

| \*= .... |

+----[SHA256]-----+

Claudias-iMac:profiles-rest-api claudiaacerra$ **ls ~/.ssh**

id\_rsa id\_rsa.pub

CAT THE PUBLIC KEY TO SAVE IT ON GITHUB FOR THIS COMPUTER

Claudias-iMac:profiles-rest-api claudiaacerra$ **cat ~/.ssh/id\_rsa.pub**

**ssh-rsa ** [**claudia.acerra@gmail.com**](mailto:claudia.acerra@gmail.com)

## Store SSH Public Key (Never give away your private key)

* 1. Login to GitHub
  2. Got to settings
  3. Go to SSH and GPG Keys
  4. Add new SSH key- which gets locked to a given host
* Paste the entire key into the “key text field” below



**After you add it, you will see this in the SSH link on Github**

****

## Create a new repo for this course on Github

### Do a git add origin to add the new repo to your local git repo through the CMD

Claudias-iMac:profiles-rest-api claudiaacerra$ pwd

/Users/claudiaacerra/courses/profiles-rest-api

Claudias-iMac:profiles-rest-api claudiaacerra$ **git remote add origin** [**https://github.com/cacerra1/profiles-rest-api.git**](https://github.com/cacerra1/profiles-rest-api.git)

### Then do a git push orgin master

Claudias-iMac:profiles-rest-api claudiaacerra$ **git push -u origin master**

Username for 'https://github.com': claudia.acerra@gmail.com

Password for 'https://claudia.acerra@gmail.com@github.com':

Counting objects: 5, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (5/5), done.

Writing objects: 100% (5/5), 1.85 KiB | 1.85 MiB/s, done.

Total 5 (delta 0), reused 0 (delta 0)

To https://github.com/cacerra1/profiles-rest-api.git

\* [new branch] master -> master

Branch 'master' set up to track remote branch 'master' from 'origin'.

# Create Local Dev Server

## Configure a Vagrantfile

Claudias-iMac:profiles-rest-api claudiaacerra$ **vagrant init ubuntu/bionic64**

* What this does it inits our project with a new vagrant file and bases it on the ubuntu/bionic64 image. This images are pibliclly available on the vagrant calatog box.
* The vagrant file now gets created automatically into our project folder as that is where we ran this command



## Configure Vagrant Box

Modifying the vagrant file with one we generated for us



I had to change my ports away from 8000 as they were already in use.

The host machine is the physical box we are running this project on and the guest machine is the development server itself. We need to make these ports accessible as we are doing here. **UPDATE: Changed by ports both back to 8000 after I killed some ports and rebooted**

Notice how we set python3 to the default python version. This lets us not need to type “python3” anymore but instead “python”… YAH!!!

## Running and connecting to our dev server

Claudias-iMac:profiles-rest-api claudiaacerra$ **vagrant up**

This downloads the image we have specified in our vagrant file and it will then use virtual box to create a new virtual machine and run our provisioning script when it starts the machine

## Connect to dev box

Claudias-iMac:profiles-rest-api claudiaacerra$ **vagrant ssh**

Now you will see we are in our virtual machine (by the green header on this command line)

Then CD into a folder called vagrant that gets automatically created

**vagrant@ubuntu-bionic**:**~**$ cd /vagrant

The create a new file could text.txt

**vagrant@ubuntu-bionic**:**/vagrant**$ touch test.txt

You will see that this gets created automatically in our project root



**vagrant@ubuntu-bionic**:**/vagrant**$ ls

LICENSE cheatSheet.txt readme.md ubuntu-bionic-18.04-cloudimg-console.log

Vagrantfile hello\_world.py test.txt

## Exit vargrant dev server:

**$exit**

Create a test hello.world file and run it from the cmd and it will print out its text:

**vagrant@ubuntu-bionic**:**/vagrant**$ python hello\_world.py

Hello World

This confirms all is working well

# Section 5: Creating our Django Application

## Create python virtual env on vagrant server

We create in env in the home dir of our vagrant server as opposed to the vagrant server which is synced to our local machine

## Activate the virtual server

**vagrant@ubuntu-bionic**:**/vagrant**$ source ~/env/bin/activate

(env) **vagrant@ubuntu-bionic**:**/vagrant**$

## Close virtual env

$ **deactivate**

## Install required Python Packages for our project

First create a requirements.txt file in the root of our project



Then run this command to install these two packages

(env) **vagrant@ubuntu-bionic**:**/vagrant**$ **pip install -r requirements.txt**

Collecting django==2.2 (from -r requirements.txt (line 1))

Downloading https://files.pythonhosted.org/packages/54/85/0bef63668fb170888c1a2970ec897d4528d6072f32dee27653381a332642/Django-2.2-py3-none-any.whl (7.4MB)

100% |████████████████████████████████| 7.5MB 258kB/s

Collecting djangorestframework==3.9.2 (from -r requirements.txt (line 2))

Downloading https://files.pythonhosted.org/packages/cc/6d/5f225f18d7978d8753c1861368efc62470947003c7f9f9a5cc425fc0689b/djangorestframework-3.9.2-py2.py3-none-any.whl (911kB)

100% |████████████████████████████████| 921kB 1.8MB/s

Collecting pytz (from django==2.2->-r requirements.txt (line 1))

Downloading https://files.pythonhosted.org/packages/3d/73/fe30c2daaaa0713420d0382b16fbb761409f532c56bdcc514bf7b6262bb6/pytz-2019.1-py2.py3-none-any.whl (510kB)

100% |████████████████████████████████| 512kB 2.5MB/s

Collecting sqlparse (from django==2.2->-r requirements.txt (line 1))

Downloading https://files.pythonhosted.org/packages/ef/53/900f7d2a54557c6a37886585a91336520e5539e3ae2423ff1102daf4f3a7/sqlparse-0.3.0-py2.py3-none-any.whl

Installing collected packages: pytz, sqlparse, django, djangorestframework

Successfully installed django-2.2 djangorestframework-3.9.2 pytz-2019.1 sqlparse-0.3.0

(env) **vagrant@ubuntu-bionic**:**/vagrant**$

## Verify what got installed:

(env) **vagrant@ubuntu-bionic**:**/vagrant**$ pip freeze

Django==2.2

djangorestframework==3.9.2

pkg-resources==0.0.0

pytz==2019.1

sqlparse==0.3.0

(env) **vagrant@ubuntu-bionic**:**/vagrant**$

## Create New Django Project

(env) **vagrant@ubuntu-bionic**:**/vagrant**$ **django-admin.py startproject profiles\_project .**

(env) **vagrant@ubuntu-bionic**:**/vagrant**$



## Create a Django application

(env) **vagrant@ubuntu-bionic**:**/vagrant**$ **python manage.py startapp profiles\_api**

(env) **vagrant@ubuntu-bionic**:**/vagrant**$



## Enable our app in Django settings.py file



We added the last three entries in the list above

## Start The server

(env) **vagrant@ubuntu-bionic**:**/vagrant**$ **python manage.py runserver 0.0.0.0:8000**

Watching for file changes with StatReloader

Performing system checks...

System check identified no issues (0 silenced).

You have 19 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, authtoken, contenttypes, sessions.

Run 'python manage.py migrate' to apply them.

July 23, 2019 - 20:41:01

Django version 2.2, using settings 'profiles\_project.settings'

Starting development server at http://0.0.0.0:8000/

Quit the server with CONTROL-C.

Exit

Then go to:

<http://127.0.0.1:8000/>

In your web browser to see the Django Rocket



# Models:

We are creating customer user authentication



## **Substituting a custom**User**model**[¶](https://docs.djangoproject.com/en/1.11/topics/auth/customizing/#substituting-a-custom-user-model)

Some kinds of projects may have authentication requirements for which Django’s built-in [**User**](https://docs.djangoproject.com/en/1.11/ref/contrib/auth/#django.contrib.auth.models.User) model is not always appropriate. For instance, on some sites it makes more sense to use an email address as your identification token instead of a username.

Django allows you to override the default user model by providing a value for the [**AUTH\_USER\_MODEL**](https://docs.djangoproject.com/en/1.11/ref/settings/#std:setting-AUTH_USER_MODEL) setting that references a custom model:

AUTH\_USER\_MODEL = 'myapp.MyUser'

This dotted pair describes the name of the Django app (which must be in your [**INSTALLED\_APPS**](https://docs.djangoproject.com/en/1.11/ref/settings/#std:setting-INSTALLED_APPS)), and the name of the Django model that you wish to use as your user model.

### Using a custom user model when starting a project[¶](https://docs.djangoproject.com/en/1.11/topics/auth/customizing/#using-a-custom-user-model-when-starting-a-project)

If you’re starting a new project, it’s highly recommended to set up a custom user model, even if the default [**User**](https://docs.djangoproject.com/en/1.11/ref/contrib/auth/#django.contrib.auth.models.User)model is sufficient for you. This model behaves identically to the default user model, but you’ll be able to customize it in the future if the need arises:

**from** **django.contrib.auth.models** **import** AbstractUser

**class** **User**(AbstractUser):

**pass**

Don’t forget to point [**AUTH\_USER\_MODEL**](https://docs.djangoproject.com/en/1.11/ref/settings/#std:setting-AUTH_USER_MODEL) to it. Do this before creating any migrations or running **manage.pymigrate** for the first time.

Also, register the model in the app’s **admin.py**:

**from** **django.contrib** **import** admin

**from** **django.contrib.auth.admin** **import** UserAdmin

**from** **.models** **import** User

admin.site.register(User, UserAdmin)

## Create customer User Manager



## Set Custom User Model in settings.py



Make Migrations and Migrate:



(env) **vagrant@ubuntu-bionic**:**/vagrant**$ python manage.py migrate

**Operations to perform:**

**Apply all migrations:** admin, auth, authtoken, contenttypes, profiles\_api, sessions

**Running migrations:**

Applying contenttypes.0001\_initial... **OK**

Applying contenttypes.0002\_remove\_content\_type\_name... **OK**

Applying auth.0001\_initial... **OK**

Applying auth.0002\_alter\_permission\_name\_max\_length... **OK**

Applying auth.0003\_alter\_user\_email\_max\_length... **OK**

Applying auth.0004\_alter\_user\_username\_opts... **OK**

Applying auth.0005\_alter\_user\_last\_login\_null... **OK**

Applying auth.0006\_require\_contenttypes\_0002... **OK**

Applying auth.0007\_alter\_validators\_add\_error\_messages... **OK**

Applying auth.0008\_alter\_user\_username\_max\_length... **OK**

Applying auth.0009\_alter\_user\_last\_name\_max\_length... **OK**

Applying auth.0010\_alter\_group\_name\_max\_length... **OK**

Applying auth.0011\_update\_proxy\_permissions... **OK**

Applying profiles\_api.0001\_initial... **OK**

Applying admin.0001\_initial... **OK**

Applying admin.0002\_logentry\_remove\_auto\_add... **OK**

Applying admin.0003\_logentry\_add\_action\_flag\_choices... **OK**

Applying authtoken.0001\_initial... **OK**

Applying authtoken.0002\_auto\_20160226\_1747... **OK**

Applying sessions.0001\_initial... **OK**

(env) **vagrant@ubuntu-bionic**:**/vagrant**$

### Then push Model Changes to git and Github

This from new command window- do not need to be in virtual env, just at root of our project folder

Last login: Thu Jul 25 09:05:38 on ttys000

Claudias-iMac:profiles-rest-api claudiaacerra$ pwd

/Users/claudiaacerra/courses/profiles-rest-api

Claudias-iMac:profiles-rest-api claudiaacerra$ git add .

Claudias-iMac:profiles-rest-api claudiaacerra$ git commit -am "Model and DB changes"

[master de75d57] Model and DB changes

4 files changed, 98 insertions(+), 1 deletion(-)

create mode 100644 profiles\_api/migrations/0001\_initial.py

Claudias-iMac:profiles-rest-api claudiaacerra$ git push origin

Counting objects: 9, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (9/9), done.

Writing objects: 100% (9/9), 863.73 KiB | 10.53 MiB/s, done.

Total 9 (delta 5), reused 0 (delta 0)

remote: Resolving deltas: 100% (5/5), completed with 5 local objects.

remote:

remote: GitHub found 2 vulnerabilities on cacerra1/profiles-rest-api's default branch (2 moderate). To find out more, visit:

remote: https://github.com/cacerra1/profiles-rest-api/network/alerts

remote:

To https://github.com/cacerra1/profiles-rest-api.git

b7dbfcd..de75d57 master -> master

Claudias-iMac:profiles-rest-api claudiaacerra$

# Create Super User

(env) **vagrant@ubuntu-bionic**:**/vagrant**$ python manage.py createsuperuser

Email: claudia.acerra@gmail.com

Name: admin

Password:

Password (again):

**This password is too common.**

Bypass password validation and create user anyway? [y/N]: y

Superuser created successfully.

(env) **vagrant@ubuntu-bionic**:**/vagrant**$

(PW is admin123)

## Register New Model in admin.py file in your application



# Looking at Admin Console

First thing is we loggin with our email address instead of user name

The Rest framework creates the other two things



# Section 8 – API Views

Django Rest Framework offers helper classes:

APIView and **ViewSet**

## APIView

Allows us define functions that match the standard HTTP methods

1. HTTP Get
2. HTTP Post
3. HTTP Put- Update
4. HTTP patch partially update an item
5. HTTP delete

Give us the most control of our HTTP logic

## When to use APIViews?

When you need full control over application logic (updating multiple data sources in one API call)?

1. Processing files and rendering a synchronous response
2. Calling other API services in one request
3. Access to local files or data

## Create APIView



## Configure a View URL

# Projects url first





## Render our new view

