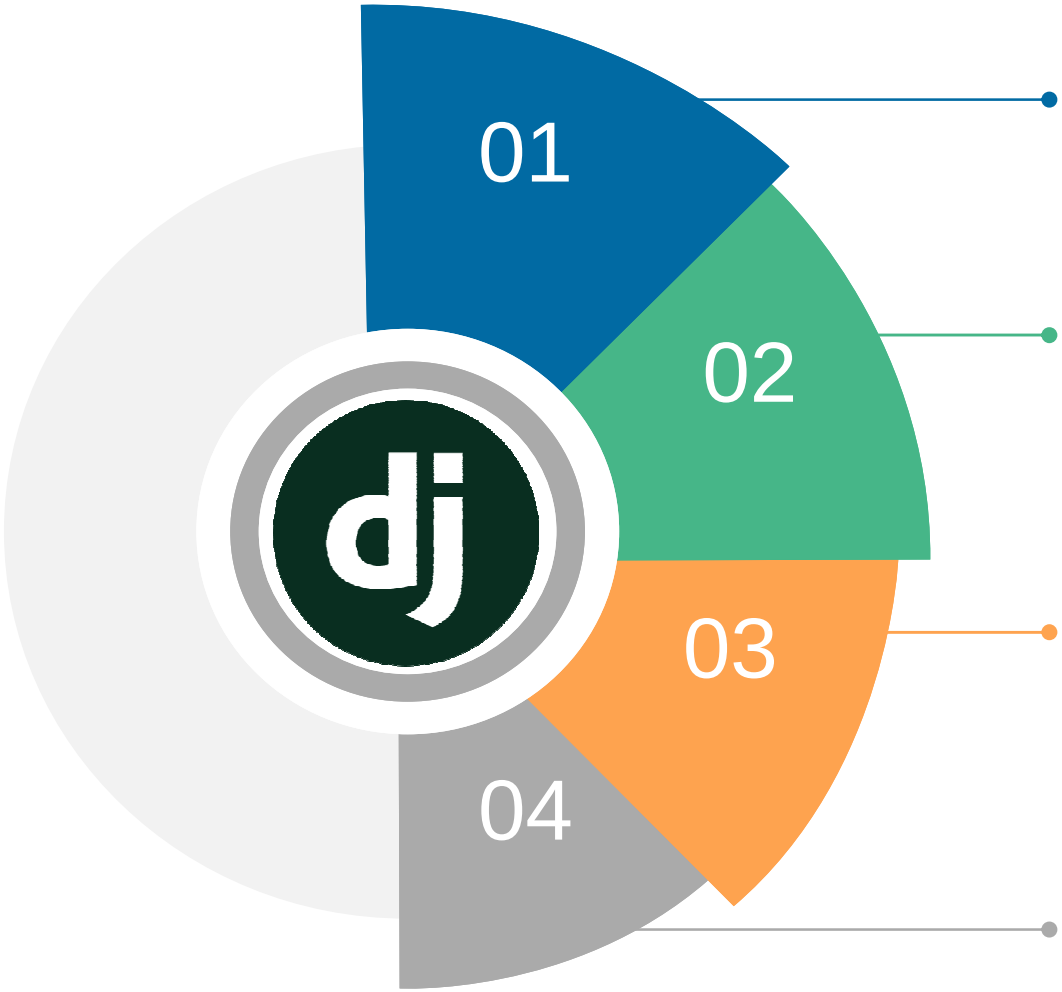
DJANGO MODELS

What is a Model ?



Model is a data access layer of Django.

It gives way to access it, validate it and describes the relationship between the data.

It maps single database table to a python class.

It handles the inconsistency of SQL across different platform.

Django ORM

01 Each Model is a python class that subclasses django.db.models.Model

02 Each attribute of model represent a database field

03 With all of this django gives you an automat ically-generated database-access API

Model Creation

Example

```
from django.db import models

class Person(models.Model):
        first_name = models.CharField(max_length=30)
        last_name  = models.CharField(max_length=30)
```

Model Fields

- Fields are specified by class attributes which represents columns in the database table(Model)
- Each Field in model class should be an instance of appropriate Field class(Which is an abstract class)
- These are the things to know for creating model fields

Fields Types

AutoField()

An integer field that automatically increments

BooleanField()

Store true/false value and generally used for check boxes

DateField()

A date field represents python datetype.date instance

IntegerField()

It stores value from -2147483648 to 2147483647

CharField()

A string field for small to large sized strings

FloatField()

It stores floating point number represented in python by a float instance

Special Field Type

EmailField()

A CharField that check valid email address

Relationship Fields

ForeignKey()

A many to one relationship requires one positional argument to define related model

ManyToManyField()

Defines a many to many relationship with another model

OneToOneField()

Define a one-to-one relationship

Field Options

- Field option are used to customize and put constraint on table rows
- Each field takes certain field specific arguments

Eg:

```
name = models.CharField(max_length=60)
```

Here “max_length” specifies the size of the VARCHAR field

The following are some common and mostly used field options:

01

null

To store empty values as NULL in database

02

blank

If True, the field is allowed to be blank. Default is False

03

default

Stores default value for the field

04

primary_key

This key will be the primary key for the table

05

unique_key

Puts unique ke constraint for column

Relationships

- **Django provides ways to define the three most common types of database relationships:**

01 many-to-one

02 many-to-many

03 Httpresponse

Choices

A sequence of 2-tuples to use as choices for this field.
The default form widget will be a select box instead of the standard text field and will limit choices to the choices given.

choices list looks like this:

```
YEAR_IN_SCHOOL_CHOICES = [
            ('FR', 'Freshman’),
            ('SO', 'Sophomore’),
            ('JR', 'Junior’),
            ('SR', 'Senior’),
            ('GR', 'Graduate’),
 ]
```

se PostgreSQL with your Django Application on Ubunt

- Django is a powerful web framework that can help you g your Python application or website off the ground.

- Django includes a simplified development server for testin your code locally, but for anything even slightly productio related, a more secure and powerful web server is required.

Prerequisites and Goals

1. We will be installing Django within a virtual environment.

2. Installing Django into an environment specific to your project will allow your projects and their requirements to be handled separately.

Install the Packages from the Ubuntu Repositories

- To begin the process, we'll download and install all of the items we need from the Ubuntu repositories.
- We will use the Python package manager pip to install additional components a bit later.
- We need to update the local apt package index and then download and install the packages.
- The packages we install depend on which version of Python your project will use.

If you are using Django with Python 3, type:

```
$ sudo apt-get update
```

```
(ENV) amit@amit:~/Django_projects/MyProject/Myproject$ sudo apt-get update
[sudo] password for amit:
Hit:1 http://in.archive.ubuntu.com/ubuntu bionic InRelease
Hit:2 http://in.archive.ubuntu.com/ubuntu bionic-updates InRelease
Hit:3 http://in.archive.ubuntu.com/ubuntu bionic-backports InRelease
Ign:4 http://dl.google.com/linux/chrome/deb stable InRelease
Hit:5 http://security.ubuntu.com/ubuntu bionic-security InRelease
Hit:6 http://dl.google.com/linux/chrome/deb stable Release
Hit:8 http://apt.postgresql.org/pub/repos/apt bionic-pgdg InRelease
Reading package lists... Done
```

$ sudo apt-get install python3-pip python3-dev libpq-dev postgresql postgresql-contrib nginx

```
) amit@amit:~/Django_projects/MyProject/Myproject$ sudo apt-get install python3-pip python3-dev libpq-dev postgresql postgresql-contrib nginx
ing package lists... Done
ding dependency tree
ing state information... Done
on3-dev is already the newest version (3.6.7-1~18.04).
on3-dev set to manually installed.
on3-pip is already the newest version (9.0.1-2.3~ubuntu1.18.04.1).
q-dev is already the newest version (12.1-1.pgdg18.04+1).
following package was automatically installed and is no longer required:
bllvm8
'sudo apt autoremove' to remove it.
following additional packages will be installed:
bnginx-mod-http-geoip libnginx-mod-http-image-filter libnginx-mod-http-xslt-filter libnginx-mod-mail libnginx-mod-stream nginx-common nginx-cor
ested packages:
giwrap nginx-doc postgresql-doc
following NEW packages will be installed:
bnginx-mod-http-geoip libnginx-mod-http-image-filter libnginx-mod-http-xslt-filter libnginx-mod-mail libnginx-mod-stream nginx nginx-common
inx-core postgresql postgresql-contrib
graded, 10 newly installed, 0 to remove and 60 not upgraded.
 to get 723 kB of archives.
r this operation, 2,255 kB of additional disk space will be used.
ou want to continue? [Y/n] y
```

his will install pip, the Python development files needed to build Gunicorn
ter, the Postgres database system and the libraries needed to interact with
, and the Nginx web server.

Create the PostgreSQL Database and User

We're going to jump right in and create a database and database user for ou Django application.

- **By default, Postgres uses an authentication scheme called "pee authentication" for local connections.**
- **Basically, this means that if the user's operating system usernam matches a valid Postgres username, that user can login with no furthe authentication.**
- **During the Postgres installation, an operating system user name postgres was created to correspond to the postgres PostgreSQ administrative user.**
- **We need to use this user to perform administrative tasks. We can us sudo and pass in the username with the -u option.**

og into an interactive Postgres session by typin

```
$ sudo -u postgres psql
```

```
(ENV) amit@amit:~/Django_projects/MyProject/Myproject$ sudo -u postgres psql
psql (12.1 (Ubuntu 12.1-1.pgdg18.04+1))
Type "help" for help.

postgres=#
```

You will be given a PostgreSQL prompt where we can set up our requirements.

First, create a database for your project:

```
postgres=# CREATE DATABASE mydatabase;
```

```
(ENV) amit@amit:~/Django_projects/MyProject/Myproject$ sudo -u postgres psql
psql (12.1 (Ubuntu 12.1-1.pgdg18.04+1))
Type "help" for help.

postgres=# CREATE DATABASE mydatabase;
CREATE DATABASE
```

Note:
Every Postgres statement must end with a semi-colon, so make sure that your command ends with one if you are experiencing issues.

Create a database user for our project.

Make sure to select a secure password:

```
postgres=# CREATE USER myprojectuser WITH PASSWORD '12345';
```

```
(ENV) amit@amit:~/Django_projects/MyProject/Myproject$ sudo -u postgres psql
psql (12.1 (Ubuntu 12.1-1.pgdg18.04+1))
Type "help" for help.

postgres=# CREATE DATABASE mydatabase;
CREATE DATABASE
postgres=# CREATE USER myprojectuser WITH PASSWORD '12345';
CREATE ROLE
```

- Afterwards, we'll modify a few of the connection parameters for the user we just created.
- This will speed up database operations so that the correct values do not have to be queried and set each time a connection is established.
- We are setting the default encoding to UTF-8, which Django expects.
- We are also setting the default transaction isolation scheme to "read committed", which blocks reads from uncommitted transactions.
- Lastly, we are setting the timezone. By default, our Django projects will be set to use UTC.
- These are all recommendations from the Django project itself:

postgres=# ALTER ROLE myprojectuser SET client_encoding TO 'utf8';

```
ostgres=# CREATE DATABASE mydatabase;
REATE DATABASE
ostgres=# CREATE USER myprojectuser WITH PASSWORD '12345';
REATE ROLE
ostgres=# ALTER ROLE myprojectuser SET client_encoding TO 'utf8';
LTER ROLE
```

postgres=# ALTER ROLE myprojectuser SET default_transaction_isolation TO 'read committed';

```
stgres=# CREATE DATABASE mydatabase;
EATE DATABASE
stgres=# CREATE USER myprojectuser WITH PASSWORD '12345';
EATE ROLE
stgres=# ALTER ROLE myprojectuser SET client_encoding TO 'utf8';
TER ROLE
stgres=# ALTER ROLE myprojectuser SET default_transaction_isolation TO 'read committed';
TER ROLE
```

postgres=# ALTER ROLE myprojectuser SET timezone TO 'UTC';

```
ostgres=# CREATE DATABASE mydatabase;
REATE DATABASE
ostgres=# CREATE USER myprojectuser WITH PASSWORD '12345';
REATE ROLE
ostgres=# ALTER ROLE myprojectuser SET client_encoding TO 'utf8';
LTER ROLE
ostgres=# ALTER ROLE myprojectuser SET default_transaction_isolation TO 'read committed';
LTER ROLE
ostgres=# ALTER ROLE myprojectuser SET timezone TO 'UTC';
LTER ROLE
```

Now, we can give our new user access to administer our new database:

postgres=# GRANT ALL PRIVILEGES ON DATABASE mydatabase TO myprojectuser;

```
res=# CREATE DATABASE mydatabase;
E DATABASE
res=# CREATE USER myprojectuser WITH PASSWORD '12345';
E ROLE
res=# ALTER ROLE myprojectuser SET client_encoding TO 'utf8';
 ROLE
res=# ALTER ROLE myprojectuser SET default_transaction_isolation TO 'read committed';
 ROLE
res=# ALTER ROLE myprojectuser SET timezone TO 'UTC';
 ROLE
res=# GRANT ALL PRIVILEGES ON DATABASE mydatabase TO myprojectuser;
```

When you are finished, exit out of the PostgreSQL prompt by typing:

```
postgres=# \q
```

```
ostgres=# CREATE DATABASE mydatabase;
REATE DATABASE
ostgres=# CREATE USER myprojectuser WITH PASSWORD '12345';
REATE ROLE
ostgres=# ALTER ROLE myprojectuser SET client_encoding TO 'utf8';
LTER ROLE
ostgres=# ALTER ROLE myprojectuser SET default_transaction_isolation TO 'read committed';
LTER ROLE
ostgres=# ALTER ROLE myprojectuser SET timezone TO 'UTC';
LTER ROLE
ostgres=# GRANT ALL PRIVILEGES ON DATABASE mydatabase TO myprojectuser;
RANT
ostgres=# \q
```

Create a Python Virtual Environment for your Project

- Now that we have our database, we can begin getting the rest of ou project requirements ready. We will be installing our Python requirement within a virtual environment for easier management.

- To do this, we first need access to the virtualenv command. We can instal this with pip.

If you are using Python 3, upgrade pip and install the package by typing:

```
$ sudo -H pip3 install --upgrade pip
$ sudo -H pip3 install virtualenv
```

Within the project directory, create a Python virtual environment by typing:

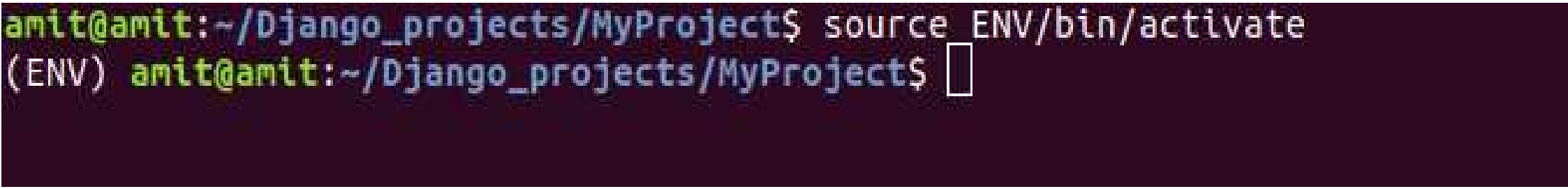
$ virtualenv ENV

- **This will create a directory called ENV within your Myproject directory.**
- **Inside, it will install a local version of Python and a local version of pip. We can use this to install and configure an isolated Python environment for our project.**

```
mit@amit:~/Django_projects/MyProject$ virtualenv ENV
sing base prefix '/usr'
ew python executable in /home/amit/Django_projects/MyProject/ENV/bin/python3
lso creating executable in /home/amit/Django_projects/MyProject/ENV/bin/python
nstalling setuptools, pip, wheel...
one.
```

Before we install our project's Python requirements, we need to activate the virtual environment. You can do that by typing:

```
$ source ENV/bin/activate
```



With your virtual environment active, install Django, Gunicorn, and the psycopg2 PostgreSQL adaptor with the local instance of pip:

$ pip install django gunicorn psycopg2

```
amit@amit:~/Django_projects/MyProject/Myproject$ pip install django gunicorn psycopg2
ement already satisfied: django in /home/amit/Django_projects/MyProject/ENV/lib/python3.6/site-packages (3.0.3)
ting gunicorn
oading gunicorn-20.0.4-py2.py3-none-any.whl (77 kB)
                                     | 77 kB 53 kB/s
ting psycopg2
oading psycopg2-2.8.4.tar.gz (377 kB)
                                     | 377 kB 69 kB/s
ement already satisfied: asgiref~=3.2 in /home/amit/Django_projects/MyProject/ENV/lib/python3.6/site-packages (from django) (3.2.3)
ement already satisfied: sqlparse>=0.2.2 in /home/amit/Django_projects/MyProject/ENV/lib/python3.6/site-packages (from django) (0.3.0)
ement already satisfied: pytz in /home/amit/Django_projects/MyProject/ENV/lib/python3.6/site-packages (from django) (2019.3)
ement already satisfied: setuptools>=3.0 in /home/amit/Django_projects/MyProject/ENV/lib/python3.6/site-packages (from gunicorn) (45.1.0)
g wheels for collected packages: psycopg2
ding wheel for psycopg2 (setup.py) ... done
ted wheel for psycopg2: filename=psycopg2-2.8.4-cp36-cp36m-linux_x86_64.whl size=419362 sha256=7c999246287e35a38655fa42a685da0e82b2f8b91ebc
ac1dcffdf
ed in directory: /home/amit/.cache/pip/wheels/1e/cb/03/20479d63812e4a01da92afa8753f6ad37f87806c24a620aaa0
sfully built psycopg2
ling collected packages: gunicorn, psycopg2
sfully installed gunicorn-20.0.4 psycopg2-2.8.4
```

Create and Configure a New Django Project

With our Python components installed, we can create the actual Django project files.

Create the Django Project

Since we already have a project directory, we will tell Django to install the files here. It will create a second level directory with the actual code, which is normal, and place a management script in this directory.

```
$ django-admin.py startproject Myproject
```

```
(ENV) amit@amit:~/Django_projects/MyProject$ django-admin startproject Myprojec
(ENV) amit@amit:~/Django_projects/MyProject$ code .
(ENV) amit@amit:~/Django_projects/MyProject$
```

At this point, your project directory (~/Myproject in our case) should have the following content:

~/Myproject/manage.py: A Django project management script.
~/Myproject/: The Django project package. This should contain the __init__.py, settings.py, urls.py, and wsgi.py files.
~/Myproject/ENV/: The virtual environment directory we created earlier.

Adjust the Project Settings

The first thing we should do with our newly created project files is adjust the settings.

```
$ nano ~/Myproject/settings.py
```

- Start by locating the ALLOWED_HOSTS directive.
- This defines a list of the server's addresses or domain names may be used to connect to the Django instance.
- Any incoming requests with a Host header that is not in this list will raise an exception.
- Django requires that you set this to prevent a certain class of security vulnerability.

In the square brackets, list the IP addresses or domain names that ar
associated with your Django server.

Each item should be listed in quotations with entries separated by
comma.

If you wish requests for an entire domain and any subdomains, prepend
period to the beginning of the entry. In the snippet below, there are a fe
commented out examples used to demonstrate:

Myproject/settings.py

```
# The simplest case: just add the domain name(s) and IP addresses
of your Django server
# ALLOWED_HOSTS = [ 'example.com', '203.0.113.5’]
# To respond to 'example.com' and any subdomains, start the domain
with a dot
# ALLOWED_HOSTS = ['.example.com',
'203.0.113.5']ALLOWED_HOSTS = ['your_server_domain_or_IP',
'second_domain_or_IP', . . .]
```

Next, find the section that configures database access. It will start with DATABASES. The configuration in the file is for a SQLite database. We already created a PostgreSQL database for our project, so we need to adjust the settings.

Change the settings with your PostgreSQL database information. We tell Django to use the psycopg2 adaptor we installed with pip. We need to give the database name, the database username, the database user's password, and then specify that the database is located on the local computer. You can leave the PORT setting as an empty string:

Myproject/settings.py

```
DATABASES = {
'default':
        {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'mydatabase',
        'USER': 'myprojectuser',
        'PASSWORD': '12345',
        'HOST': 'localhost',
        'PORT': '',
    }
}
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'mydatabase',
        'USER': 'myprojectuser',
        'PASSWORD': '12345',
        'HOST': 'localhost',
        'PORT': '',
    }
}
```

Next, move down to the bottom of the file and add a setting indicating where the static files should be placed.

This is necessary so that Nginx can handle requests for these items. The following line tells Django to place them in a directory called static in the base project directory:

Myproject/settings.py

```
STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static/')
```

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'static/')
```

Save and close the file when you are finished.

~/myproject/python manage.py makemigrations
~/myproject/python manage.py migrate

```
ENV) amit@amit:~/Django_projects/MyProject/Myproject$ python manage.py makemigrations
o changes detected
ENV) amit@amit:~/Django_projects/MyProject/Myproject$ python manage.py migrate
perations to perform:
 Apply all migrations: admin, auth, contenttypes, sessions
unning migrations:
 Applying contenttypes.0001_initial... OK
 Applying auth.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 contenttypes.0002_remove_content_type_name... 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 sessions.0001_initial... OK
```

reate an administrative user for the project by typing

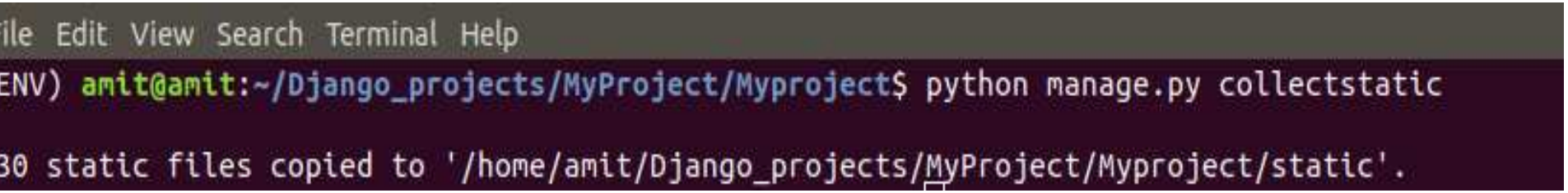
~/myproject/python manage.py createsuperuser

```
ENV) amit@amit:~/Django_projects/MyProject/Myproject$ python manage.py createsuperuser
sername (leave blank to use 'amit'): admin
mail address:
assword:
assword (again):
```

You will have to select a username, provide an email address, and choose and confirm a password.

e can collect all of the static content into the director
cation we configured by typing:

~/myproject/python manage.py collectstatic



You will have to confirm the operation. The static files will then be placed in a directory called static within your project directory.

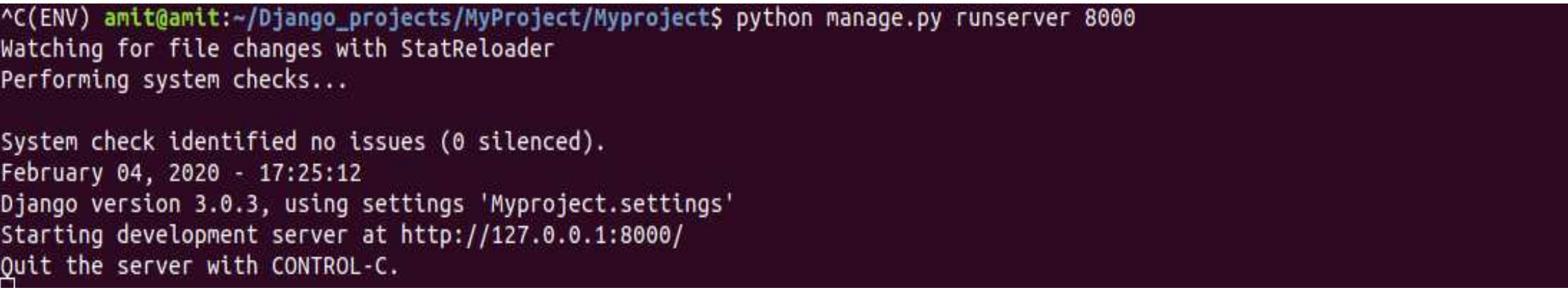
- If you followed the initial server setup guide, you should have a UF firewall protecting your server.
- In order to test the development server, we'll have to allow access to th port we'll be using.

Create an exception for port 8000 by typing:

```
(ENV) amit@amit:~/Django_projects/MyProject/Myproject$ sudo ufw allow 8000
[sudo] password for amit:
Rules updated
Rules updated (v6)
```

Finally, you can test our your project by starting up the Django development server with this command:

```
$ python manage.py runserver 8000
```



f you append /admin to the end of the URL in th ddress bar, you will be prompted for the administrativ sername and password you created with th reatesuperuser command:

- **After authenticating, you can access the default Django admin interface:**
- **When you are finished exploring, hit CTRL-C in the terminal window to shut down the development server.**

Use Oracle Database With Django In Windows

In order to connect to Oracle from Python, we need to install cx_Oracle package, which is DB-API 2.0 implementation, using PIP.

Getting Oracle Database ready

We work with Oracle Database 11g Express Edition (XE). If you have not yet installed it, go to Oracle Database 11g Express Edition and download the version relevant to your platform.

- Download and install Python 3.6 for our platform
- The python package that is used to connect to Oracle from Python is cx_Oracle
- Go to directory where pip.exe (or pip3.6.exe) is present and give following command. Pip.exe is present in /Scripts folder in Windo installation. Check your platform for details.

```
pip install cx_Oracle
```

Installing Instant Client

- In order to access Oracle from Python, you need to install Instant Client that is specific to your platform.
- Go to Instant Client and select download for your platform.
- You need to download Instant Client 64 bit if your Python is 64 bit otherwise download 32 bit version. Python edition and Instant Client edition must be same.
- It is enough to download Instant Client Basic Light version as we need support for only English.

- You can check which version of Python you are using

```
C:\python>python
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)]
on win32 Type "help", "copyright", "credits" or "license" for more information.
 >>>
```

- Make sure **Oracle's BIN** directory or **InstantClient** folder is in system **PATH**.

- The following command shows how to add Oracle's BIN directory to system PATH:

```
\>set PATH=%PATH%;C:\oraclexe\app\oracle\product\11.2.0\server\bi
```

- If you are using InstantClient, use folder into which you installed InstantClient ex: c:\python\instantclient) as follows:

```
:\>set PATH=%PATH%;C:\python\instantclient
```

The following program will connect to Oracle Database using username gktcs and password gktcs.

In case you are trying to use a different account or a different version of Oracle database then feel free to change the details and experiment.

```
import os
import cx_Oracle

# Connect to hr account in Oracle Database 11g Express Edition(XE)
con = cx_Oracle.connect("gktcs", "gktcs", "localhost/xe")
print("Connected!")
con.close()
```

Creating Django Project and Application

We need to install Django Framework using PIP as follows:

```
pip install django
```

Installing Django, installs django-admin.exe in Scripts folder of Python installation directory.

Let's create a new Django Project called **oracledemo** and Application called **blog** in that project with the following commands:

django-admin startproject oracledemo

- It will create a new folder oracledemo and places manage.py and anothe folder oracledemo inside that. Get into oracledemo folder and then run manage.py to create a new application called **blog** inside that project.

python manage.py startapp blog

Configure Django settings.py

Open settings.py file that is placed in oracledemo folder, which is inside oracledemo (project) folder.

- Add application hr as one of the installed applications by adding it to INSTALLED_APPS list.

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

- Modify default database configuration so that Oracle is used as default database.
- we are using Oracle Database 11g XE running on the current system.

```
DATABASES = {
'default': {
        'ENGINE': 'django.db.backends.oracle’,
        'NAME': 'XE’,
        'USER': ‘gktcs’,
        'PASSWORD': ‘gktcs’,
        'HOST': 'localhost’,
         'PORT': '1521' } }
```

Creating Model

- Create a class that represents JOBS table in Oracle database in oracledemo/hr/models.py file.

```
ass Post(models.Model):
  title =models.CharField(max_length=120)
  content=models.TextField()
  updated=models.DateTimeField(auto_now=True, auto_now_add=False)
  timestamp=models.DateTimeField(auto_now=False, auto_now_add=True
  class Meta:
      db_table = “posts"
```

Creating view and template

- Create the following function view in oracledemo/blog/views.py to display details of posts from Post table.

```
def blog_posts (request):
        return render(request,'blog_post.html',{`posts' : Post.objects.all()})
```

- Here is oracledemo/blog/templates/blog_post.html to display details of Jobs in HTML table.

og_post.html

```
<!DOCTYPE html>
<html lang="en">
    <head> <meta charset="UTF-8">
        <title>Blogs</title>
    </head>
<body>
<h1>Blog</h1>

<table width="100%" border="1">
    <tr style="background-color:lightgray">
        <th>Title</th>
        <th>Content</th>
        <th>Updated Time</th>
    </tr>
```

```
    {% for post in posts %}
        <tr>
                <td>{{post.title}}</td>
                <td>{{post.content}}</td>
                 <td>{{post.updated}}</td>
        </tr>
     {% endfor %}

</table>
</body>
</html>
```

- Finally add a new URL in oracledemo/oracledemo/urls.py that invokes blog_posts().

```
from django.urls
import path import blog.views as blog_views

urlpatterns = [
        path('post', blog_views. blog_posts )
 ]
```

- Start server after adding Oracle's bin directory or InstantClient director
  to system PATH.

```
>set PATH=%PATH%;C:\oraclexe\app\oracle\product\11.2.0\server\bi
>python manage.py runserver
```

- Now go to browser and enter the following URL to get list of Jobs.

http://localhost:8000/post

reating a Django Web Application with a PostgreSQL atabase on Windows

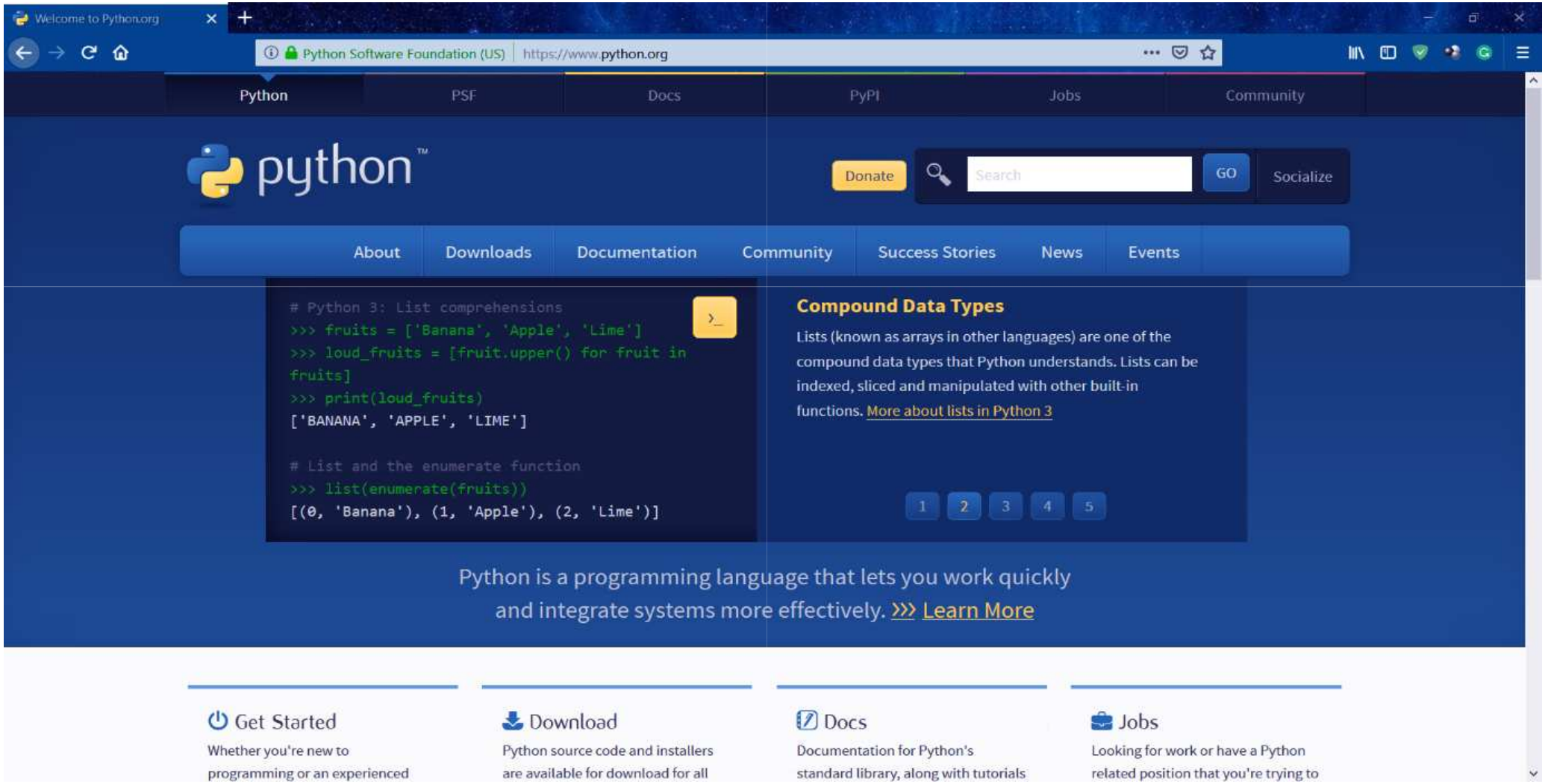
Installing Python 3

To run the Django Framework on your system you would need Python 3 installed on your system.

You just have to download the package from the official website, www.python.org, according to your operating system.

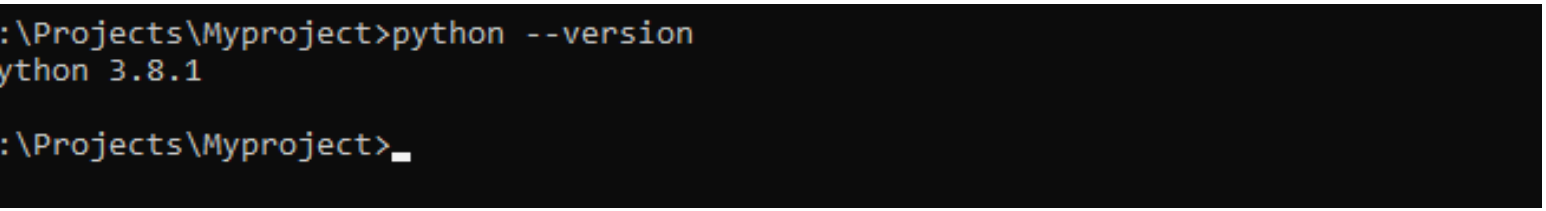
Keep in mind you have to install a Python 3 version, not the version 2.

- While installing Python 3 don’t forget to tick the option “Add Python 3 to your path” when the installation prompt opens.



- You can check that the Python version is on your system by typing this command in PowerShell:

```
python --version
```



Installing Virtualenv

Virtualenv is a Python package that lets you create different vir
nvironments for multiple projects requiring different versions of the softwa

Pip install virtualenv

```
\Projects\Myproject>pip install virtualenv
ollecting virtualenv
 Downloading https://files.pythonhosted.org/packages/05/f1/2e07e8ca50e047b9cc9ad56cf4291f4e041fa73207d000a095fe478abf84/virtualenv-16.7.9-py2.py3-none-any.whl
    |                                | 3.4MB 386kB/s
nstalling collected packages: virtualenv
uccessfully installed virtualenv-16.7.9
ARNING: You are using pip version 19.3.1; however, version 20.0.2 is available.
ou should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

This will download and install virtualenv on your system globally

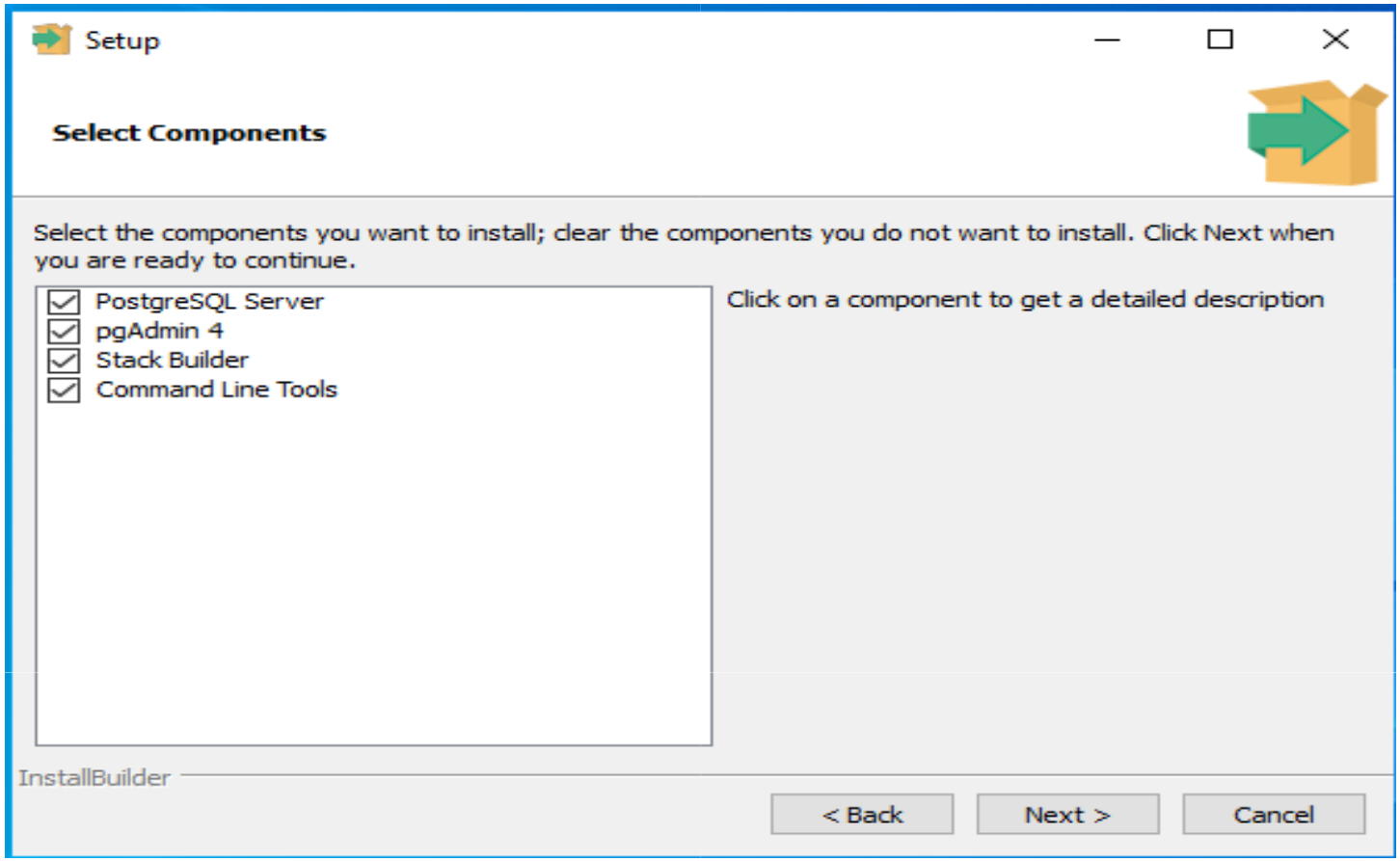
Install PostgreSQL

Go to the download page for PostgreSQL installers and install the latest version of PostgreSQL that is suitable to your system (64-bit Windows).

The installation process is pretty standard but here are some things you should look out for:

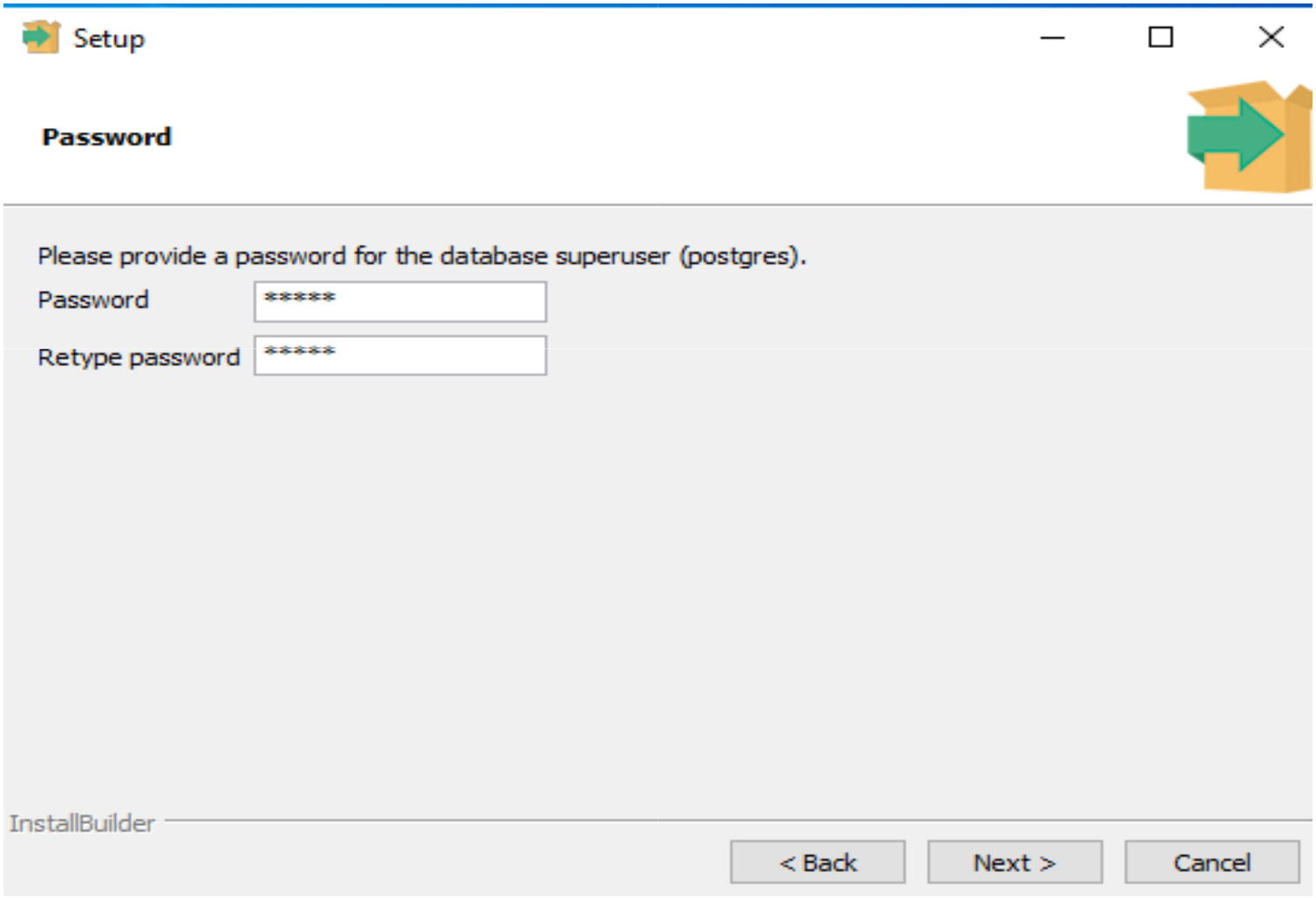
PostgreSQL Database Download

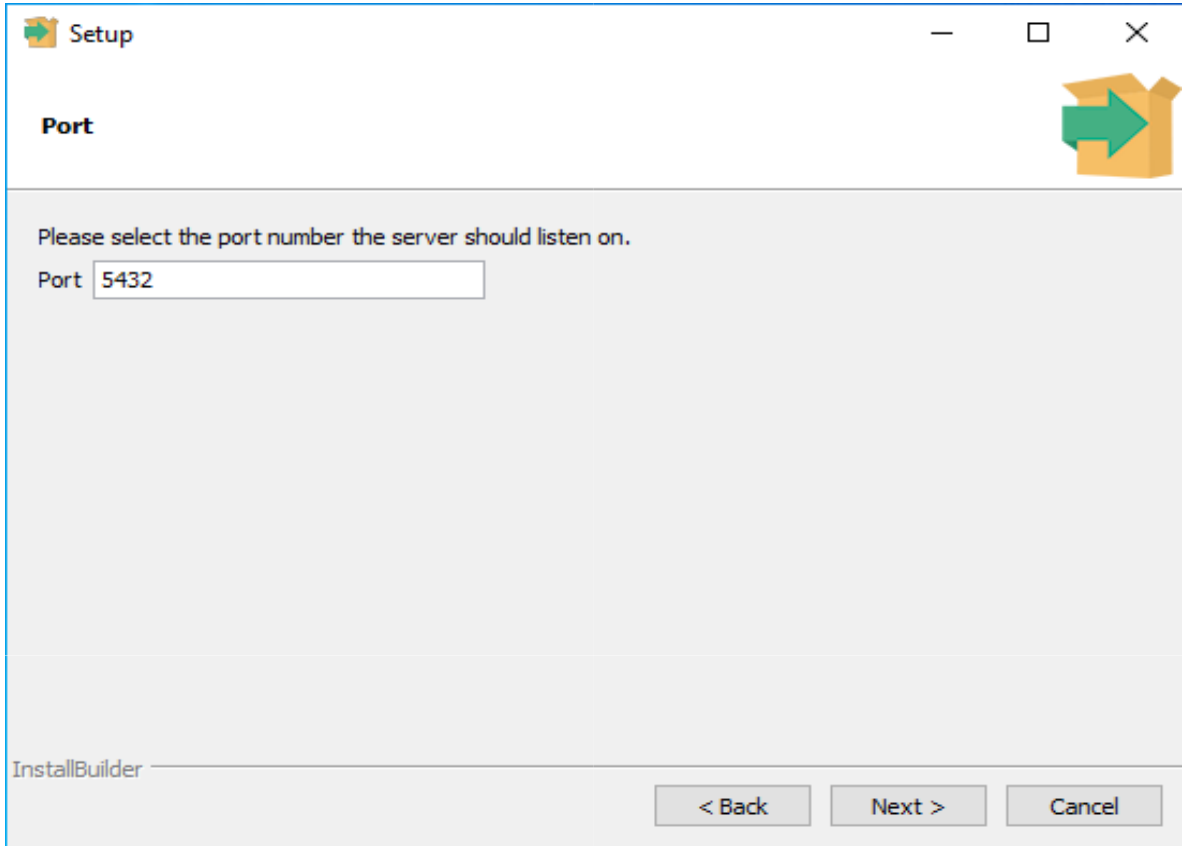
PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
12.1	N/A	N/A	Download	Download	N/A
11.6	N/A	N/A	Download	Download	N/A
10.11	Download	Download	Download	Download	Download
9.6.16	Download	Download	Download	Download	Download
9.5.20	Download	Download	Download	Download	Download
9.4.25	Download	Download	Download	Download	Download
9.3.25 (Not Supported)	Download	Download	Download	Download	Download



- In the PostgreSQL Setup Wizard, in the *Select Components* page, uncheck the components that you do not want to install, or just leave it be.
- If you uncheck anything, don't worry, just launch the installer later and select the component you need, and PostgreSQL will be updated accordingly.

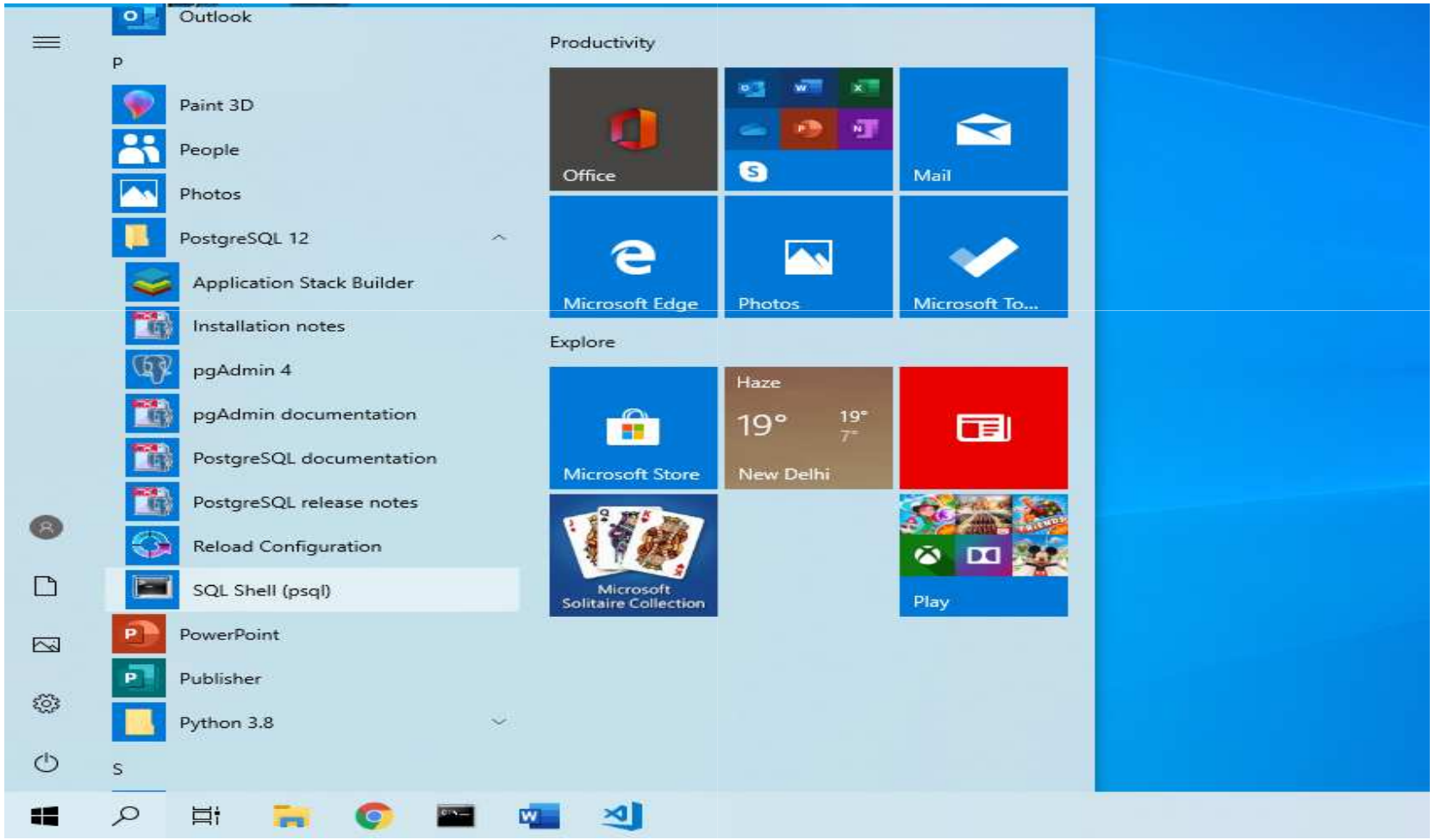
- At the Password page, enter the password for the database superuse (postgres). This account will be used to access your SQL Shell (pqsl later on.



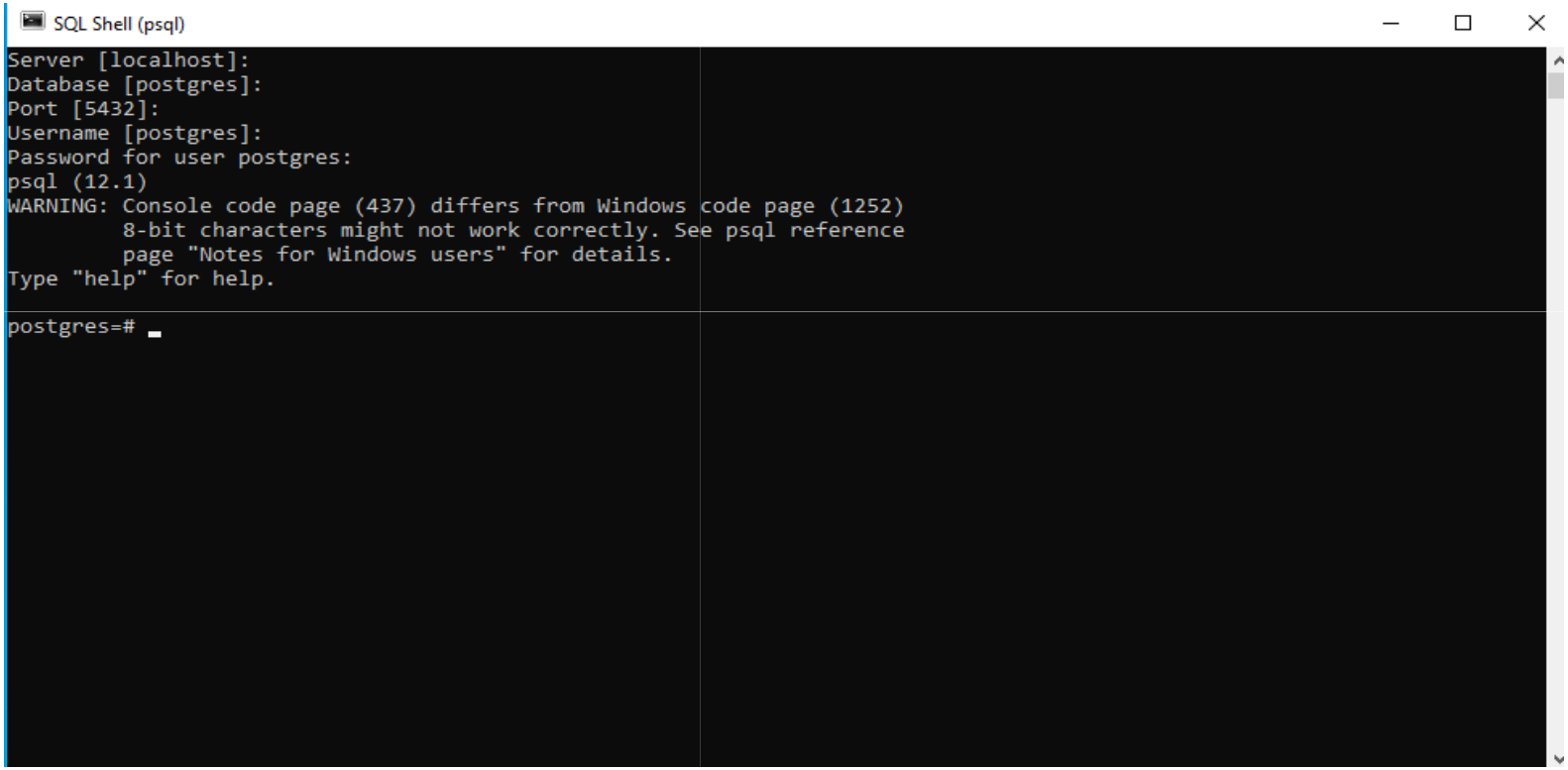


- At the Port page, choose the port number that the server should listen on, or stick to the default 5432. Just make sure that the port is not currently used by any other applications.

- **Proceed with the rest of the installation. To verify the installation, find the SQL Shell (pqsl) program and click on it to launch it. The pqsl command line will appear.**



- Open the command line. Accept the default for the Server, Database, Port, and Username fields by pressing Enter.
- However, at the Password field, you must enter the password that you chose during in the Setup Wizard.



- If you your window is same as the above, then you have successfully installed PostgreSQL!

- **You will be given a PostgreSQL prompt where we can set up our requirements.**

First, create a database for your project:

postgress=# CREATE DATABASE mydatabase;

```
ostgres=# CREATE DATABASE mydatabase;
REATE DATABASE
ostgres=#
```

Note:

Every Postgres statement must end with a semi-colon, so make sure tha your command ends with one if you are experiencing issues.

- Next, create a database user for our project. Make sure to select a secure password:

```
postgress=# CREATE USER myprojectuser WITH PASSWORD 'password';
```

```
ostgres=# CREATE USER myprojectuser WITH PASSWORD '12345';
REATE ROLE
ostgres=#
```

- Afterwards, we'll modify a few of the connection parameters for the use we just created.
- This will speed up database operations so that the correct values do no have to be queried and set each time a connection is established.

We are setting the default encoding to UTF-8, which Django expects.

We are also setting the default transaction isolation scheme to "read committed", which blocks reads from uncommitted transactions.

By default, our Django projects will be set to use UTC. These are all recommendations from the Django project itself:

```
postgress=# ALTER ROLE myprojectuser SET client_encoding TO 'utf8';
```

```
ostgres=# ALTER ROLE myprojectuser SET client_encoding TO 'utf8'
LTER ROLE
ostgres=#
```

postgress=# ALTER ROLE myprojectuser SET default_transaction_isolation
TO 'read committed';

```
LTER ROLE
ostgres=# ALTER ROLE myprojectuser SET default_transaction_isolation TO 'read committed'
LTER ROLE
ostgres=#
```

postgress=# ALTER ROLE myprojectuser SET timezone TO 'UTC';

```
ostgres=# ALTER ROLE myprojectuser SET timezone TO 'UTC';
LTER ROLE
ostgres=#
```

Now, we can give our new user access to administer our new database:

```
postgress=# GRANT ALL PRIVILEGES ON DATABASE mydatabase TO myprojectuser;
```

```
tgres=# GRANT ALL PRIVILEGES ON DATABASE mydatabase TO myprojectuser;
NT
tgres=#
```

When you are finished, exit out of the PostgreSQL prompt by typing:

```
postgress=# \q
```

Installing Django

- We can install Django on our system, for that again we will have to just execute some commands on our system.

pip install django

```
Projects\Myproject>pip install django
llecting django
Downloading https://files.pythonhosted.org/packages/c6/b7/63d23df1e311ca0d90f41352a9efe7389ba353df95deea5676652e615420/Django-3.0.3-py3-none-any.whl (7.5MB)
    |                                | 7.5MB 3.2MB/s
llecting asgiref~=3.2
Downloading https://files.pythonhosted.org/packages/a5/cb/5a235b605a9753ebcb2730c75e610fb51c8cab3f01230080a8229fa36adb/asgiref-3.2.3-py2.py3-none-any.whl
llecting pytz
Downloading https://files.pythonhosted.org/packages/e7/f9/f0b53f88060247251bf481fa6ea62cd0d25bf1b11a87888e53ce5b7c8ad2/pytz-2019.3-py2.py3-none-any.whl (509k
    |                                | 512kB 2.2MB/s
llecting sqlparse>=0.2.2
Downloading https://files.pythonhosted.org/packages/ef/53/900f7d2a54557c6a37886585a91336520e5539e3ae2423ff1102daf4f3a7/sqlparse-0.3.0-py2.py3-none-any.whl
talling collected packages: asgiref, pytz, sqlparse, django
cessfully installed asgiref-3.2.3 django-3.0.3 pytz-2019.3 sqlparse-0.3.0
NING: You are using pip version 19.3.1; however, version 20.0.2 is available.
 should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

- This command will install Django's latest stable version and we will be working on the same.

Setting Up a Virtual Environment

All the necessary requirements have been fulfilled and now we can start our project.

Our first step will be to set up a virtual environment for our project.

To make a virtual environment, go to the directory of your choice via PowerShell and execute this command:

```
virtualenv your_project_name
```

fter the virtual environment has been created it should look like this.

```
Projects\Myproject>virtualenv ENV
ng base prefix 'c:\\users\\hadol\\appdata\\local\\programs\\python\\python38-32'
 python executable in D:\Projects\Myproject\ENV\Scripts\python.exe
talling setuptools, pip, wheel...
e.
```

Now activate virual environment

or that execute this command:

n Windows, virtualenv creates a batch file

\Environment Name\Scripts\activate.bat

```
Projects\Myproject>ENV\Scripts\activate.bat

V) D:\Projects\Myproject>
```

e will also need to install psycopg2. psycopg2 is a package that will allo
ango to use the PostgreSQL database that we just configured. Similarly,
stall, write:

pip install psycopg2

```
) D:\Projects\Myproject>pip install psycopg2
ecting psycopg2
ownloading psycopg2-2.8.4-cp38-cp38-win32.whl (986 kB)
   |                                | 986 kB 1.1 MB/s
talling collected packages: psycopg2
cessfully installed psycopg2-2.8.4
```

Now, it’s time to start a Django project!

Django-admin startproject myproject

```
NV) D:\Projects\Myproject>Django-admin startproject myproject

NV) D:\Projects\Myproject>
```

- We are ready to configure the Django database settings!
- In the project directory, find the file “settings.py”. This file contains the configurations for the app.
- Open a section labelled “DATABASES”. The section should currently look like this:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

- Django is automatically configured with SQLite.

We'll need to change this section to tell Django to use PostgreSQL instead

```
DATABASES = {
'default':
    {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'mydatabase',
        'USER': 'myprojectuser',
        'PASSWORD': '12345',
        'HOST': 'localhost',
        'PORT': '',
    }
}
```

- **The username and password you set there will be used later when we create a superuser for the database. Create the administrative account by typing:**

python manage.py createsuperuser

```
(ENV) D:\Projects\Myproject\myproject>python manage.py createsuperuser
Username (leave blank to use 'amit'): admin
Email address:
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
This password is too common.
This password is entirely numeric.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

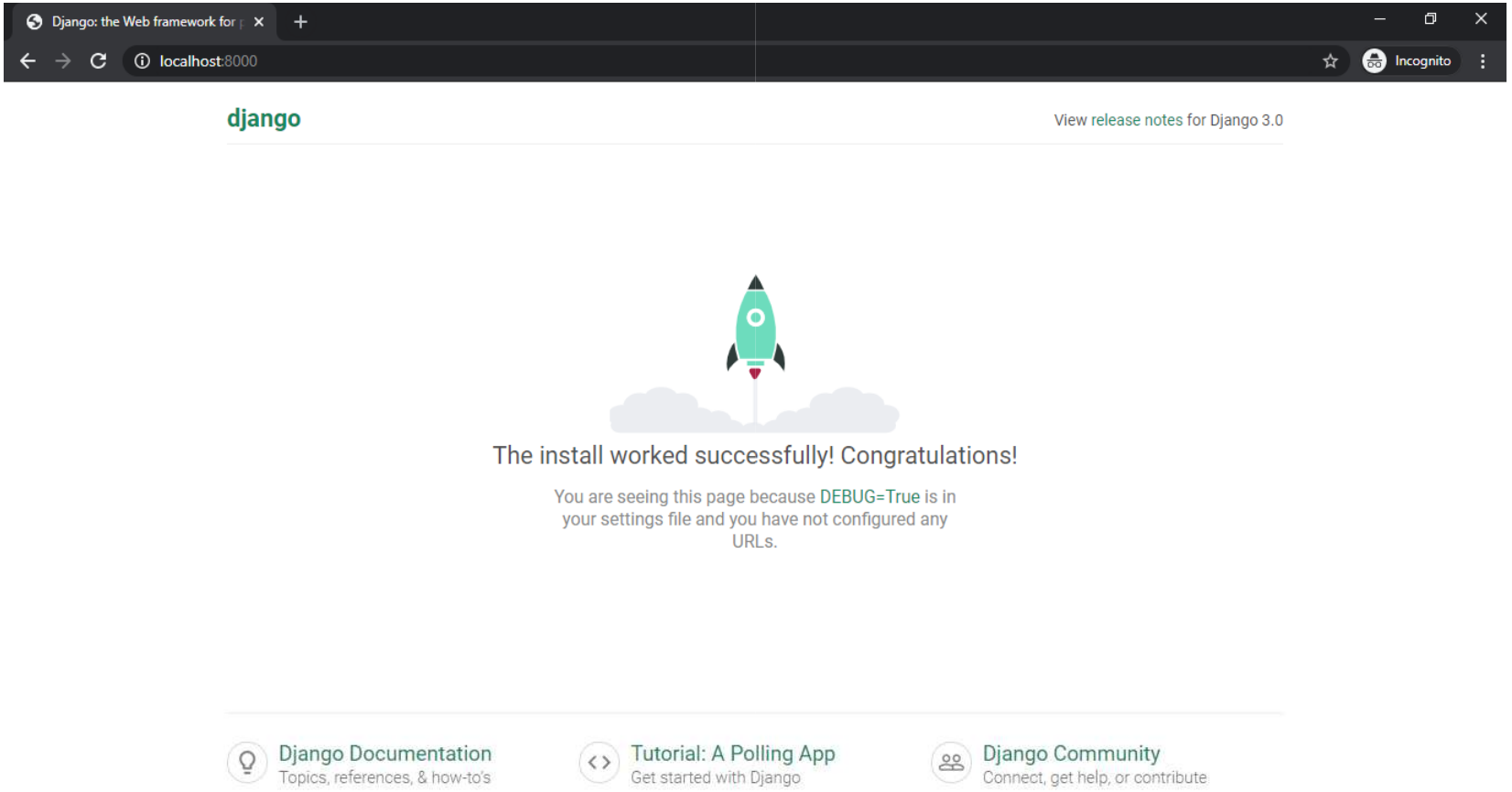
Now that everything is set up, test whether your database is performing by running the Django development server.

```
python manage.py runserver
```

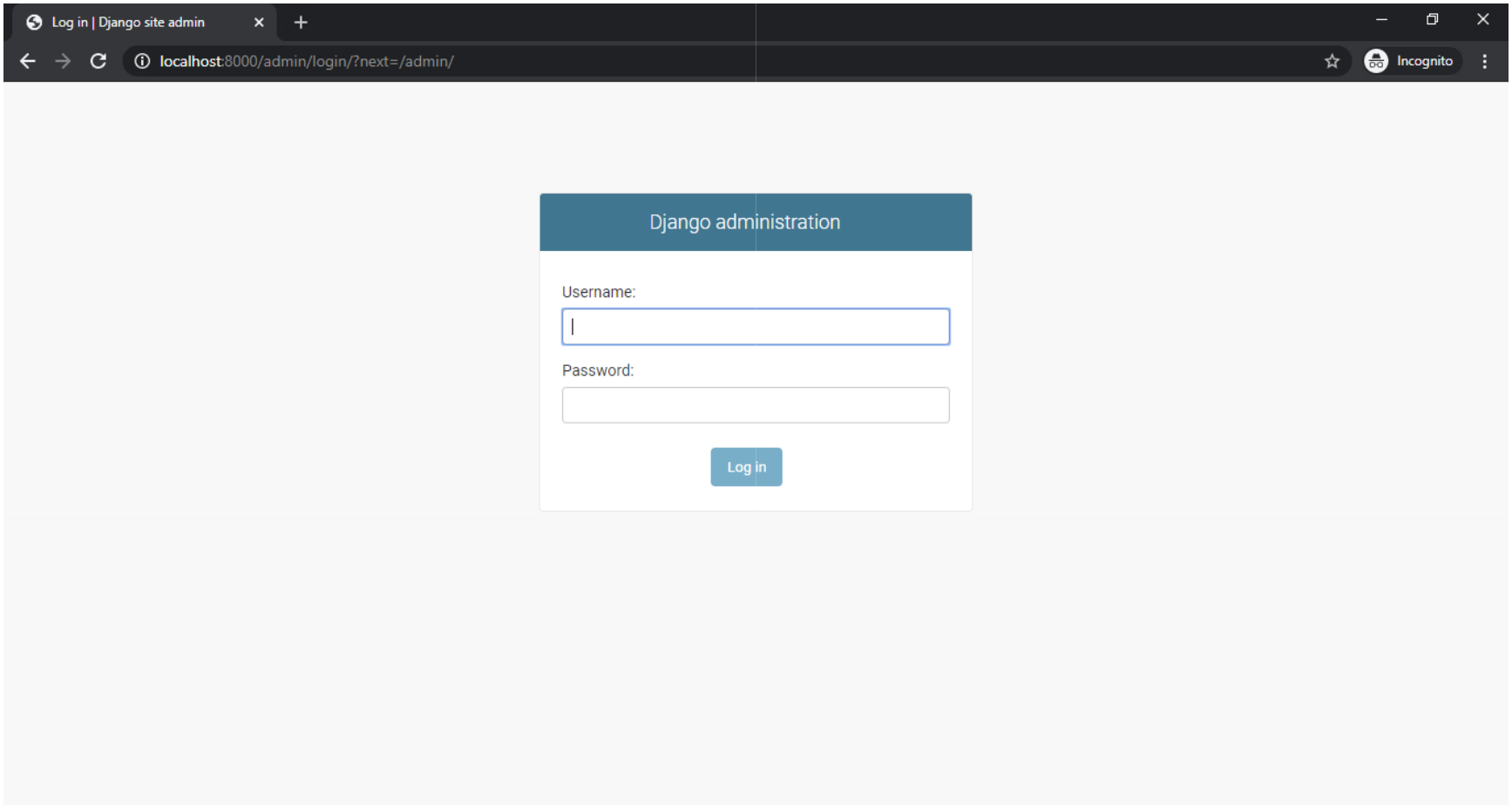
- Test it by going to the development server which can be found on your command prompt.

```
(ENV) D:\Projects\Myproject\myproject>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
February 05, 2020 - 12:23:11
Django version 3.0.3, using settings 'myproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```



Appending /admin to the url will redirect you to the Django administration page where you can log in with the same credentials that you earlier used to create a superuser.



Congratulations! You have set up a Django app with PostgreSQL as the database!