**Cloud Copasi Project Setup Guide**

# Initial Setup

Using Conda framework, start off by setting up the virtual environment. Assuming that Conda framework is installed on your computer. Proceed ahead after installing conda.

1. Create a virtual environment name “CCEnv”

>> conda create --name ccEnv Django

1. Activate the virtual environment name “CCEnv”

>> conda activate ccEnv

1. Check if the Django is installed.

>> django-admin --version

If not, install it using: >> conda install django

# Installations

We need to make the following installations first

1. Python interface to AWS (current version 2.49.0)  
   >> pip install boto
2. Cycler (current version 0.10.0)  
   >> pip install cycler
3. Django-extensions (current version 2.2.9)  
   >> pip install django-extensions
4. LXML (to process xml and html files in python) (current version 4.5.1)  
   >> pip install lxml
5. matplotlib (current version 3.2.1)  
   >> pip install matplotlib
6. Psycopgy2 – It is the most popular PostgreSQL database adapter for python programming language. (current version 2.8.5).  
   >> python -m pip install psycopg2-binary

Or

>> conda install -c anaconda psycopg2

1. Pyparsing – Classes and methods to define and execute parsing grammars. (current version 2.4.7 – already installed).  
   >> pip install pyparsing
2. Python-dateutil – A built-in date time module which is used for manipulating dates and times from simple to complex ways. While this may be enough for a number of use cases, the dateutil module provides powerful extensions to this. (current version 2.8.1).
3. >> pip install python-dateutil
4. pytz – pytz brings the Olson tz database into Python. This library allows accurate and cross platform timezone calculations using Python 2.4 or higher. It also solves the issue of ambiguous times at the end of daylight saving time, which you can read more about in the Python Library Reference (datetime.tzinfo). (already installed 2020.1).

>> pip install pytz

1. six – It provides utility functions for smoothing over the differences between the Python versions with the goal of writing Python code that is compatible on both Python versions. See the documentation for more information on what is provided. (already installed with cycler, current version 1.15.0).

>> pip install six

1. subprocess32 – allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. (current version 3.5.4).

>> pip install subprocess32

1. typing - Type Hints for Python (current version 3.7.4.1)

>> pip install typing

1. Database Installation

Download and Install PostGreSQL database alongwith PGAdmin 4 UI from the following link   
<https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>

Set the password and port as shown below:

Password: password

Port = 5432

# Steps to setup cloud-copasi Django Project

1. Create a Django project named “cloud\_copasi”

>> django-admin startproject cloud\_copasi

Change directory to /cloud\_copasi and run the server

>> python manage.py runserver

1. Change the directory to /cloud\_copasi/cloud\_copasi/ and Create a Django application named “web\_interface”

>> python manage.py startapp web\_interface

1. Creating an app

To start a django app in different directory. follow the following steps

* You need to first create a directory appname (web\_interface) inside /cloud\_copasi/.

>> mkdir cloud\_copasi/web\_interface

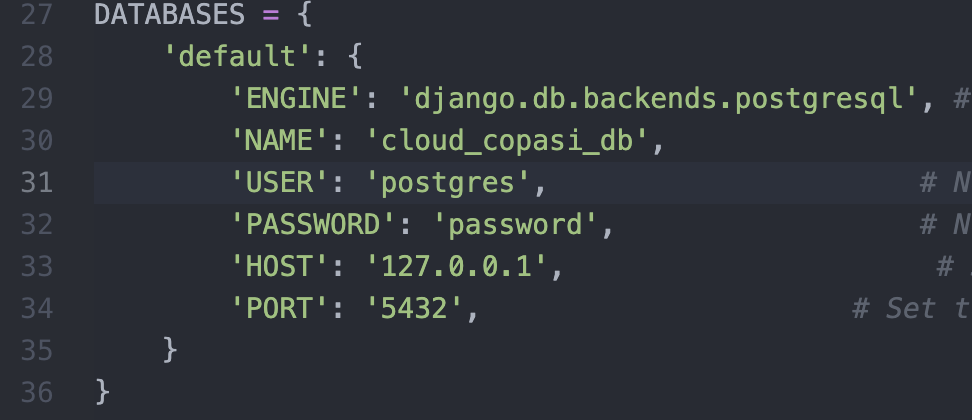
* Then, run the startapp command to create the app.

>> django-admin.py startapp web\_interface ./cloud\_copasi/web\_interface

1. Setting up the settings.py file

Now set the settings.py file according to the settings.py.EXAMPLE comes with the cloud copasi repository.

1. Run PgAdmin4 database UI. Enter password and create a database named “cloud\_copasi\_db”. Select user “postgres”. Also reflect the changes in settings.py file as shown below:

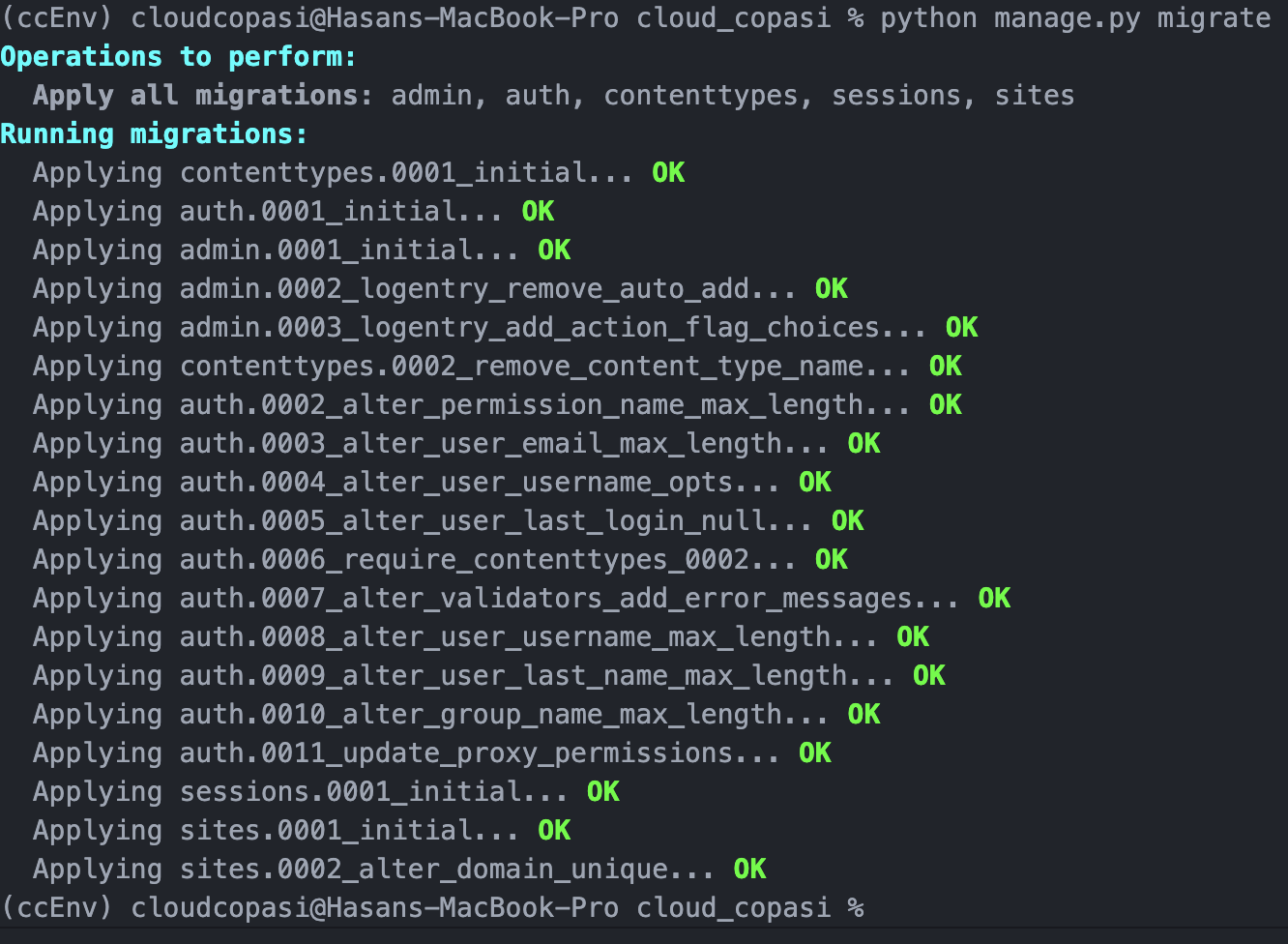


1. Make sure psycopg2 is installed by checking it with the following command. It will show the version of psycopg2

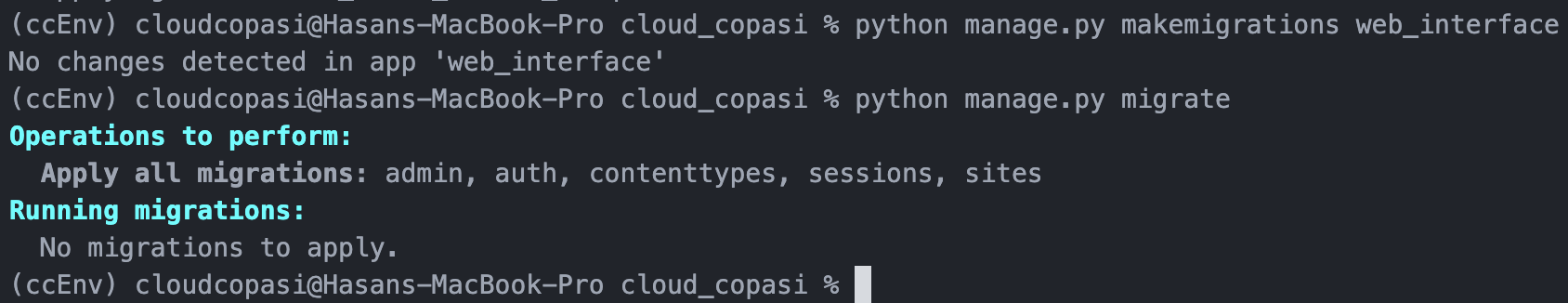
>> pip freeze | grep psycopg2

1. Now migrate the project to see if it is working fine.

>> python manage.py migrate



1. Now perform makemigrations and migrate again as shown below:

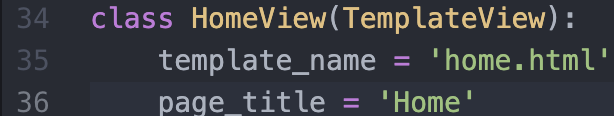


1. Now run the server to verify the changes we have made in settings.py file is not creating any problem. It will only display the default Django webpage at the moment.

# Creating a Webpage, with VIEWs and URLs

## HomeView

Creating a Home page VIEW



1. Application’s (web\_interface) url file



1. Project’s (cloud\_copasi) url file

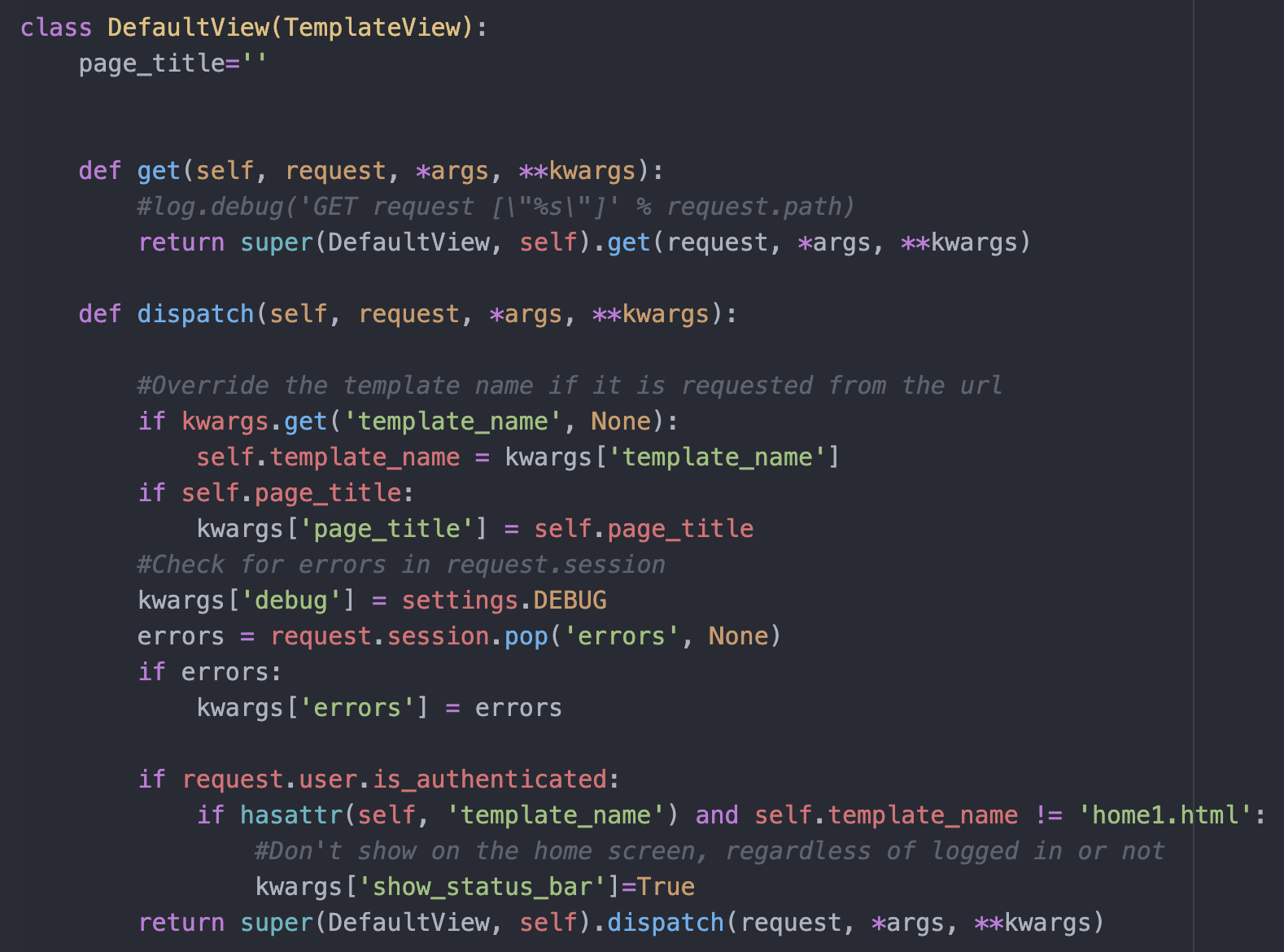


NOTE: Index view is only added for checking purposes.

Notice that the page title does not appear correctly as shown below:



## DefaultView



NOTE: Read more about super() class [here](https://stackoverflow.com/questions/50192463/explain-return-super-getrequest-args-kwargs/50192672).

Now update the HomeView class and inherit the DefaultView in it as follows:

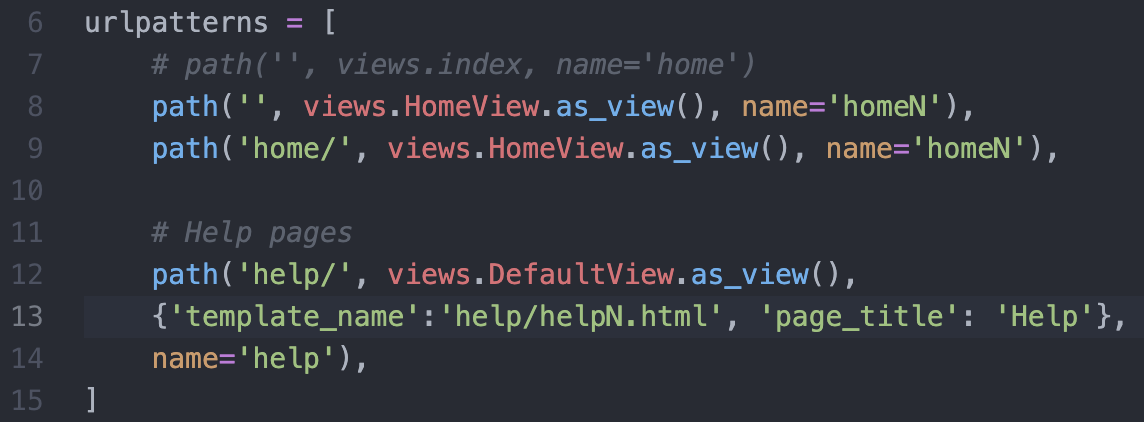
HomeView(DefaultView)

Now update the url files – the project’s and application’s both.

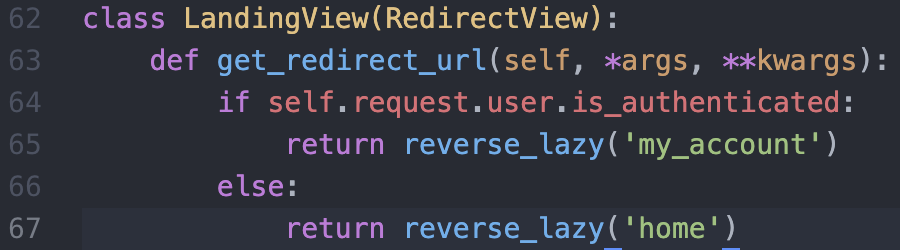
1. Project’s (cloud\_copasi) url file



1. Application’s (web\_interface) url file



## LandingView



NOTE: The reverse\_lazy function is contained with the django.urls module within the Django project code base. This function is actually defined in base.py of the django.urls directory but it is typically imported directly from django.urls, without base in the import module path.

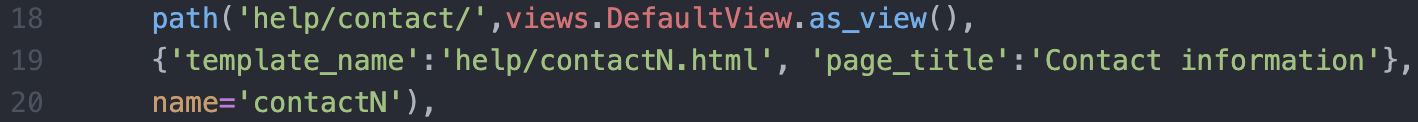
reverse\_lazy is used for resolving Django URL names into URL paths. The resolution is not seen by the end user client as all of this work occurs within the Django application code and framework code.

Now we need to update the application’s (web\_interface) url file.



### Adding other help pages

We now create other html pages of help.

1. Contact page.   
   add the following lines in application’s urls.py file  
     
     
     
   Also enable the page by linking it ({% url ‘helpN’ %}) to the respective <a> anchor in baseN.html file
2. Now add tasks, pools, and terms html pages. Also, add the following lines in application’s url.py file  
   

## Adding Registration form and its associated views

File: cloud\_copasi/web\_interface/account\_viewsN.py

We can optionally add logging capabilities to log the events, using the following lines of code

import logging

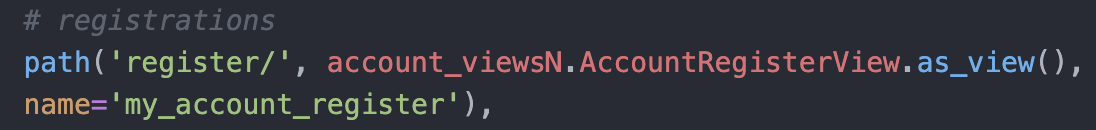
log = logging.getLogger(\_\_name\_\_)

Read more about logging [here](https://data-flair.training/blogs/django-logging/).

1. Add the AccountRegisterForm class  
     
   
2. Add AccountRegisterView class



1. Add respective urls in application’s urls.py file.



## Adding Login and Logout Views

File: cloud\_copasi/web\_interface/views.py

1. Adding view classes



1. Adding urls.