# Cloud Copasi Project Setup Guide

## 1. 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.

a. Create a virtual environment name “CCEnv”
   ```
   >> conda create --name ccEnv Django
   ```

b. Activate the virtual environment name “CCEnv”
   ```
   >> conda activate ccEnv
   ```

c. Check if the Django is installed.
   ```
   >> django-admin --version
   ```
   If not, install it using: `>> conda install django`

## 2. Installations

We need to make the following installations first

a. Python interface to AWS (current version 2.49.0)
   ```
   >> pip install boto
   ```

b. Cycler (current version 0.10.0)
   ```
   >> pip install cycler
   ```

c. Django-extensions (current version 2.2.9)
   ```
   >> pip install django-extensions
   ```

d. LXML (to process xml and html files in python) (current version 4.5.1)
   ```
   >> pip install lxml
   ```

e. matplotlib (current version 3.2.1)
   ```
   >> pip install matplotlib
   ```

f. 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
   ```

g. Pyparsing – Classes and methods to define and execute parsing grammars. (current version 2.4.7 – already installed).
   >> pip install pyparsing

h. 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).

i. >> pip install python-dateutil

j. 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

k. 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

l. 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

m. typing - Type Hints for Python (current version 3.7.4.1)
   >> pip install typing

n. 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

## 3. Steps to setup cloud-copasi Django Project

a. 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

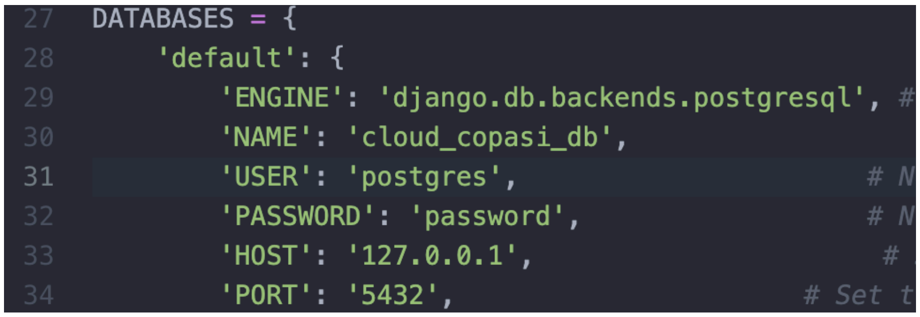
b. Change the directory to /cloud_copasi/cloud_copasi/ and Create a Django application named “web_interface”
   >> python manage.py startapp web_interface

c. 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

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

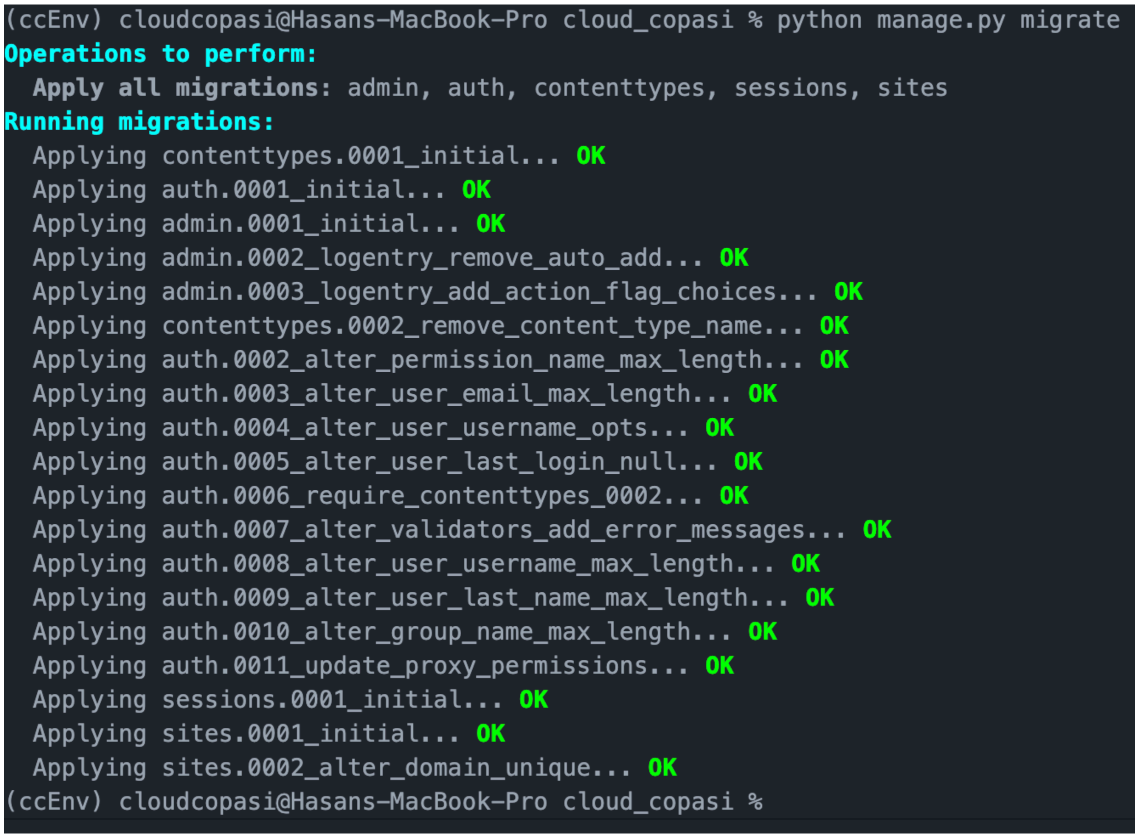
e. Run PgAdmin4 database UI. Enter password and create a database named “cloud_copasi_db”. Select user “postgres”. Also reflect the changes in settings.py file as shown below:



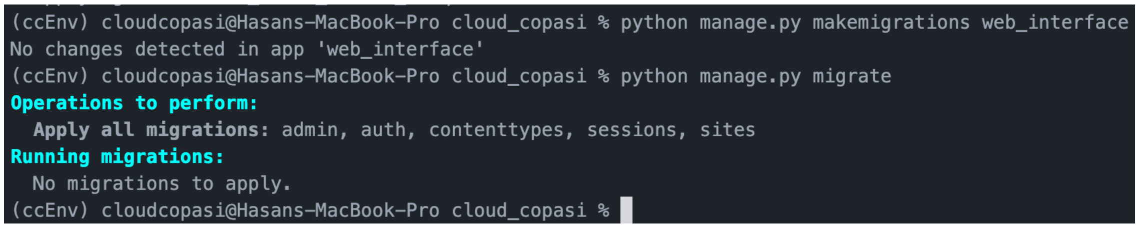
```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql', #
        'NAME': 'cloud_copasi_db',
        'USER': 'postgres',                        # N
        'PASSWORD': 'password',                    # N
        'HOST': '127.0.0.1',                        #
        'PORT': '5432',                          # Set t
```

f. Make sure psycopg2 is installed by checking it with the following command. It will show the version of psycopg2
   >> pip freeze | grep psycopg2

g. Now migrate the project to see if it is working fine.
   >> python manage.py migrate


```
(ccEnv) cloudcopasi@Hasans-MacBook-Pro cloud_copasi % python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, sites
Running 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
  Applying sites.0001_initial... OK
  Applying sites.0002_alter_domain_unique... OK
(ccEnv) cloudcopasi@Hasans-MacBook-Pro cloud_copasi %
```

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

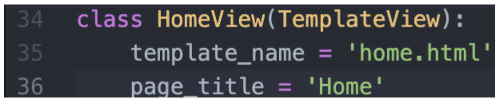

```
(ccEnv) cloudcopasi@Hasans-MacBook-Pro cloud_copasi % python manage.py makemigrations web_interface
No changes detected in app 'web_interface'
(ccEnv) cloudcopasi@Hasans-MacBook-Pro cloud_copasi % python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, sites
Running migrations:
  No migrations to apply.
(ccEnv) cloudcopasi@Hasans-MacBook-Pro cloud_copasi %
```

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

## 4. Creating a Webpage, with VIEWs and URLs

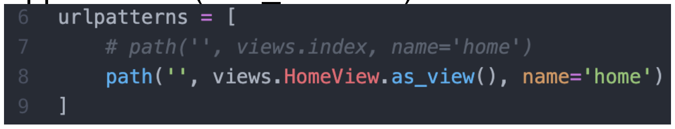
### *i. HomeView*

Creating a Home page VIEW



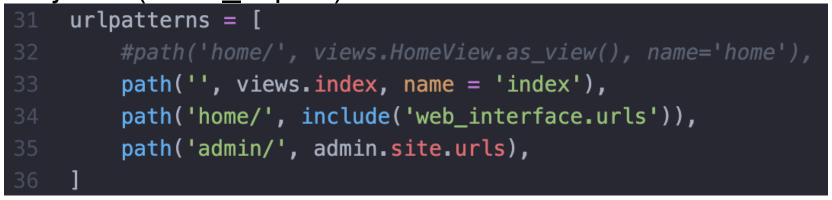
```
class HomeView(TemplateView):
    template_name = 'home.html'
    page_title = 'Home'
```

a. Application's (web_interface) url file



```
urlpatterns = [
    # path('', views.index, name='home')
    path('', views.HomeView.as_view(), name='home')
]
```

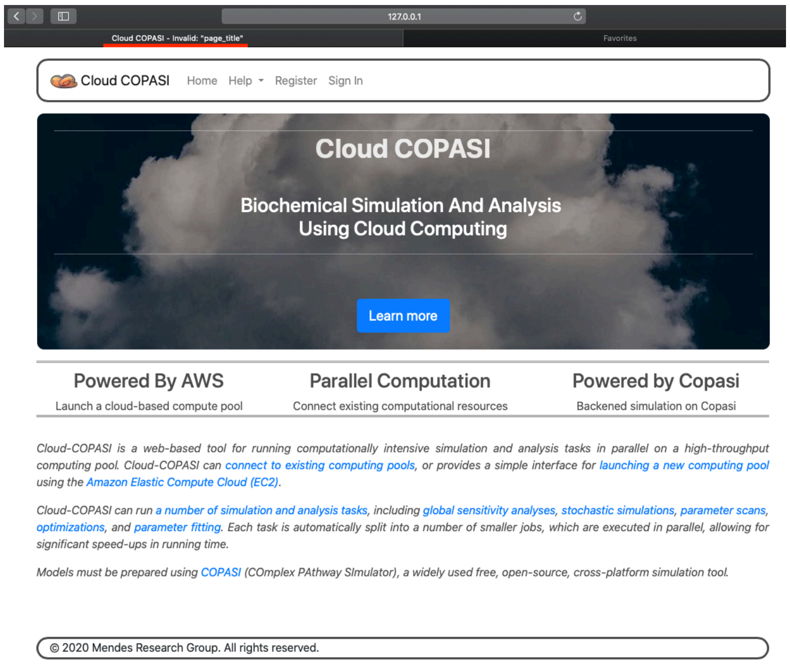
b. Project's (cloud_copasi) url file



```
urlpatterns = [
    #path('home/', views.HomeView.as_view(), name='home'),
    path('', views.index, name = 'index'),
    path('home/', include('web_interface.urls')),
    path('admin/', admin.site.urls),
]
```

NOTE: Index view is only added for checking purposes.

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



127.0.0.1

Cloud COPASI - Invalid: "page_title"

Favorites

Cloud COPASI Home Help Register Sign In

Cloud COPASI

Biochemical Simulation And Analysis Using Cloud Computing

Learn more

Powered By AWS
Launch a cloud-based compute pool

Parallel Computation
Connect existing computational resources

Powered by Copasi
Backened simulation on Copasi

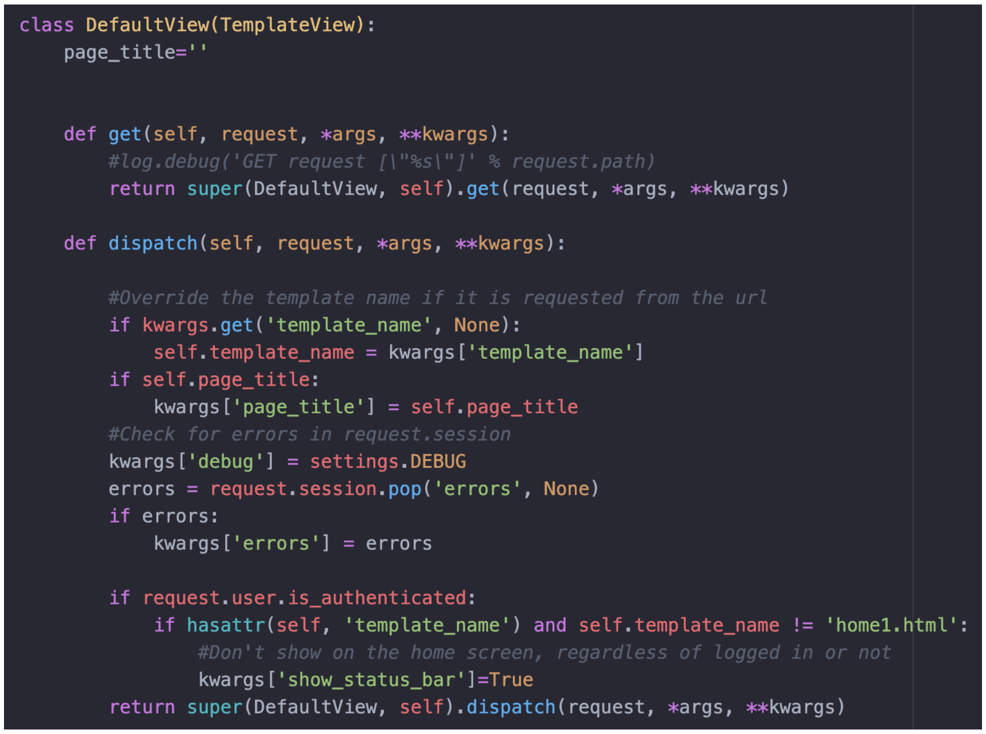
*Cloud-COPASI is a web-based tool for running computationally intensive simulation and analysis tasks in parallel on a high-throughput computing pool. Cloud-COPASI can connect to existing computing pools, or provides a simple interface for launching a new computing pool using the Amazon Elastic Compute Cloud (EC2).*

*Cloud-COPASI can run a number of simulation and analysis tasks, including global sensitivity analyses, stochastic simulations, parameter scans, optimizations, and parameter fitting. Each task is automatically split into a number of smaller jobs, which are executed in parallel, allowing for significant speed-ups in running time.*

*Models must be prepared using COPASI (COmplex PAthway SImulator), a widely used free, open-source, cross-platform simulation tool.*

© 2020 Mendes Research Group. All rights reserved.

## ii. DefaultView



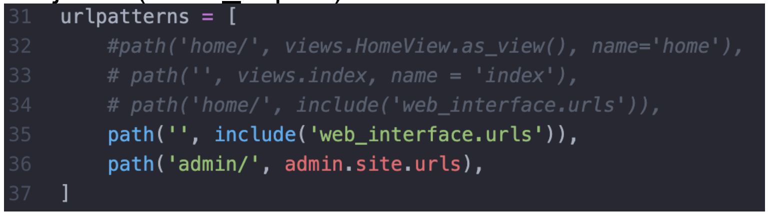
```python
class DefaultView(TemplateView):
    page_title=''


    def get(self, request, *args, **kwargs):
        #log.debug('GET request [\"%s\"]' % request.path)
        return super(DefaultView, self).get(request, *args, **kwargs)

    def dispatch(self, request, *args, **kwargs):

        #Override the template name if it is requested from the url
        if kwargs.get('template_name', None):
            self.template_name = kwargs['template_name']
        if self.page_title:
            kwargs['page_title'] = self.page_title
        #Check for errors in request.session
        kwargs['debug'] = settings.DEBUG
        errors = request.session.pop('errors', None)
        if errors:
            kwargs['errors'] = errors

        if request.user.is_authenticated:
            if hasattr(self, 'template_name') and self.template_name != 'home1.html':
                #Don't show on the home screen, regardless of logged in or not
                kwargs['show_status_bar']=True
        return super(DefaultView, self).dispatch(request, *args, **kwargs)
```

NOTE: Read more about super() class here.

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

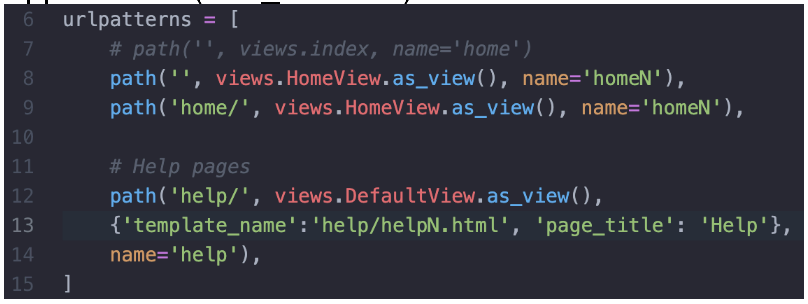
Now update the url files – the project's and application's both.

a. Project's (cloud_copasi) url file



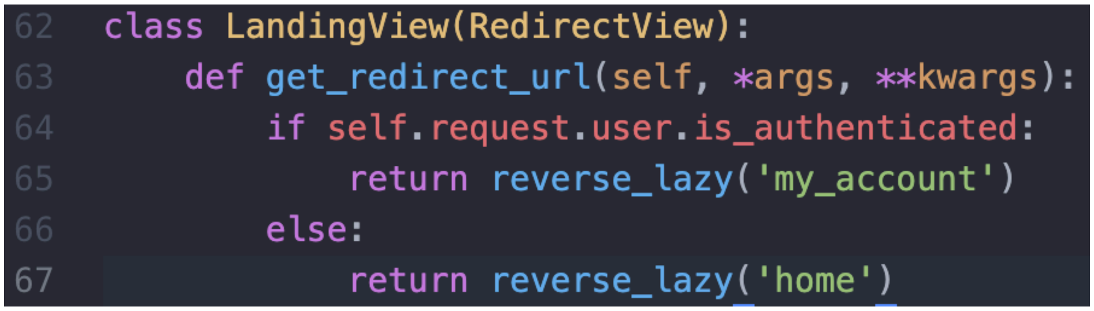
```python
urlpatterns = [
    #path('home/', views.HomeView.as_view(), name='home'),
    # path('', views.index, name = 'index'),
    # path('home/', include('web_interface.urls')),
    path('', include('web_interface.urls')),
    path('admin/', admin.site.urls),
]
```

b. Application's (web_interface) url file



```python
urlpatterns = [
    # path('', views.index, name='home')
    path('', views.HomeView.as_view(), name='homeN'),
    path('home/', views.HomeView.as_view(), name='homeN'),

    # Help pages
    path('help/', views.DefaultView.as_view(),
    {'template_name':'help/helpN.html', 'page_title': 'Help'},
    name='help'),
]
```

### *iii. LandingView*

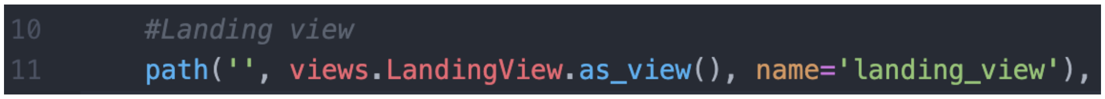


```
class LandingView(RedirectView):
    def get_redirect_url(self, *args, **kwargs):
        if self.request.user.is_authenticated:
            return reverse_lazy('my_account')
        else:
            return reverse_lazy('home')
```

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.

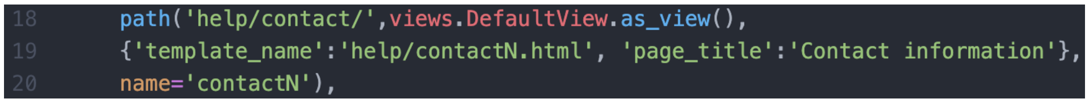


```
    #Landing view
    path('', views.LandingView.as_view(), name='landing_view'),
```

## Adding other help pages

We now create other html pages of help.

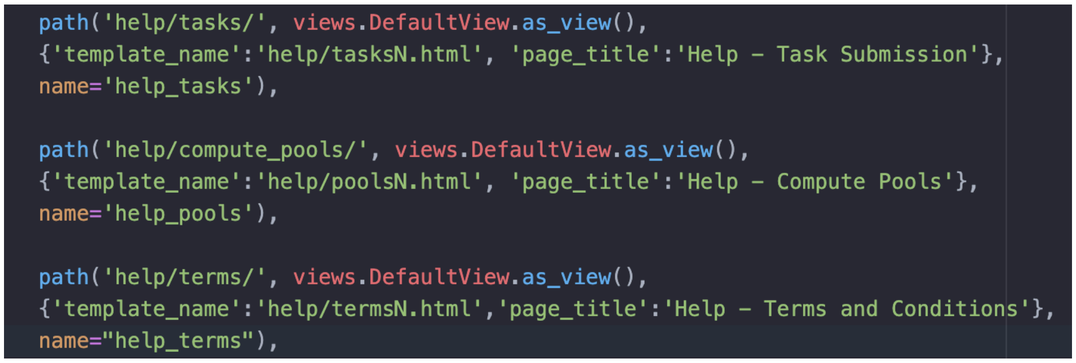
a. Contact page.
   add the following lines in application's urls.py file



```
    path('help/contact/',views.DefaultView.as_view(),
    {'template_name':'help/contactN.html', 'page_title':'Contact information'},
    name='contactN'),
```

   Also enable the page by linking it ({% url 'helpN' %}) to the respective <a> anchor in baseN.html file

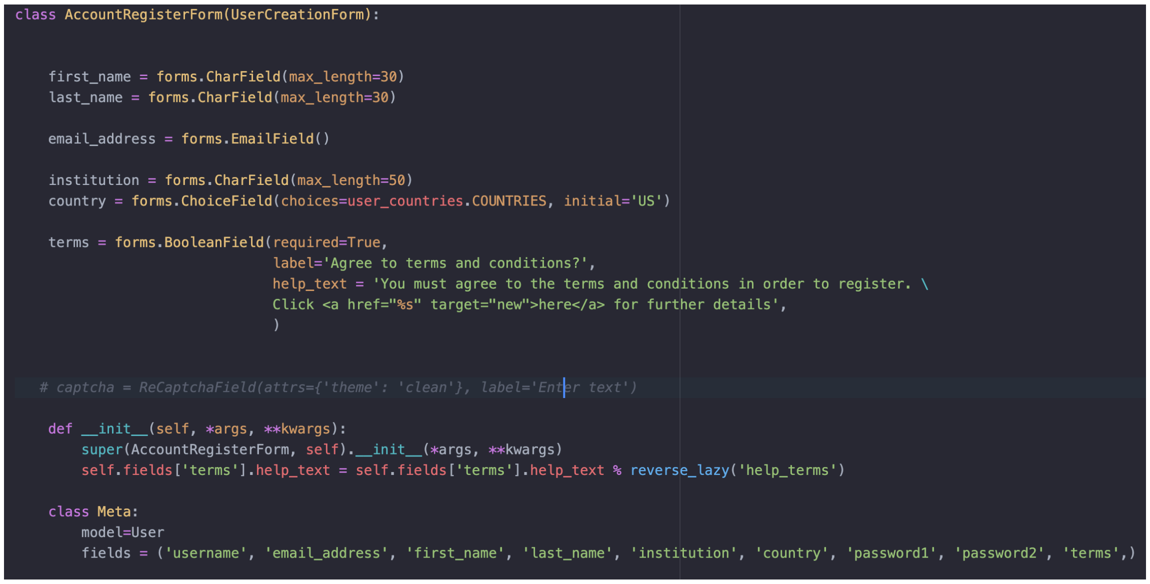
b. Now add tasks, pools, and terms html pages. Also, add the following lines in application's url.py file



```
path('help/tasks/', views.DefaultView.as_view(),
{'template_name':'help/tasksN.html', 'page_title':'Help - Task Submission'},
name='help_tasks'),

path('help/compute_pools/', views.DefaultView.as_view(),
{'template_name':'help/poolsN.html', 'page_title':'Help - Compute Pools'},
name='help_pools'),

path('help/terms/', views.DefaultView.as_view(),
{'template_name':'help/termsN.html','page_title':'Help - Terms and Conditions'},
name="help_terms"),
```

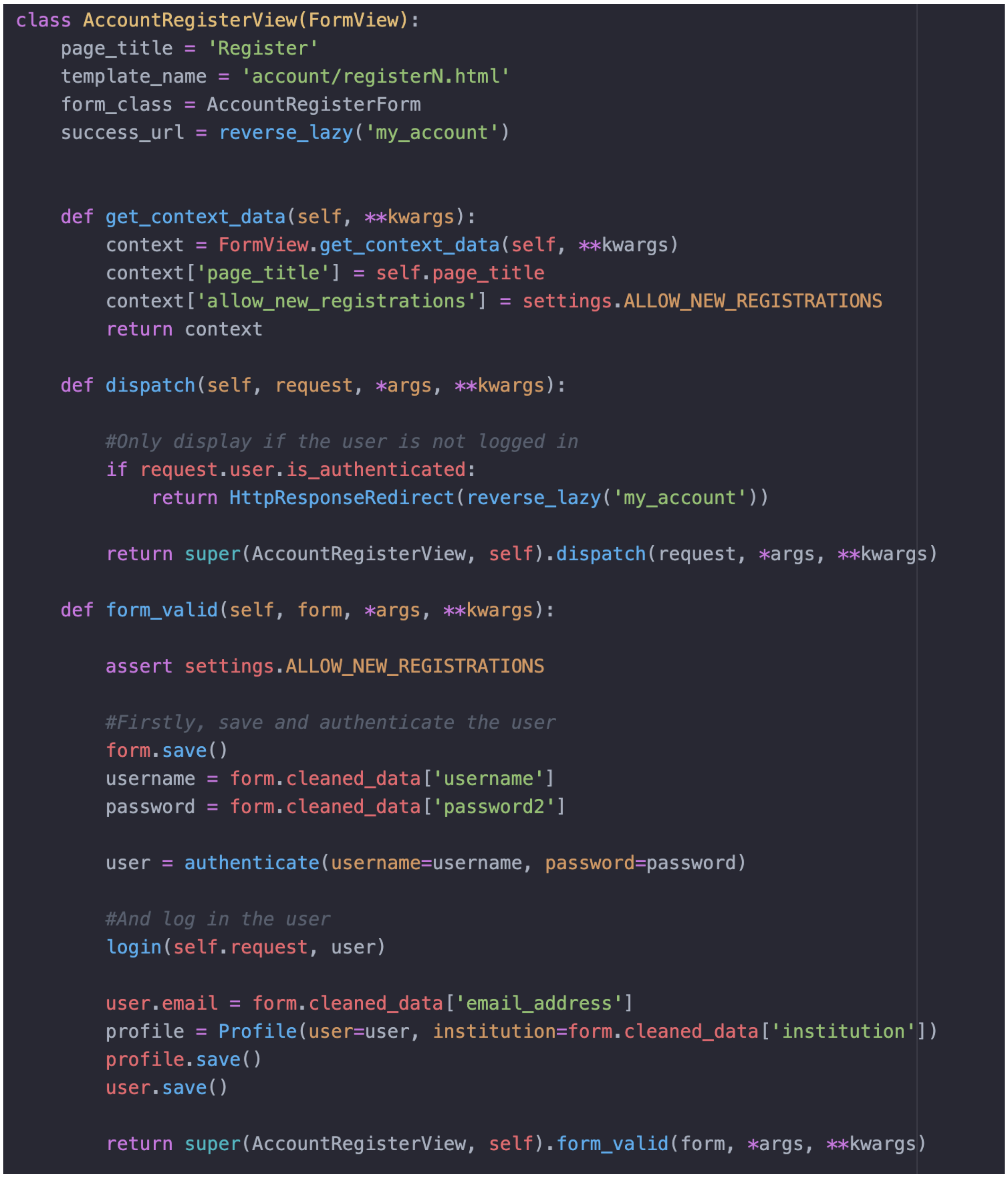
## iv. *Adding Registration form and its associated views*

a. Add the AccountFormView class



```
class AccountRegisterForm(UserCreationForm):


    first_name = forms.CharField(max_length=30)
    last_name = forms.CharField(max_length=30)

    email_address = forms.EmailField()

    institution = forms.CharField(max_length=50)
    country = forms.ChoiceField(choices=user_countries.COUNTRIES, initial='US')

    terms = forms.BooleanField(required=True,
                               label='Agree to terms and conditions?',
                               help_text = 'You must agree to the terms and conditions in order to register. \
                               Click <a href="%s" target="new">here</a> for further details',
                               )


   # captcha = ReCaptchaField(attrs={'theme': 'clean'}, label='Enter text')

    def __init__(self, *args, **kwargs):
        super(AccountRegisterForm, self).__init__(*args, **kwargs)
        self.fields['terms'].help_text = self.fields['terms'].help_text % reverse_lazy('help_terms')

    class Meta:
        model=User
        fields = ('username', 'email_address', 'first_name', 'last_name', 'institution', 'country', 'password1', 'password2', 'terms',)
```

b. Add AccountRegisterView class



```python
class AccountRegisterView(FormView):
    page_title = 'Register'
    template_name = 'account/registerN.html'
    form_class = AccountRegisterForm
    success_url = reverse_lazy('my_account')


    def get_context_data(self, **kwargs):
        context = FormView.get_context_data(self, **kwargs)
        context['page_title'] = self.page_title
        context['allow_new_registrations'] = settings.ALLOW_NEW_REGISTRATIONS
        return context

    def dispatch(self, request, *args, **kwargs):

        #Only display if the user is not logged in
        if request.user.is_authenticated:
            return HttpResponseRedirect(reverse_lazy('my_account'))

        return super(AccountRegisterView, self).dispatch(request, *args, **kwargs)

    def form_valid(self, form, *args, **kwargs):

        assert settings.ALLOW_NEW_REGISTRATIONS

        #Firstly, save and authenticate the user
        form.save()
        username = form.cleaned_data['username']
        password = form.cleaned_data['password2']

        user = authenticate(username=username, password=password)

        #And log in the user
        login(self.request, user)

        user.email = form.cleaned_data['email_address']
        profile = Profile(user=user, institution=form.cleaned_data['institution'])
        profile.save()
        user.save()

        return super(AccountRegisterView, self).form_valid(form, *args, **kwargs)
```

c.