Optional: Create a Standalone Django ORM Project Template

In this lab, you will create a standalone Django ORM project which you can use a template for building further Django ORM apps.

Estimated time needed: 20 minutes

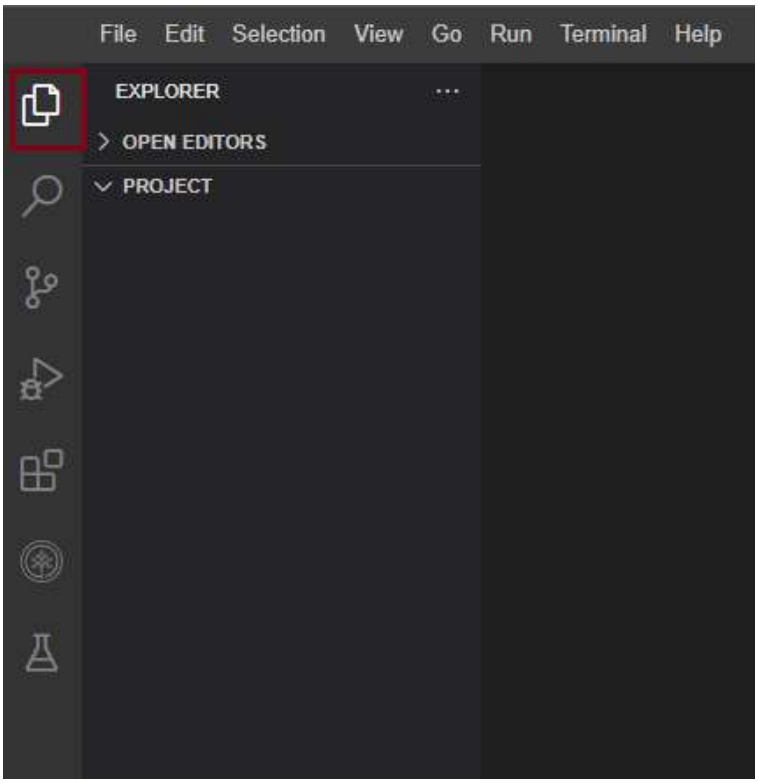
Learning Objectives

- Create a standalone Django ORM project and app
- Save this project as a template to learn and build more complex Django ORM apps

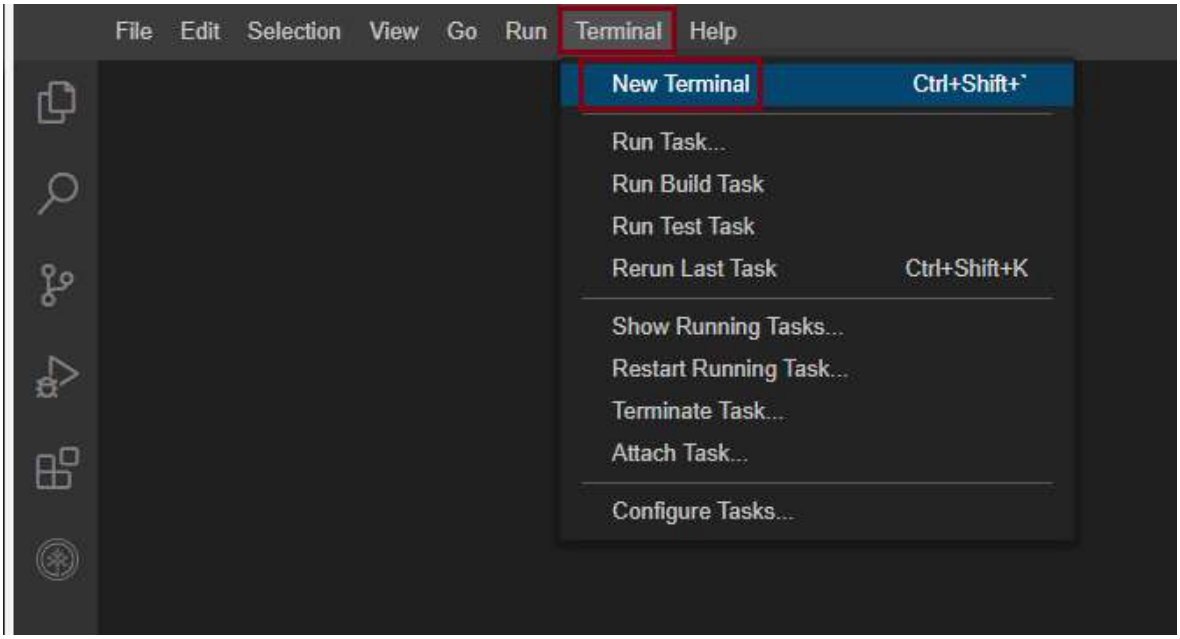
Working with files in Cloud IDE

If you are new to Cloud IDE, this section will show you how to create and edit files, which are part of your project, in Cloud IDE.

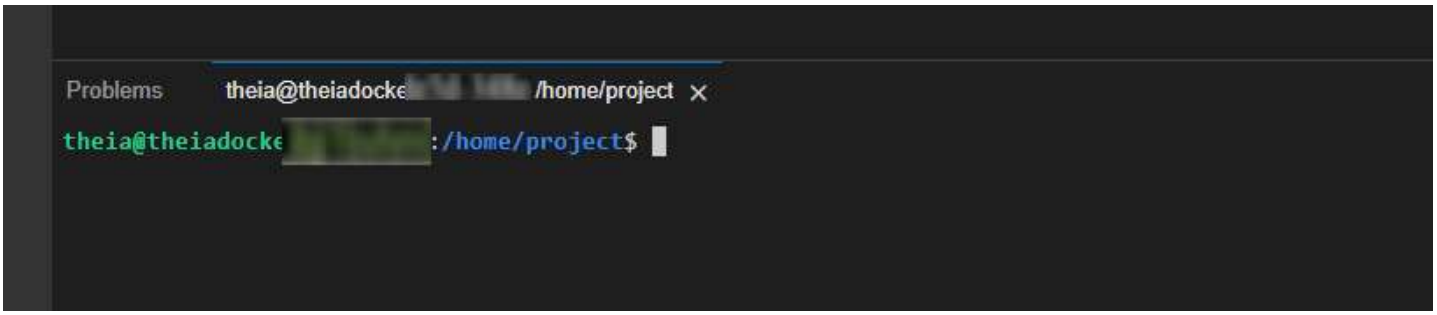
To view your files and directories inside Cloud IDE, click on this files icon to reveal it.



Click on `New`, and then `New Terminal`.



This will open a new terminal where you can run your commands.



Concepts covered in the lab

1. `Django ORM`: A Python ORM component of the Django web application framework, where each Django model maps to a database table.
2. `PostgreSQL` or `Postgres`: An open-source relational database management system used by Django.
3. `Psycopg`: An interface used by Django for working with PostgreSQL.
4. `Database migrations`: The management of version-controlled, incremental and reversible changes to relational database schemas to update or revert the schema to a newer or older version.
5. `manage.py`: A command-line interface for managing a Django project.

Start PostgreSQL in Theia

If you are using the Theia environment hosted by Skills Network Labs, a pre-installed `PostgreSQL` instance is provided for you which can be started with one simple command line.

You can skip this step if you have already started it in previous labs.

- If the terminal was not open, go to `Terminal > New Terminal` and run:

```
1. 1
1. start_postgres
```

Copied!

- Once PostgreSQL is started, you can check the server connection information in the terminal. You need to save the connection information such as generated `username`, `password`, and `host`, etc, which will be used to configure the Django app to connect to this database.

```
1
2
3
4
5
6
7
8
9
10
```

```
Starting your Postgres database....
This process can take up to a minute.

Postgres database started, waiting for all services to be ready....

Your Postgres database is now ready to use and available with username: postgres password: Nzg3Mi15bHVvLTIz

You can access your Postgres database via:
 Browser at: https://yluo-5050.theiadocker-1.proxy.cognitiveclass.ai
 Command Line: psql --username=postgres --host=localhost
```

Copied!

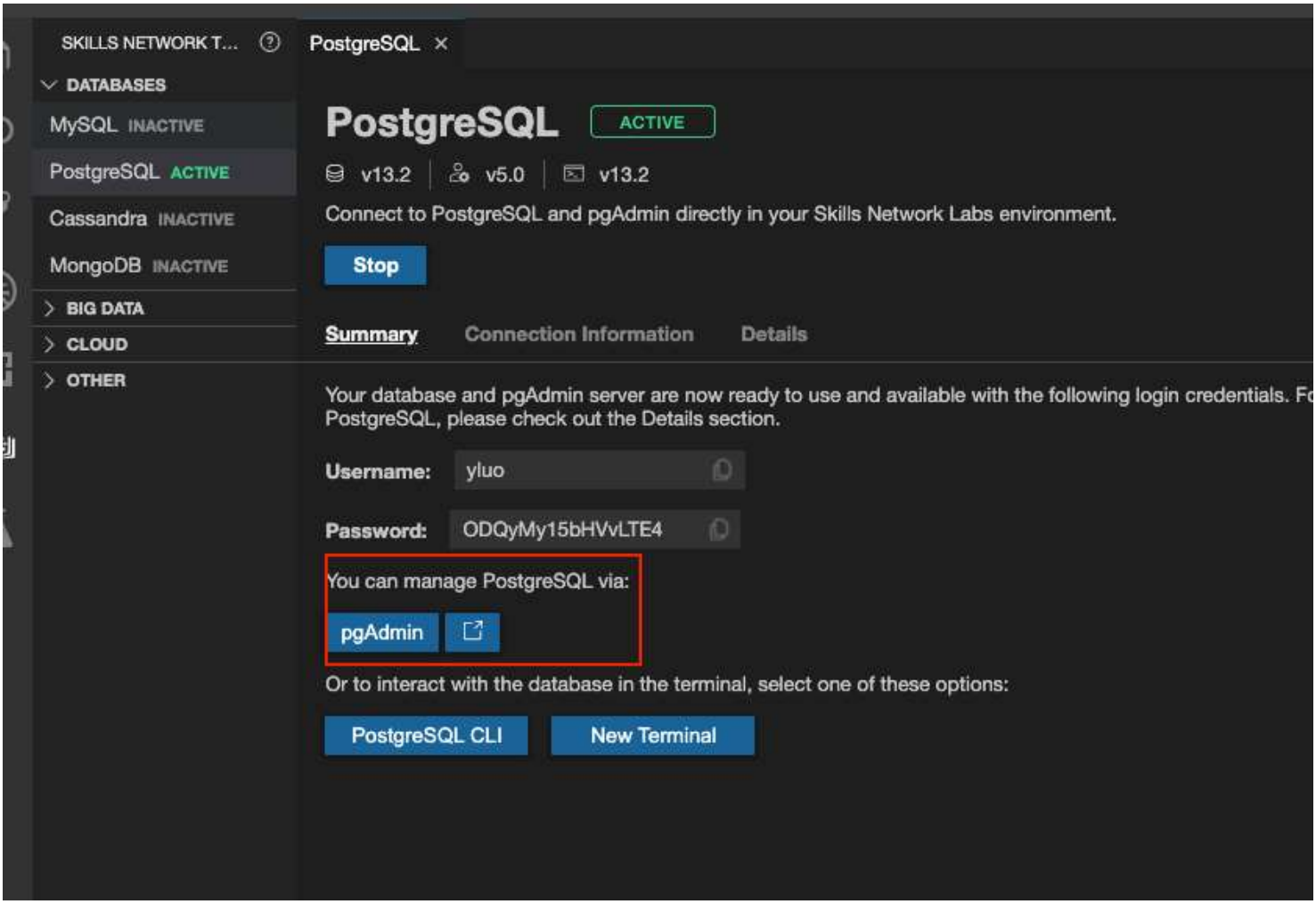
- Install these must-have packages before you setup the environment to access postgres.

```
1
2
```

```
pip install --upgrade distro-info
pip3 install --upgrade pip==23.2.1
```

Copied! Executed!

- A `pgAdmin` instance is also installed and started for you.



Create a Simplified Django ORM Project

If the terminal was not open, go to `Terminal > New Terminal` and make sure your current Theia directory is `/home/project`.

In this lab, instead of creating a complete Django web project and app using command-line utilities,you will be creating a simplified app with only ORM component from scratch.

Within the ORM-only app, you can define your models and easily perform CRUD operations on your model objects. More importantly, you can create and run Python script files to do that instead of typing Python code into shell line by line.

Let's start by creating an empty project folder

- In the Theia menu, click `File > New Folder` and type `ormtemplate` which acts as the container folder for an empty Django project.
- Right-click the `ormtemplate` folder, and create a subfolder `standalone` which acts as the container folder for an empty Django app called `standalone`
- Right-click the `ormtemplate` folder again, and create an empty `settings.py` file and a `manage.py` file

Next, we will add the content to `manage.py` file acting as a command-line interface managing our Django project

- Open the empty `manage.py` file, and copy and paste the following code snippet

```
1. 1
2. 2
3. 3
4. 4
5. 5
```

```
6
7
8
9
10
11
12
13
14
15
16
17
18
```

```
import os
import sys

if __name__ == "__main__":
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "settings")
    try:
        from django.core.management import execute_from_command_line
    except ImportError:
        try:
            import django
        except ImportError:
            raise ImportError(
                "Couldn't import Django. Are you sure it's installed and "
                "available on your PYTHONPATH environment variable? Did you "
                "forget to activate a virtual environment?"
            )
        raise
    execute_from_command_line(sys.argv)
```

Copied!

The code snippet be added to the `manage.py` file as the setting module of our `ormtemplate` project and be able to execute Django built-in commands such as migrations.

Next, let's add a simple database settings to `settings.py`

- Open the empty `settings.py`, copy and paste the following code snippet

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

```
# PostgreSQL
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'postgres',
        'USER': 'postgres',
        'PASSWORD': 'Place with your password generated in Step 1',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}

INSTALLED_APPS = (
```

```
    'standalone',
)

SECRET_KEY = 'SECRET KEY for this Django Project'
```

Copied!

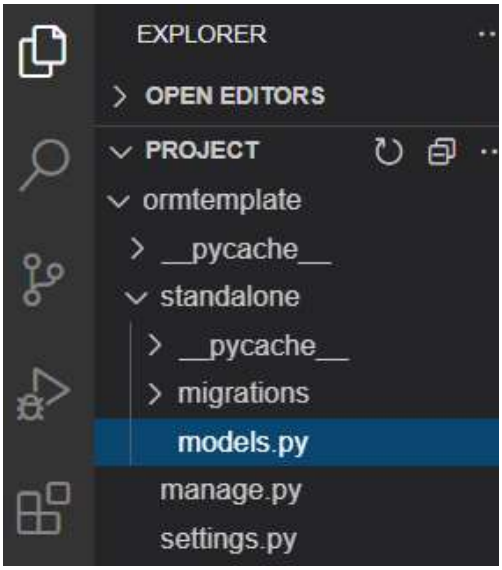
The above code snippet adds `standalone` app as an installed app and adds the default database to be the pre-installed PostgreSQL we created in Step 1.

You just need to update the `PASSWORD` field to be the password generated in Step 1.

So far we have created a very simple Django project. Next let's add content to our `standalone` app.

- Click `ormtemplate/standalone` folder and create an empty `models.py` containing model definitions and a folder named `migrations` containing migration scripts in `standalone` app.

After that, your app structure should look like the following:



- Now your Django `standalone` app is ready and you can start test if the `standalone` app is working.

Test the standalone App

- Open `orm/models.py` file and copy and paste the following code snippet to define a simple `Test` model

```
1
2
3
4
5
```

```
from django.db import models

# Test model
class Test(models.Model):
    name = models.CharField(max_length=30)
```

Copied!

Next, we can ask `ormtemplate` app to generate the `Test` table by running migration command-lines.

- `cd` into `ormtemplate` folder

```
1
```

```
cd ormtemplate
```

Copied!

and now your current Theia folder should be `/home/project/ormtemplate` shown in the terminal

Let's set upa a virtual environment which will contain all the packages we need.

```
1
2
3
4
5
```

```
pip install virtualenv

virtualenv djangoenv

source djangoenv/bin/activate
```

Copied!

Install the required packages.

```
1
```

```
pip install django==4.2.4 psycopg2-binary==2.9.7
```

Copied! Executed!

- Then, you need to generate migration scripts for `standalone` app

```
1
```

```
python3 manage.py makemigrations standalone
```

Copied!

and you should see migration scripts generated for your `Test` model

```
1
2
3
```

```
Migrations for 'standalone':
  standalone/migrations/0001_initial.py
    - Create model Test
```

Copied!

Note, if you see errors like

`django.db.utils.OperationalError: FATAL: password authentication failed for user "postgres"`
please re-run `start_postgres` to reset the `PostgreSQL` server and use the new password in `settings.py`.

- and run migration

```
1
```

```
python3 manage.py migrate
```

Copied! Executed!

Next, you can write some Python testing code in a Python script file (`*.py`) to test your model.

- Click the `ormtemplate` folder and create a new file called `test.py`
- Open the empty `test.py` and add following code snippet to test your `test` model:

```
1
2
3
```

```
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
```

```
# Django specific settings
import inspect
import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "settings")
from django.db import connection
# Ensure settings are read
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
# Your application specific imports
from standalone.models import Test

# Delete all data
def clean_data():
    Test.objects.all().delete()

# Test Django Model Setup
def test_setup():
    try:
        clean_data()
        test = Test(name="name")
        test.save()
        # Check test table is not empty
        assert Test.objects.count() > 0
        print("Django Model setup completed.")
    except AssertionError as exception:
        print("Django Model setup failed with error: ")
        raise(exception)
    except:
        print("Unexpected error")

test_setup()
```

Copied!

The above code snippet first cleans the database and then inserts a test object.
Then it checks if the test object was inserted correctly.

- At last, in the terminal, run the `test.py`:

```
1
```

```
python3 test.py
```

Copied! Executed!

and you should see

Django Model setup completed.

Now you have successfully created a standalone Django ORM app and tested it with a simple `Test` model. You also get yourself familiar with Django app structure by creating them from scratch.

Next, you could download and save this project locally as a template for your future learning and Django ORM development activities.

- Right-click the root folder `ormtemplate`, and click `Download` to save your workspace.

For your practice, you could import it to a new Theia environment or your local Python environment as a starting point to develop more complex Django ORM models.

Summary

In this lab, you have created a standalone Django ORM app without creating a full Django web project. You could use this simple ORM app as a template for you to learn Django ORM as well as build more complex Django ORM apps in the future.

Author(s)

Yan Luo