BIT2008A/ITEC2000A: Multimedia Data Management
Fall 2022

Lab 8

Django Models

Please take the following steps:

i) Create a sample Django project. If you are using Docker, you can use the template of Week 6 as a starting point.

ii) Add an app to your project by executing the following line in your terminal.

```
python manage.py startapp flights
```

On docker, you can go to *Containers* tab, and open a terminal like the image below suggests.

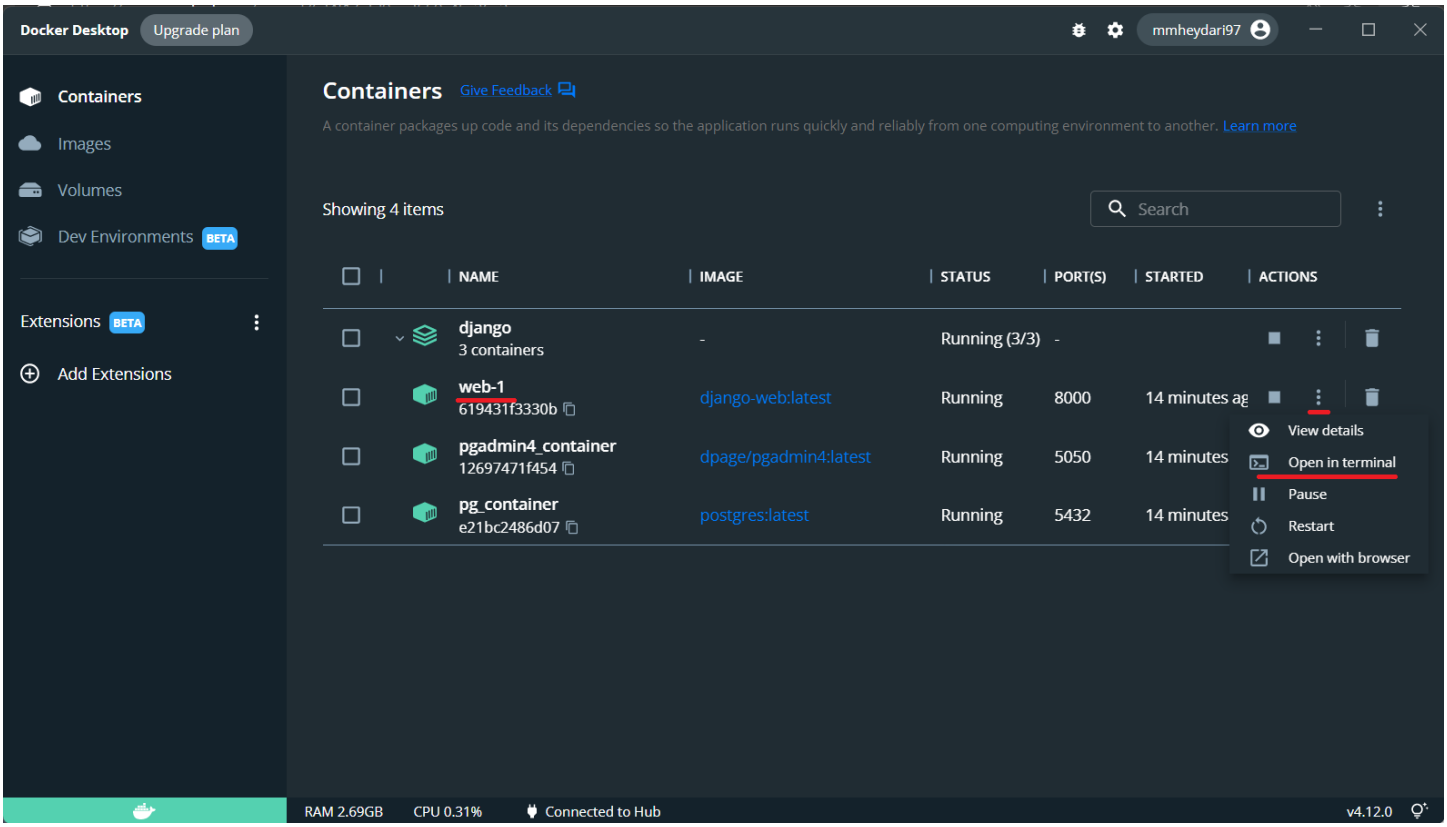

Figure 1. open terminal to run Django commands.

iii) go to *settings.py* and add flights app to your installed apps.

iv) go to *models.py* in your flights app and add the following class:

- Flight (origin, destination, duration). The first two attributes should be of *VARCHAR* equivalent type in Django, and duration should be **INTEGER** equivalent.

v) Execute the commands below to reflect new changes to your database:

```
python manage.py makemigrations
python manage.py migrate
```

vi) Now, register your *settings.py* file then navigate your terminal to the interactive Python environment by executing:

```
export DJANGO_SETTINGS_MODEUL=composeexample.settings
ipython
```

vii) In that environment you can execute Python code in real-time. first set up Django using:

```
import django
django.setup()
from flights.models import Flight
```

viii) In this environment, we can add new instances to our models that are directly reflected to the database. Try to create a new instance of the class *Flight* with $origin =$ "$NewYork$", $destination =$ "$London$", and $duration = 415$. After that you can call *save()* method to store it in your database.

ix) Change the string representation of Flight class by overriding *__str__* method and make it return this value for the instance you just created:
{id}: {New York} to {London}

x) Save a few more Flight instances of your choice, then try to print them all. you have access to those instances via *objects* property of the class. When you want to get all of them you can simply call *all()* method.

xi) Now, create another class called Airport with the schema as below:
- Airport (code, city)

Here, the code has three alphabetical characters. For instance, YUL stands for Montreal Pierre Elliott Trudeau International Airport. Also, change the string representation of Airport class to:
{city} ({code})

xii) Afterward, change Flight model to take two Airports as its origin and destination. To update the table of Flights after deleting each airport, all of the records involved should be removed.

xiii) Django makes it possible to use foreign keys reversely. For instance, you should be able to find all of the Flights departing or arriving to a certain airport. To do so you should use an input argument called *related_name* on the foreign key attribute. Assign departures and arrivals to origin and destination respectively. Before applying those changes to your database, make sure that you removed all of the rows from Flight class using *delete()* method.

xiv) Now create some Airport instances listed below: You should choose informative variable names for

code	city
JFK	New York
LHR	London
CDG	Paris
NRT	Tokyo

the airports because we want to use them for flight instances, too.

xv) Now create some Flight instances as listed below: Now, use *arrivals* to list all of the flights.

origin	destination	duration
JFK	LHR	415
JFK	CDG	435

xvi) On the instance that you created for *JFK* airport, use arrivals property to check how the reverse foreign key relation works.

xvii) You can also mimic SQL queries using Django *filter()* method on the QuerySets of each Model. To test it, try to select all of the airport instances in "New York" city. Alternatively, you can use *get* method in your python code but it produces an error when facing none or many instances with the criteria that you mention as input arguments.

xviii) Django has an intuitive way of handling many-to-many relationships. Create another class for passengers in the models.py which has first_name, last_name, and flights attributes. Here flights is a *ManyToManyField* that stands for the relationship between the passengers and flights. This field also supports *related_name* input argument. give it the value *passengers* to use it reversely.

xix) After applying the latest changes to your database, register your models to the admin panel using *admin.py*, then create a superuser through the terminal if you haven't already done that.

```
1        python manage.py createsuperuser
```

xx) Now, populate your *Passenger* model via admin panel.

xxi) Try to extract the passengers of each flight by using the reverse *ManyToManyField*.

References

This tutorial was prepared using the topics *Django Models*, *Migrations*, *Shell*, and *Many-to-Many Relationships* of this video. It is a good practice to go through it completely!