Creating a Web Application in Python using Django

Topics

- Python – Working with virtual environments
- Creating a Web Application in Python using Django

Prerequisite

- This tutorial assumes that you are using the AWS Cloud9 Environment that you have already set up for developing in Python. Otherwise please complete the section *Setting up Cloud9 for Cloud Application Development in Python* of the Tutorial on AWS Cloud9 for Python.
- Ensure that you have installed pip for the version of Python you are using! (see Appendix: Verifying pip version & Installing pip for details).

You MUST ensure that

- **!!!You use the AWS resources responsibly!!!**

  To prevent ongoing charges to the AWS account after you're done with this tutorial, **STOP** the EC2 instance that hosts your AWS Cloud9 environment.

Note In this document, all the commands that you should run from the terminal are prefixed by the '$' character.

```
$ command
```

1 Python – Working with virtual environments

Virtual environments allow us to manage separate package installations for different projects. For more information please consult https://packaging.python.org/guides/installing-using-pip-and-virtual-environments/

1. Create a virtual environment: to create a virtual environment, navigate (by using the command *cd*) to your project's directory and run the command *venv* at the terminal. The following command will create the virtual environment named env.

   ```
   $ python3 -m venv env
   ```

2. Activate a virtual environment: before we can start installing or using packages in the virtual environment we have to activate the virtual environment we created, by running the following command at the terminal

   ```
   $ source env/bin/activate
   ```

   Note that once the virtual environment is activated the terminal's prompt is prefixed by the name of your virtual environment (i.e. (env) in this example).

3. Deactivate the virtual environment: when you want to exit the virtual environment (for example when you finish a working session, or would like to work on another project) run the following command at the terminal

   ```
   $ deactivate
   ```

4. Each time you want to enter again the virtual environment for your project follow step 2. Note that typically we run only once *item 1* per project, and as many times needed *item 2* and *item 3*.

Freezing dependencies

We can export a list off all installed packages and their versions by running the command *pip freeze* at the terminal

```
$ pip freeze
```

2 Creating a Web Application in Python using Django

In this tutorial you will create a simple web application in Python. The guidelines presented in this tutorial are similar with and based on Django's Official Tutorial[1]. For more information please consult the following resources:

- Writing your first Django app, part 1 https://docs.djangoproject.com/en/4.2/intro/tutorial01/
- Writing your first Django app, part 2 https://docs.djangoproject.com/en/4.2/intro/tutorial02/
- Writing your first Django app, part 3 https://docs.djangoproject.com/en/4.2/intro/tutorial03/
- Create a new directory/folder using the *mkdir* command, and then navigate into this directory using the *cd* command. That folder will contain your Django project.

```
$ mkdir simple_proj
$ cd simple_proj
```

- Recall that in Python, it's a good approach to work with virtual environments. Create a virtual environment

```
... simple_proj $  python -m venv env
```

- Activate the virtual environment

```
... simple_proj $  source env/bin/activate
```

- Use pip to install Django by running the following command at the terminal

```
(env) ... simple_proj $  pip install django==2.1.15
```

[1] https://docs.djangoproject.com/en/4.2/intro/

- Use the *django-admin startproject* command to create a Django project. Run the following command at the terminal to create a project, in this example, named *demoproj*

```
(env) ... simple_proj $  django-admin startproject demoproj
```

The above command will create a folder named *demoproj* in your current folder (in this example, the current folder is *simple_proj*). The folder *demoproj* has the following structure

```
demoproj/
    manage.py
    demoproj/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

- Navigate to the parent directory of your project (i.e. by using the *cd* command)

```
(env) ... simple_proj $  cd demoproj/
```

- Run the server for your Django project by specifying also the port to be able to preview it in Cloud 9 (see *Appendix: Django – preview your application in Cloud9*)

```
(env) ... demoproj  $  python manage.py runserver 8080
```

Use two terminals!!! i.e. in one terminal leave the server running, and in another terminal run the commands required to further develop your project.

2.1 Create a movies Django application

Create a movies application

```
(env) ... demoproj  $  python manage.py startapp movies
```

The above command will create a folder named *movies* within your Django project with the following structure

```
movies/
    migrations/
        __init__.py
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    views.py
```

2.1.1 Views and URL Conf

When a user accesses a web application's functionalities, the user accesses the application via browser by specifying the URL of that particular application. The functionalities are made available via views (i.e. a specific type of web page). To obtain the view that corresponds to a particular URL, Django uses URL configurations knows as *URLconfs*. A URLconf maps URL patterns to views.

- Let's update the *views.py* file to create the first view. Update the *movies* → *views.py* file with the following content

  ```
  from django.http import HttpResponse

  # Create your views here.
  def index(request):
      return HttpResponse("You're at the movies index.")
  ```

- Create a URLconf in the *movies* folder, by creating a file named *urls.py* in the *movies* folder

```
(env) ... demoproj  $  touch movies/urls.py
```

- Update the content of the *movies* → *urls.py* file with the following content

```
from django.urls import path

from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

- Next, update the root URLconf to point to the *movies.urls* module. Update the file *demoproj* → *urls.py*, by adding an import for *django.urls.include* and inserting an *include()* in the *urlpatterns* list. The updated *demoproj* → *urls.py* file should look similar with the following content

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('movies/', include('movies.urls')),
    path('admin/', admin.site.urls),
]
```

- Check that the view (i.e. index) you created is working by starting the server (if it's not already started) and by appending, in the browser, to the URL of your application the suffix */movies*

```
(env) ... demoproj  $  python manage.py runserver 8080
```

  - **Note**: If you get a *Page not found (404)* error, ensure that the URL does have the suffix */movies/* i.e. the URL, if you are using Cloud 9 for the development of your application, should look similar with *https://....vfs.cloud9.eu-west-1.amazonaws.com/movies/*

2.1.2 Database

Next, initialize your Django project local database by running the next command at the terminal. Note that when the *migrate* command is run, *migrate* looks at the INSTALLED_APPS list from the *settings.py* file and creates any necessary database tables according to the migrations that exist for those applications.

```
(env) ... demoproj  $  python manage.py migrate
```

2.1.3 Models

- Include the *movies* app in our Django project, namely we have to add a reference to the *movies* configuration class in the INSTALLED_APPS list of the *settings.py* file. The *movies* configuration class is *MoviesConfig* class, which is in the *movies* → *apps.py* file, therefore its dotted path is 'movies.apps.MoviesConfig'. Update the *settings.py* file from the Django project with the dotted path 'movies.apps.MoviesConfig' by adding it to the INSTALLED_APPS list. The INSTALLED_APPS of the *settings.py* should look similar with the following

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

- Next, let's define a model in the application. This will represent a table in the database. A model is represented in Python by a class which inherits from django.db.models.Model. Update the *movies* → *models.py* file to define a *Movie* model with the class members as shown next

```
from django.db import models

# Create your models here.

class Movie(models.Model):
   # each class variable represents a database i.e. table field in the model
    title = models.CharField(max_length=200)
    director = models.CharField(max_length=30)
    release_date = models.DateTimeField('release date')
    genre = models.CharField(max_length=200)
    duration = models.FloatField()
```

> Note that all the models for the movies application will be defined as classes in the *same models.py* file.

- As we updated the models, we have to run the following command from the terminal so that the changes are stored as a migration.

```
(env) ... demoproj $ python manage.py makemigrations movies
```

- Next, run the *migrate* command to create the model table in the database:

```
(env) ... demoproj $ python manage.py migrate
```

  - Next, let's add a method (i.e. __str__() method) to the Movie model to provide a representation of a Movie object as this will be used by the Django's automatically-generated admin. Update the Movie model (i.e. the Movie class) from the *movies* → *models.py* file with the representation you'd like to have for that object. Let's assume that we'd like to have the following implementation for the (i.e. __str__() method):

```
def __str__(self):
    return self.title + " - " + self.director
```

  - The updated *models.py* file from the *movies* folder should look similar with the following code

```
from django.db import models

# Create your models here.
class Movie(models.Model):
# each class variable represents a database i.e. table field in the model
    title = models.CharField(max_length=200)
    director = models.CharField(max_length=30)
    release_date = models.DateTimeField('release date')
    genre = models.CharField(max_length=200)
    duration = models.FloatField()

    def __str__(self):
        return self.title + " - " + self.director
```

Note that the migrations enables us to change the models of an application over time, as we develop our project, without having to manually delete the database and/or tables and create new database and/or tables. Each time we want to create or update a particular table for an application we have to perform the following steps

1. Update the models, by updating the *models.py* file of that application.
2. Run from the terminal the following command to create migrations for those updates

   ```
   $  python manage.py makemigrations
   ```

3. Run from the terminal the following command to apply the updates to the database

   ```
   $  python manage.py migrate
   ```

2.1.4 Django Admin

One can create an administrator site for their Django project to access the admin console directly from the web application. Recall that the admin site is provided by the *admin* application which is listed as *django.contrib.admin* in the *INSTALLED_APPS* list of the *settings.py* file of the Django project.

- Initialize your Django application's local database if you haven't already done so

  ```
  (env) ... demoproj  $  python manage.py migrate
  ```

- Create an admin user (i.e. the user who can log in to the admin site) by running the following command. *Note that you'll be prompted to provide the username, email and the password for the admin user.*

  ```
  (env) ... demoproj  $  python manage.py createsuperuser
  ```

- The admin site is accessed by appending, in the browser, to the URL of your application the suffix */admin/*. Recall that the web server has to be running.

  ```
  (env) ... demoproj  $  python manage.py runserver 8080
  ```

  - **Note**: If you get a *Page not found (404)* error, ensure that the URL does have the suffix */admin/* i.e. the URL, if you are using Cloud 9 for the development of your application, should look similar with *https://....vfs.cloud9.eu-west-1.amazonaws.com/admin/*

- If you'd like to enable one of the applications from your Django project to be updated from the admin site, you have to register the specific models or model of that application with the admin site. In order to do that, you have to update the *admin.py* file from that application's folder. In this example, we will update the *admin.py* file from the *movies* folder to register the *Movie* model with the admin application, such that there is an admin interface for the *Movie* objects. Update the *movies* → *admin.py* file with the following content

```
from django.contrib import admin

# Register your models here.
from .models import Movie
admin.site.register(Movie)
```

- Refresh the application in the browser, and now you should see that the admin user can also perform CRUD (create, read, update, delete) operations on Movie objects from the admin site. For example, try to create a movie, and then modify its details. **Note** that the form is automatically generated from the Movie model (i.e. that you defined earlier in the *movies* → *models.py* file.

2.1.5 Views and templates

In Django, the web page are provided by views. Each view is implemented as a Python function (or method).

It's a good practice to separate the design of a web page from the actual content displayed on that page. To achieve this, we can use the template system available in Django, by creating a template (i.e. a .html page) that is used by a view (i.e. the Python function). A template[2] contains the static parts of the HTML output and the code to insert the dynamic data. By convention, Django will look for a *templates* folder in each of the applications mentioned in the *INSTALLED_APPS* list of the *settings.py* file.

- In this example we create templates for the movies application, therefore we will create a *templates* folder in the *movies* folder.

```
(env) ... demoproj $ mkdir movies/templates
```

- Using the Django conventions, we have to create another folder within the previously created *templates* folder with the very same name as the name of the application for which we want to create those templates. In our example, we have to create a folder named *movies* as we create templates for the *movies* application

```
(env) ... demoproj $ mkdir movies/templates/movies
```

[2] https://docs.djangoproject.com/en/4.2/topics/templates/

***index* view and corresponding template used by the view**

- In the *movies* → *templates* → *movies* folder create a template named *index.html* (i.e. this will be the template used by the *index* view defined in the file *views.py* of the *movies* folder).

```
(env) ...  demoproj  $  touch movies/templates/movies/index.html
```

  - Please note that within the Django project we can identify the above template with *movies/index.html*

- Update the *index* view from *movies* → *views.py* file; the updated *index* view will use the *index.html* template we created earlier. The index view will compute the newest 15 movies (if there are any).

```
from django.shortcuts import render

from .models import Movie


def index(request):
    newest_movies = Movie.objects.order_by('-release_date')[:15]
    context = {'newest_movies': newest_movies}
    return render(request, 'movies/index.html', context)
```

- Next, update the *index.html* to display the newest movies, or a corresponding message if there are no movies.

```
<h1>Newest Movies<h1>

{% if newest_movies %}
    <ul>
    {% for movie in newest_movies %}
        <li><a href="/movies/{{ movie.id }}/">{{ movie.title }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>There are no movies recorded.</p>
{% endif %}
```

Next, let's create a view to display the details of a specific movie.

***show* view and corresponding template used by the view**

- In the *movies* → *templates* → *movies* folder create a template named *show.html* (i.e. this will be the template used by the *show* view defined in the file *views.py* of the *movies* folder).

```
(env) ... demoproj $ touch movies/templates/movies/show.html
```

  - Please note that within the Django project we can identify the above template with *movies/show.html*

- Add a new view named *show* to the *views.py* file from the *movies* folder by implementing a function named *show*. The *show* view will display the details of a given movie (i.e. identified by its id – the primary key). The *show* view will use the *show.html* template we previously created.

```
def show(request, movie_id):
    try:
        movie = Movie.objects.get(pk=movie_id)
    except Movie.DoesNotExist:
        raise Http404("Movie does not exist")
    return render(request, 'movies/show.html', {'movie': movie})
```

  - Note that in the previous code if the requested movie does not exist a `Http404` exception is raised. Therefore, we must import the `django.http.Http404` class, i.e. add the following import statement to the *views.py* file

```
from django.http import Http404
```

– **!Checkpoint!** Your *views.py* file from the *movies* folder should look similar with the next content

```
from django.http import Http404
from django.shortcuts import render

from .models import Movie


def index(request):
    newest_movies = Movie.objects.order_by('-release_date')[:15]
    context = {'newest_movies': newest_movies}
    return render(request, 'movies/index.html', context)


def show(request, movie_id):
    try:
        movie = Movie.objects.get(pk=movie_id)
    except Movie.DoesNotExist:
        raise Http404("Movie does not exist")
    return render(request, 'movies/show.html', {'movie': movie})
```

- Next, we need to map the *show* view to an URL. Update the content of the *movies* → *urls.py* file by adding the mapping to the *urlpatterns* list via a *path()* call. For example, update the *urls.py* according to the following example

```
from django.urls import path

from . import views

urlpatterns = [
    # /movies/
    path('', views.index, name='index'),
    # /movies/id  e.g. /movies/1
    path('<int:movie_id>/', views.show, name='show'),
]
```

- ***Namespacing URL names*** In order to enable Django to distinguish between different URLs that would correspond to views with the very same name, e.g. *show* view, across multiple applications (recall that a Django project can have multiple applications), we should update the *movies* → *urls.py* file to set the application namespace (via *app_ name*). Update the *urls.py* as follows

```
from django.urls import path

from . import views

app_name = 'movies'
urlpatterns = [
    # /movies/
    path('', views.index, name='index'),
    # /movies/id  e.g. /movies/1
    path('<int:movie_id>/', views.show, name='show'),
]
```

- *Avoid using hardcoded URLs!* Update *index.html* template. Now that we mapped an URL to the *show* view by adding the previous *path()* call, we can use the value we provided to the *name* argument, namely *'show'* (i.e. name='show'), to eliminate the partially hardcoded URL (i.e. <a href="/movies/ movie.id /">) from the *index.html* template. We can replace the partially hardcoded URL with the *{% url %}* template tag. Update the *index.html* from *movies* → *templates* → *movies* folder to look similar with the following code

```
<h1>Newest Movies<h1>
{% if newest_movies %}
  <ul>
  {% for movie in newest_movies %}
   <li><a href="{% url 'movies:show' movie.id %}">{{ movie.title }}</a></li>
  {% endfor %}
  </ul>
{% else %}
  <p>There are no movies recorded.</p>
{% endif %}
```

- Note that in the *{% url %}* template tag from the *index.html* file, namely in the *{% url* ***'movies:show'*** *movie.id %}*
  - ***movies*** corresponds to the namespace from the *url.py* file, namely the value we provided to the *app_name* i.e. *app_name =* ***'movies'***
  - ***show*** corresponds to the value we provided to the argument *name* in the *path()* call from the *url.py* file, namely *path('<int:movie_id>/', views.show, name=****'show'****),*

- Next, update the *show.html* to display the details of a given movie.

```
<h2>{{movie.title}} Details</h2>
<table>
    <thead>
    </thead>
    <tbody>
        <tr>
            <td><b>Director:</b></td>
            <td>{{movie.director}}</td>
        </tr>
        <tr>
            <td><b>Release date:</b></td>
            <td>{{movie.release_date}}</td>
        </tr>
        <tr>
            <td><b>Genre:</b></td>
            <td>{{movie.genre}}</td>
        </tr>
        <tr>
            <td><b>Duration (min):</b></td>
            <td>{{movie.duration}}</td>
        </tr>
    </tbody>
</table>
```

Appendix: Verifying pip version & Installing pip

To verify that *pip*[3] is installed for the version of Python you are using, check both the version of Python you are using and the version of pip by running the next two commands at the terminal:

```
$  python --version

$  python -m pip --version
```

To install *pip* run the following commands from the terminal:

- Retrieve the install script

  ```
  $  curl -O https://bootstrap.pypa.io/get-pip.py
  ```

- Install pip for the version of Python you are using

  ```
  $  sudo python get-pip.py
  ```

- Verify pip is installed for the version of Python you are using

  ```
  $  python -m pip --version
  ```

- Next, you can delete the installation script *get-pip.py*

  ```
  $  rm get-pip.py
  ```

To upgrade the version of pip run the following command from the terminal

```
$  python -m pip install --upgrade pip
```

[3] https://pip.pypa.io/en/stable/

Appendix: Django – preview your application in Cloud9

To be able to preview your application in Cloud9 perform the following tasks

1. Run the server for your Django project by specifying also the port

   ```
   $  python manage.py runserver 8080
   ```

   Use two terminals!!! i.e. in one terminal leave the server running, and in another terminal run the commands required to further develop your project.

2. Once the server has started, Cloud9 will display a window with the preview link to your application. Click on that link; a new tab should automatically open, and you may see the *Invalid HTTP_HOST header* error documented below. Next, follow the guidelines provided at item 3 to fix that error.

3. **error:** *Invalid HTTP_HOST header: '....vfs.cloud9.eu-west-1.amazonaws.com'. You may need to add '....vfs.cloud9.eu-west-1.amazonaws.com' to ALLOWED_HOSTS.*

   - Modify the *settings.py* file of the project to update the ALLOWED_HOSTS with your Cloud9 environment's domain name provided in the error message (i.e. '....vfs.cloud9.eu-west-1.amazonaws.com')

     ```
     # used for development environment
     # NOTE you must use the Cloud 9 URL of your own environment!!!
     ALLOWED_HOSTS = ['....vfs.cloud9.eu-west-1.amazonaws.com']

     #ALLOWED_HOSTS = [] #revert to this version for the production
     ```

     !!! Ensure that you undo the changes you made above at item 3 for the production i.e. deployed version of the application!!!

Resources

- pip - The Python Package Installer https://pip.pypa.io/en/stable/
- Django https://docs.djangoproject.com
- Django Official Tutorial https://docs.djangoproject.com/en/4.2/intro/
  - Writing your first Django app, part 1 https://docs.djangoproject.com/en/4.2/intro/tutorial01/
  - Writing your first Django app, part 2 https://docs.djangoproject.com/en/4.2/intro/tutorial02/
  - Writing your first Django app, part 3 https://docs.djangoproject.com/en/4.2/intro/tutorial03/
  - Writing your first Django app, part 4 https://docs.djangoproject.com/en/4.2/intro/tutorial04/
  - Writing your first Django app, part 5 https://docs.djangoproject.com/en/4.2/intro/tutorial05/
- Writing views https://docs.djangoproject.com/en/4.2/topics/http/views/
- Templates https://docs.djangoproject.com/en/4.2/topics/templates/