Django Workshop Session 1

Advanced Programming

Amirkabir University of Technology

Spring 2019

Overview

What is Venv

virtual environment or simply venv allows you to have a virtual installation of python and packages on your computer.

Why Using Venv

- Packages change and get updated often.
- Some of these changes break backwards compatibility.
- You may want to test out new features.
- However, you can't take down your website every time a package gets updated.
- So we will use venv for package (dependency) management.

How to Create a Venv

```
$ cd <project directory>
$ virtualenv -p <address of python> <name>
$ source ./<name>/bin/activate
$ pip install <packages>
$ pip freeze -l > requirements.txt **
$ ...
$ deactivate
```

it is also possible to use IDE tools related to virtual environments.
** We can also create requirements.txt file manually in order to avoid excess package installations.

How to create django projects

```
$ ... venv is activated with django installed
$ django-admin startproject <project name>
$ cd <project name>
$ django-admin startapp <app name>
$ ... magic happens
$ python manage.py migrate
$ python manage.py makemigrations
$ python manage.py migrate
$ python manage.py runserver
```

Some files generated

__init__.py

- This is a blank python script that let's our directory to be treated as a python package.

settings.py

This is a where all of our project settings exists including:

- Base directory, templates and static files directories
- installed apps (defaults and defined apps)
- database engine and file
- password validation methods
- language and timezone used in project

urls.py

- This is a Python scrip that will store all the url patterns for our project.
- Basically the different pages of our application.
- linking methods implemented in business logic to urls generated by user interactions with project.

wsgi.py

- Wsgi stands for web server gateway interface.
- It helps us deploy our web app to product.

manage.py

- Simply manages our project.
- We use it to run some of our commands as we build our project a lot.

App vs. Project

App? Project?

A Python package provides a way of grouping related Python code for easy reuse. A package contains one or more files of Python code (also known as “modules”).
A package can be imported with import foo.bar or from foo import bar.
For a directory (like polls) to form a package, it must contain a special file __init__.py, even if this file is empty.
A Django application is just a Python package that is specifically intended for use in a Django project. An application may use common Django conventions, such as having models, tests, urls, and views submodules.
Later on we use the term packaging to describe the process of making a Python package easy for others to install. It can be a little confusing, we know.

admin.py

- We can register our models here
- Changing what to show and how to show them in admin interface.

apps.py

- Here we can place application specific configuration.
- We can also register our app by setting using apps.py.

models.py

- Here we can store our application data model.
- Classes stand for tables, attributes for fields etc.
- What entities are included in our application, how they are related and what attributes each of them has.

test.py

- We can design methods and classes to test our code automatically.
- Not included in our lectures.

views.py

- This is where we have functionalities of our application.
- handling requests and return responses
- Two methodologies: class based views (CBV) and function based views (FBV).

templates

- Templates are a key part to understanding how django really works and interacts with our website.
- We have some template tags to produce dynamic data.
- it goes under our top level directory that is:

  -
    ```
    <Project_dir>/templates/<App_dir>
    ```

Template tags

There are some template tags by help of which we can insert dynamic content into HTML document. We will handle them in views.py . It might be surprising but we can define our own template tags.

- variables: {{ variable name }}
- extending: {% extends "base.html" %}
- loop: {% for obj in list %} body {% endfor %}
- condition: {% if condition %}{% elif condition %}{% else %}{% endif %}
- block: {% block name %} code {% endblock %}
- in forms: {% csrf_token %}
- statics: {% load staticfiles %}
- source: src={% static "a picture addr"%}

Migrations

- This directory stores information about the database, related to the models and their changes.
- Usually ignored in commits.

Notes to Remember

1. We usually add a urls.py to each application in order to make them more modular.
2. Then the urls.py of project will include them and refer to them in order to resolve a url address.

```
from django.conf.urls import include
urlpatterns = [ ...
url(r'<a_pattern>/', include('<app_name>.urls')),
... ]
```

3. We optionally make a forms.py script and separate our forms from other views in order to increase readability.

The Approach to build a project

It's up to the developer to take any approaches, however the steps below are recommended:

1. Designing our models carefully because it should change rarely and other parts are highly related on it.
2. Implementing a functionality or action of our project in views.
3. Mapping a url to the view and creating a template file.
4. go to 2.

Static Files

There are two ways to include static files like images in our project:

- fetching data from the database. (not covered)
- defining a folder and getting images from there.

There will be folders named: css, js, images etc. in a directory named static, located in project directory.
We should locate STATIC_DIR at the end of settings.py.

The End