Python Programming

Day 18: Introduction to Django Framework

Django Model and Admin Panel

DJANGO	Model
View	Template

Color and symbol meaning

Hint

Preferred

Student's activity

Practice code

	Keyword
	In-built functions
	Strings
	Output

Introduction to Django Framework

Django is a free and open source web application **framework**, written in **Python**. A **web framework** is a **set of components** that helps you to develop websites **faster and easier**.

Frameworks exist to save you from having to reinvent the wheel and to help alleviate some of the overhead when you're building a new site.

How Django works

When a **request** comes to a web server, it's passed to Django which tries to figure out what is actually requested. It takes a web page **address first and tries to figure out what to do**.

This part is done by **Django's urlresolver** (note that a website address is called a URL – Uniform Resource Locator)

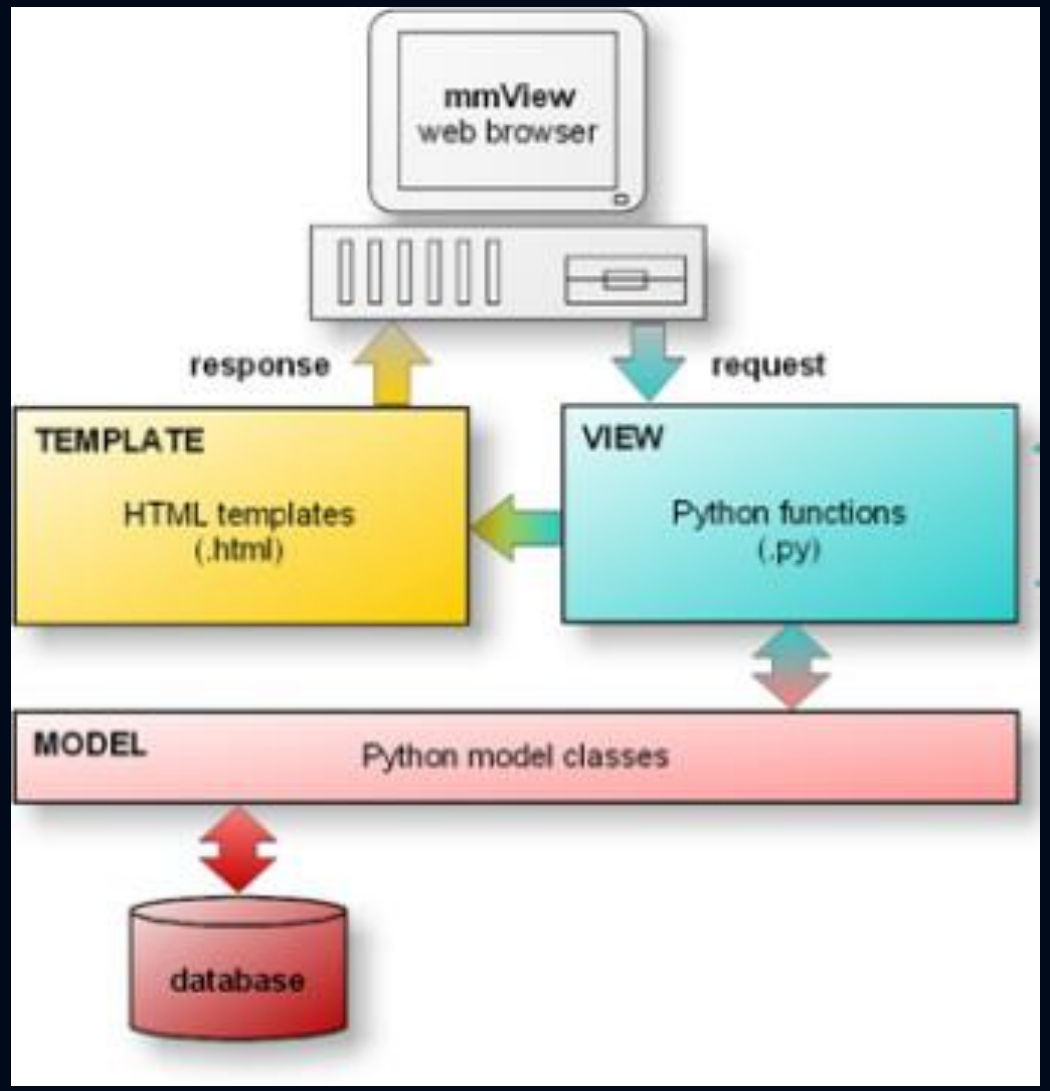
Django **compares** URL patterns with a **declared list in urls.py** file and if something is matched, then Django passes the request to the associated function (which is called **view**).

How Django works

In the view function, all the interesting things are done: we can look at a database to look for some information or save data into the database or manipulate data before displaying to the user.

Then the view generates a response and Django can send it to the user's web browser using the Template.

How Django works



Model View Template (MVT) Architecture

Python Programming

Django Application Development

Getting Started

We shall learn how to develop web application with Django by developing a "ToDo" App.

Before we progress, it is expected that we have installed Django and a rich text editor (Pycharm)

Getting Started

The first step is to start a new Django project. Basically, this means that we'll run some scripts provided by Django that will create the skeleton of a Django project for us. This is just a bunch of directories and files that we will use later.

Remember to run everything in the virtualenv. If you don't see a prefix (myvenv) in your console, you need to activate your virtualenv. We explained how to do that earlier. Typing myvenv\Scripts\activate on Windows.

Getting Started

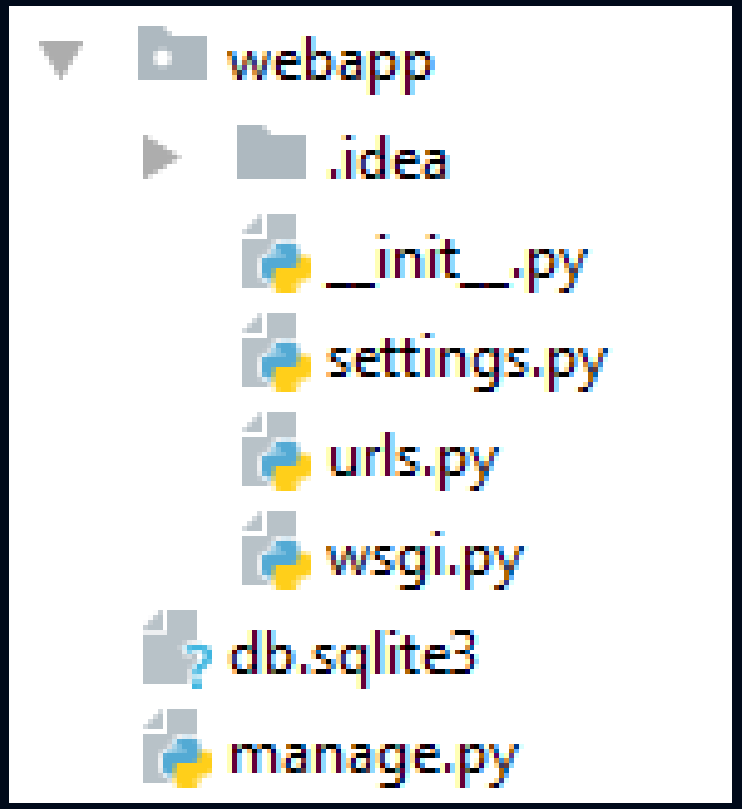
Let's run the following command.

(myvenv)C:\Users\xxxx\environment>
django-admin startproject webapp .

The period "." at the end of the command is crucial because it tells the script to install Django in your current directory.

Getting Started

You should now have a directory structure which looks like this:



- **The manage.py is a script that helps with management of the site. With it we will be able (amongst other things) to start a web server on our computer without installing anything else.**
- **The settings.py file contains the configuration of your website.**
- **The urls.py file contains a list of patterns used by urlresolver.**

Getting Started

The "webapp" project is not the application rather a special directory for one or more applications.

We need to create a new application within the "webapp" directory to get really STARTED!

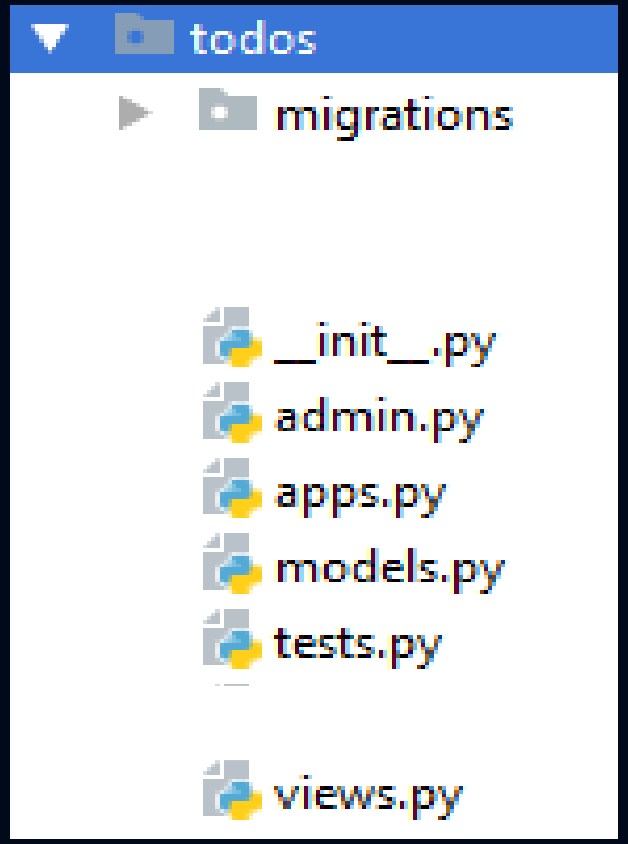
Run the command shown by the side within the webapp directory

```
(myvenv)C:\Users\xxxx\environment\webapp> python manage.py startapp todos
```

todos is the name of our application

Django Basic Configuration

We should have something similar to the figure shown



Django Basic Configuration

There are 3 basic modifications to be made to the settings.py file.

- **Installed Applications setting**
- **Database setting**
- **Templates setting**

Django Basic Configuration

❖ **Installed Applications**

You must enable the application by adding the name to the list of Installed Apps

```
# Application definition

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

Make sure you add it to the top of the list.

Django Basic Configuration

❖ Database

There is a lot of different database software that can store data for your site. We'll use the default one, **sqlite3**. This is already set up in this part of your **webapp/settings.py** file:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Django includes support out of the box for MySQL, PostgreSQL, SQLite3, and Oracle.

Django Basic Configuration

❖ Static files settings

We need to add a path for static files (css, javascript).

Go down to the end of the file, and just underneath the STATIC_URL entry, add a new one called STATIC_ROOT as shown below.

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.11/howto/static-files/

STATIC_URL = '/static/'
STATIC_ROOT = os.path.join(BASE_DIR,'static')
```

Django Basic Configuration

❖Starting the web server

We **need to be in the directory that contains the manage.py file**, in this case, "environment". In the console, we can start the web server by running: > **python manage.py runserver**:

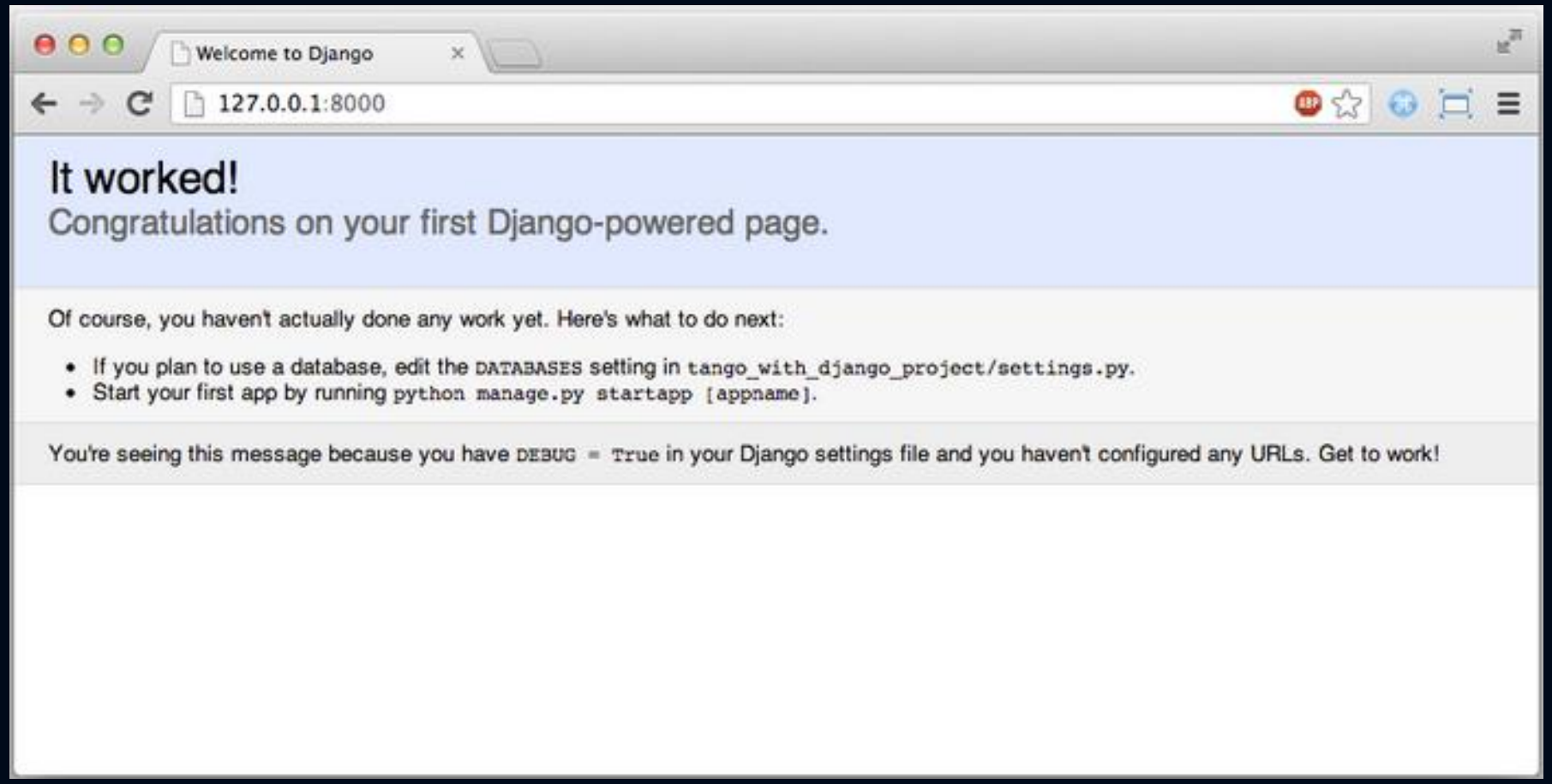
```
C:\Users\ ━━━━ \environment>myvenv\Scripts\activate
(myvenv) C:\Users\ ━━━━ \environment>python manage.py runserver
Performing system checks...

System check identified some issues:

WARNINGS:
?: (urls.W001) Your URL pattern '^$' uses include with a regex en

System check identified 1 issue (0 silenced).
June 19, 2017 - 14:35:26
Django version 1.11.2, using settings 'webapp.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Django Basic Configuration

Open your web browser and enter this address:

http://127.0.0.1:8000/

Now you should be live on the browser



Django Basic Configuration

While the web server is running, you won't see a new command-line prompt to enter additional commands.

To type additional commands while the web server is running, open a new terminal window and activate your virtualenv.

Django Model

A **model** is a **class** that **represents table** or collection in our database, and where **every attribute of the class is a field of the table** or collection.

Models are defined in the app/models.py (in our example: todos/models.py)

Let's open todos/models.py and include the following lines of code

Django Model

- **Datetime** is required due to **timestamp** we want to add to each record.
- **Todo** is the **name of the table** to be created while **title**, **details** and **status** are the other field to be created in the table.
- The **__str__** magic method is overridden to **return the title** for each record

```
from datetime import datetime
# Create your models here.
class Todo(models.Model):
    title = models.CharField(max_length=100)
    details = models.TextField()
    status = models.CharField(max_length=10)
    created_at =
models.DateTimeField(default=datetime.now,
blank=True)

    def __str__(self):
        return self.title
```

Django Model

After our **model** is created we need to **add** it to the **database** as a **table** by creating **migrations** that implement the linking between the model and the database.

Run the following command from the console

```
\webapp> python manage.py makemigrations todos
\webapp> python manage.py sqlmigrate todos 0001
\webapp> python manage.py migrate
```

The **0001** is the number returned from **makemigrations todos command**

Django Model

Basic Datatypes in Django model.

models.CharField – defines text with a limited number of characters.

models.IntegerField – defines an integer field

models.TextField – used to define long text without a limit.

models.DateTimeField – defines date and time field.

models.ForeignKey – this is used to link to another model.

Django Admin

One of the most powerful parts of Django is the automatic admin interface.

In this document we discuss how to activate, use, and customize Django's admin interface.

Django Admin

We must create a *superuser* (a user account that has control over everything on the site) to access the admin section of Django

Run the following command from the console

- **\webapp> python manage.py createsuperuser --username=covenant --email= python@covenant.com**
- **Set password by entering it twice at the prompt**

Django Admin

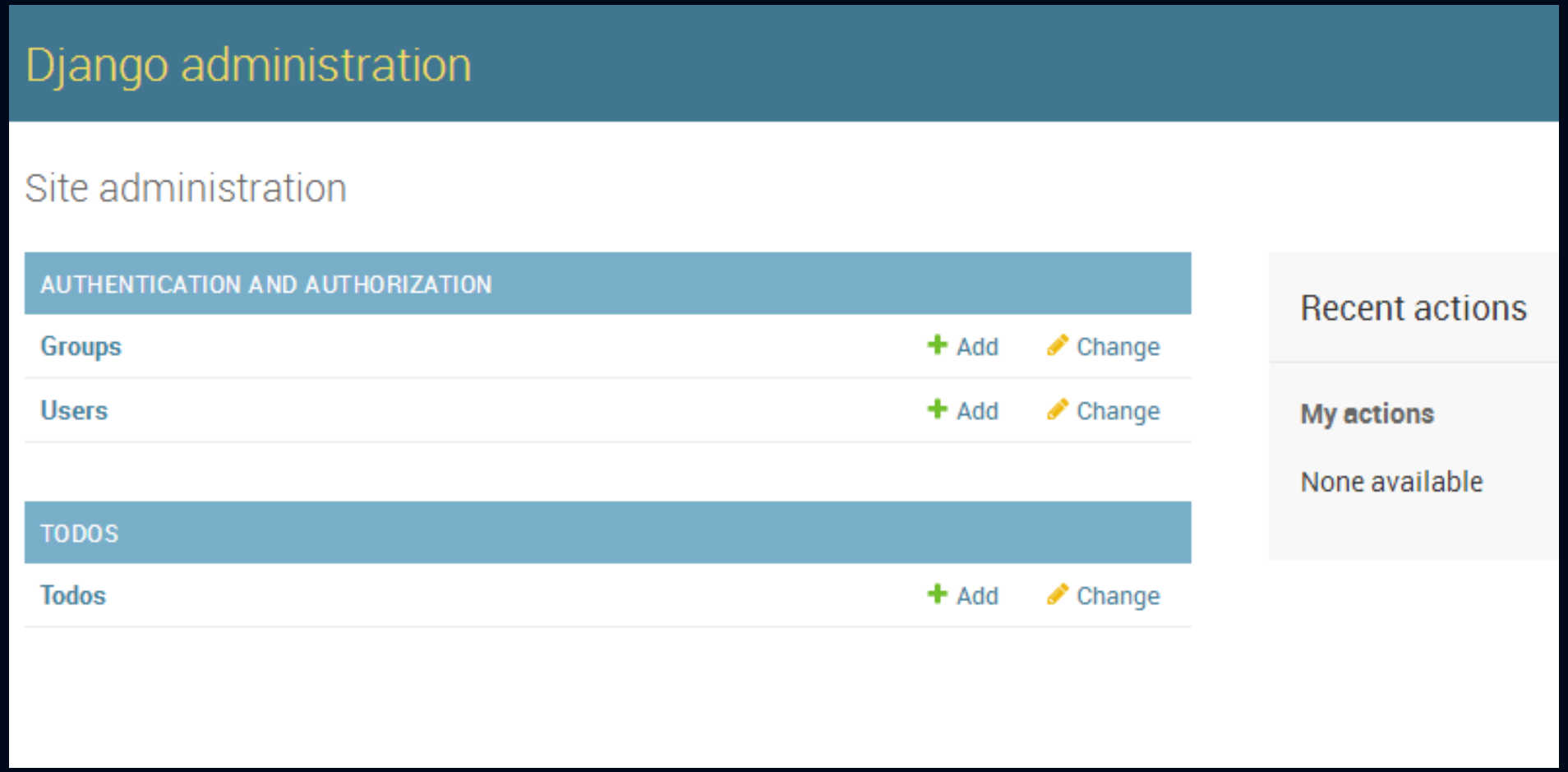
With *superuser* access, let us now include our model into the Django admin area.

Goto admin.py in the todos directory and include the following lines of code

- **from . models import Todo**
- **admin.site.register(Todo)**

Django Admin

- **Navigate to http://localhost:8000/admin/**
- **Provide the username and password for the *superuser* to access the admin panel**



Django Admin

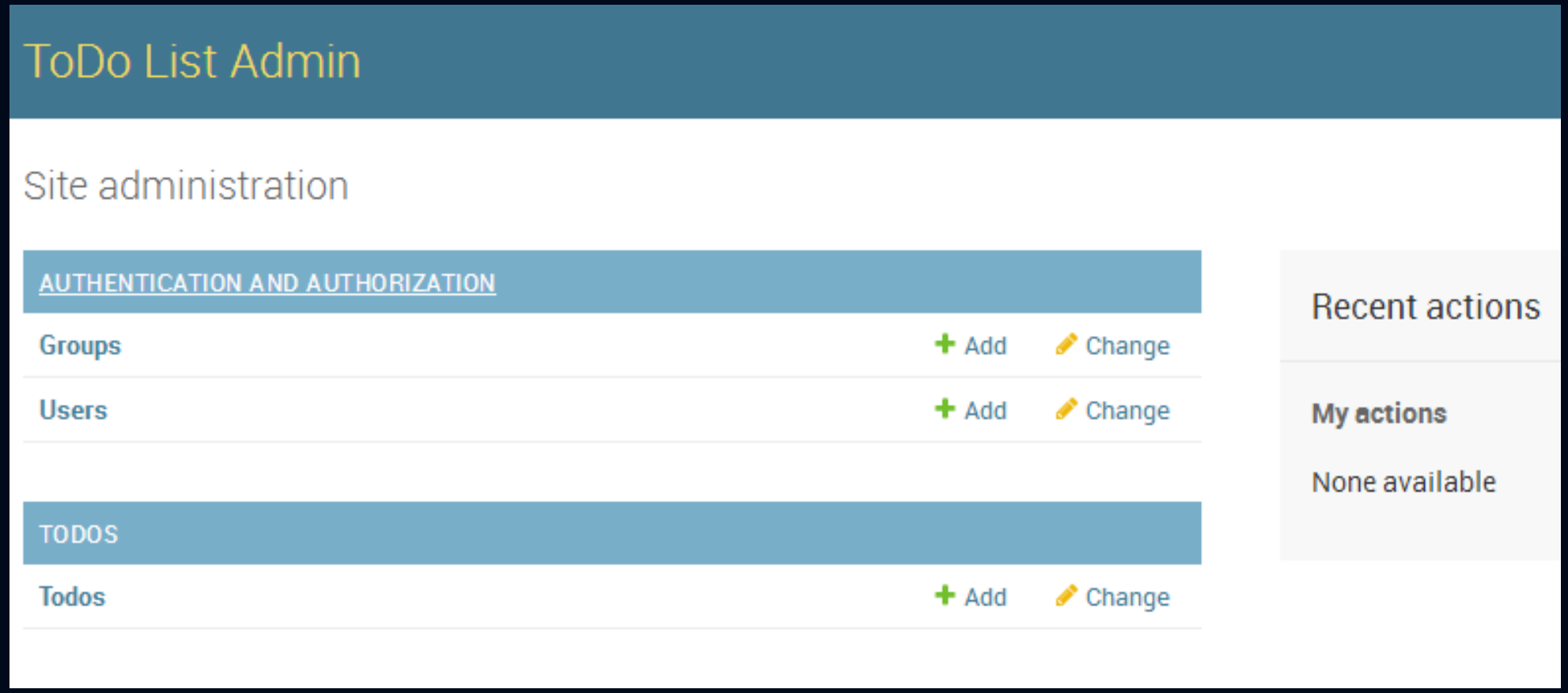
To **Personalize** the Admin panel, do the following.

- Create a new directory called **templates** where we shall **store all our HTML files**
- Create **another** directory called **admin inside the templates** directory
- Add **base_site.html** to the **admin** directory
- **Include** the code block shown in the **base_site.html** file.

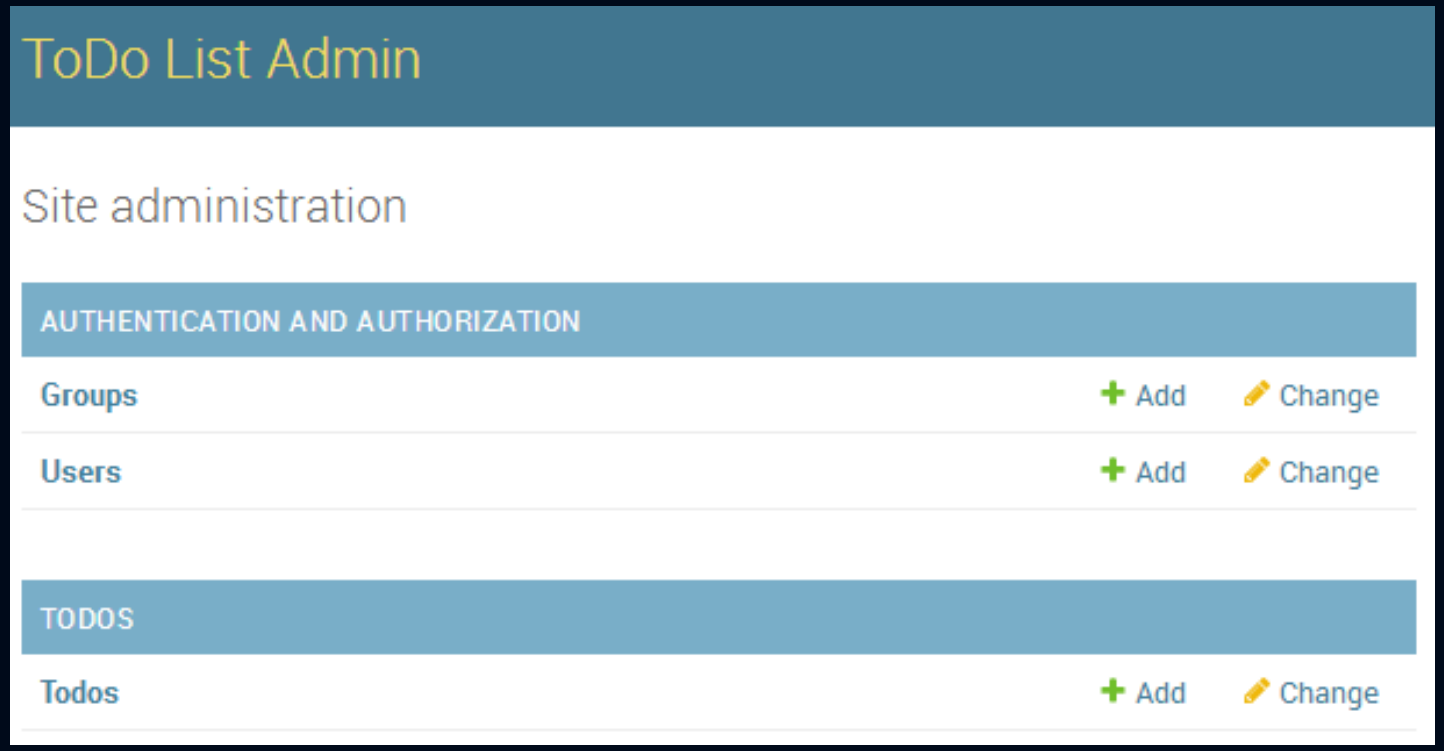
```
{% extends "admin/base.html" %}

{% block branding %}
  <h1 id="site-name">ToDo List Admin</h1>
{% endblock %}
```

Django Admin

This changes the Header name of the admin panel. Several other personalization can be done.



Class Activity



Use the add button to add 3 records to the ToDo database

Python Programming

Tutorials

Exercise 1:

Create a new project named "cu" having an app named "myapps". Create a model "myapp" with matricno, name, level and gpa.

Enable the admin panel and use it to add 5 records

Note: Create the project and app within the virtual environment

Next Lecture ...

Python  Programming

Day 19: Django URL and Template