Django Introduction

SoftUni Team

Technical Trainers

Software University

https://softuni.bg

Have a Question?

Software University

sli.do

#python-web

Table of Contents

Django Framework

Full-Stack Framework for Perfectionists with Deadlines

What is Framework?

- A foundational **structure** or set of tools
- Provides a **pre-defined structure** and **reusable** components
- Streamlines the development of software **applications**
- Allows developers to **focus** on specific **functionalities**
- Includes **libraries**, **templates**, and **predefined patterns** helping developers to work **efficiently** and **consistently**

What is Django?

- **High-level** Python Web Framework, known for its
  - **Speed**
  - **Security**
  - **Scalability**
  - **Open-source** nature

What is MVT?

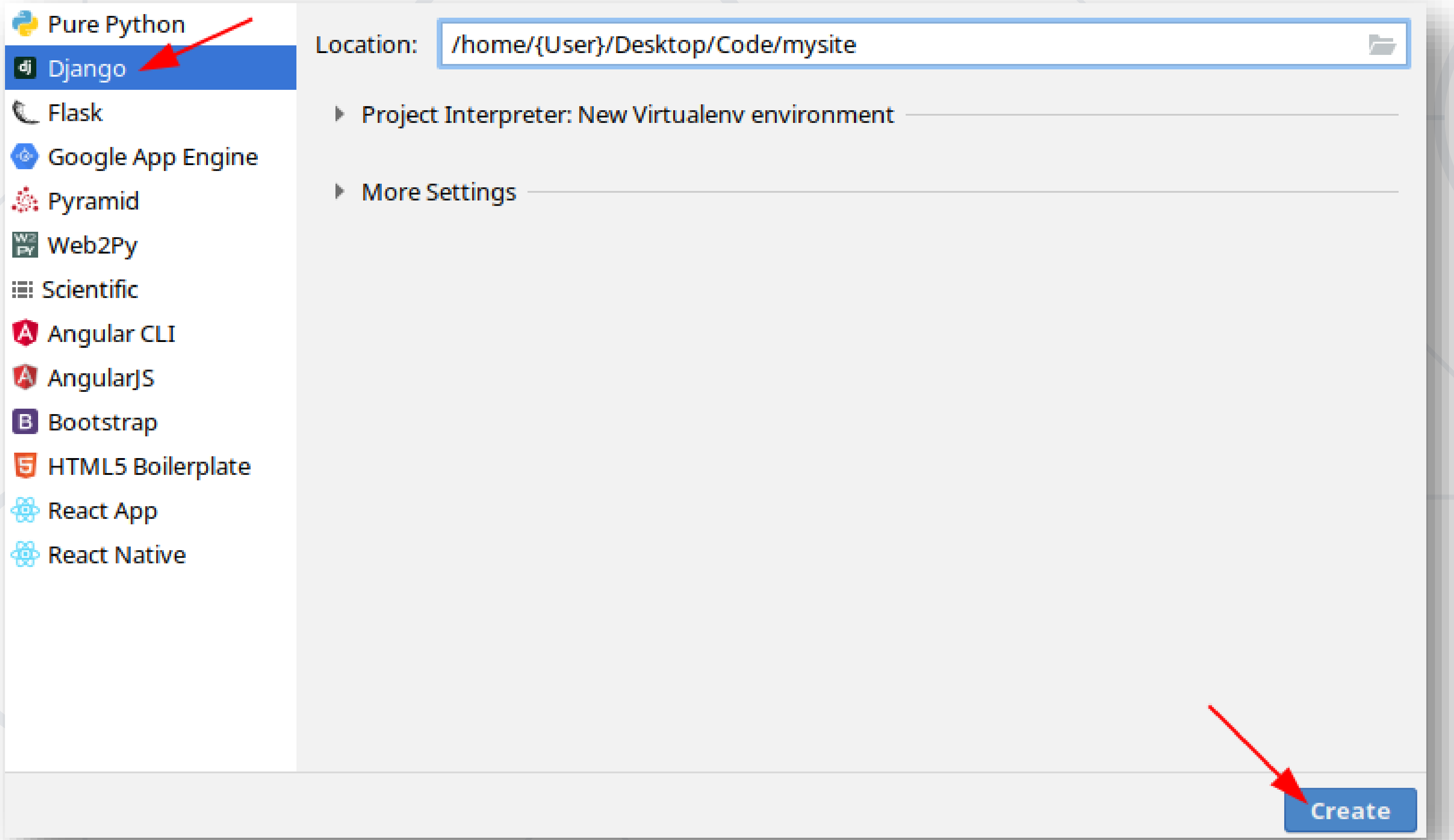
- Django follows the **MVT design pattern** to develop web applications
- MVT stands for **Model-View-Template**
  - **`Model`** - defines the structure and behavior of data
  - **`View`** – receives an **HTTP request** and returns an **HTTP response**
    - Contains the application's business logic
  - **`Template`** - the presentation (front-end) layer
    - Provides a convenient way to generate **dynamic** HTML pages by using a **special** template syntax (**DTL**)

Creating a Django Project

Where the magic happens

Creating a Django Project

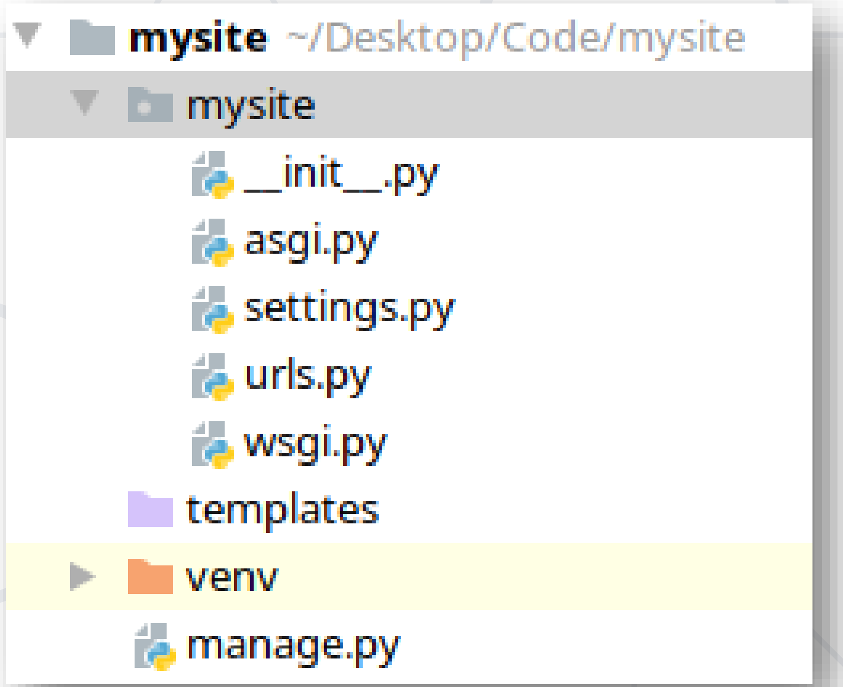
- Open PyCharm Professional -> File -> New Project



Location: /home/{User}/Desktop/Code/mysite

Project Structure

- **__init__.py**
  - The directory is a Python package
- **settings.py**
  - The configuration file for the Django Project
- **urls.py**
  - Table of Content
- **manage.py**
  - Tool for executing commands



```
mysite ~/Desktop/Code/mysite
  mysite
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
  templates
  venv
  manage.py
```

Running a Django Project

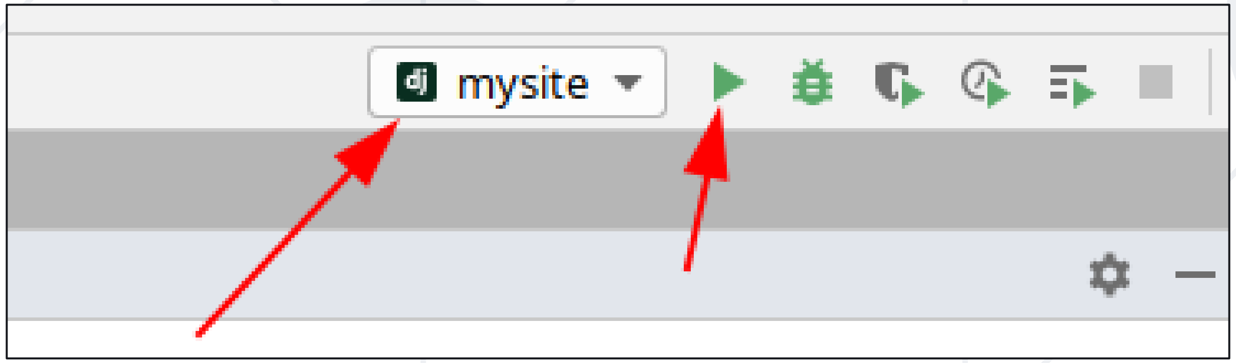
- Using **Terminal command**

```
python manage.py runserver
```

- Using **Keyboard Shortcut** in PyCharm

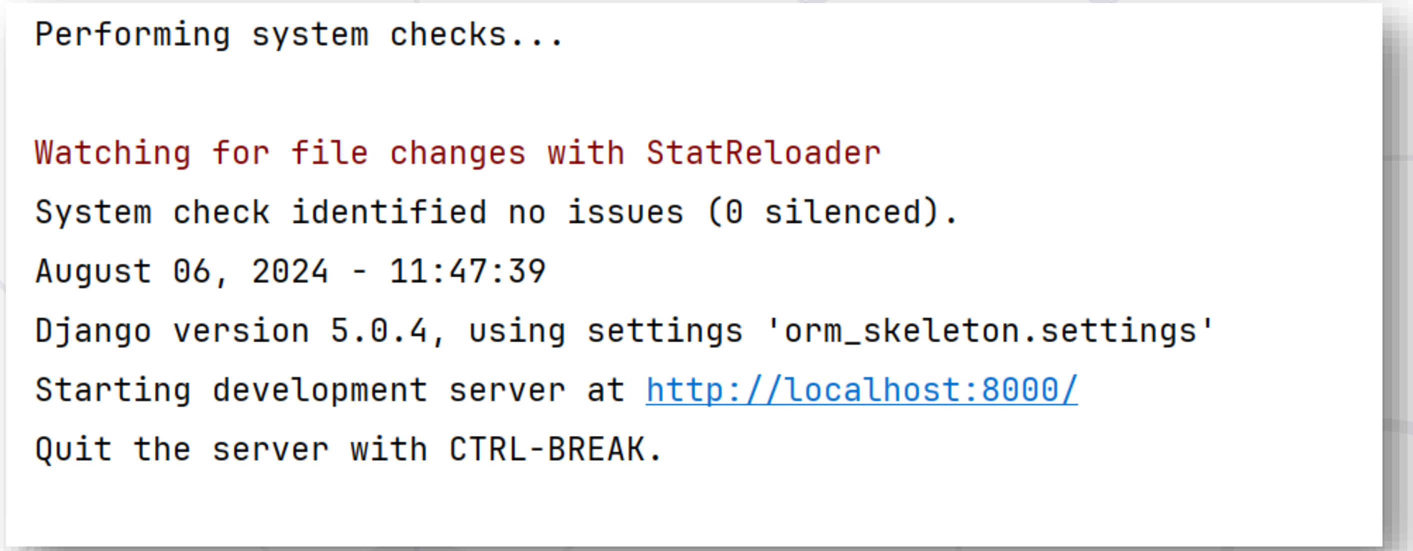
```
Shift + F10
```

- Using PyCharm **Run button**



Running a Django Project

- You'll see the following output on the command line:

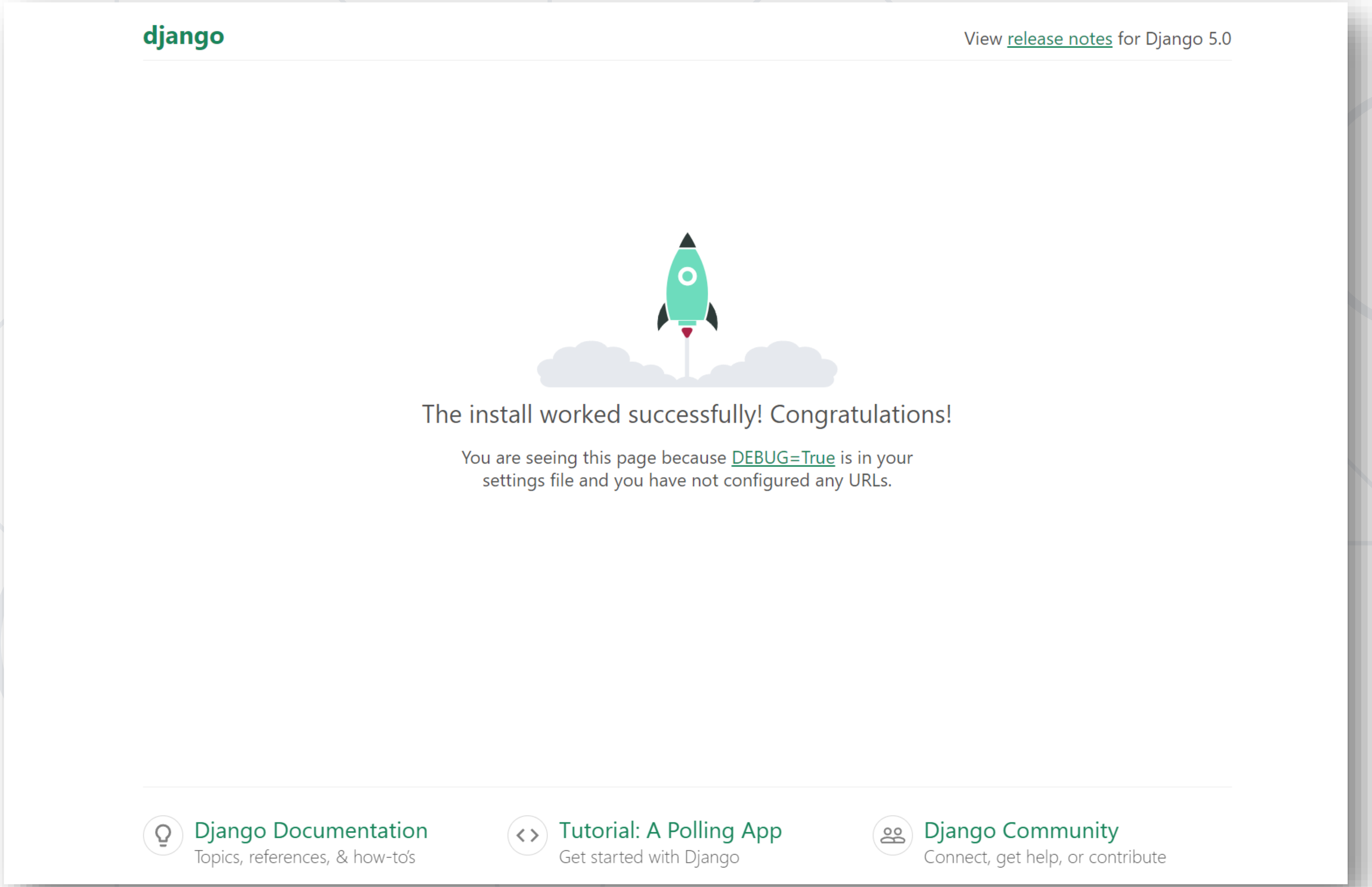


```
Performing system checks...

Watching for file changes with StatReloader
System check identified no issues (0 silenced).
August 06, 2024 - 11:47:39
Django version 5.0.4, using settings 'orm_skeleton.settings'
Starting development server at http://localhost:8000/
Quit the server with CTRL-BREAK.
```

- The **runserver** command starts the **development server** on the internal IP at **port 8000** by **default**
- Note: This server is used for **development** purposes **only**

Running a Django Project



django

View release notes for Django 5.0

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Django Documentation
Topics, references, & how-to's

Tutorial: A Polling App
Get started with Django

Django Community
Connect, get help, or contribute

Django Application

The Bread and Butter of a Django Project

App vs Project

- Django App:
  - A **Web application that does something** - e.g., a small task app
  - An app can be **in multiple projects**
- Django Project:
  - A **collection of configurations and apps** for a particular website
  - A project can **contain multiple apps**

Creating a Django App

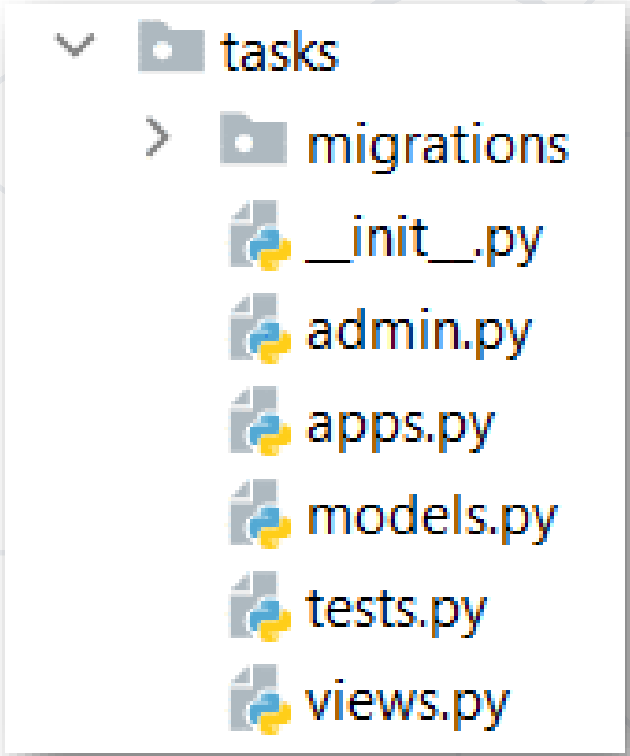
- The app is created in the **same directory** as the **`manage.py`** file
- Use the **terminal command**

```
python manage.py startapp tasks
```

- Move it **inside the project** for a **better-structured** project management
- Django **automatically** generates the **basic directory structure** of an app

Directory Structure

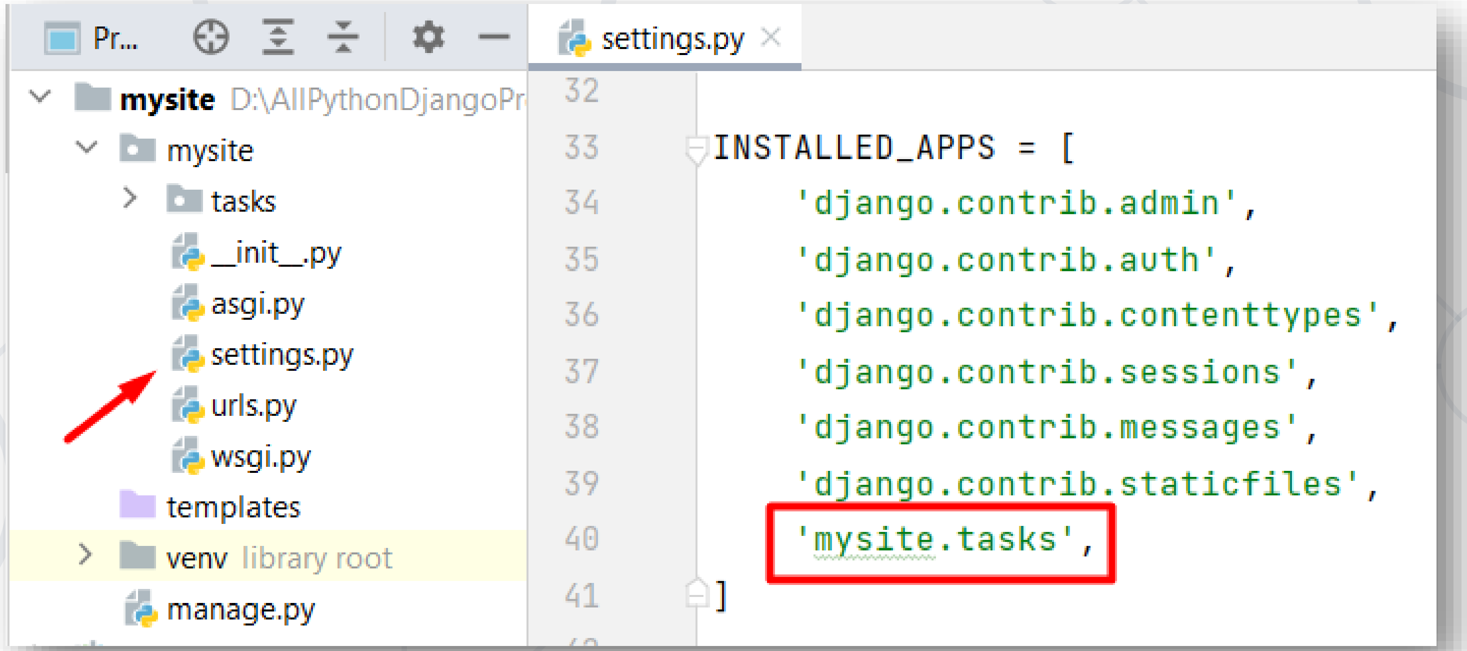
- **admin.py**
  - The admin page
- **models.py**
  - The models of the app
- **views.py**
  - The views of the app
- **migrations**
  - Command-line utility for propagating changes in models



```
tasks
  migrations
  __init__.py
  admin.py
  apps.py
  models.py
  tests.py
  views.py
```

Including an App

- To include an app in a project, **add a reference** to the app in the **`INSTALLED_APPS`** setting



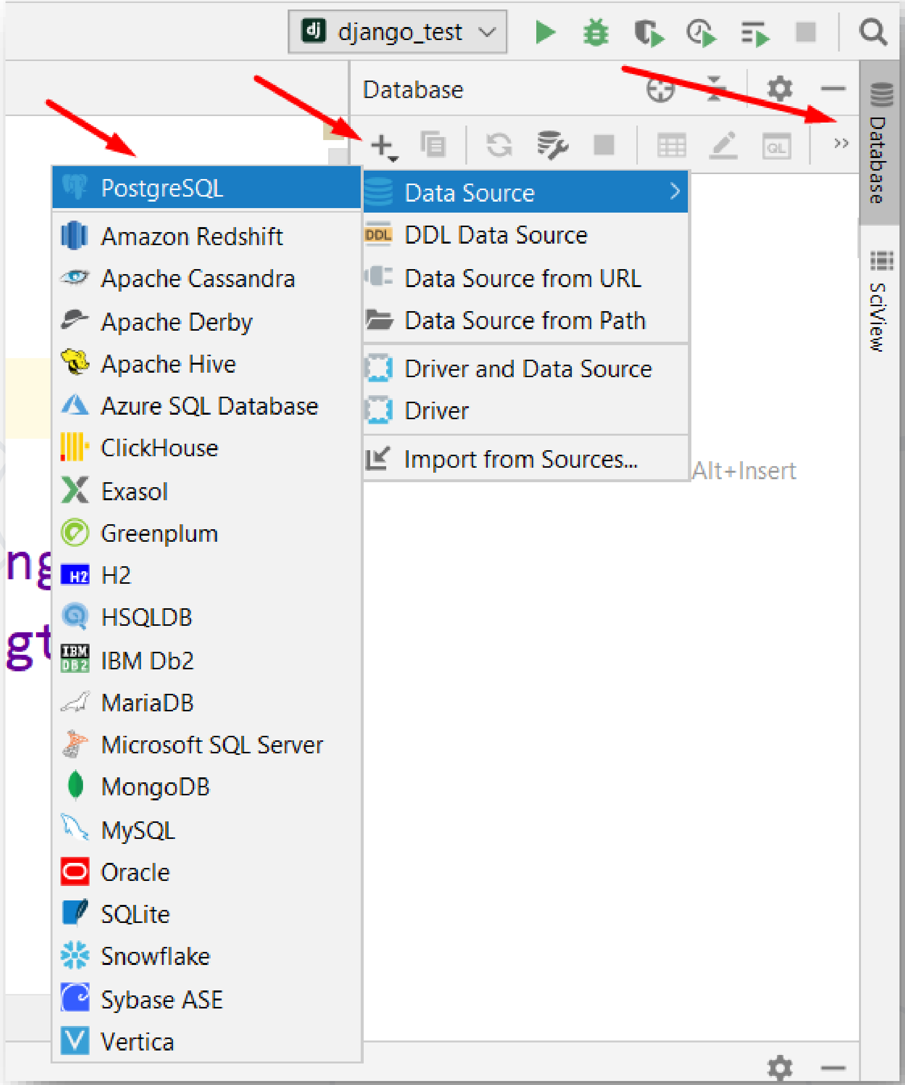
```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'mysite.tasks',
]
```

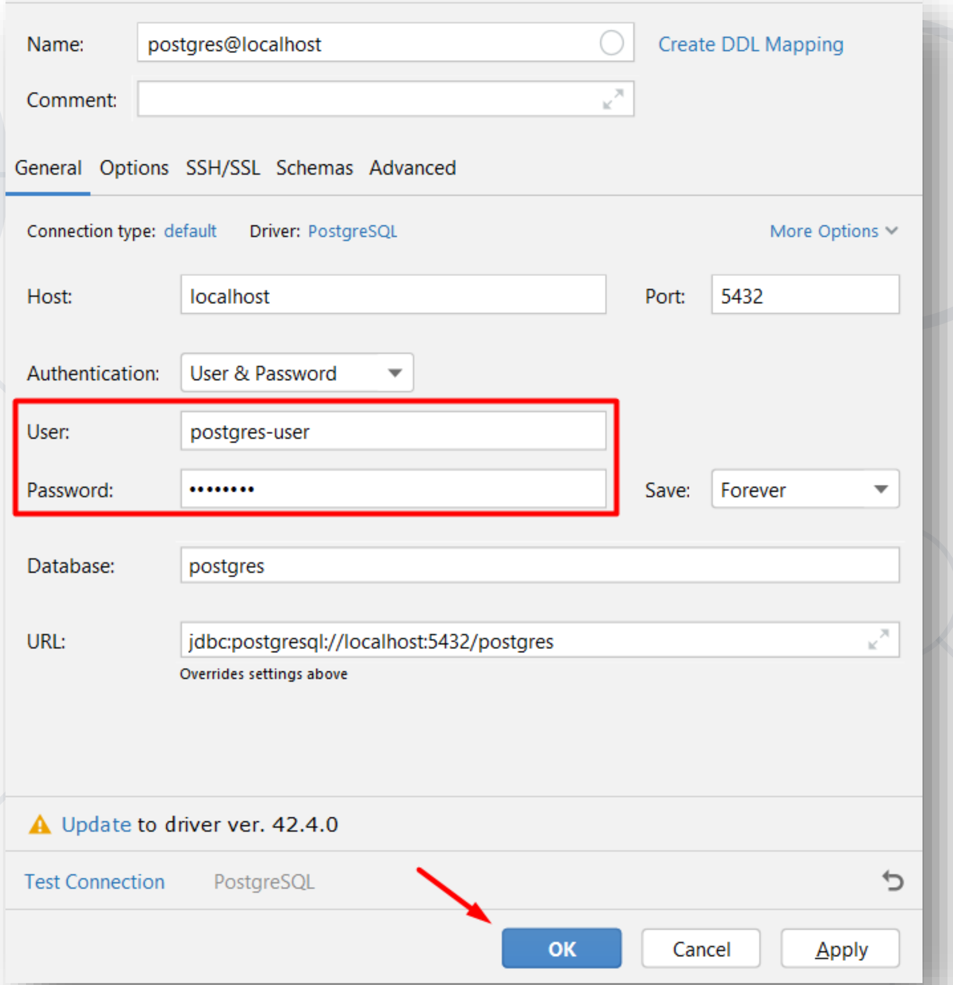
Setting up a Database

Psycopg2

- PostgreSQL database **adapter** for the Python programming language
- Use the Psycopg2 module to:
  - **Connect** to PostgreSQL
  - Perform SQL **queries** and database **operations**
- It is an **external** module

Connect to PostgreSQL





Name: postgres@localhost

Comment:

Connection type: default Driver: PostgreSQL

Host: localhost Port: 5432

Authentication: User & Password

User: postgres-user

Password: ••••••••

Save: Forever

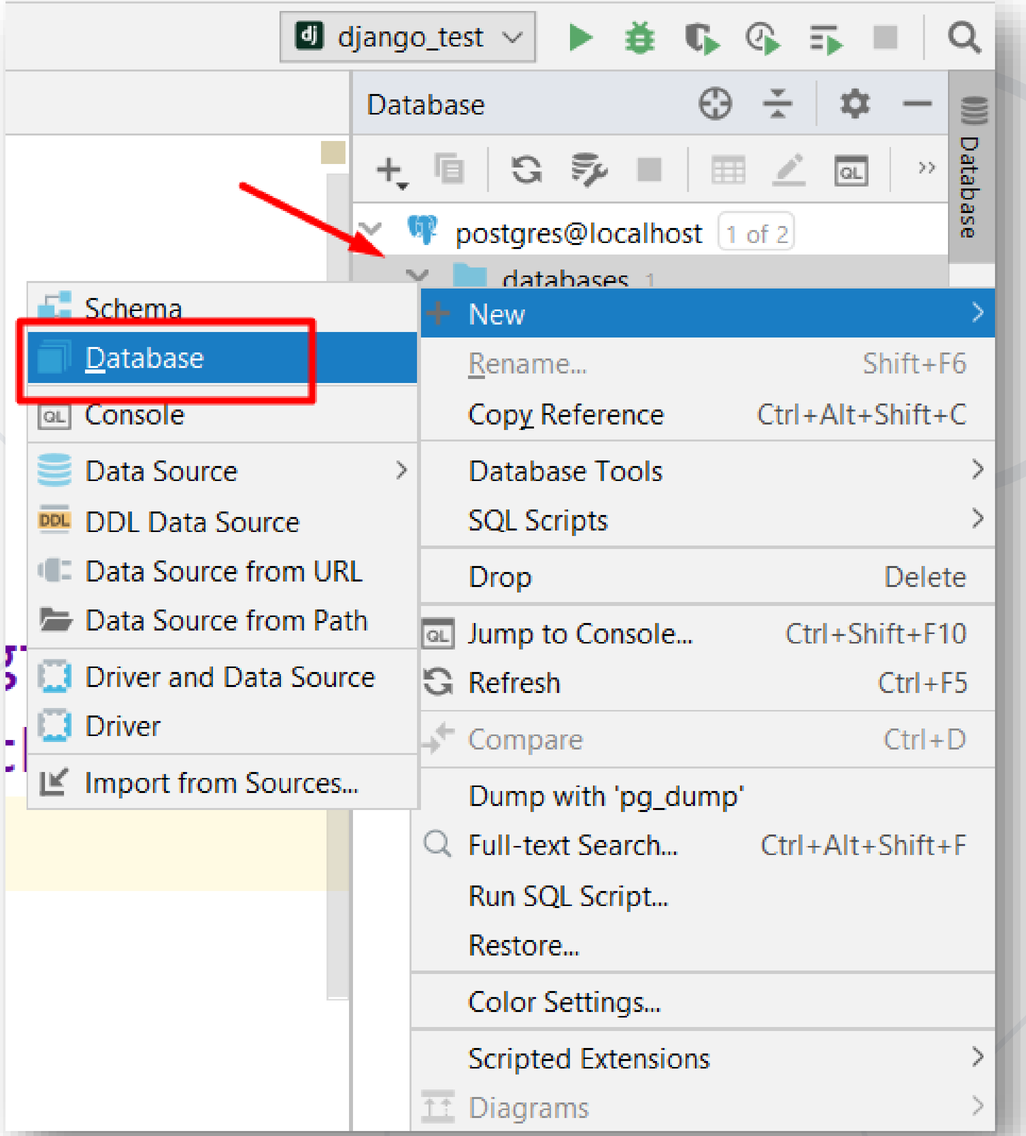
Database: postgres

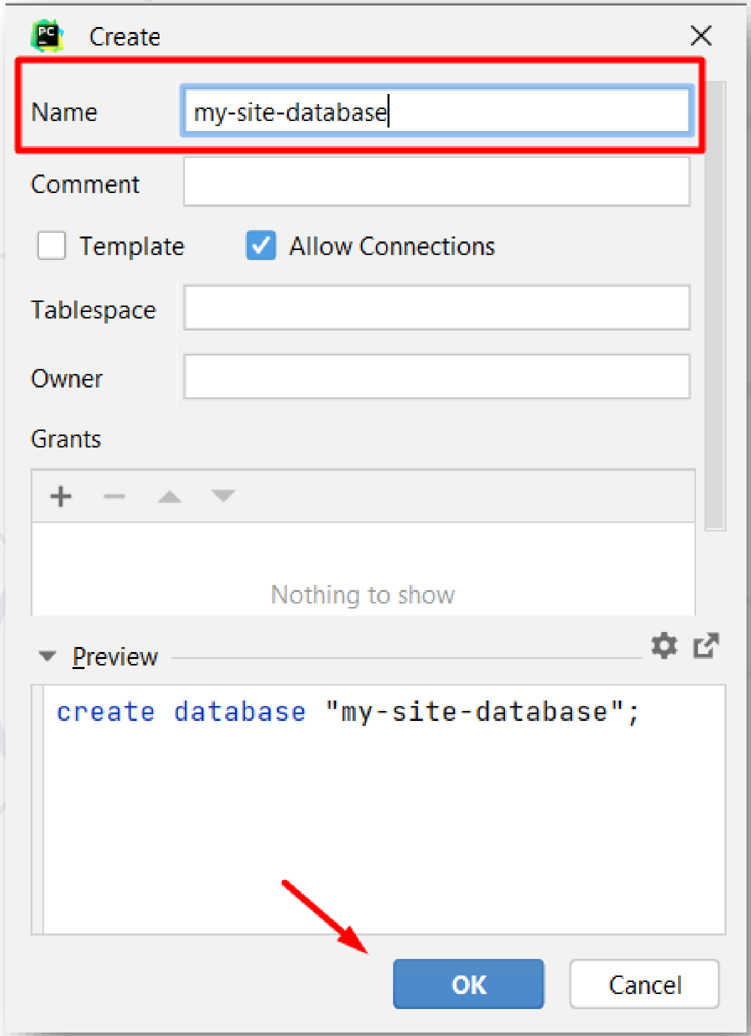
URL: jdbc:postgresql://localhost:5432/postgres

Overrides settings above

Update to driver ver. 42.4.0

Create a Database





Name: my-site-database
Comment:
[ ] Template [✓] Allow Connections
Tablespace:
Owner:

```
create database "my-site-database";
```

Set up PostgreSQL

- To configure our project to work with **PostgreSQL**, we need to set it in the **settings.py** file

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'my-site-database',
        'USER': 'postgres',
        'PASSWORD': 'postgres',
        'HOST': '127.0.0.1',
        'PORT': '5432'
    }
}
```

Use PostgreSQL

Name of the database

Database user credentials

Writing a Simple Task App

Django Model

- Models represent your **application's data**
  - The essential **fields** and **behaviors** of the stored data
- Each model maps to a single **database table**
- Model is a Python class that subclasses `django.db.models.Model`
- Each attribute of the model represents a **database field**

Adding a Model

- Each application has a **`models.py`** file
- Create **models** that will be used in the **application**

tasks/models.py

```
from django.db import models

class Task(models.Model):
    title = models.CharField(max_length=50)
    text = models.TextField()
```

Model Name

Field Types

Fields

Activating Models

- Use models to create a **database schema** for the app
- Use **migrations** to apply **changes** and **update** the database schema
  - First, **create migrations** for the added model

```
python manage.py makemigrations
```

  - Next, **apply those changes** to the database

```
python manage.py migrate
```

Django View

- The **`views.py`** file contains view functions or classes
- Each view takes an **HTTP request** and returns an **HTTP response**
- Implements the **business logic** that needs to be executed when a given **URL** is reached
- The **names** of the functions are usually related to the **URL** that is being reached

Simple View Example

tasks/views.py

```
from django.http import HttpResponse
from tasks.models import Task

def index(request):
    tasks_list = Task.objects.all()
    output = "; ".join(f"{t.title}: {t.text}"
                       for t in tasks_list)
    if not output:
        output = "There are no created tasks!"

    return HttpResponse(output)
```

Get all Task objects

Return the desired output

Django app/urls.py

- In the **urls.py** file you configure which function or logic should be executed when reaching a given **URL**
- **Each app** should have its own **urls.py** file

tasks/urls.py

```
from django.urls import path
from {app_name} import views

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

URL

Action

Django project/urls.py

- The created urls.py file should be **included** in the **project's** urls.py
- Import the `include()` function and use it in the `urlpatterns` list

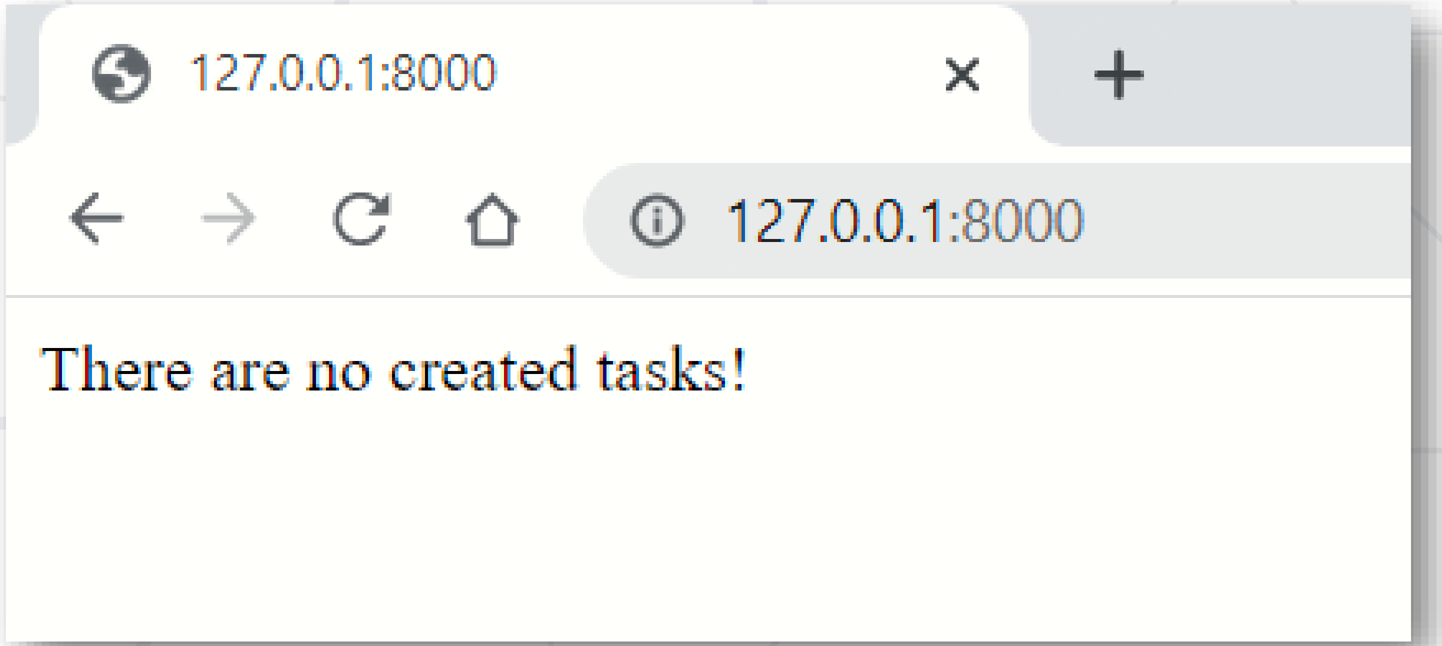
mysite/urls.py

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

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

Simple URL Example

- Start a **development server** and verify the **result**

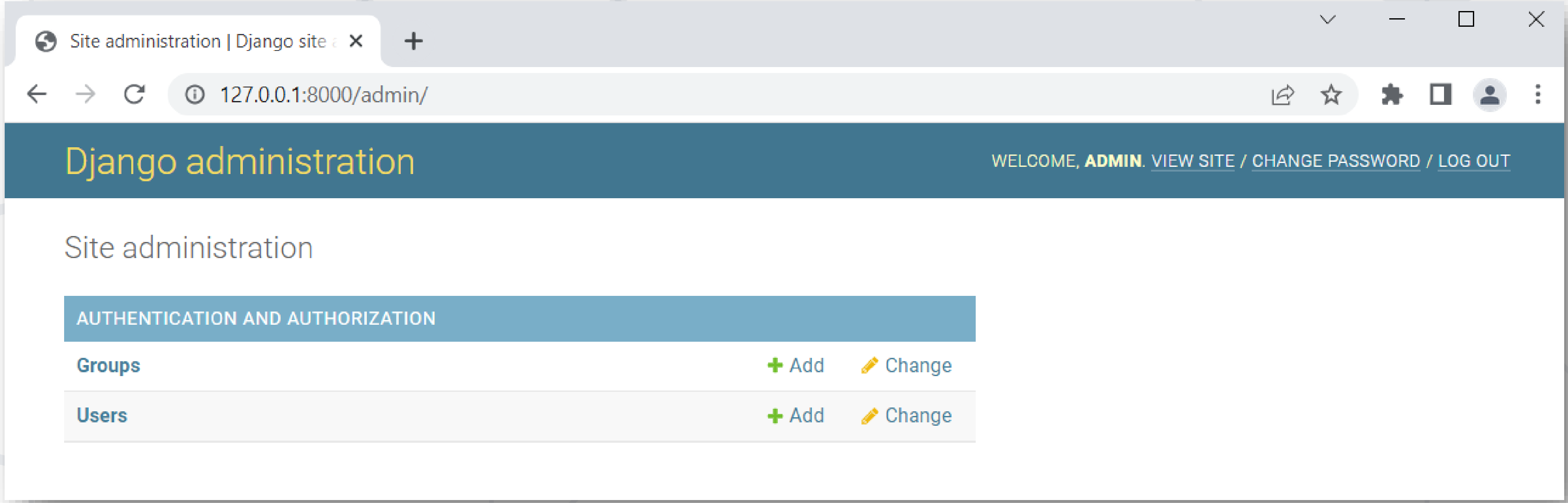


There are no created tasks!

Django Admin Site

Django Admin Site

- It is a **built-in admin** interface
  - **Trusted** users can **manage** content on the site
- One of the **benefits** of Django



Django administration

WELCOME, **ADMIN**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

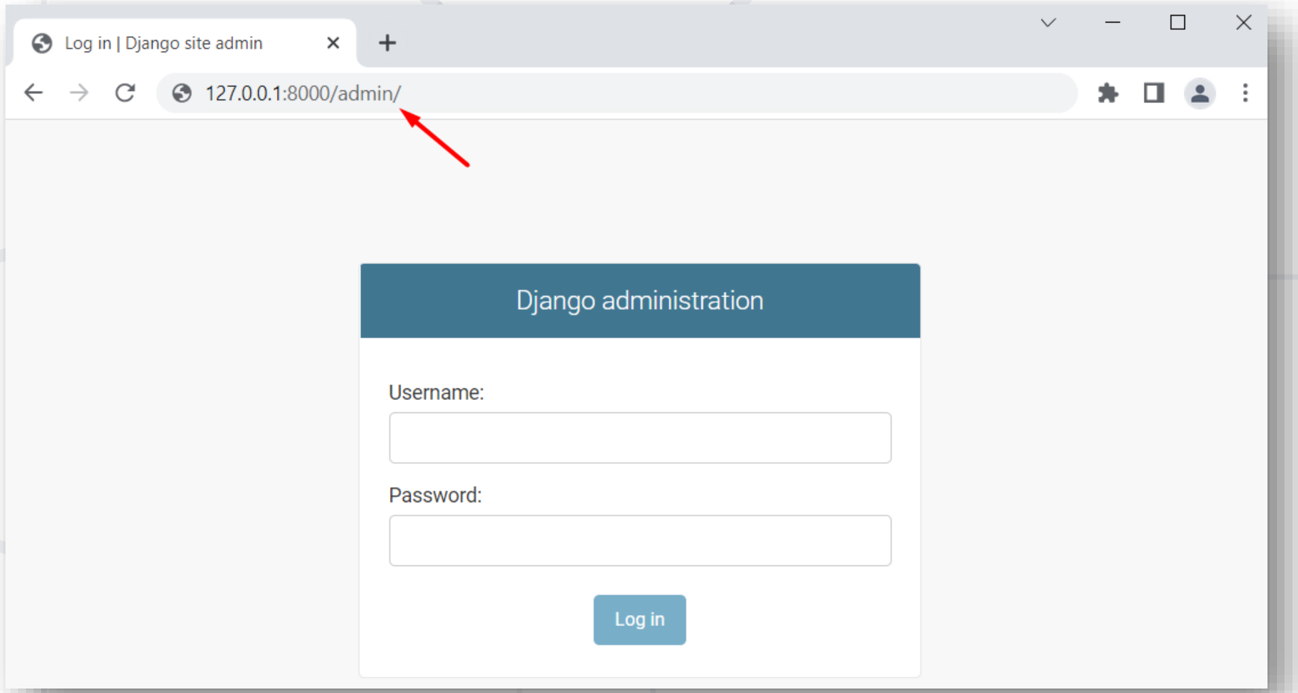
AUTHENTICATION AND AUTHORIZATION		
Groups	+ Add	Change
Users	+ Add	Change

Access Django Admin Site

- First, create a **superuser** to **log in** with

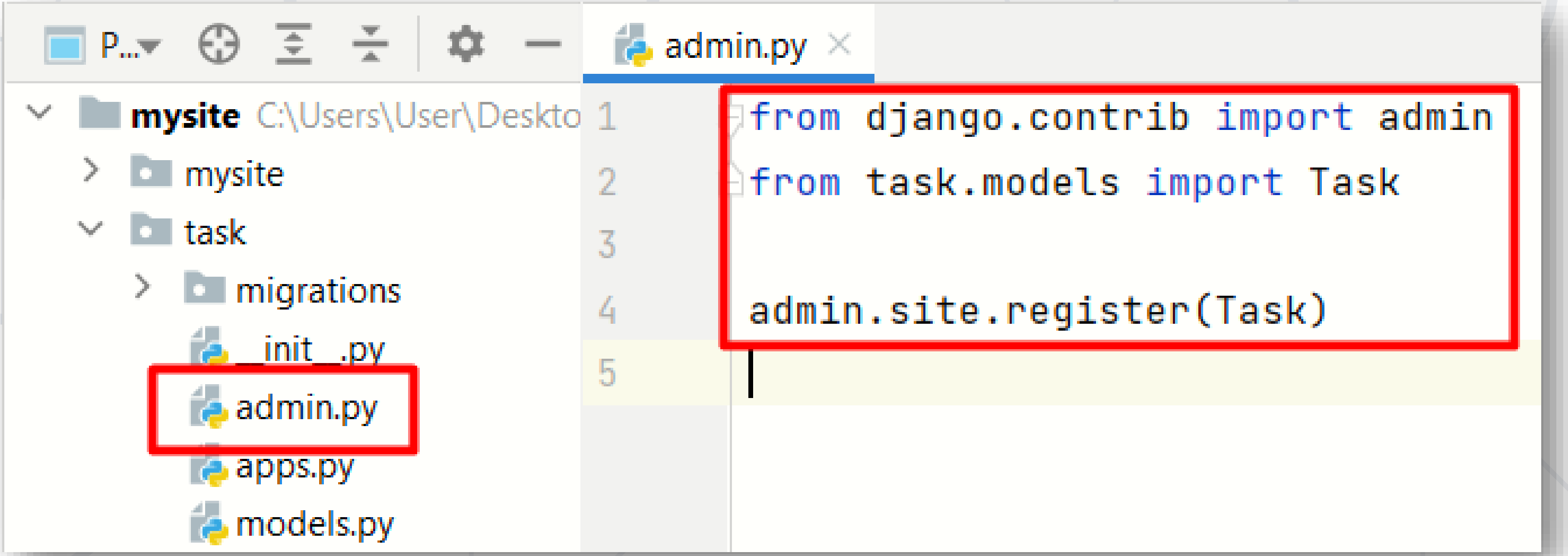
```
python manage.py createsuperuser
```

- Then, **start** the **server** and **navigate** to the **admin site**



Make Models Visible

- **Register** all models in a special file in the **app** called **`admin.py`**

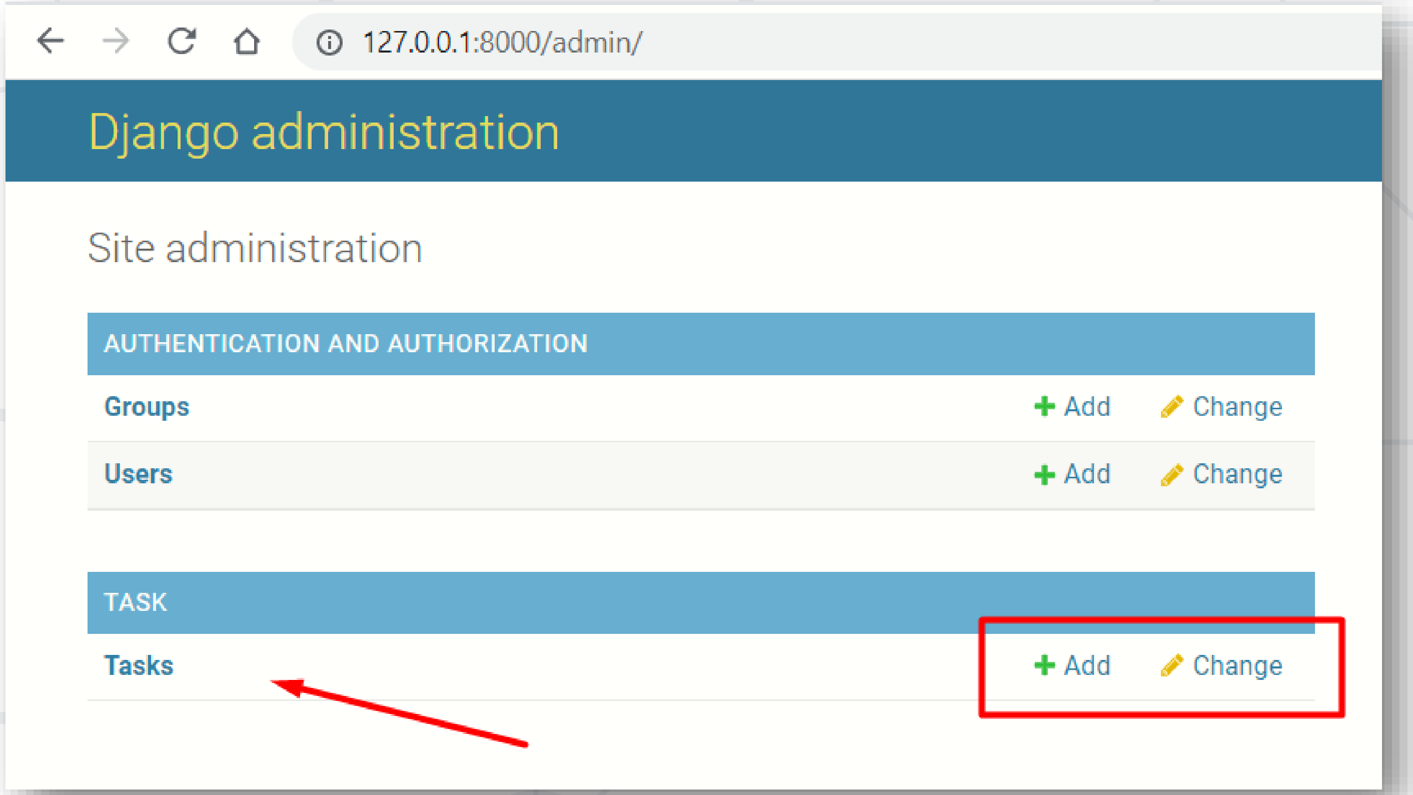


```python
from django.contrib import admin
from task.models import Task

admin.site.register(Task)
```

Django Admin Benefits

- Easily **manage** (create, update, delete) the data stored in the database



127.0.0.1:8000/admin/

Django administration

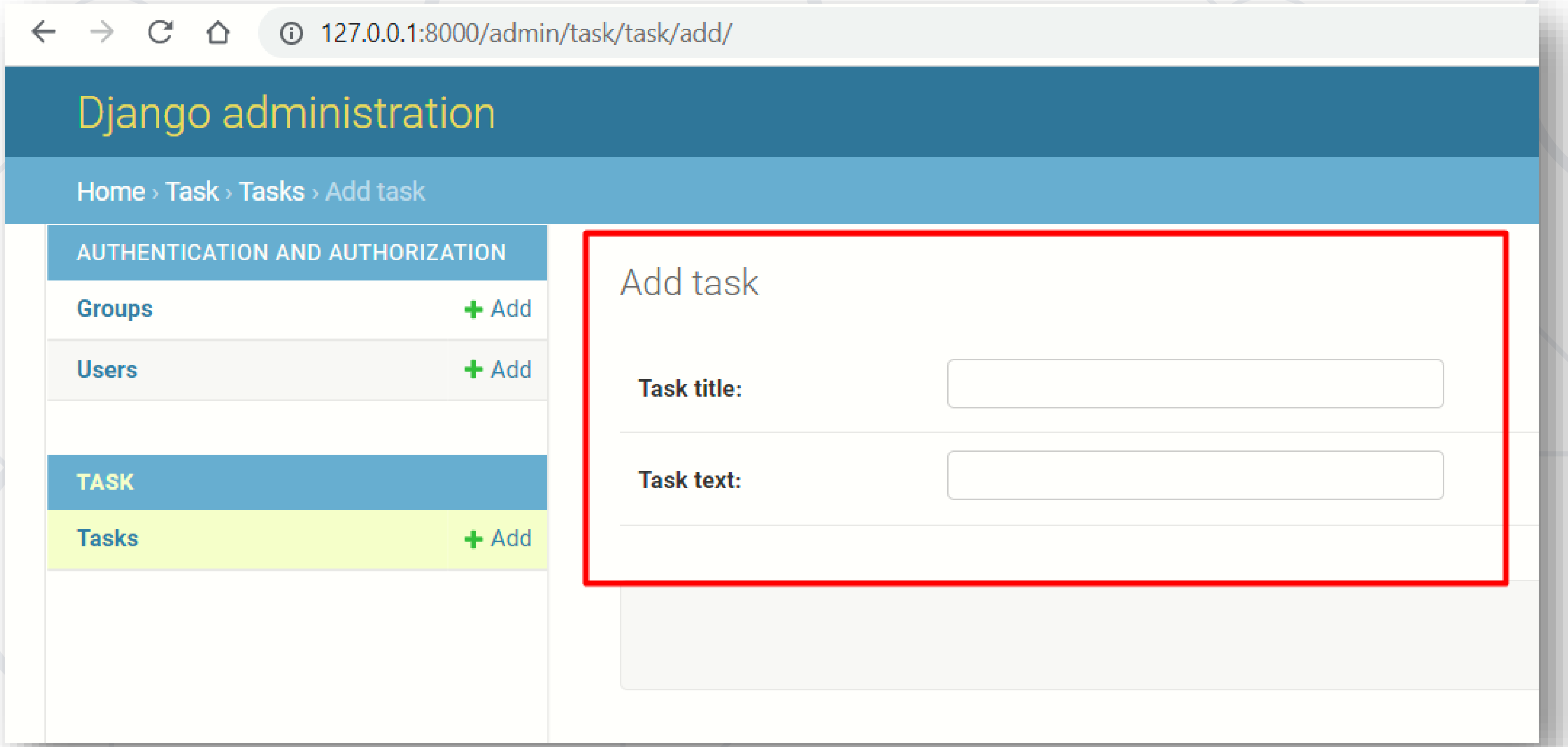
Site administration

AUTHENTICATION AND AUTHORIZATION		
Groups	Add	Change
Users	Add	Change

TASK		
Tasks	Add	Change

Django Admin Benefits

- The form is **automatically** generated



127.0.0.1:8000/admin/task/task/add/

Django administration

Home › Task › Tasks › Add task

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

TASK

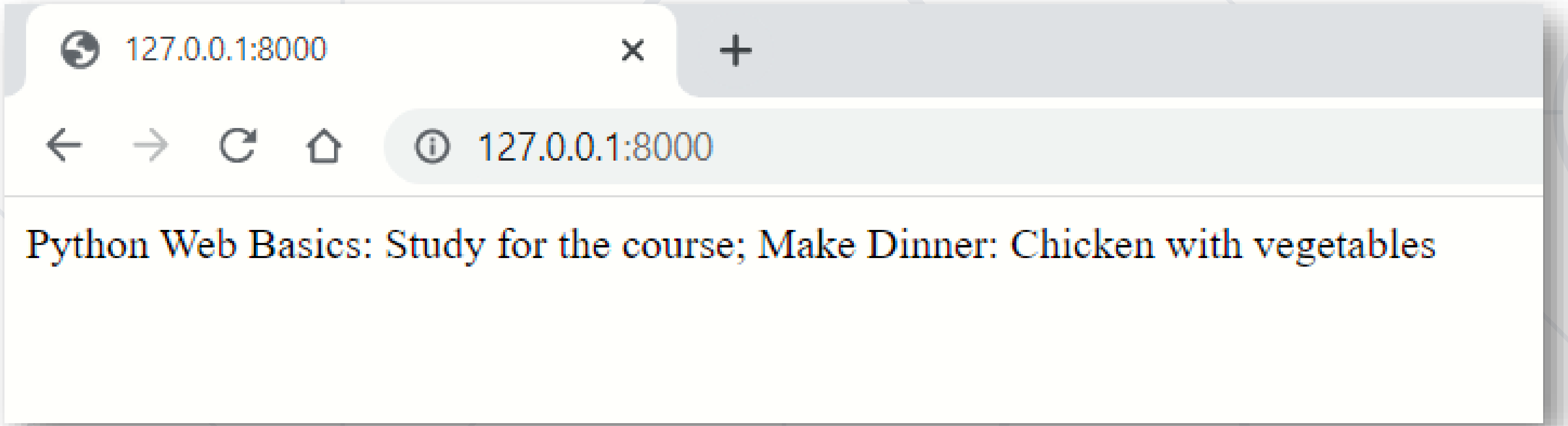
Tasks + Add

Add task

Task title:	
Task text:	

Data Visualization

- Tasks are **returned** by the view



```
Python Web Basics: Study for the course; Make Dinner: Chicken with vegetables
```

- Note: The page's design is **hard-coded in the view**

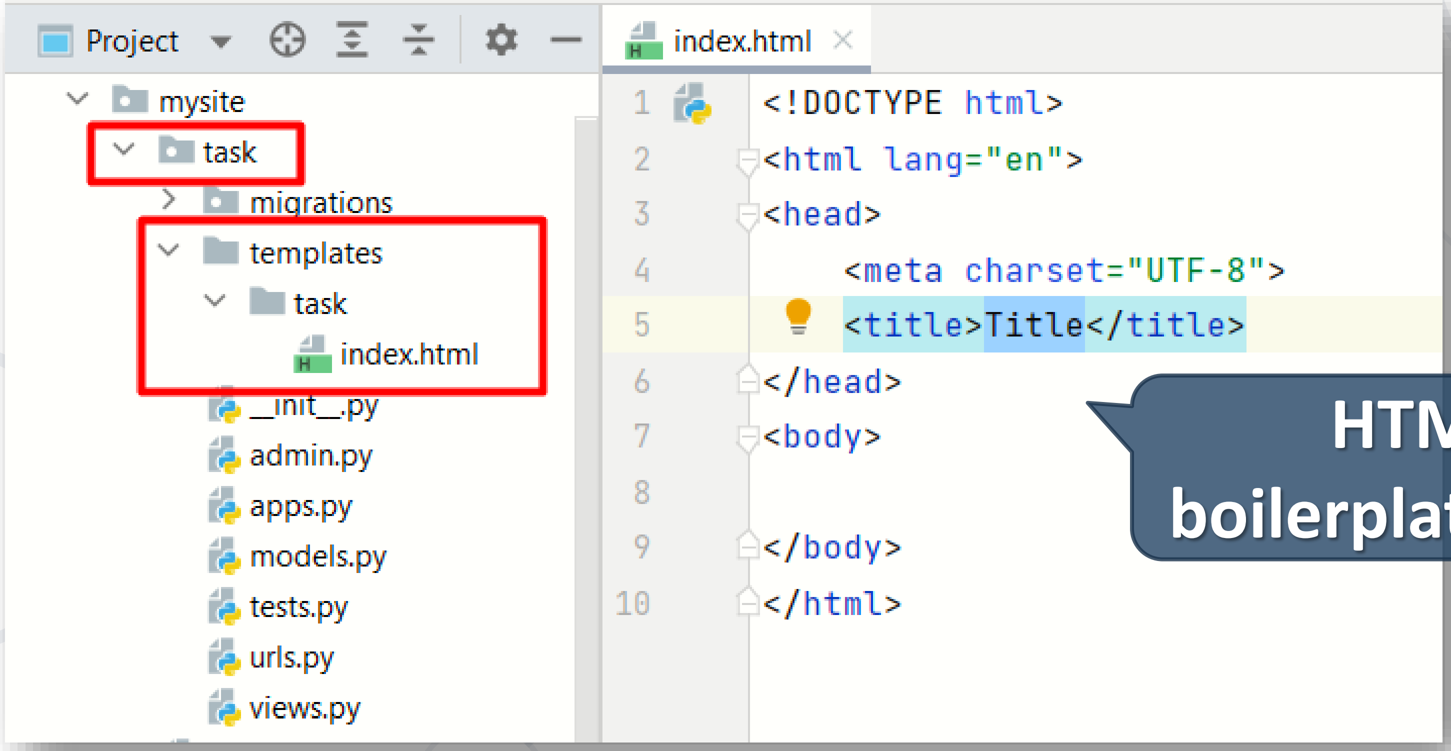
Creating a Simple Design

What is a Django Template

- A text file, written in a **special syntax** that allows a **dynamic generation** of HTML
- Serves as a **presentation layer** in the Model-View-Template (MVT) architecture
- Uses a **markup language** known as **Django Template Language** (**DTL**), which extends the standard HTML
- Plays a crucial role in **separating** the **logic** (handled by views) from the **presentation**

Creating a Templates Folder

- Create a **templates** folder inside the app's folder, and then create an HTML file named **index**.html



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

</body>
</html>
```

HTML boilerplate code

Rendering a Template

- Now that the template is created, we should refactor the **`views.py`** file in the app

task/views.py

```
from django.shortcuts import render
from task.models import Task

def index(request):
    return render(request, 'task/index.html')
```

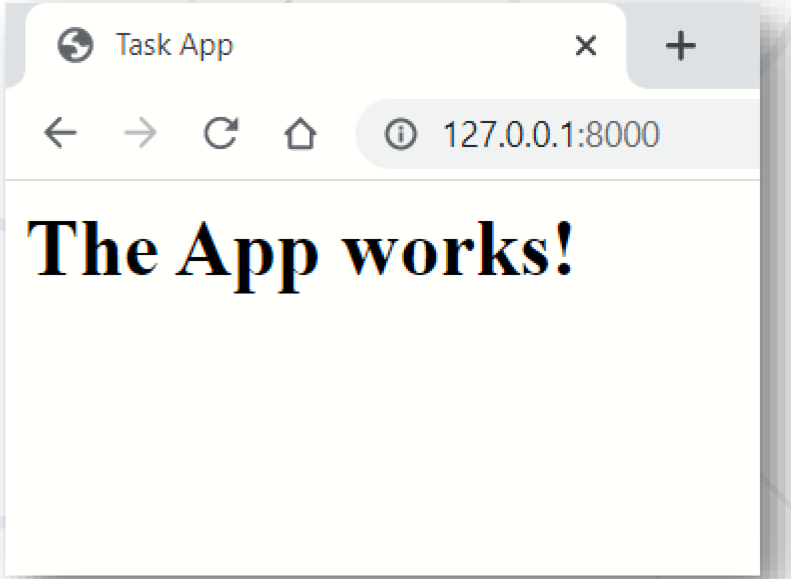
- Instead of using **HttpResponse**, we can now use the **render** function that will display the created template

Creating a Template

- Creating an `.html` file that will be a template

index.html

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Task App</title>
</head>
<body>
    <h1>The App works!</h1>
</body>
</html>
```



The App works!

Adding a Context

- The **render** function accepts a **context** as an argument
  - It is a **dictionary** passed to the template and used to display data **dynamically**

task/views.py

```
from django.shortcuts import render
from task.models import Task

def index(request):
    tasks_list = Task.objects.all()
    context = {'tasks_list': tasks_list}
    return render(request, 'task/index.html', context)
```

Basic Template Logic

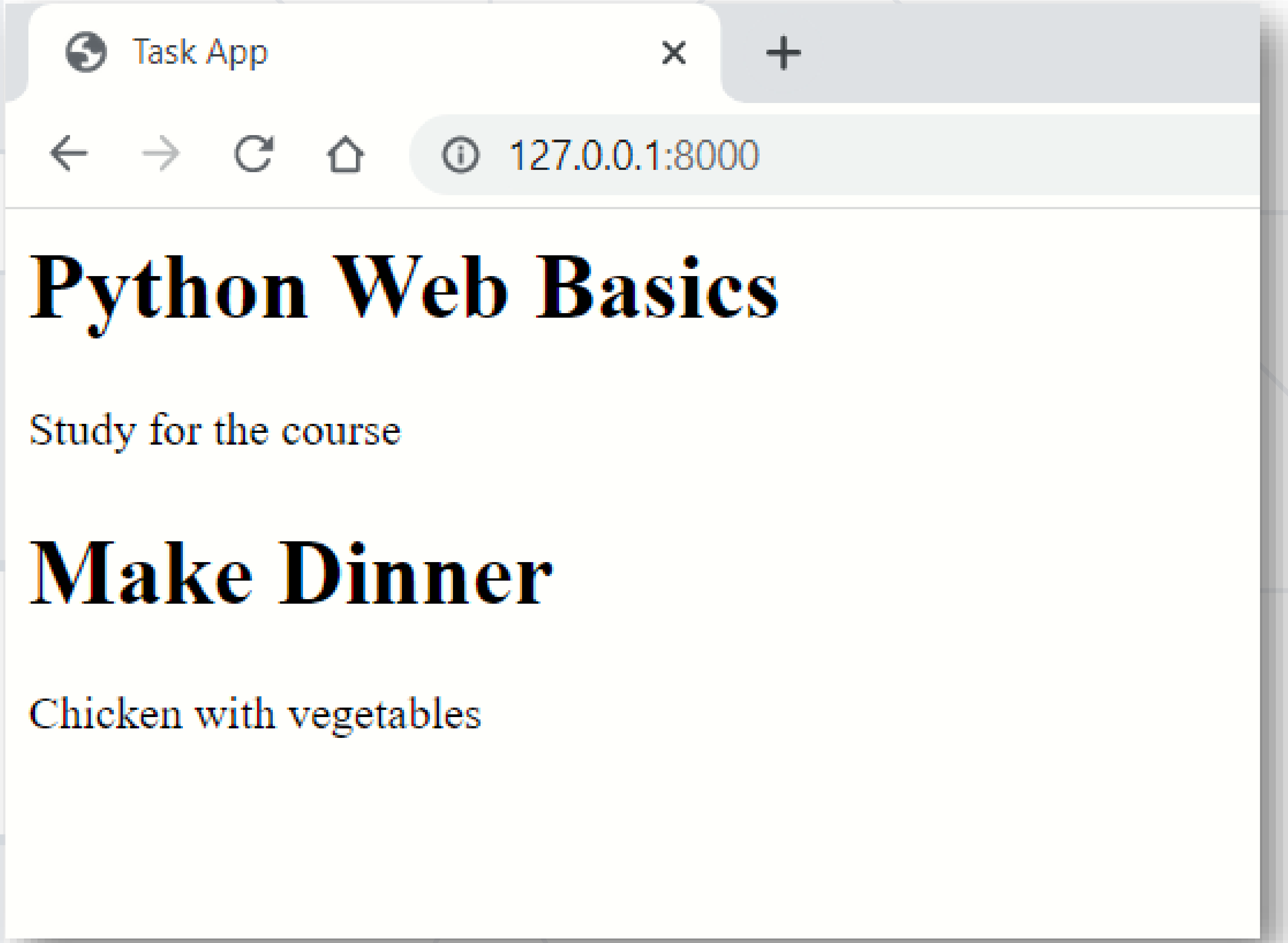
- We can have a simple **logic** using built-in **template tags**

index.html

```
{% if tasks_list %}
{% for task in tasks_list %}
<h1>{{ task.task_title }}</h1>
<p>{{ task.task_text }}</p>
{% endfor %}
{% else %}
<p>There are no created tasks!</p>
{% endif %}
```

If a "**tasks_list**" is not empty, then return a title and a text for each task

Simple Task App



Python Web Basics

Study for the course

Make Dinner

Chicken with vegetables

Summary

- Django is a **high-level** Web Framework
- Django Project can contain **multiple** apps
- Django **views**
  - Context
- Django **templates**
  - Django Template Language (**DTL**)
  - Template **tags**

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

СофтУни БУДИТЕЛ

Finance Academy

SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

THE CROWN IS YOURS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License