

# Django Introduction



SoftUni Team  
Technical Trainers



**SoftUni**



Software University

<https://softuni.bg>

sli.do

**#python-web**

# Table of Contents

1. Django Framework
2. Creating a **Django Project**
3. Creating a **Django Application**
4. Setting up a **Database**
5. Writing a Simple **Task App**
6. Creating a Simple **Design**





# **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**)





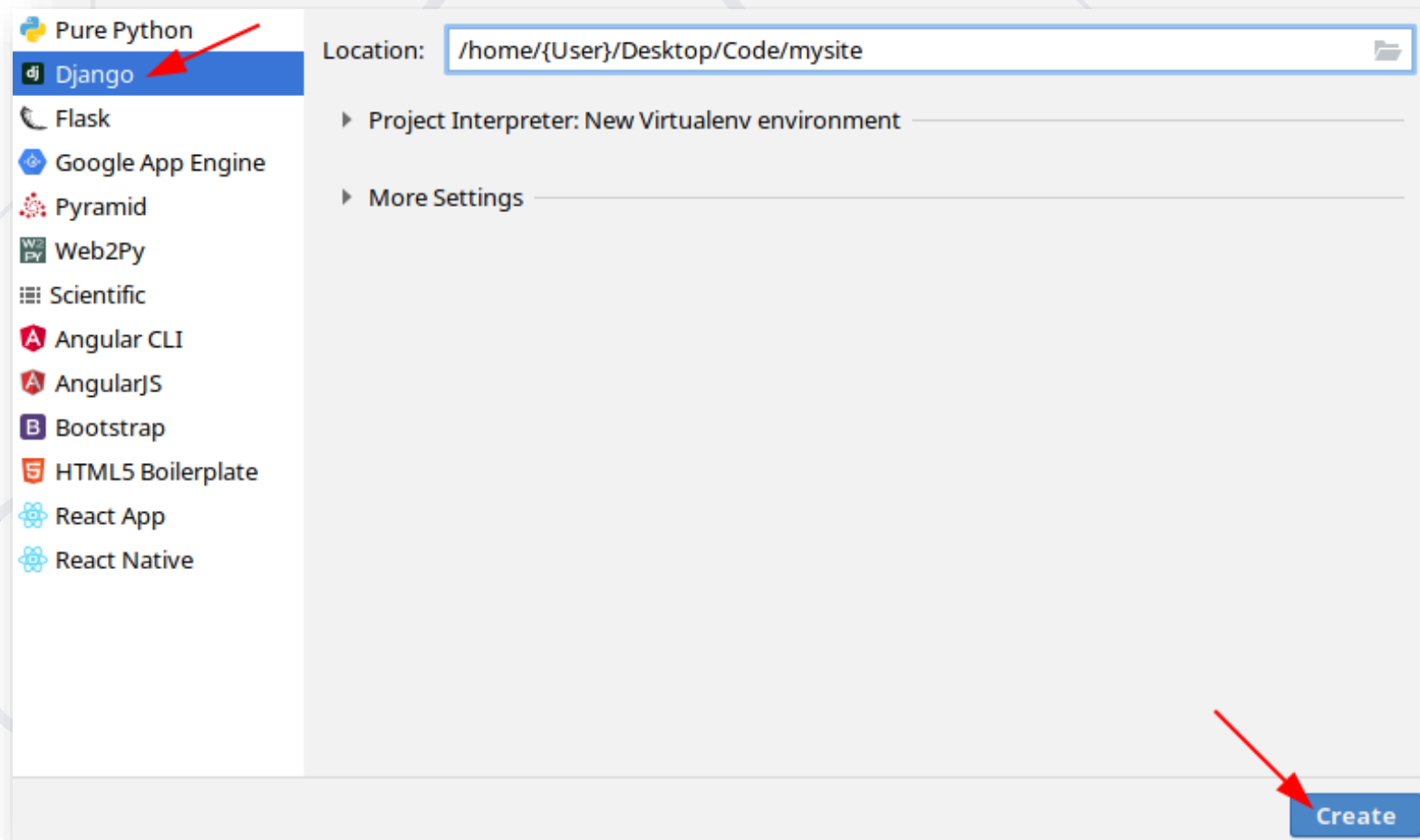
# Creating a Django Project

Where the magic happens



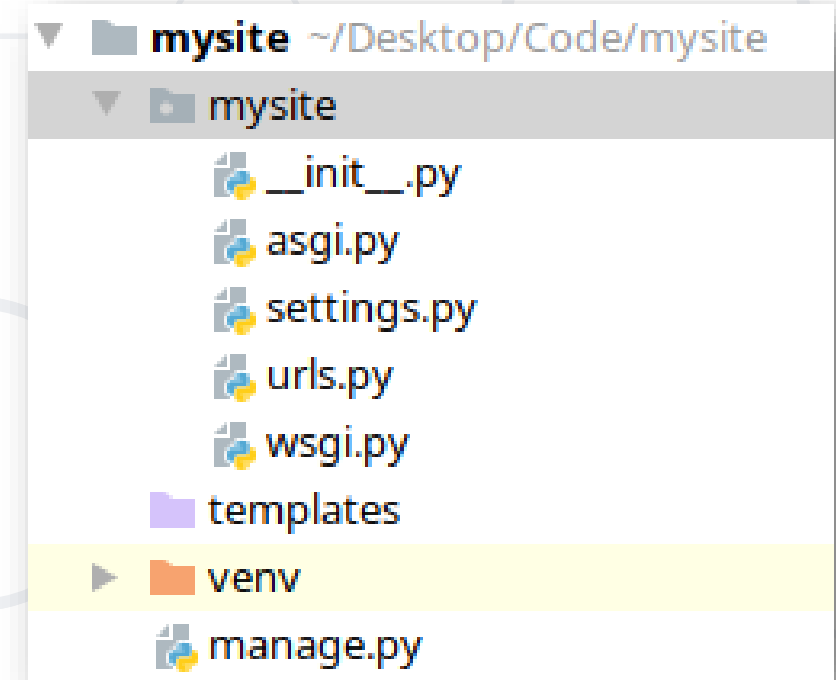
# Creating a Django Project

- Open PyCharm Professional -> File -> New Project



# 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



# Running a Django Project

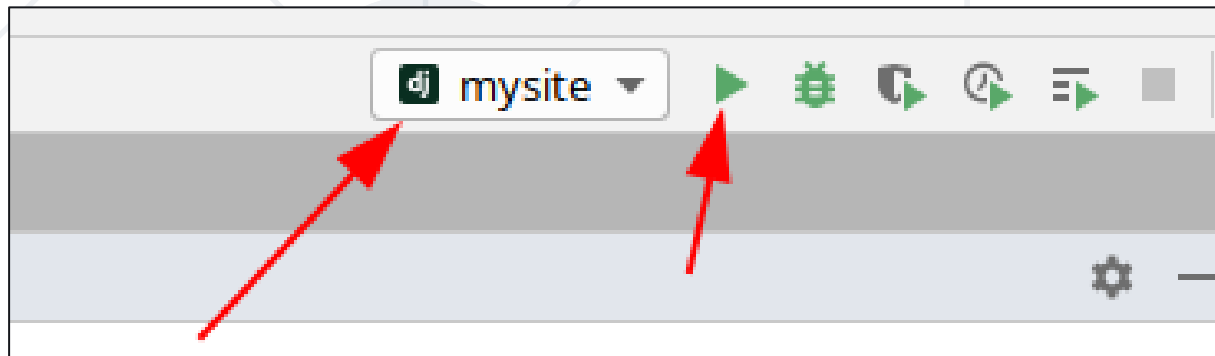
- Using **Terminal command**

```
python manage.py runserver
```

- Using **Keyboard Shortcut** in PyCharm

Shift + F10

- Using PyCharm **Run button**



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

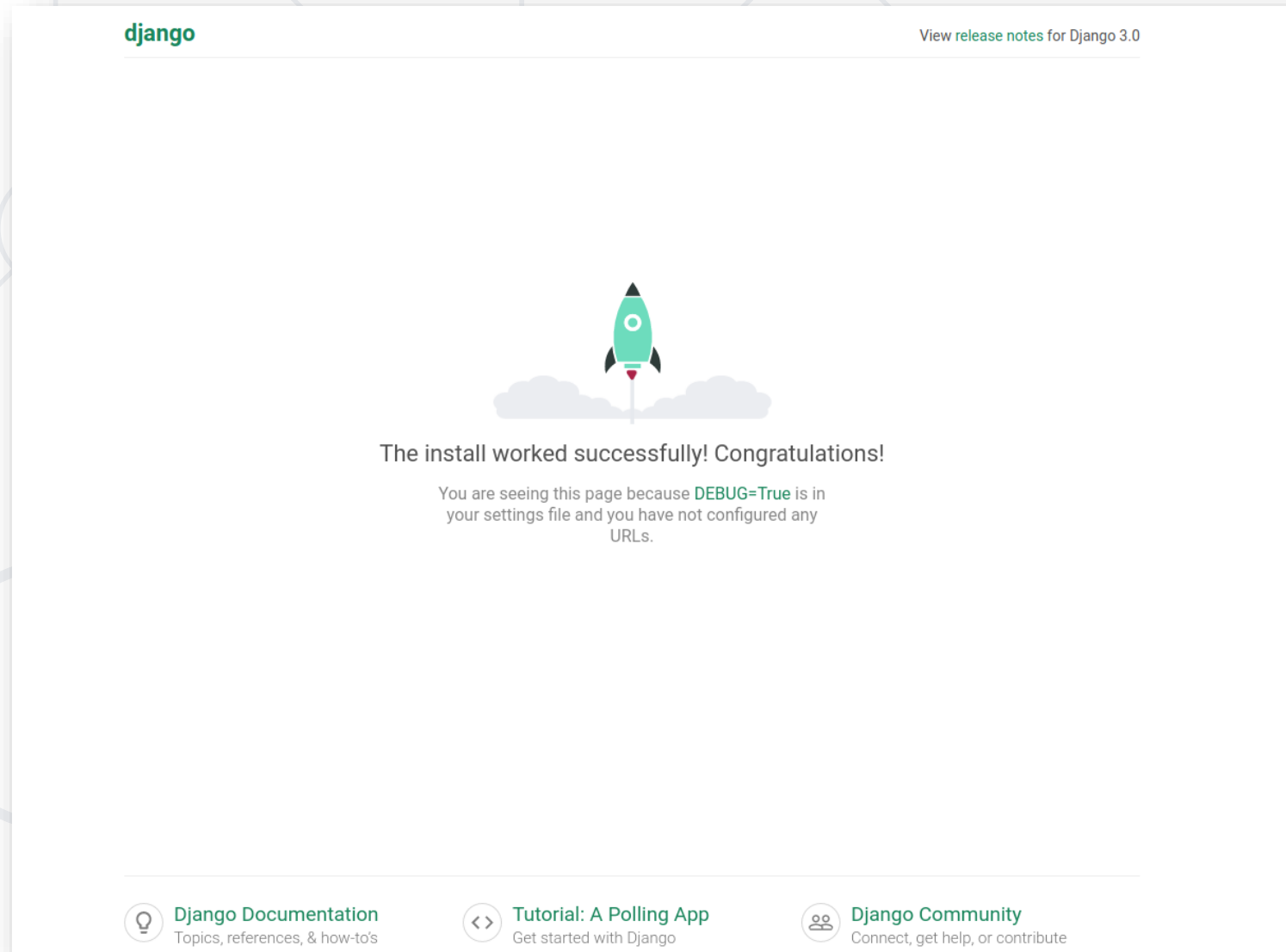
```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 17 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
February 18, 2020 - 11:15:18
Django version 3.0.3, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

- 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 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
```

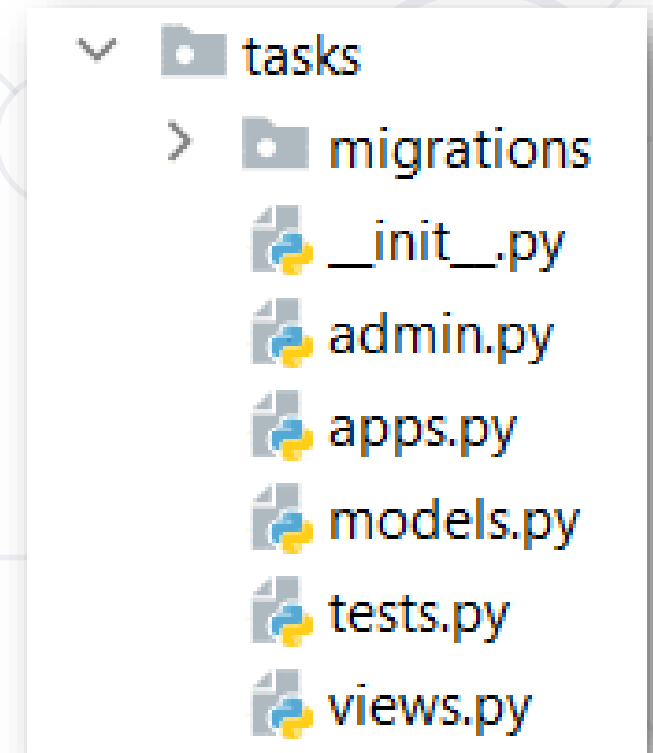
- Move it **inside the project** for a **better-structured** project management
- Django **automatically** generates the **basic directory structure** of an app





# 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



# Including an App

- To include an app in a project, **add a reference** to the app in the **INSTALLED\_APPS** setting



```
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'mysite.tasks',
41 ]
```



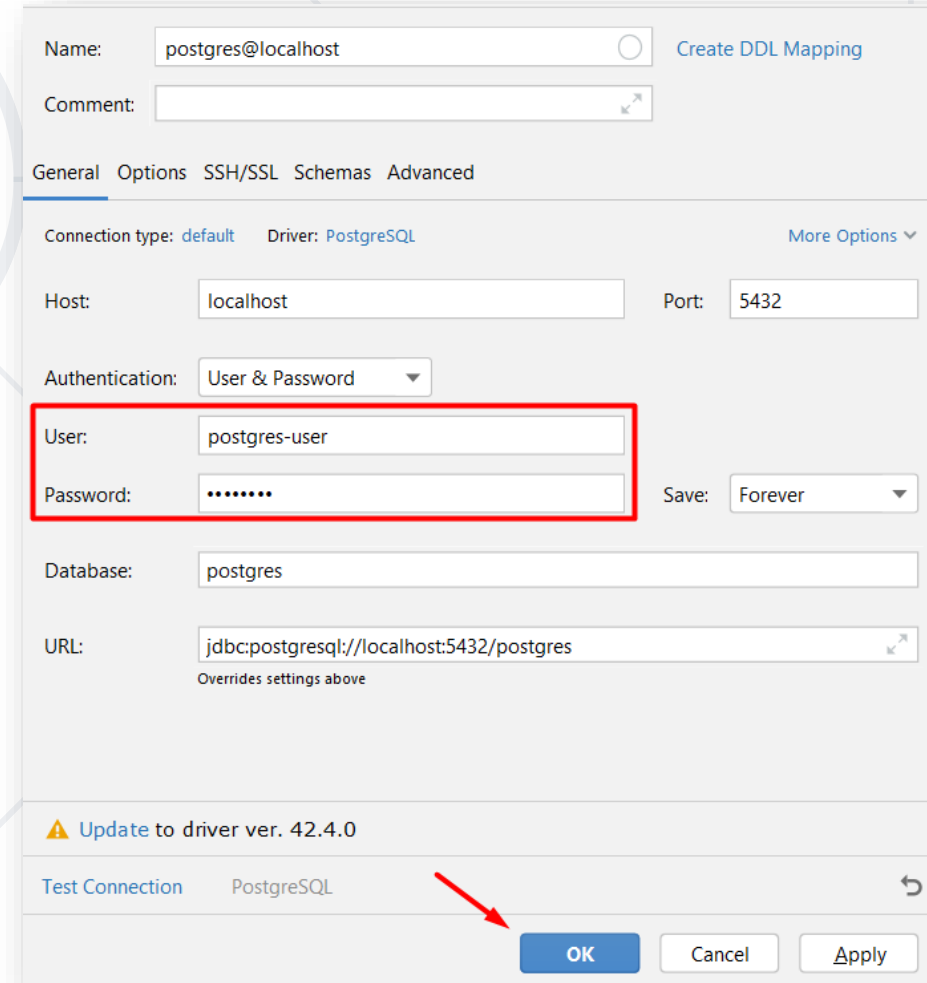
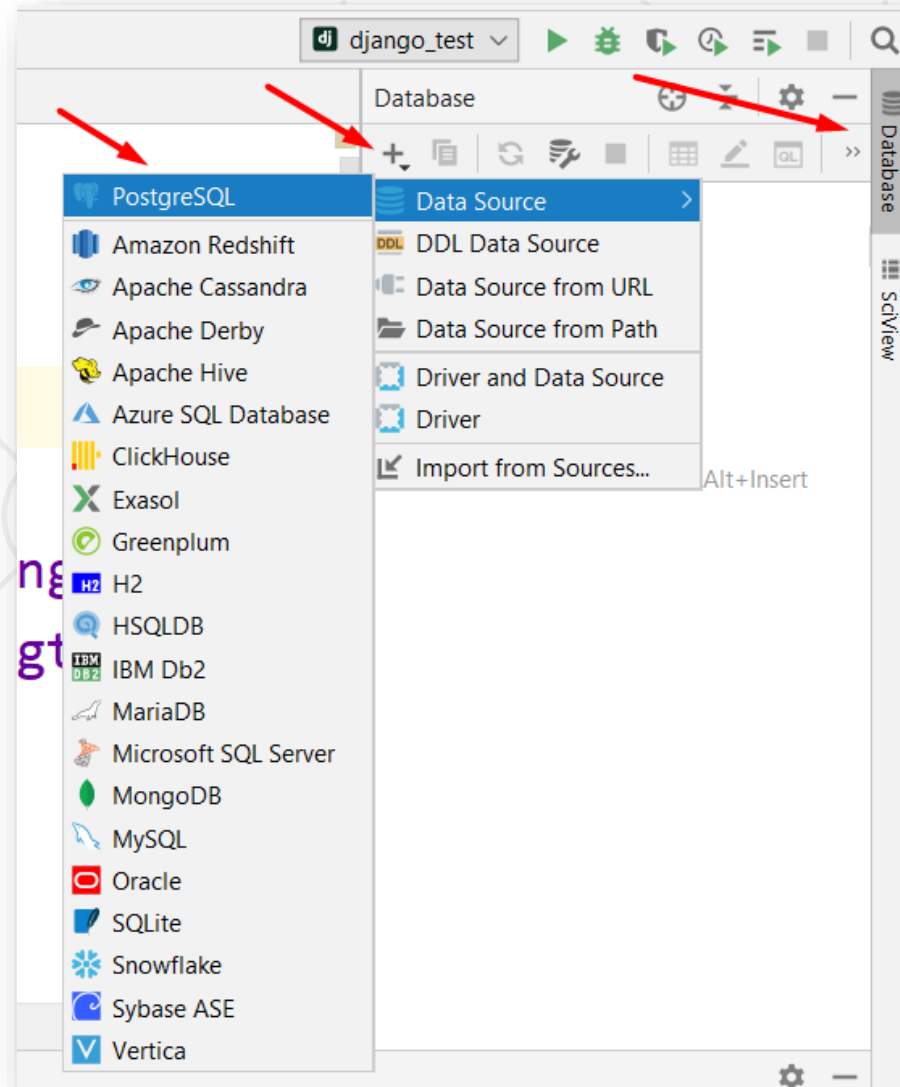
# Setting up a Database

# 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

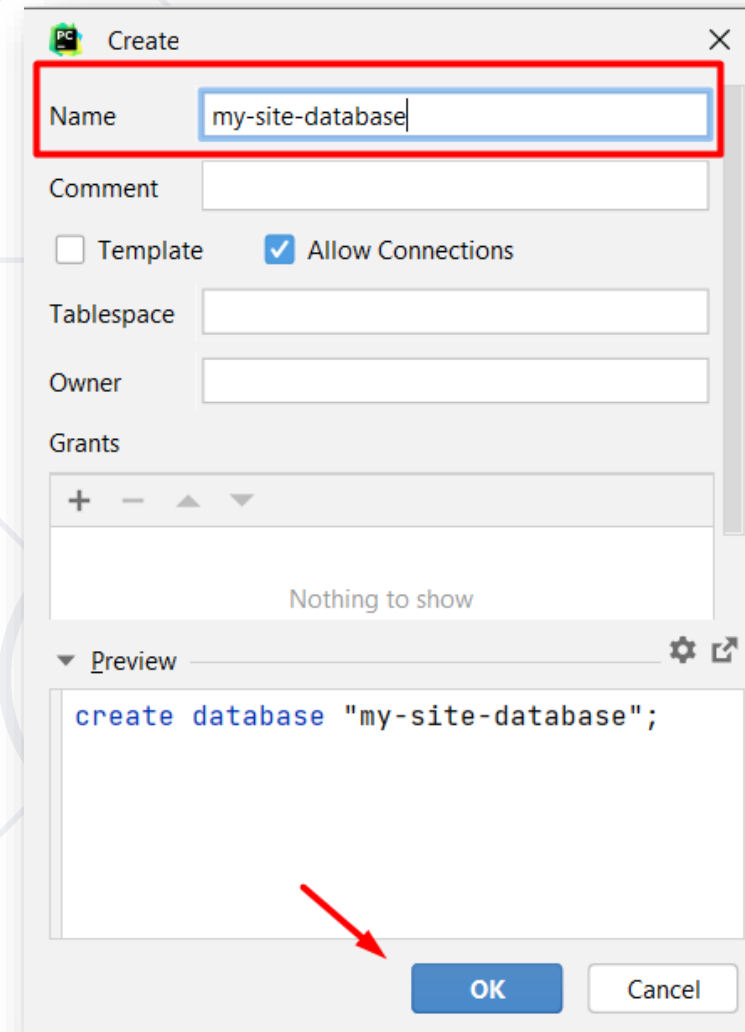
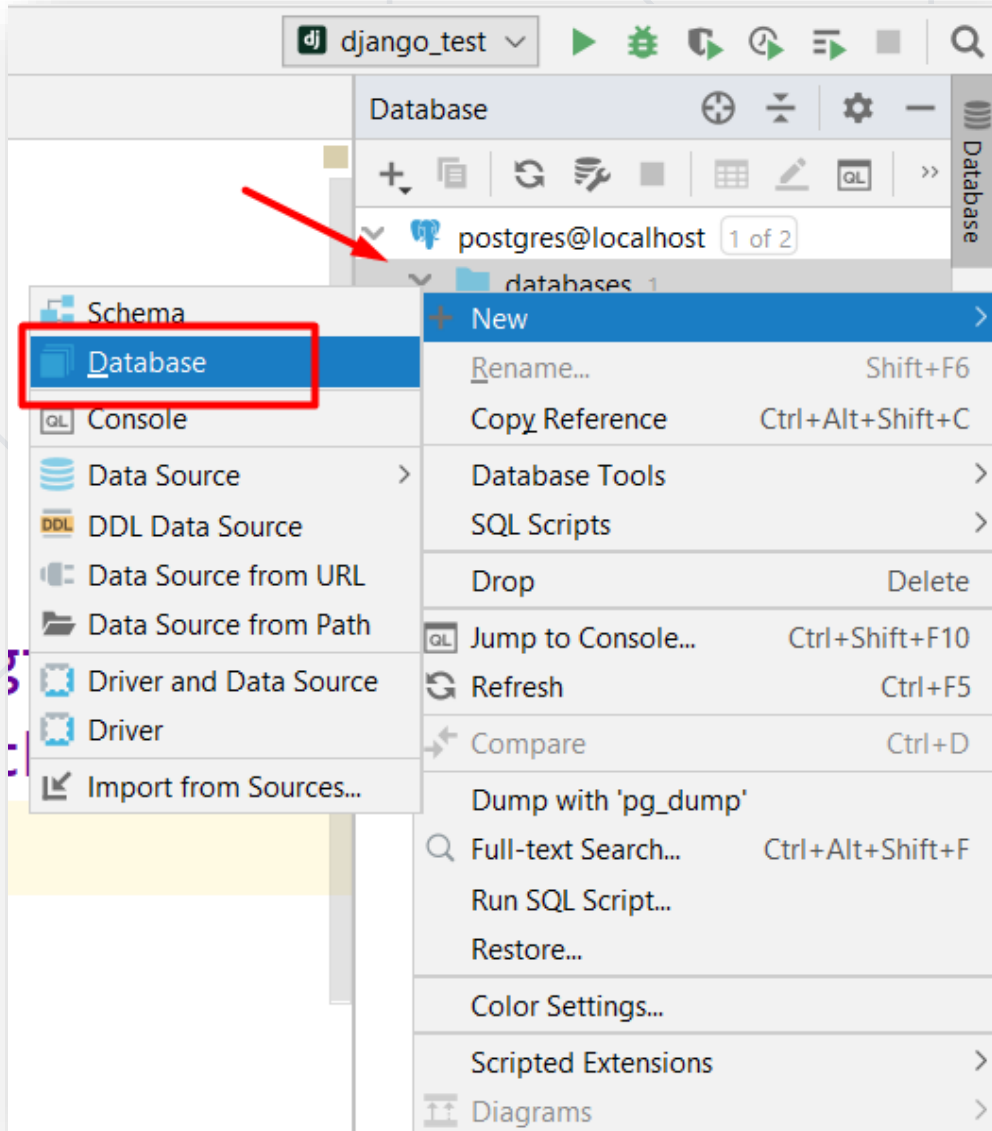


# Connect to PostgreSQL



A screenshot of the PostgreSQL connection configuration dialog. The dialog has tabs for 'General', 'Options', 'SSH/SSL', 'Schemas', and 'Advanced'. The 'General' tab is selected. Fields include: Name (postgres@localhost), Comment, Connection type (default), Driver (PostgreSQL), Host (localhost), Port (5432), Authentication (User & Password), User (postgres-user), Password (masked with dots), Database (postgres), and URL (jdbc:postgresql://localhost:5432/postgres). A red box highlights the 'User' and 'Password' fields. At the bottom, there is a 'Test Connection' button, a status indicator for 'PostgreSQL', and 'OK', 'Cancel', and 'Apply' buttons. A red arrow points to the 'OK' button. A warning message at the bottom says 'Update to driver ver. 42.4.0'.

# Create a Database



- 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'  
    }  
}
```

Name of the  
database

Use PostgreSQL

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
```

Model Name

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

Field Types

Fields

- 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 HttpResponseRedirect
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 HttpResponseRedirect(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



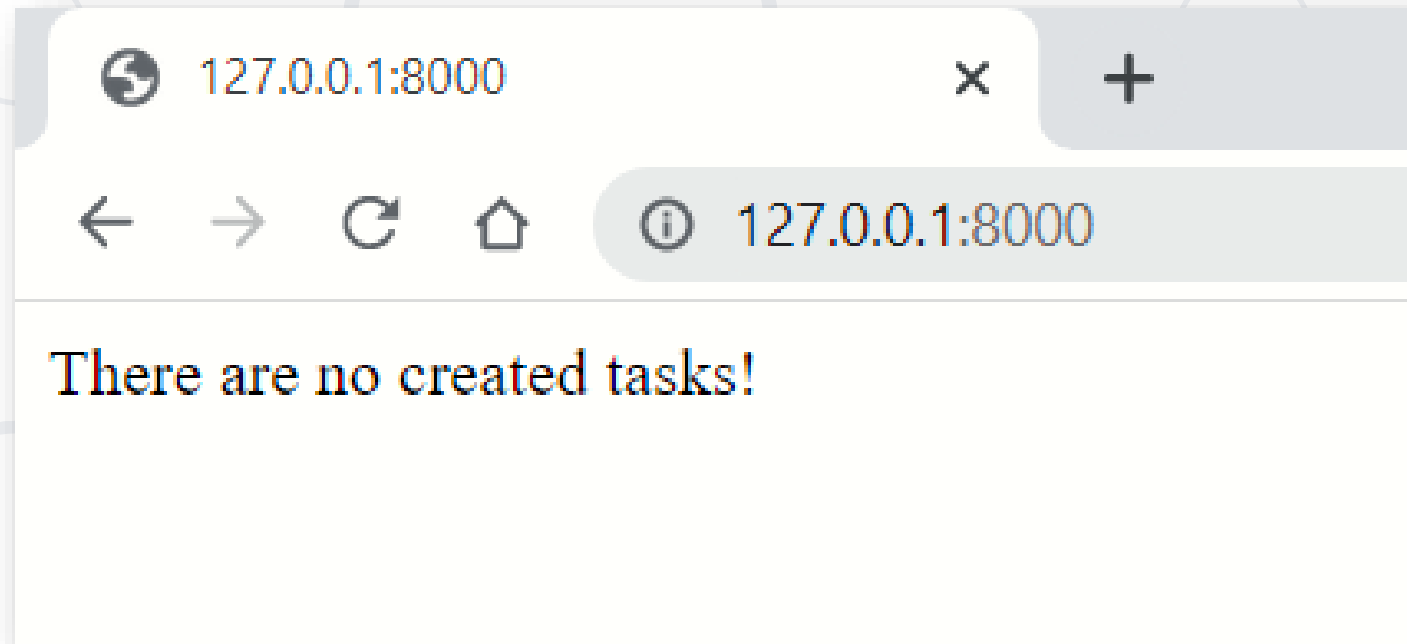
mysite/urls.py

```
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



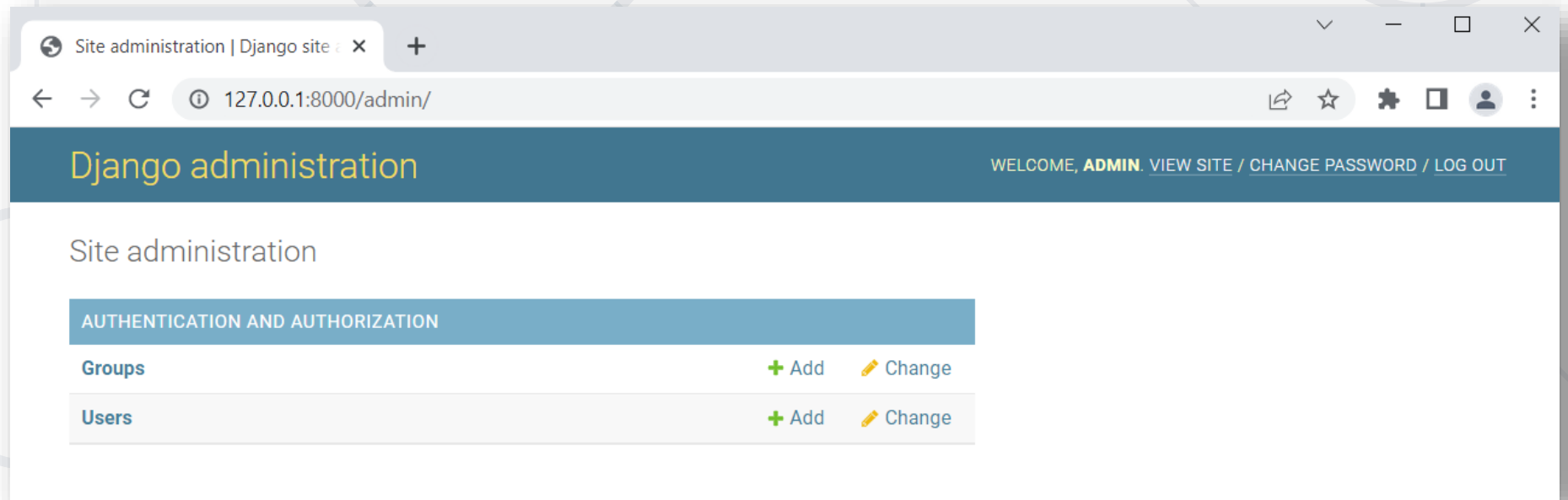




# Django Admin Site

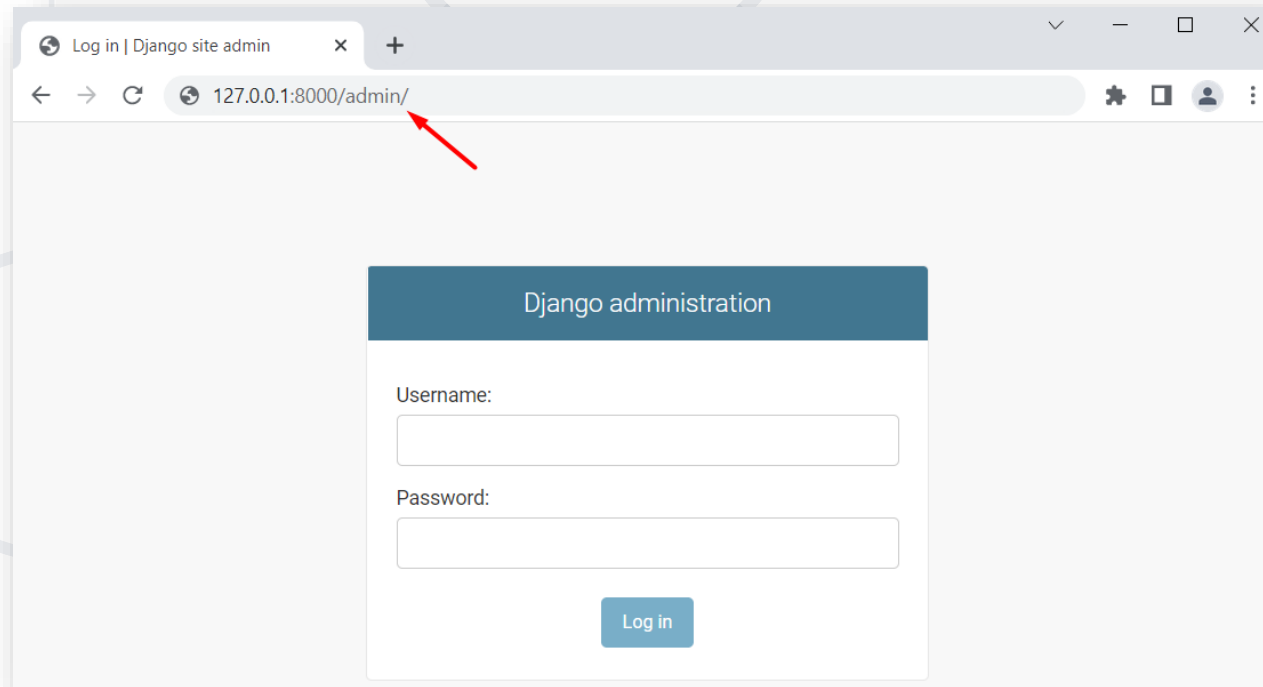
# Django Admin Site

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



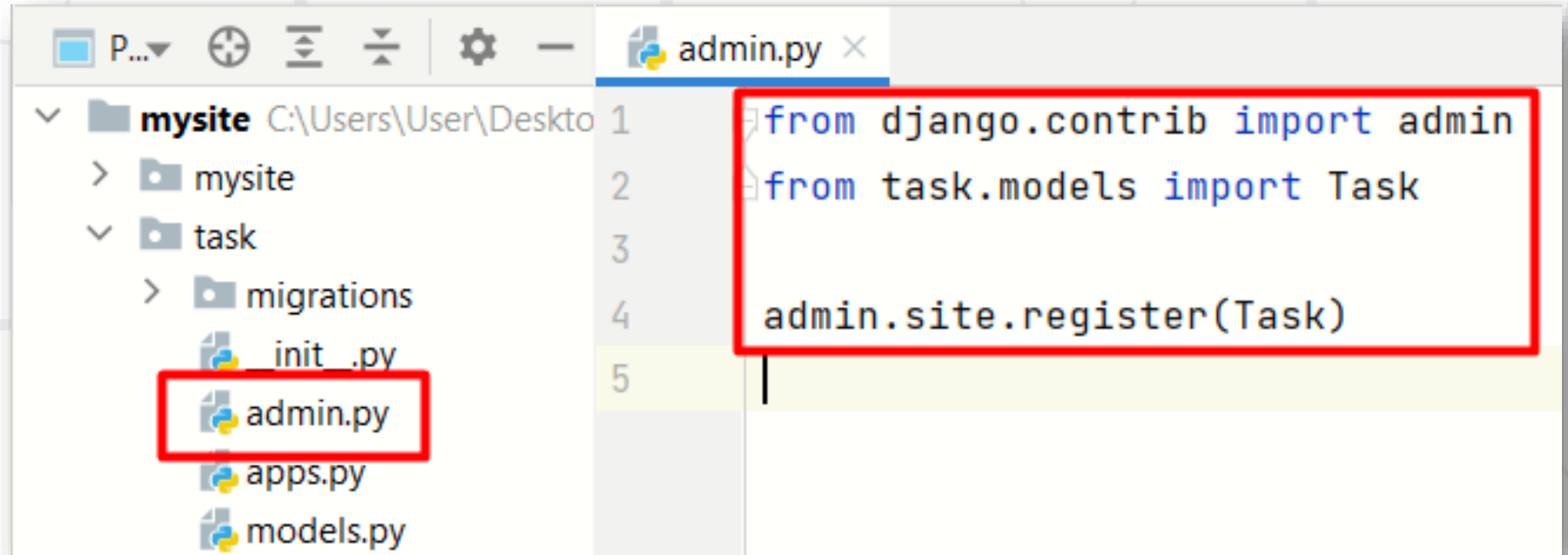
# Access Django Admin Site

- First, create a **superuser** to **log in** with  
`python manage.py createsuperuser`
- Then, **start** the **server** and **navigate** to the **admin site**



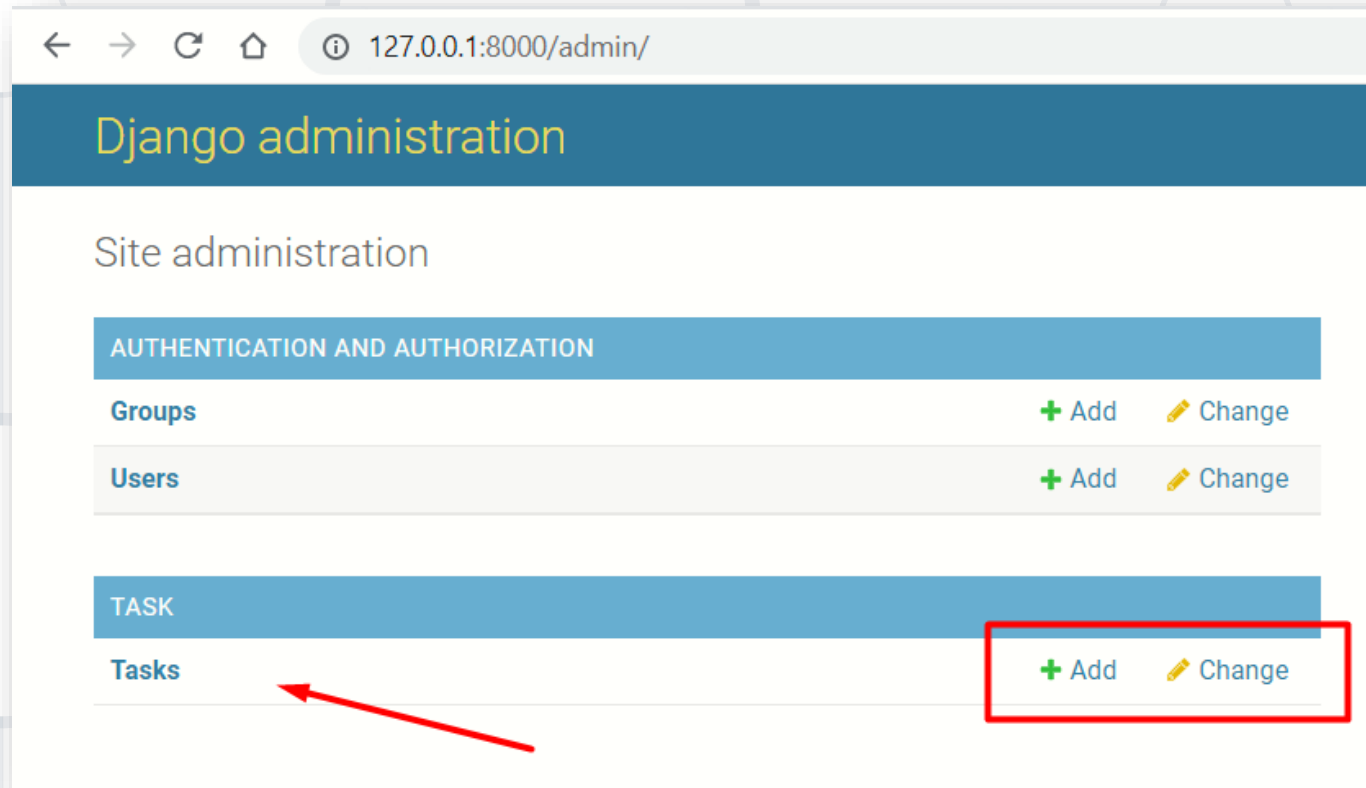
# Make Models Visible

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

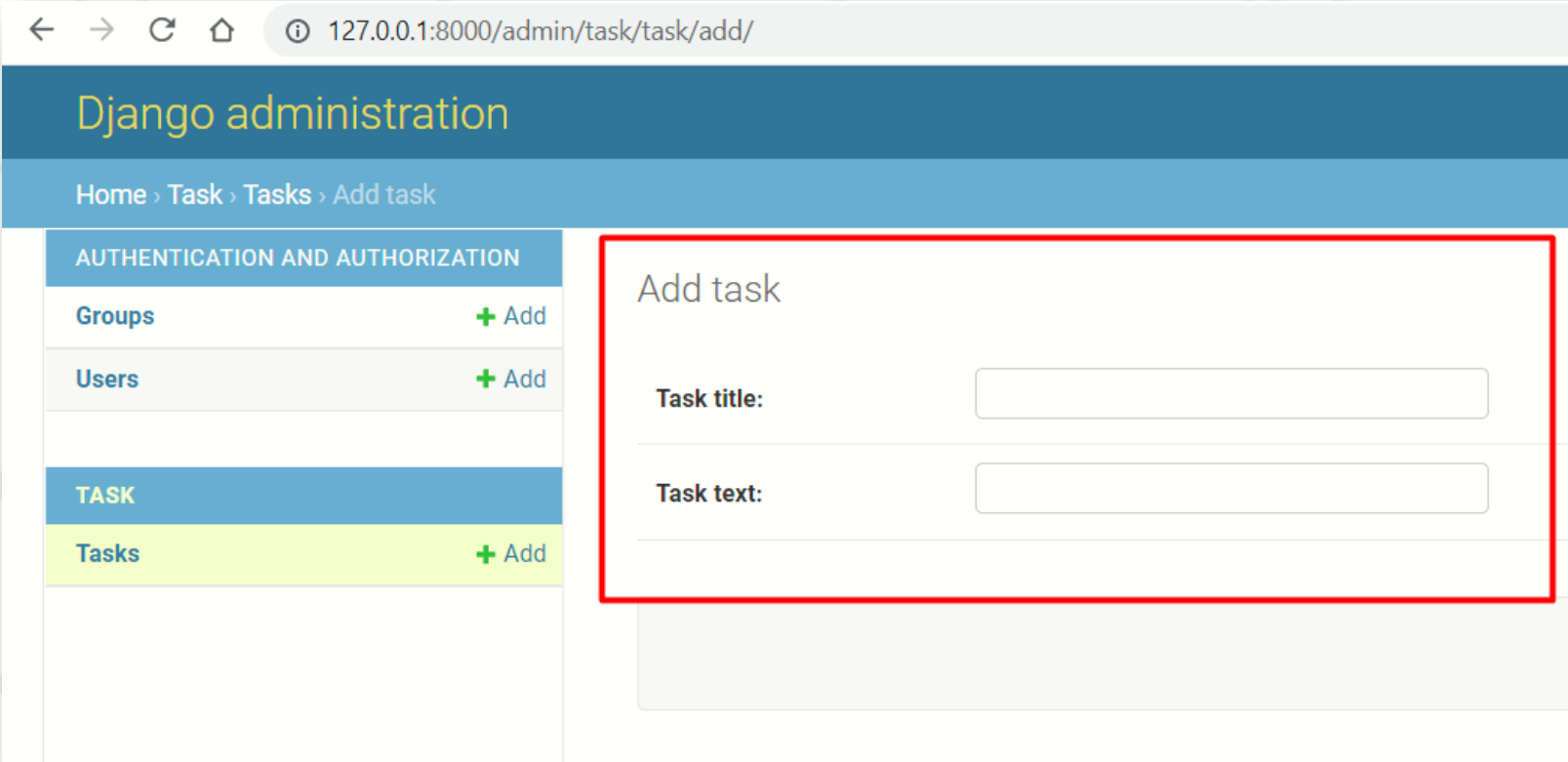


```
1 from django.contrib import admin
2 from task.models import Task
3
4 admin.site.register(Task)
5
```

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

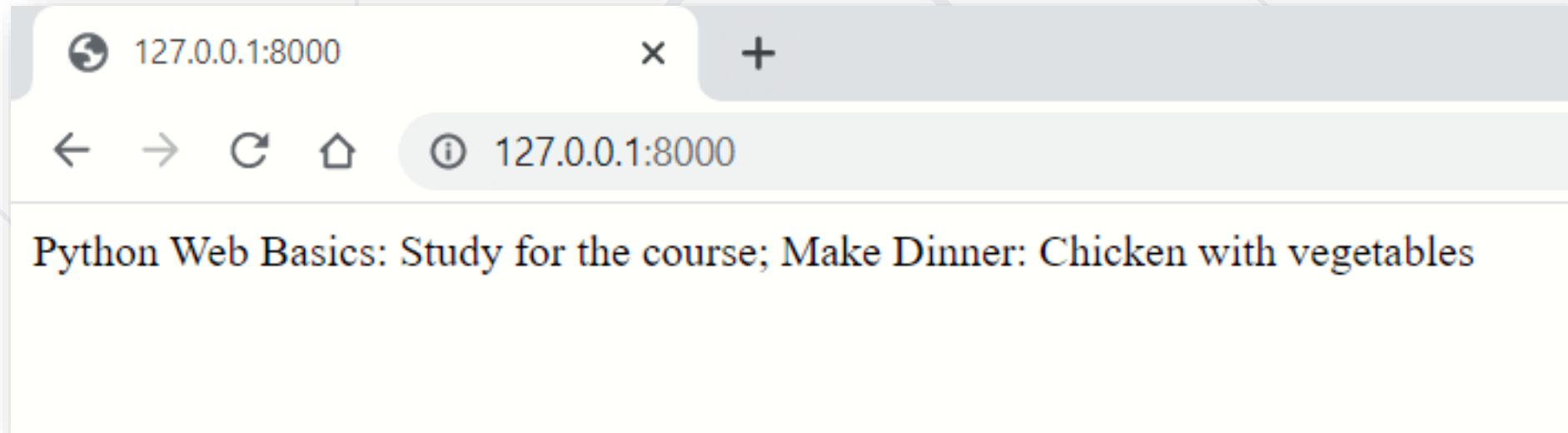


- The form is **automatically** generated



The screenshot shows the Django Admin interface in a web browser. The address bar displays the URL `127.0.0.1:8000/admin/task/task/add/`. The page title is "Django administration". The breadcrumb trail is "Home > Task > Tasks > Add task". The left sidebar contains a menu with "AUTHENTICATION AND AUTHORIZATION" (Groups, Users) and "TASK" (Tasks). The main content area, titled "Add task", contains two form fields: "Task title:" and "Task text:", each with an adjacent text input box. A red rectangular border highlights the "Add task" form area.

- Tasks are **returned** by the view



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



# Creating a Simple Design



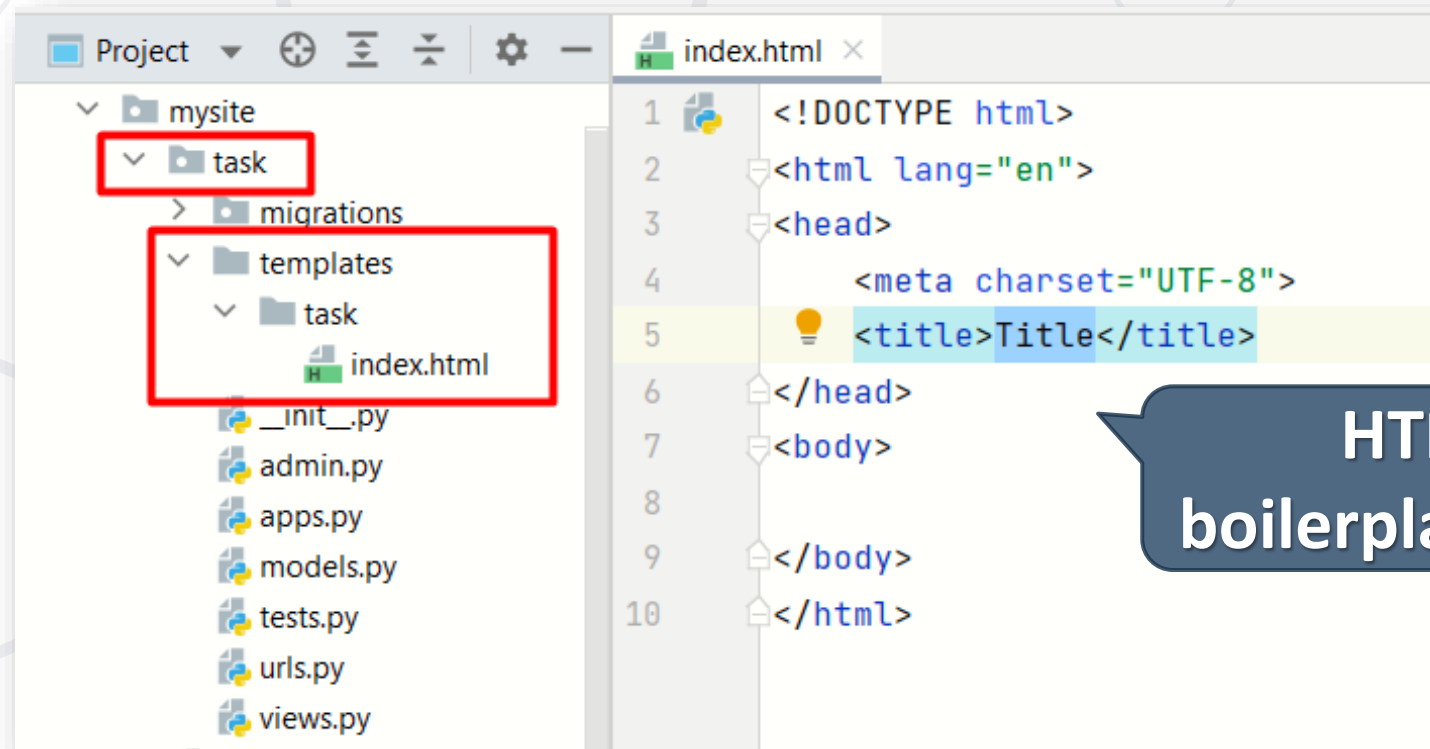
# What is a Django Template

- 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**



HTML  
boilerplate code

- 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')
```

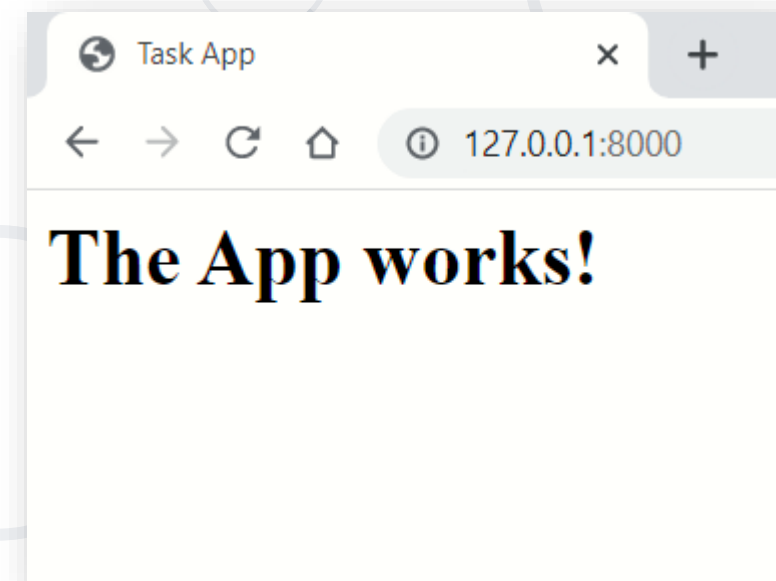
- Instead of using **HttpResponse**, we can now use the **render** function that will display the created template

# 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>
```



# 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

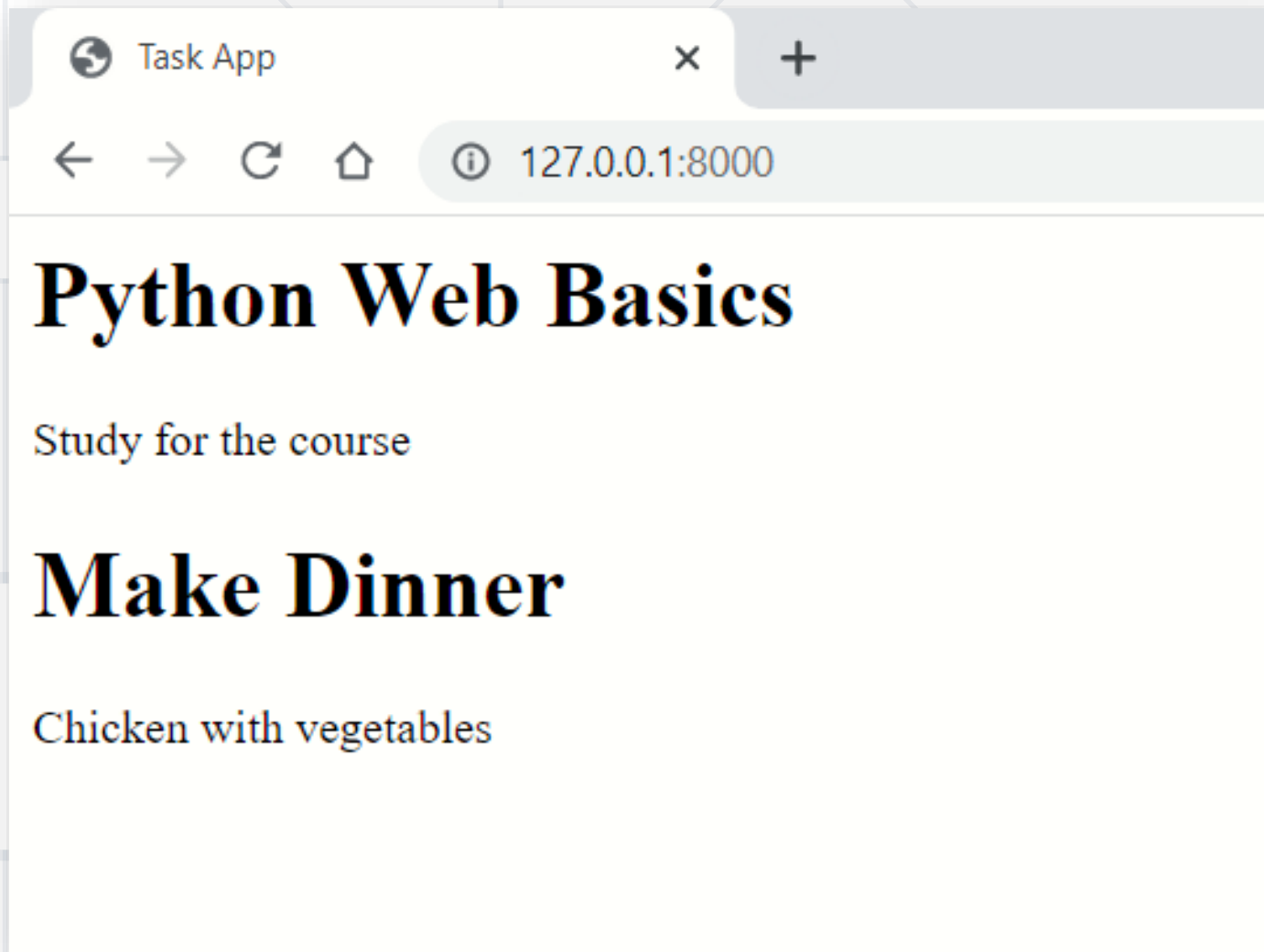
def index(request):
    tasks_list = Task.objects.all()
    context = {'tasks_list': tasks_list}
    return render(request, 'task/index.html', context)
```

- 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

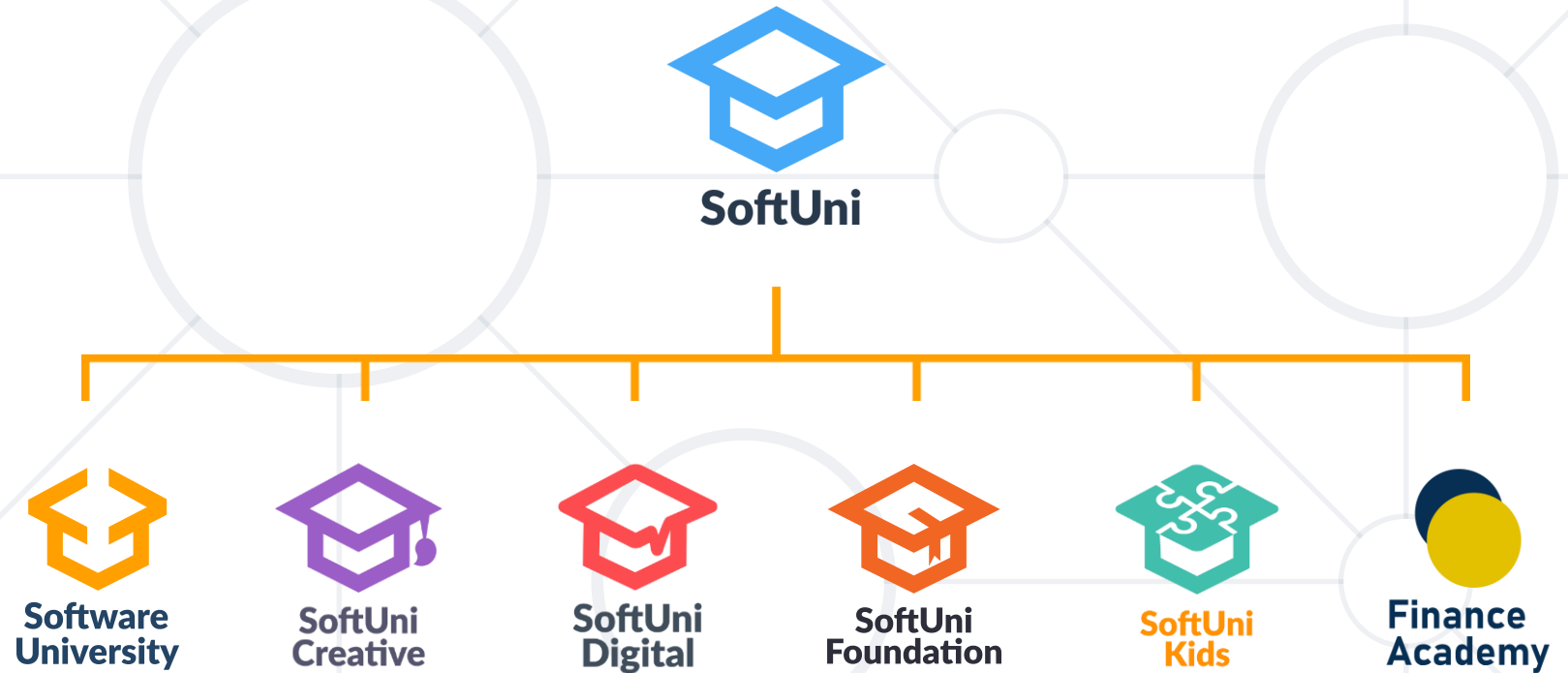


- Django is a **high-level** Web Framework
- Django Project can contain **multiple** apps
- Django **views**
  - Context
- Django **templates**
  - Django Template Language (**DTL**)
  - Template **tags**





# Questions?



# SoftUni Diamond Partners



- Software University – High-Quality Education, Profession and Job for Software Developers

- [softuni.bg](http://softuni.bg), [softuni.org](http://softuni.org)

- Software University Foundation

- [softuni.foundation](http://softuni.foundation)

- Software University @ Facebook

- [facebook.com/SoftwareUniversity](https://facebook.com/SoftwareUniversity)

- Software University Forums

- [forum.softuni.bg](http://forum.softuni.bg)



- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is **copyrighted content**
- Unauthorized copy, reproduction, or use is illegal
- © SoftUni – <https://softuni.org>
- © Software University – <https://softuni.bg>

