Django Introduction

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

Table of Contents

sli.do

#python-web

Django

Full-Stack Framework for Perfectionists with Deadlines

What is Framework?

- Platform for **developing software applications**

- Provides a **foundation** on which software developers can **build programs** for a specific platform

- A framework **includes an API**

- May include **code libraries**, a **compiler**, and other programs **used in the software development process**

What is Django?

- **High-level** python web framework

- Ridiculously **fast**

- Reassuringly **secure**

- Exceedingly **scalable**

- **Free** and **Open Source**

What is MTV?

- Django follows the **MTV architecture pattern** to develop web applications

- MTV stands for **Model-Template-View**

  - **Model** - manages **the data** and is represented by a database

  - **Template** - the presentation (front-end) layer

    - Provides a convenient way to **generate dynamic HTML pages** by using **special template syntax**

  - **View** - receives **HTTP requests** and sends **HTTP responses**

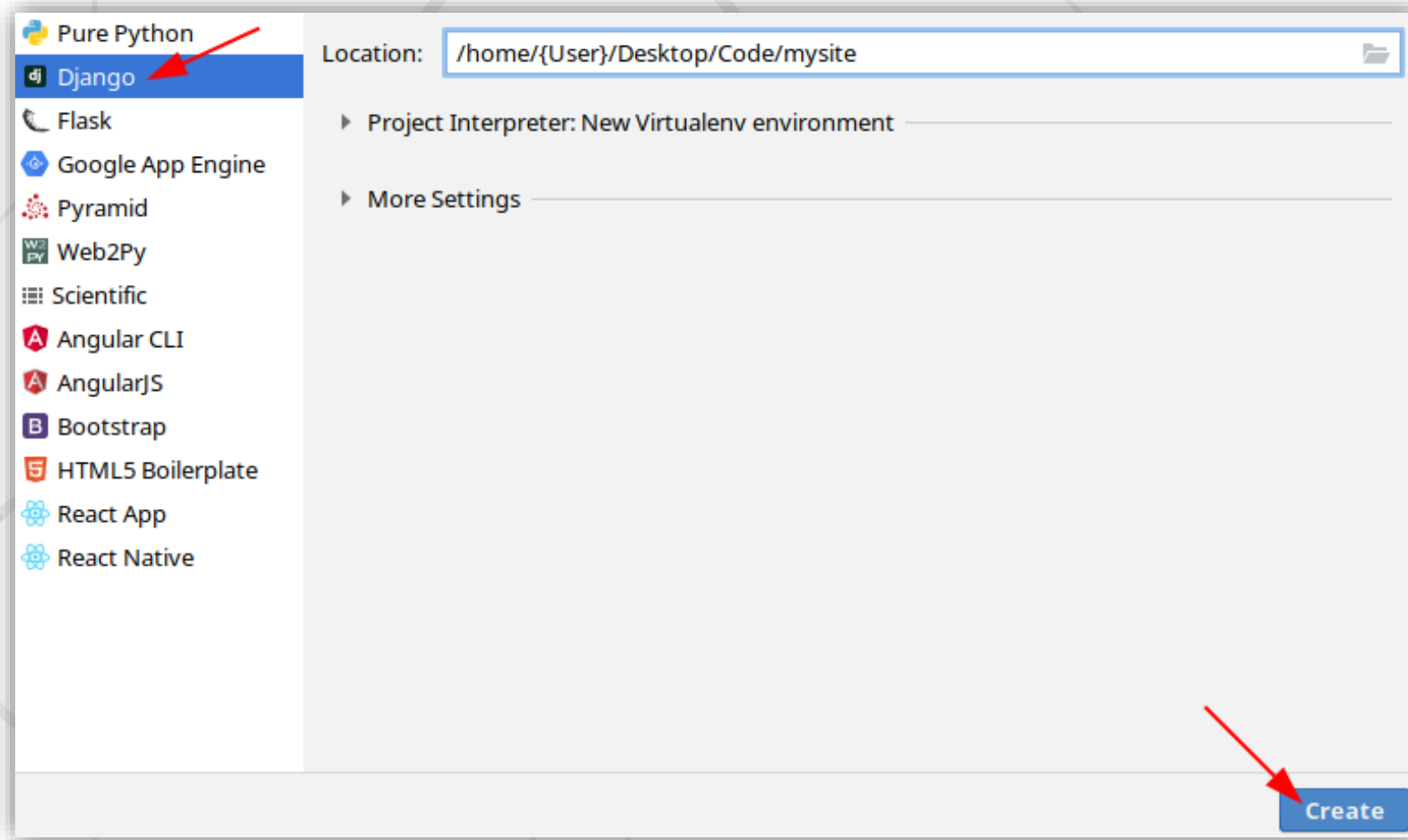   - Interacts with the model and the template to complete a response

Creating a Django Project
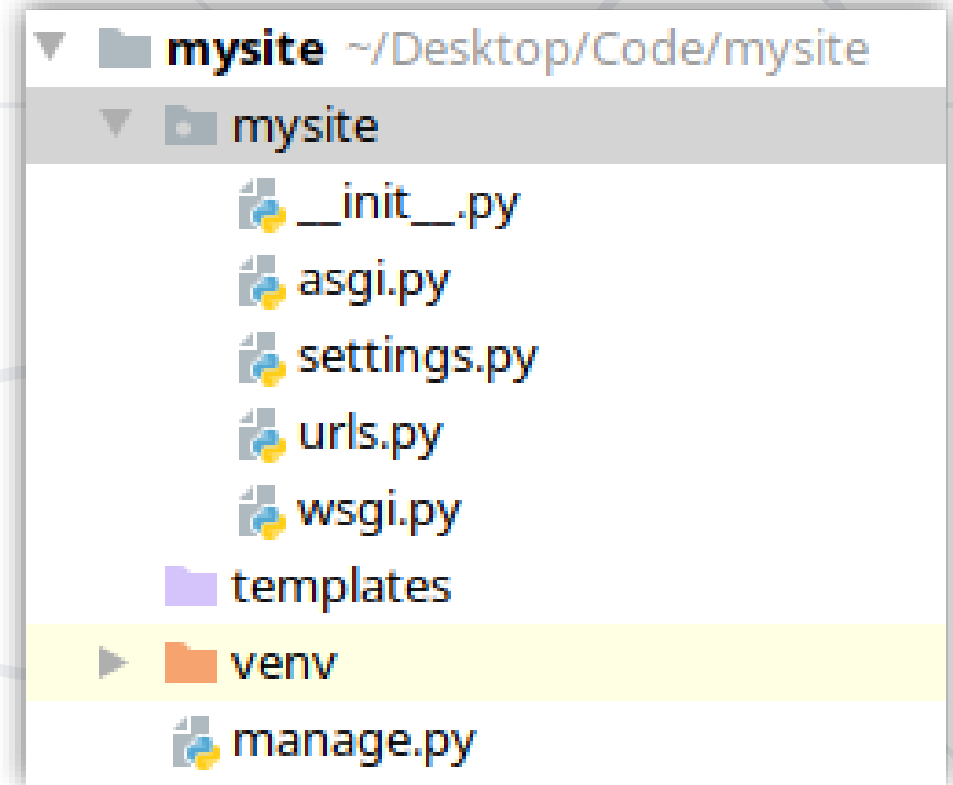
Where the magic happens

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**
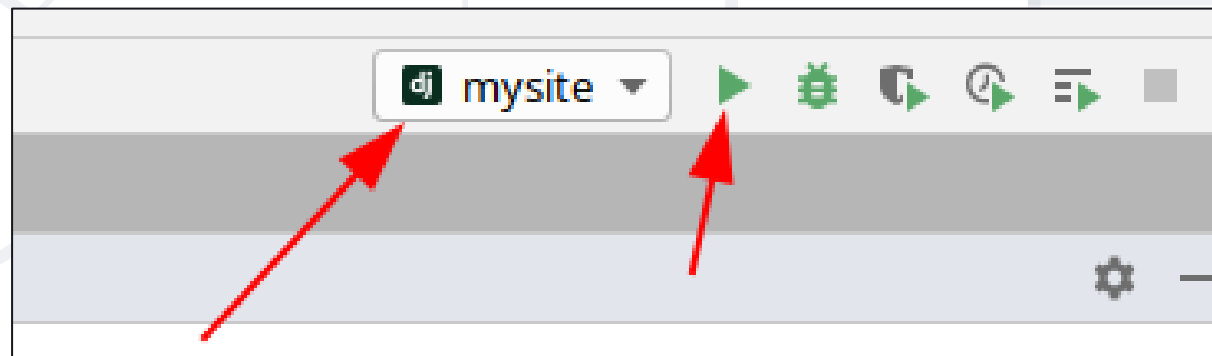 - Tool for executing commands

Running a Django Project (1)

- Using **Terminal command**

```
python manage.py runserver
```

- Using **Keyboard Shortcut** in PyCharm

```
Shift + F10
```

- Using PyCharm **Run button**

Running a Django Project (2)

- 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 intended only for use **while developing**

Running a Django Project (3)
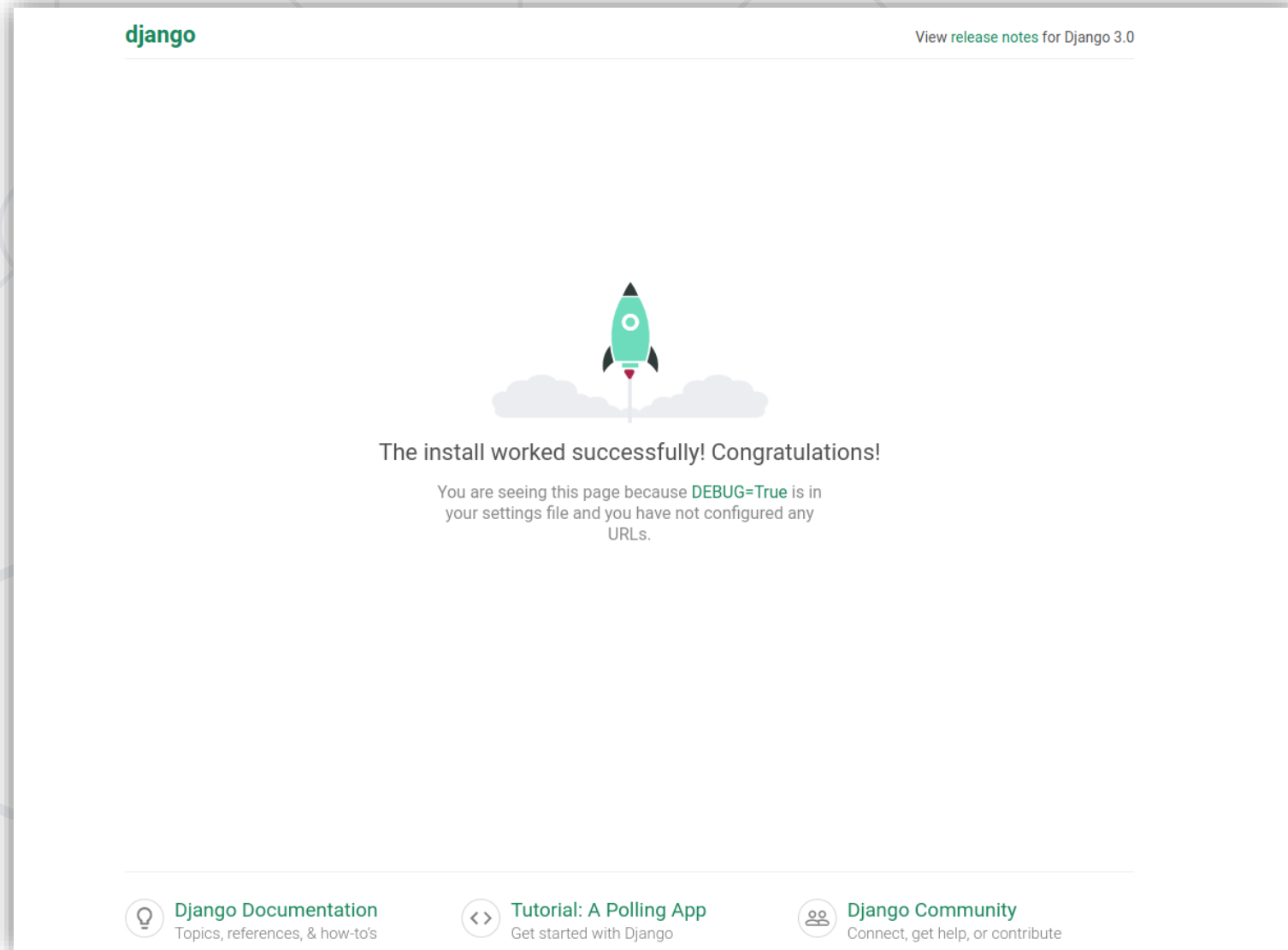
Django Application

The Bread and Butter of a Django Project

App vs Project

- Django App:

  - A **Web application that does something** - e.g., a wide web blog system or a small task app

  - An app can be **in multiple projects**

- Django Project:

  - A **collection of configuration and apps** for a particular website

  - A project can **contain multiple apps**

Creating a Django App

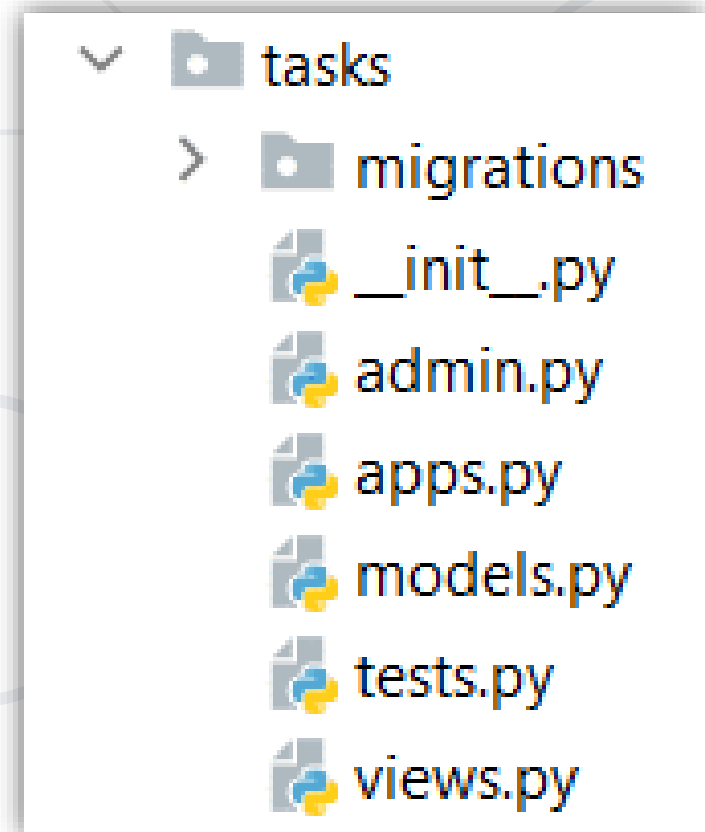- The app is created in the **same directory** as the **manage.py** file

- 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

```
v  📁 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
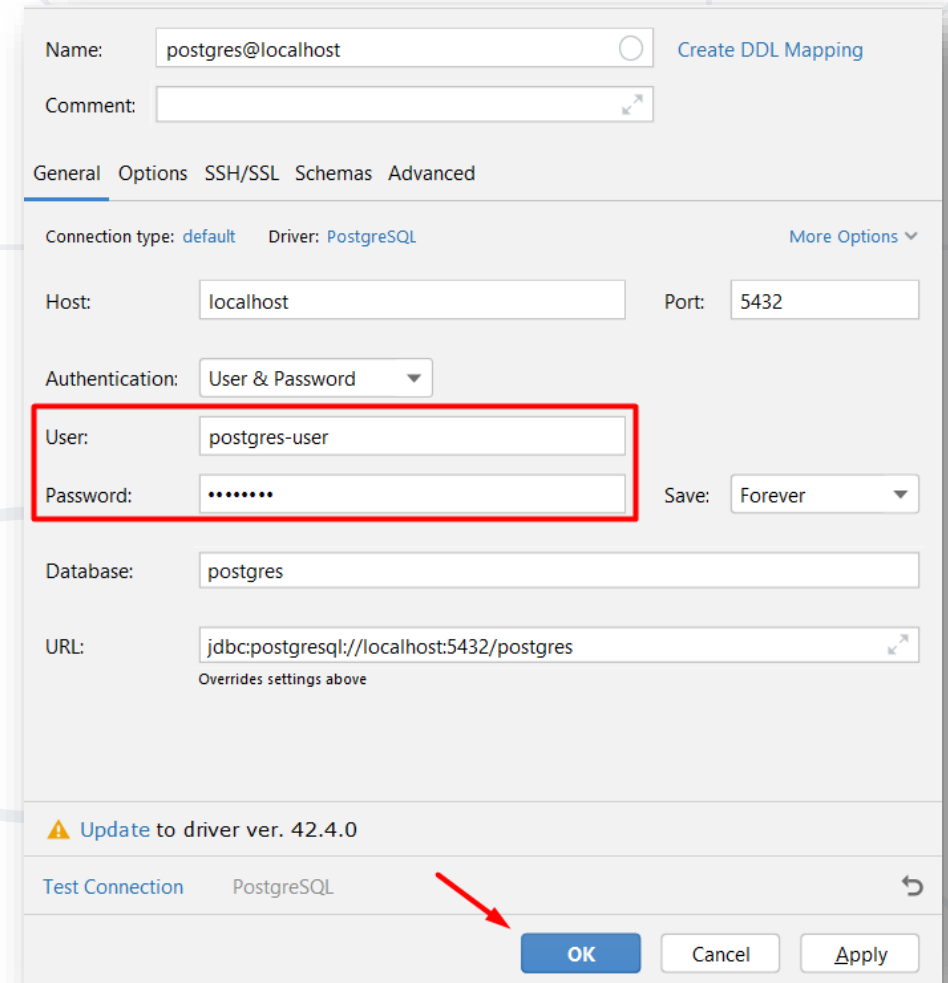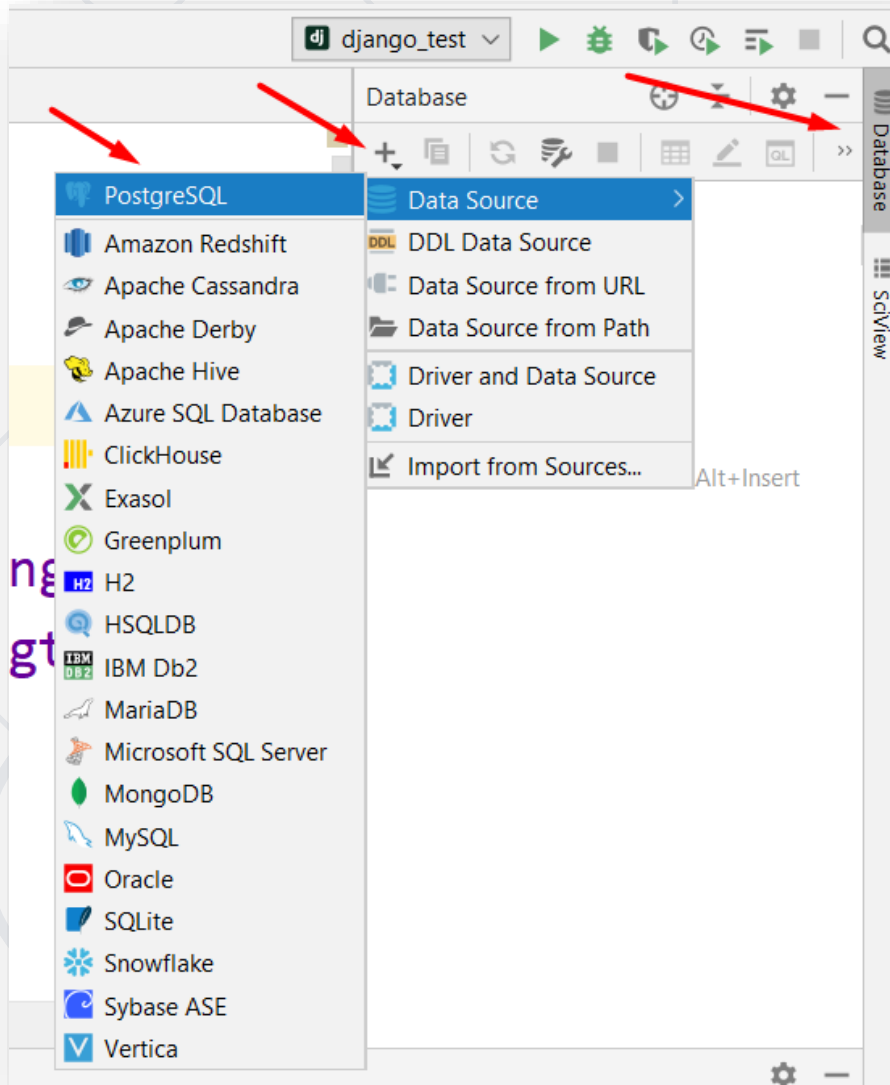
Setting up a Database

Psycopg2

- PostgreSQL database **adapter** for the Python programming language

- Use the Psycopg2 module to:

  - **Connect** to PostgreSQL

  - Perform SQL **queries** and database **operations**
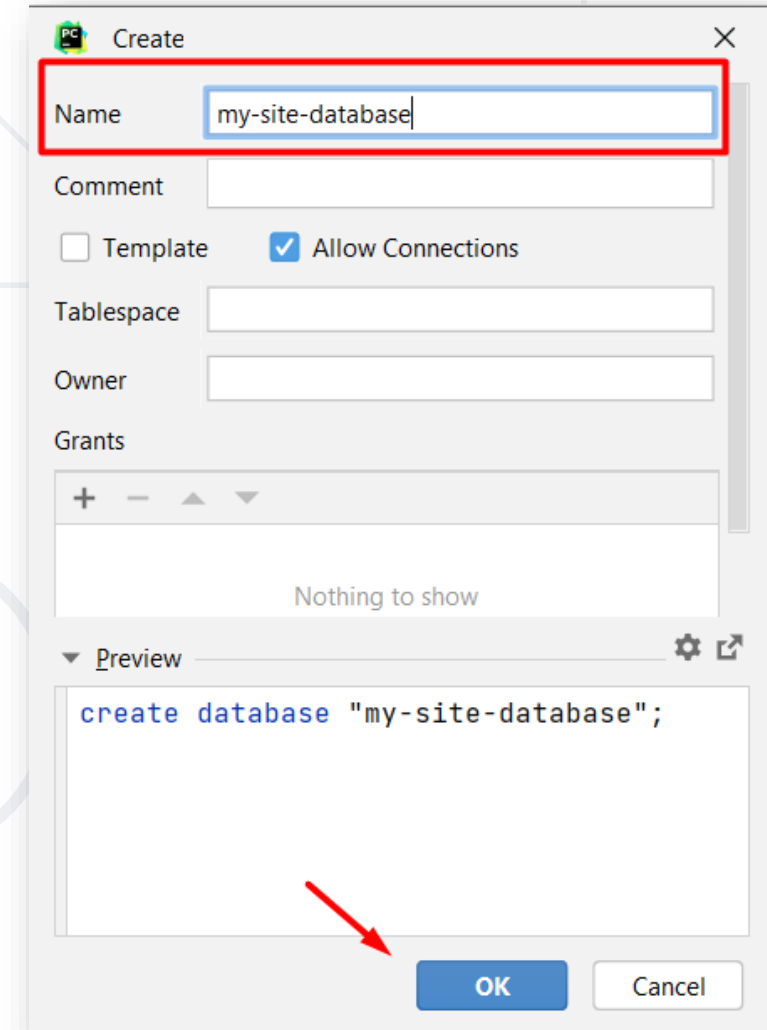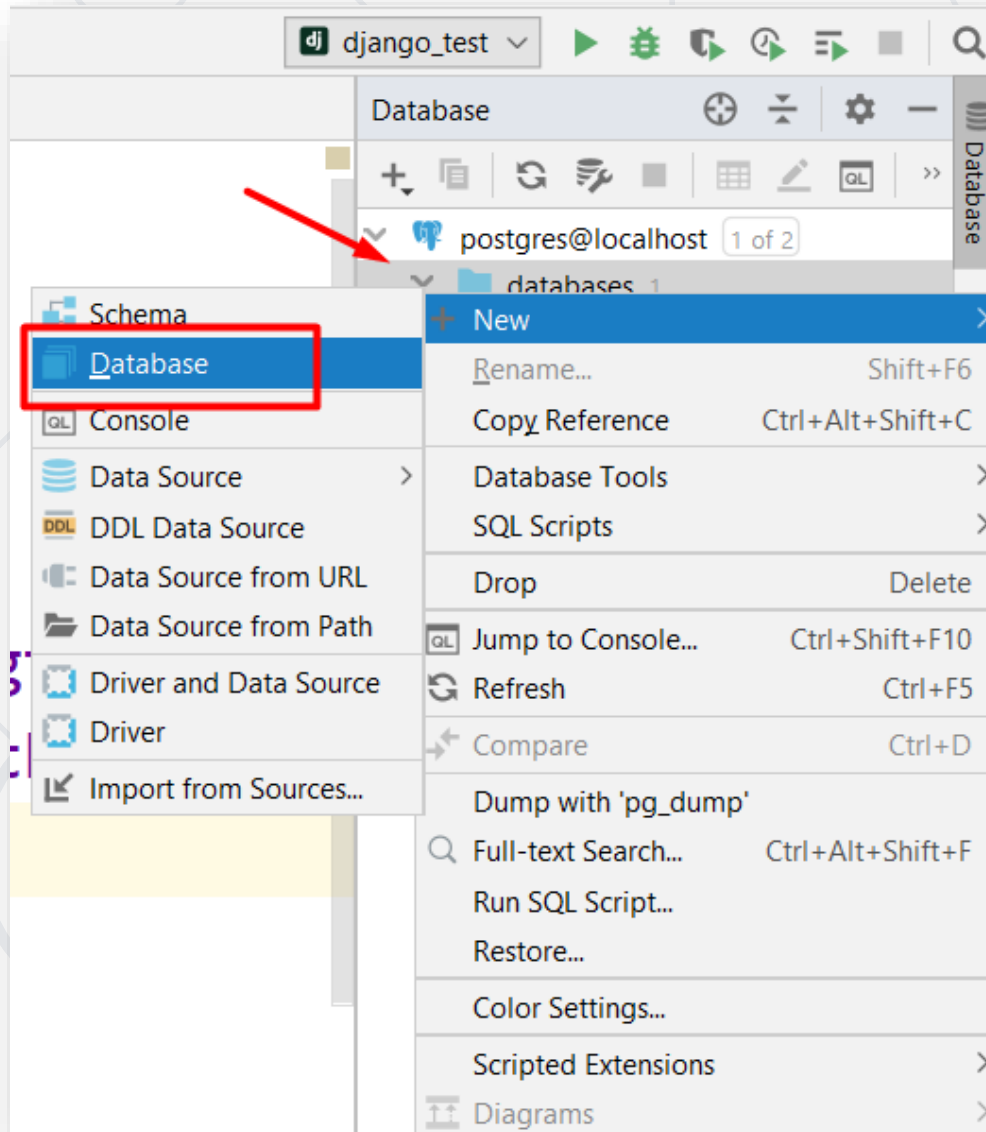
- It is an **external** module

Connect to PostgreSQL

Create a Database

Set up PostgreSQL

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

```python
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

23

Writing a Simple Task App

Django Model

- Models store your **application's data**
  - The essential **fields** and **behaviors** of the stored data
- Generally, each model maps to a single **database table**
- Each 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 have a **models.py** file

- In there all **models** that will be used in the **application** should be created

tasks/models.py

```python
from django.db import models


class Task(models.Model):
    task_title = models.CharField(max_length=50)
    task_text = models.TextField()
```

Model Name

Field Types

Fields

Activating Models

- Use models to create a **database schema** for the app

- Use **migrations** to **upgrade your database** live

    - 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/ classes
- Each view takes a **Web request** and returns a **Web response**
    - Implements the **main logic** that needs to happen 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

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

def index(request):
    tasks_list = Task.objects.all()
    output = "; ".join(f"{t.task_title}: {t.task_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 what function or logic should be executed when accessing a given **URL**

- Usually, **every app** should have its own **urls.py** file

tasks/url.py

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


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

URL

Action

Django project/urls.py

- The created url.py file should be **pointed in the project's urls.py**

- Import the **include()** function and insert it in the **urlpatterns** list

mysite/url.py

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

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

Simple URL Example
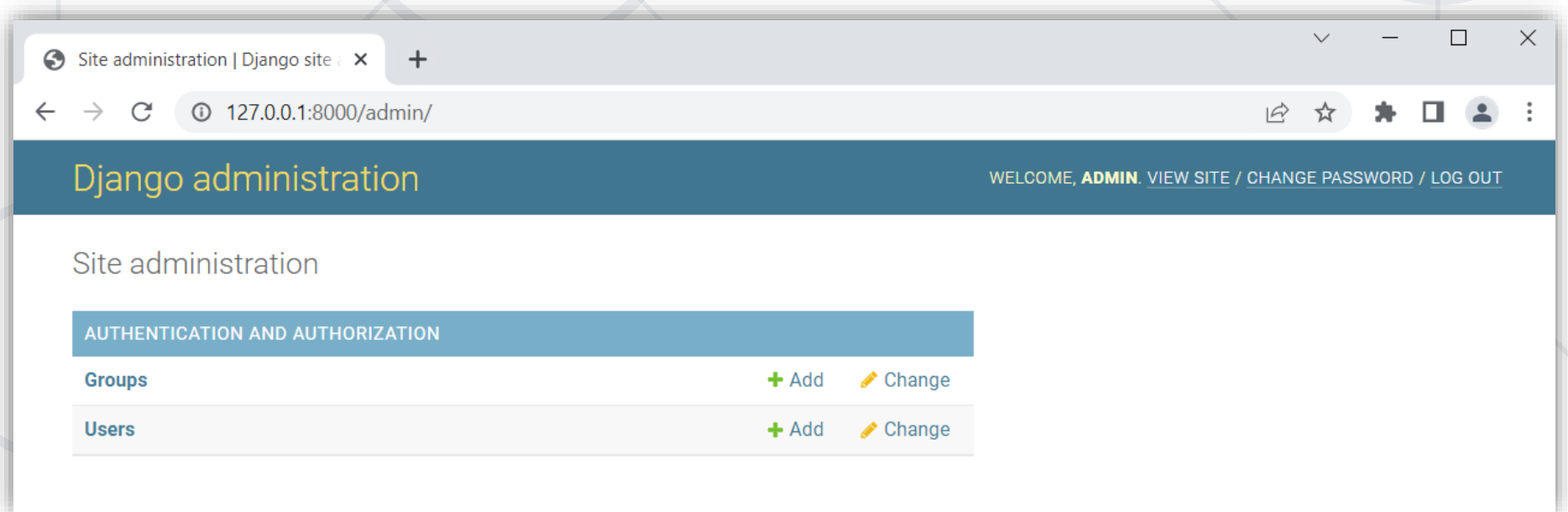
- Verify it is working by starting the development server
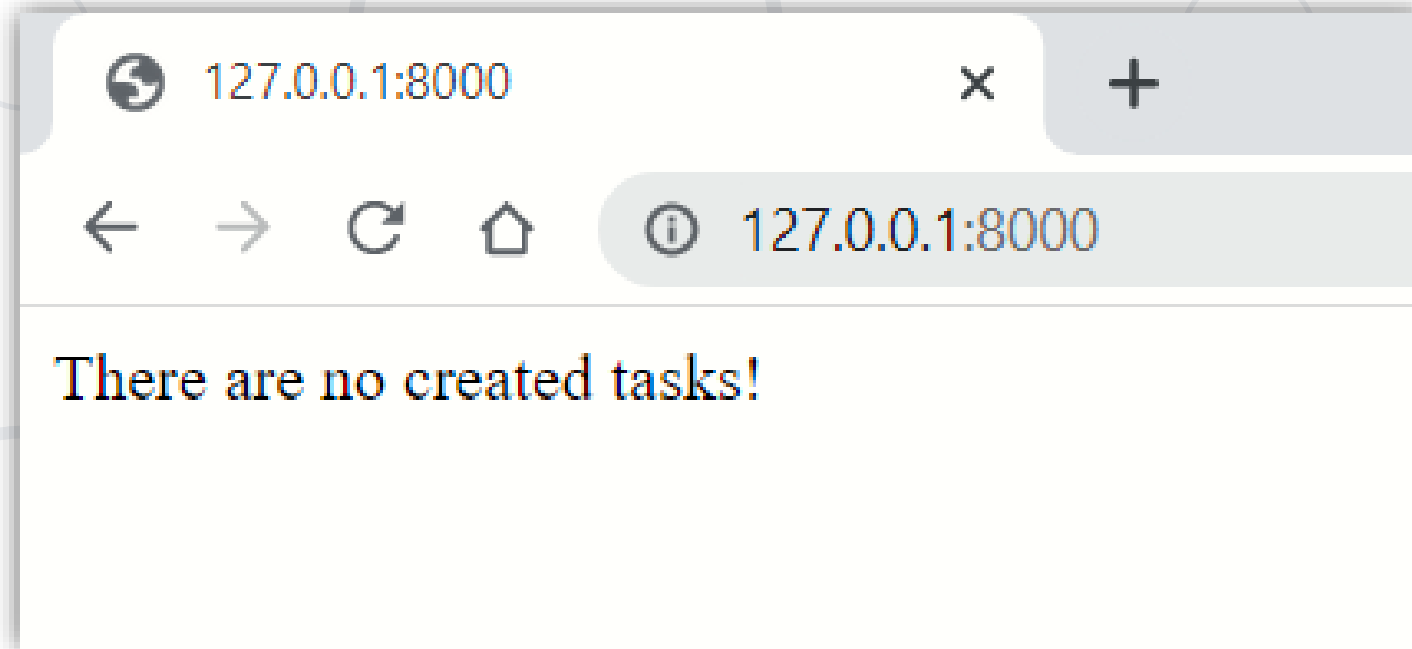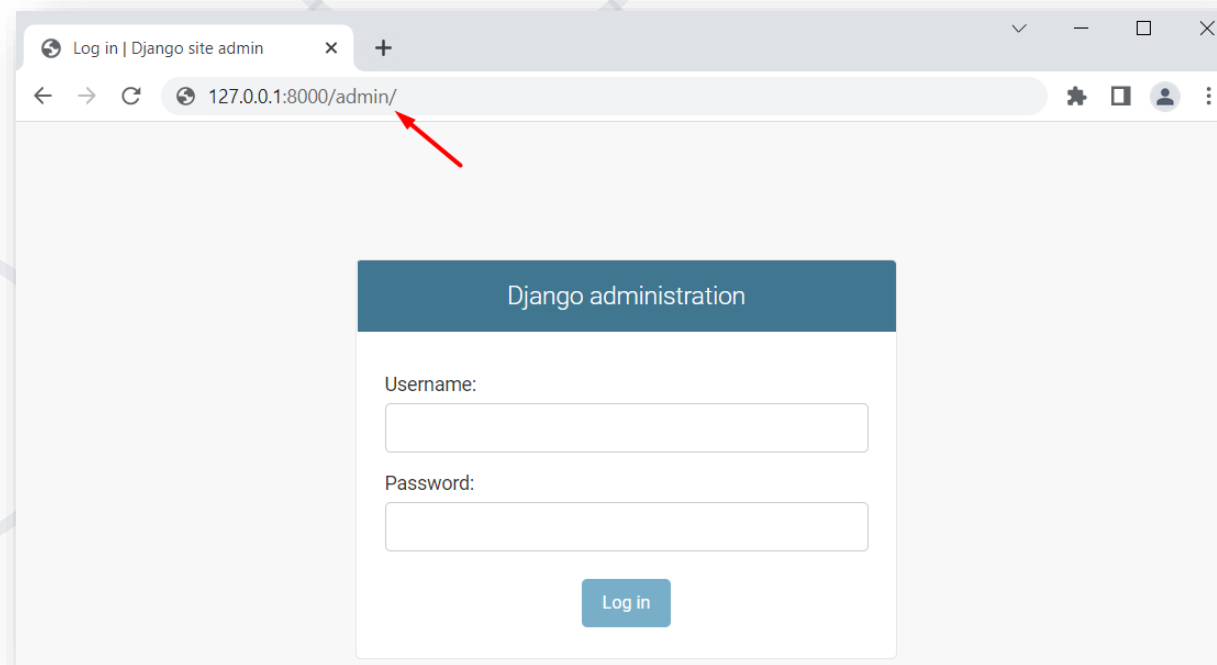
Django Admin Site

Django Admin Site

- It is an **automatic admin interface**
  - There **trusted users can manage content** on the site
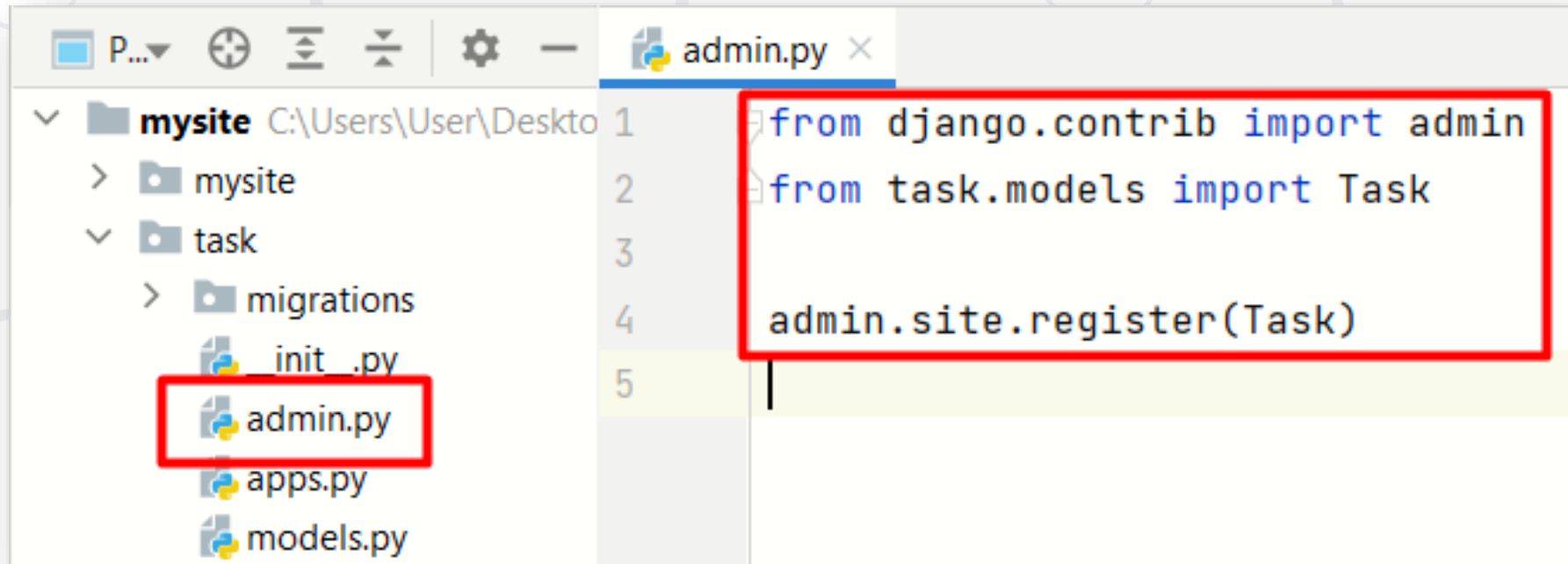- One of the most powerful parts of Django

Access Django Admin Site

- First, create a **user to login with**

```
python manage.py createsuperuser
```

- Then, **start the server** and **navigate to the admin site**
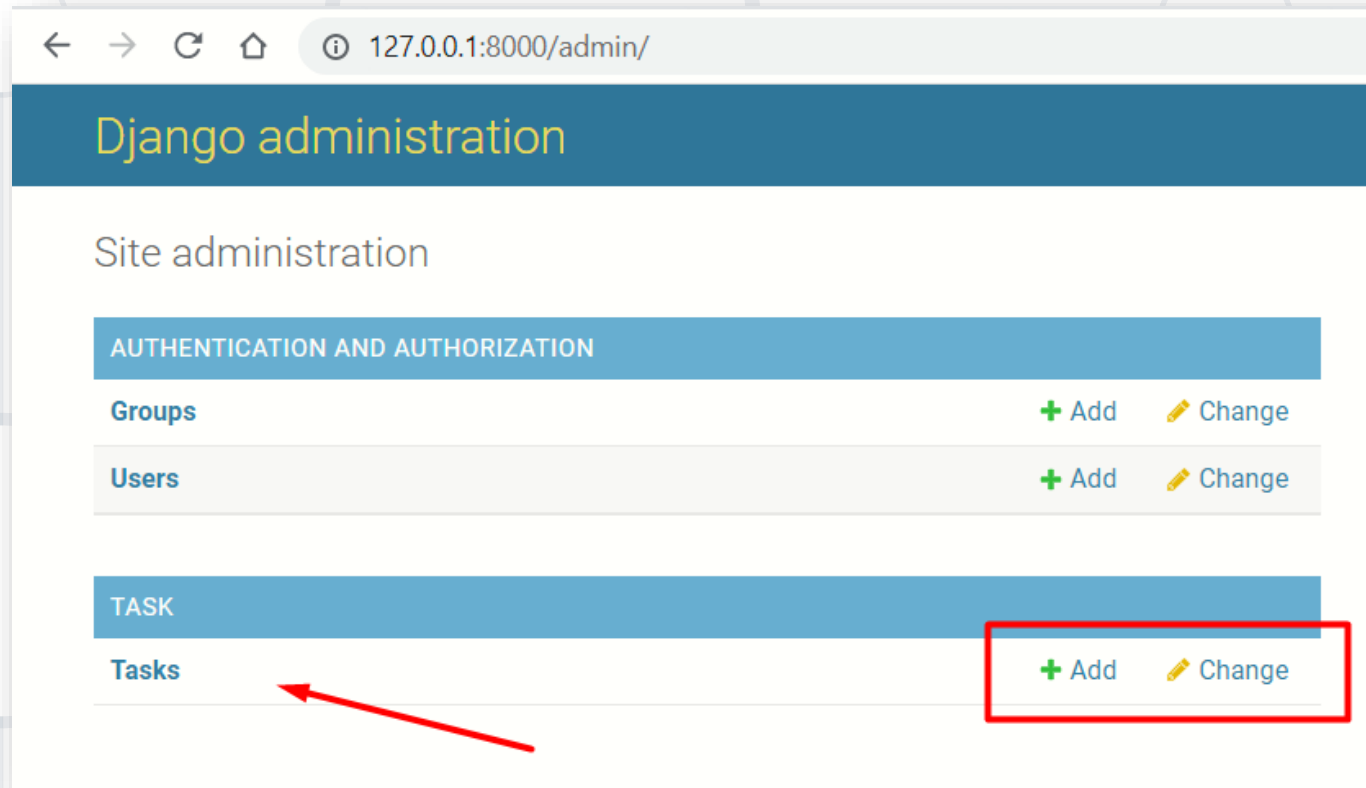
Make the App Modifiable in the Admin

- **Register** all models in a special file in the **app** called **admin.py**
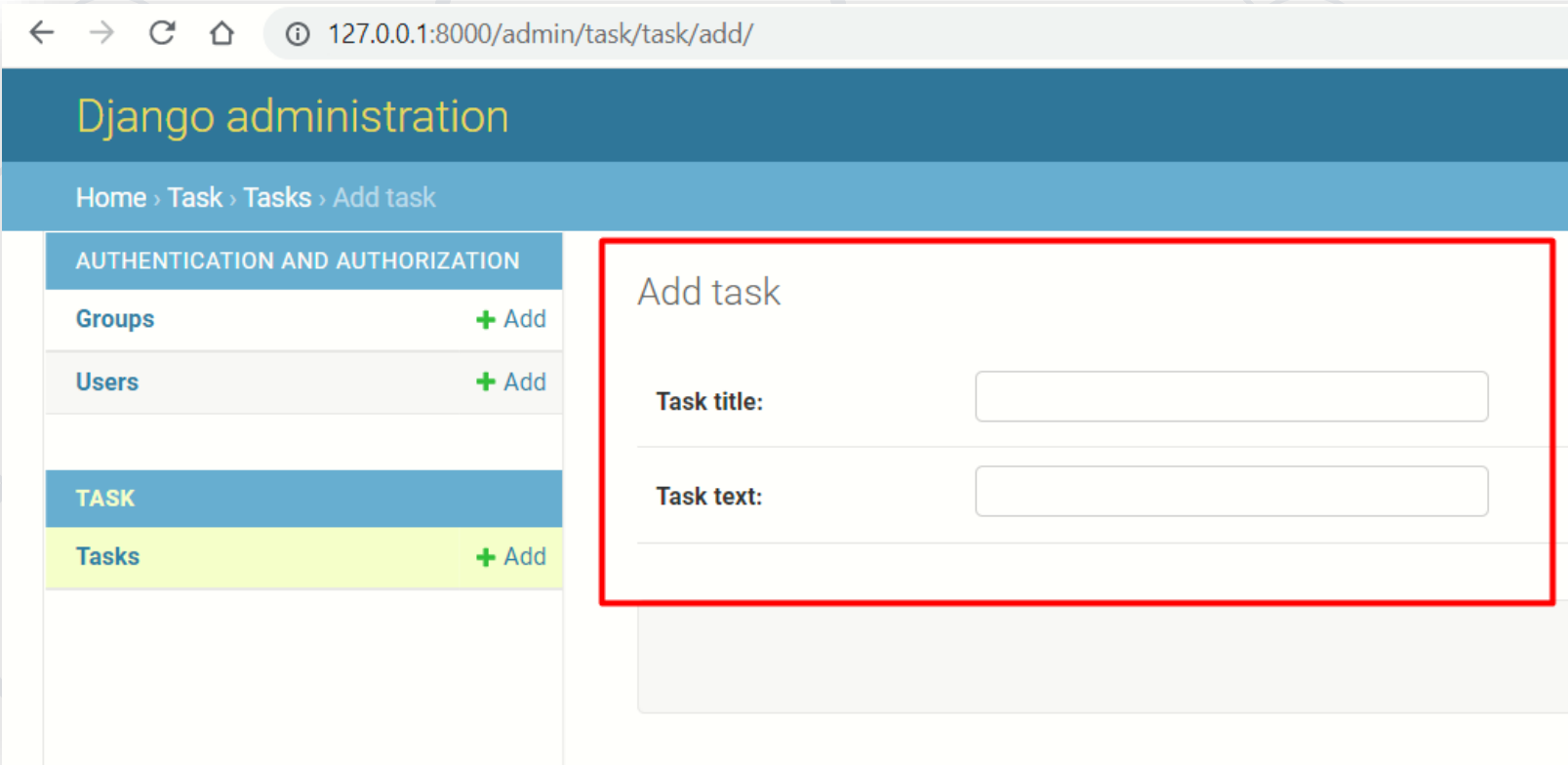
Django Admin Benefits (1)

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

Django Admin Benefits (2)

- The form is **automatically generated** from the models

Data Visualization

- When **adding tasks**, they are **returned as response** in the desired Web page (using the created view in the app)
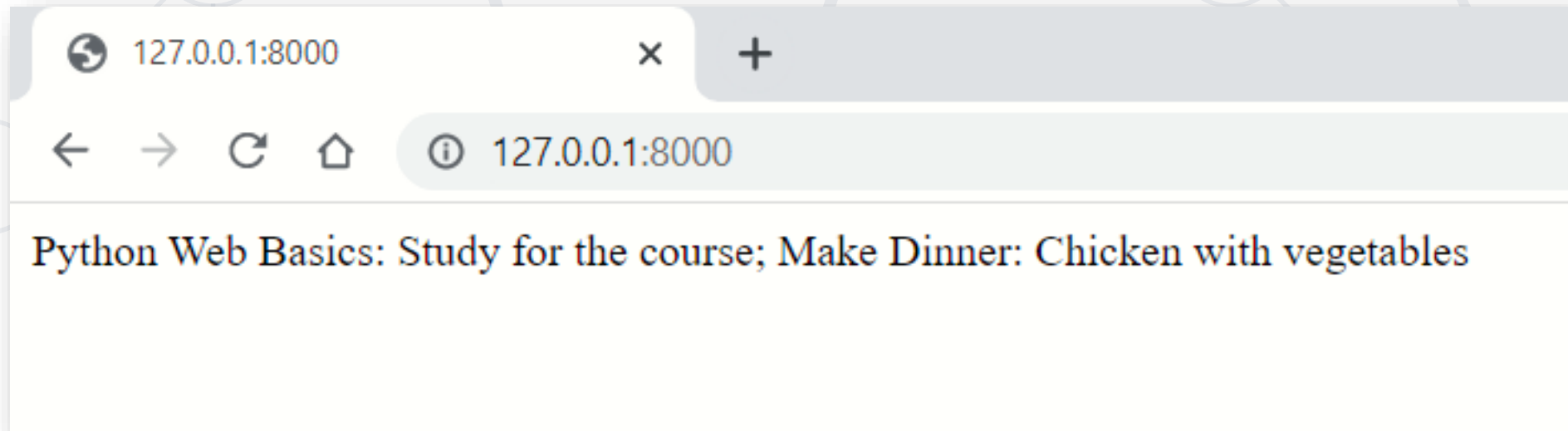
- Note: The page's design is **hard-coded in the view**

Creating a Simple Design

What is a Template

- Being a web framework, Django needs a convenient way to **generate HTML dynamically**

- The most common approach relies on **templates**

- A template contains:

  - The **static parts** of the desired HTML output

  - A **special syntax** describing how dynamic content will be inserted

Creating a Template Folder

- In order to use templates, we need a **special folder** where they should be created
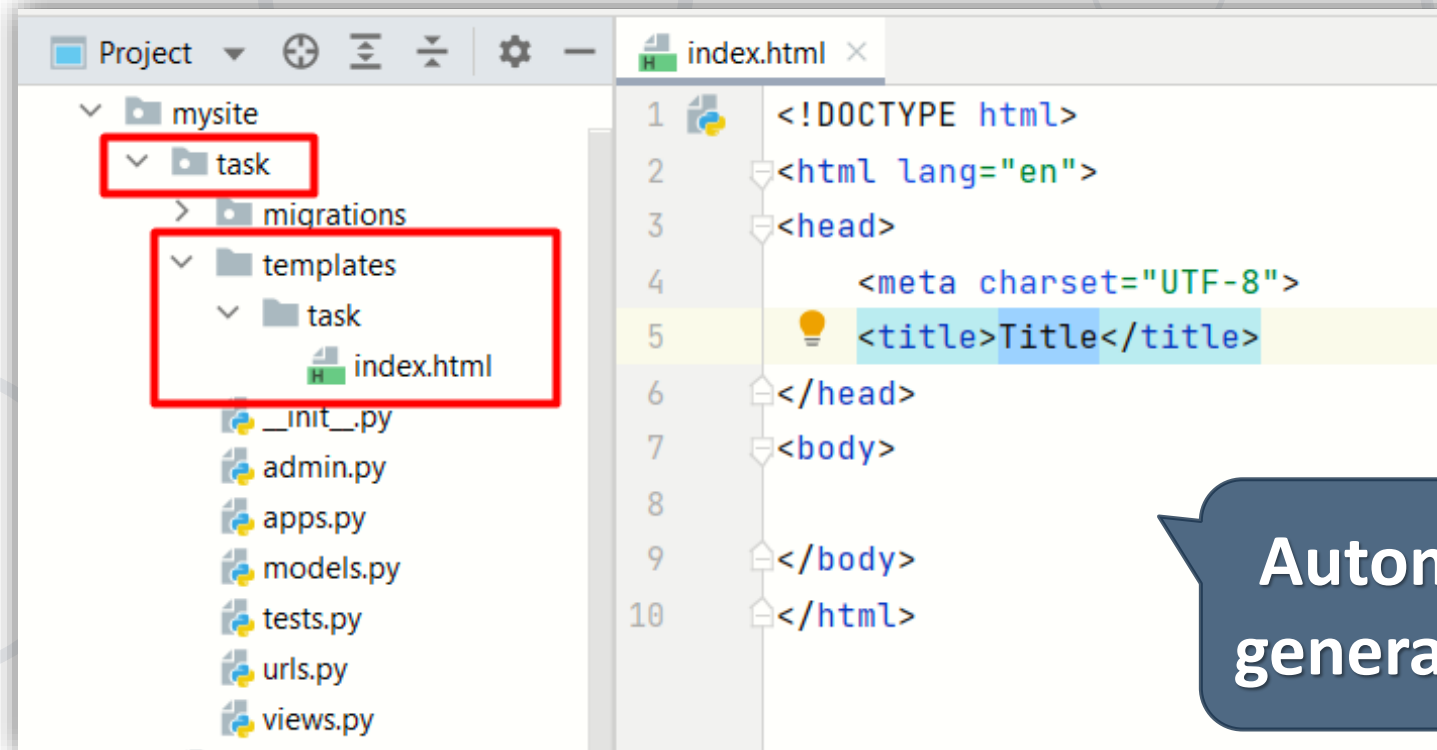
Automatically generated code

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')
```
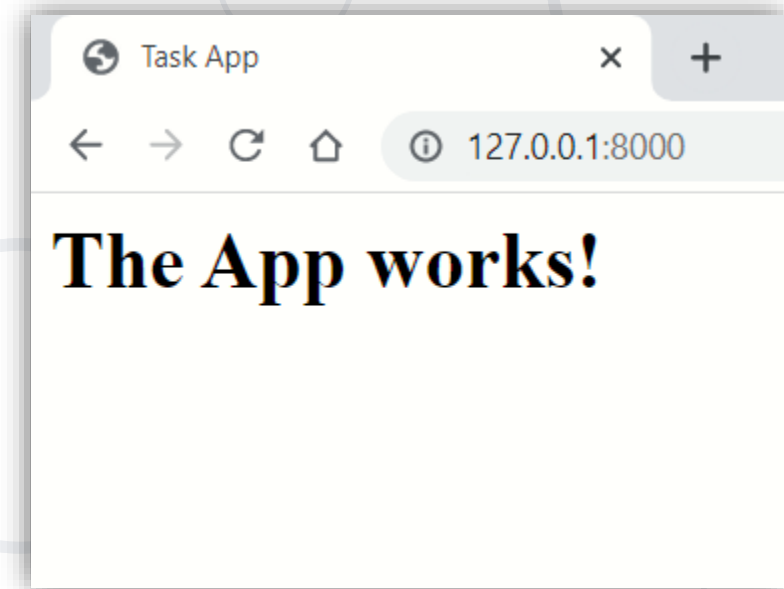
- Instead of using **HttpResponse**, we can now use the **render** function that will show the created template

Creating a Template

- Creating an **.html** file that will be the template

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

Adding Context

- The **render** function can receive a **context**
  - It is a **dictionary** passed to the template and used to display data **dynamically**

```python
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

- In the templates we can also have **programming 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 "tasks_list" is passed as context, for each task return its title and text

Simple Task App
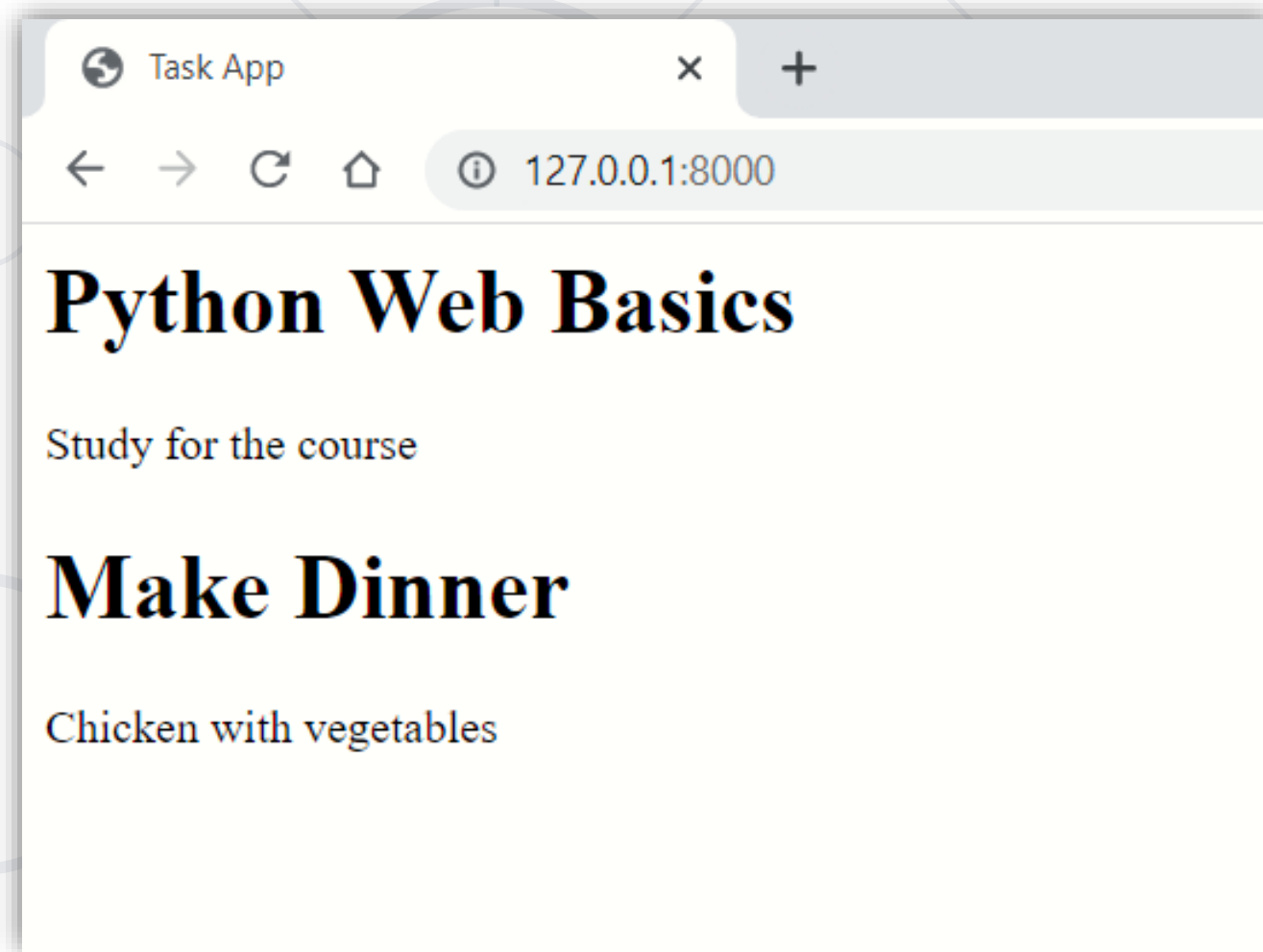
Demo

Live Exercise in Class

Summary

- Django is a **high-level** Web Framework

- Django applications are set of **features** for a Django Project

- We can use **terminal commands** to manage our projects and apps

- We can **manage the content** of the application using Django Admin

Questions?
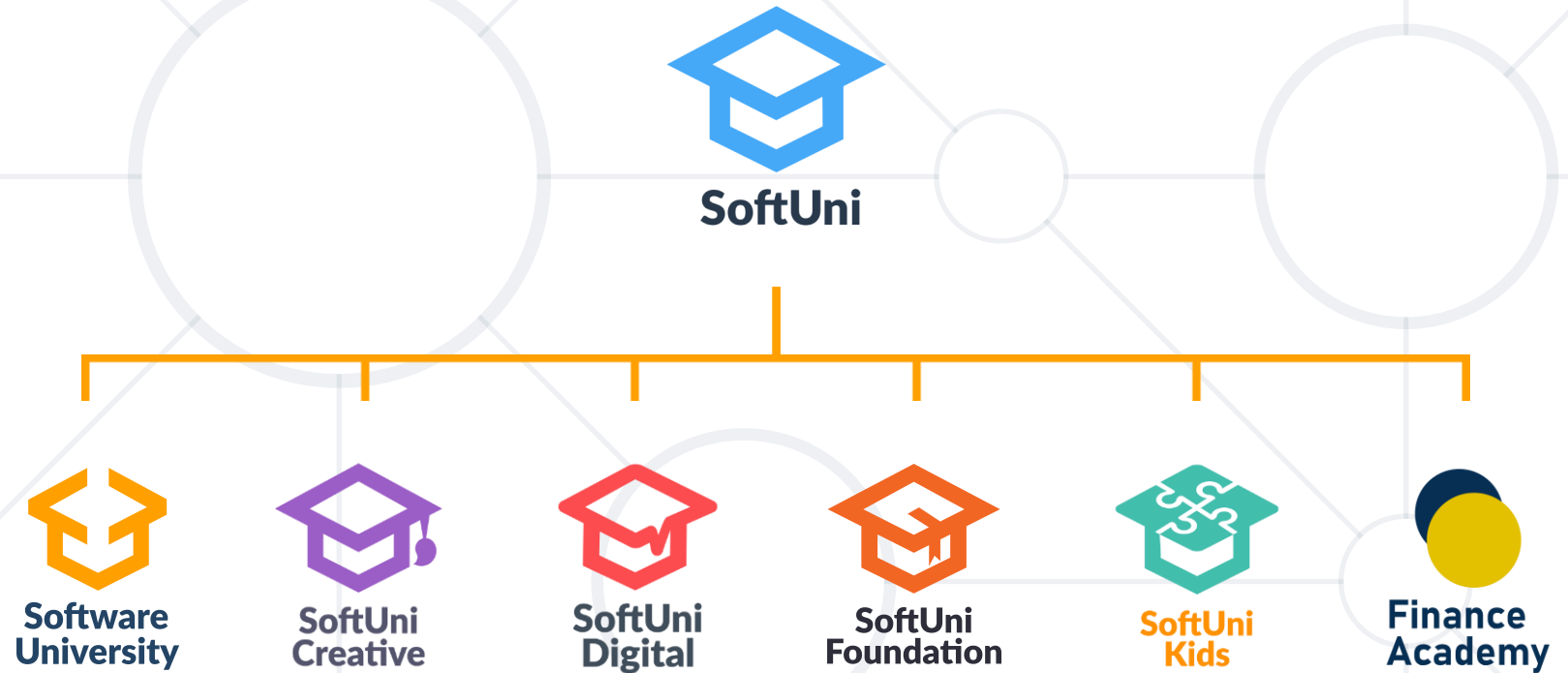
SoftUni Diamond Partners

Educational Partners

Trainings @ Software University (SoftUni)
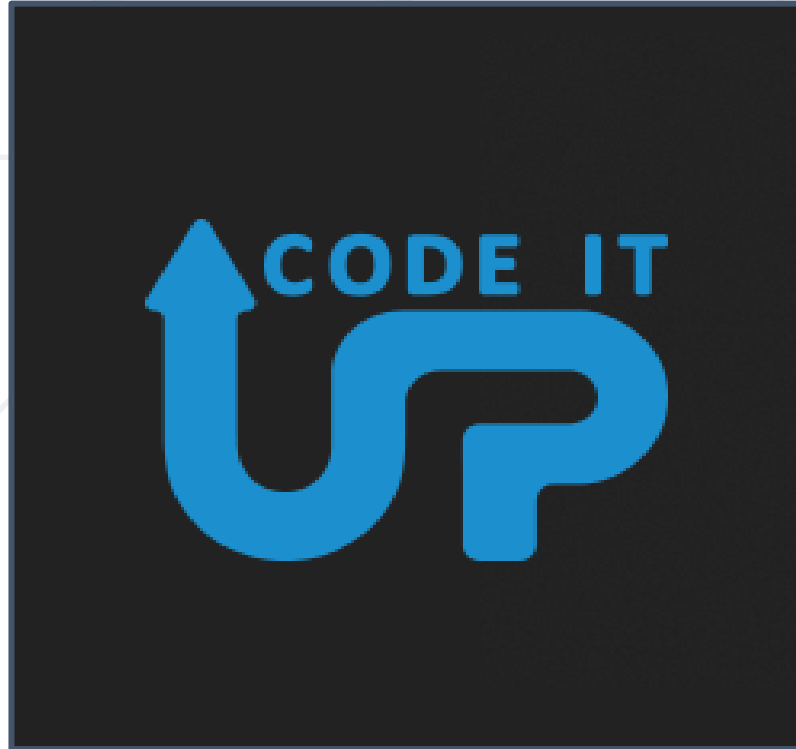
- Software University – High-Quality Education, Profession and Job for Software Developers

    - softuni.bg, softuni.org

- Software University Foundation

    - softuni.foundation

- Software University @ Facebook

    - facebook.com/SoftwareUniversity

- Software University Forums

    - forum.softuni.bg

License

- 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