Фреймворк Django

Лекцій – 4 години

Лабораторних робіт – 2 години

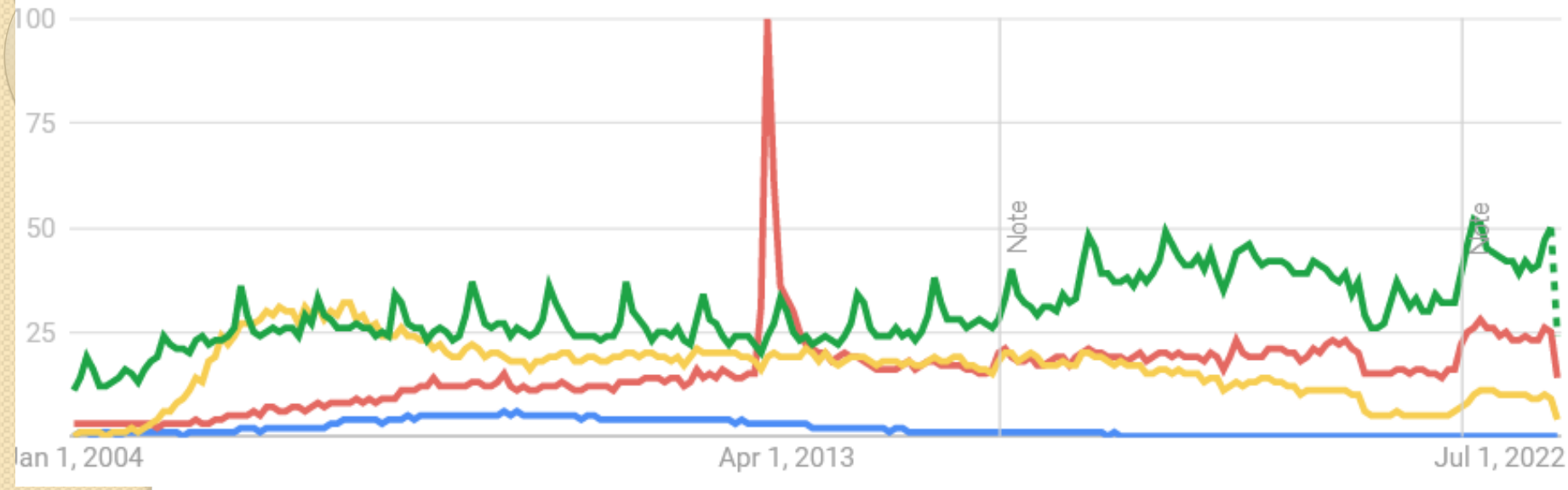
ІННІ ЗНУ, кафедра ПЗАС, доц. Попівщий В.І., 2015-2021

Вступ до web-фреймворку Django

- **Django** (Джанго) - високорівневий відкритий Python-фреймворк для розробки веб-систем.
- Названий на честь джазмена Джанго Рейнхардта

- Архітектура Django подібна MVC
  - рівневу архітектуру Django часто називають «Модель-Шаблон-Подання» (MTV)
- Сайт https://www.djangoproject.com/
- Має власний ORM для роботи з базами даних
- Вбудований інтерфейс адміністратора
- Диспетчер URL на основі регулярних виразів
- Останній реліз Django 4.2 (на квітень2023)
- Розробники: *Russell Keith-Magee, Adrian Holovaty, Simon Willison, Jacob Kaplan-Moss, Wilson Miner – перший реліз в 2003 році*

Популярність фреймворків



- http://www.google.com/trends/explore?hl=en-US#q=/m/0cdvjh,+/m/06y_qx,+/m/0505cl,+/m/0dhx5b&cmpt=q

Використання Django

Веб-фреймворк Django використовується в таких великих і відомих сайтах, як

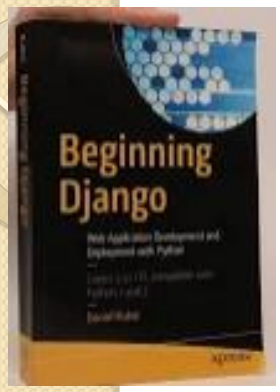
- Instagram,
- Disqus,
- Mozilla,
- The Washington Times,
- Pinterest та ін.

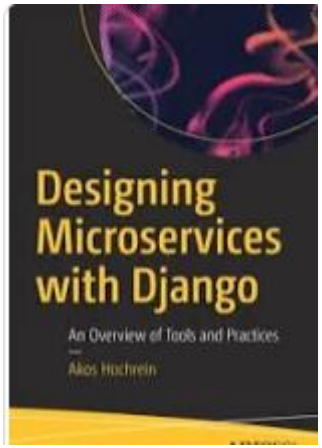
Також Django використовується в якості веб-компонента в різних проектах, таких як

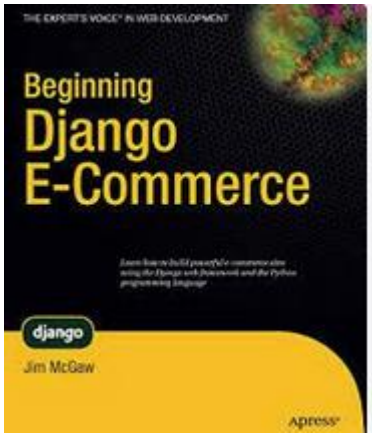
- Graphite - система побудови графіків та моніторингу,
- FreeNAS - вільна реалізація системи зберігання та обміну файлами та ін.

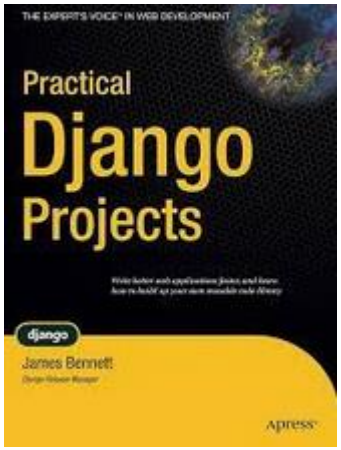

Who Uses Django?
reddit
the ONION
America's Finest News Source
Pinterest
YouTube
Dropbox
Spotify
NETFLIX
Instagram

КНИГИ









Best Django Books (2023)
https://wsvincent.com/best-django-books/

Корисні посилання

- Офіційний сайт https://www.djangoproject.com/
- Django Essential Training https://github.com/LinkedInlearning/django-esst-2894047

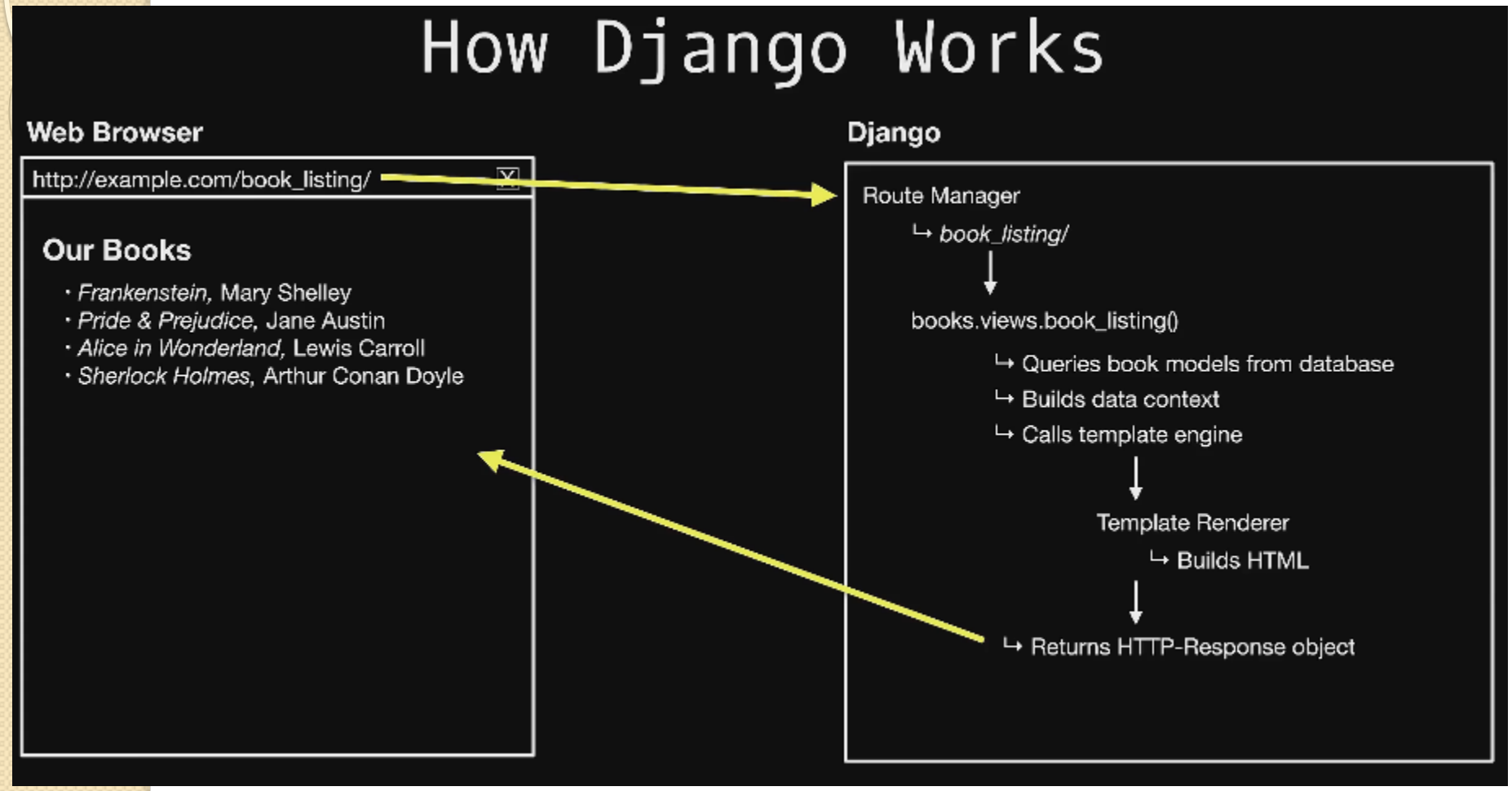
Стандартний функціонал Django

- Система конфігурування URL
- Django ORM (Object-relational mapping)
- Розширювана система шаблонів
- “Class-based views” – універсальні представлення
- Бібліотека для роботи з формами
- Вбудований інтерфейс адміністратора
- Вбудована система кешування
- Інтернаціоналізація

+ та багато іншого

Основні принципи Django

- Django прагне бути “пітонічним”
- DRY (Don’t Repeat Yourself)
- Слабка залежність та гнучкість
- Швидка розробка


How Django Works
Web Browser
http://example.com/book_listing/
Our Books
• Frankenstein, Mary Shelley
• Pride & Prejudice, Jane Austin
• Alice in Wonderland, Lewis Carroll
• Sherlock Holmes, Arthur Conan Doyle
Django
Route Manager
↳ book_listing/
books.views.book_listing()
↳ Queries book models from database
↳ Builds data context
↳ Calls template engine
Template Renderer
↳ Builds HTML
↳ Returns HTTP-Response object

URL Patterns

- Declare a list of URL mappings
- Routes
- Map to a:
  - View function
  - View class
  - List of URL mappings

Views

- View is responsible for returning content
  - `HTTPResponse` object
- Views can take arguments

- Views can be written as functions or classes
- Views typically render a template
- Django has some built-in views

Template Engine

- Separate business logic from presentation logic
- Text based rendering:
  - HTML
  - Email
- Tags for controlling repetition and conditional blocks
- Tags for block re-use: inheritance, inclusion
- Tags for rendering variables from the context
- Filters to modify data, further separating presentation logic

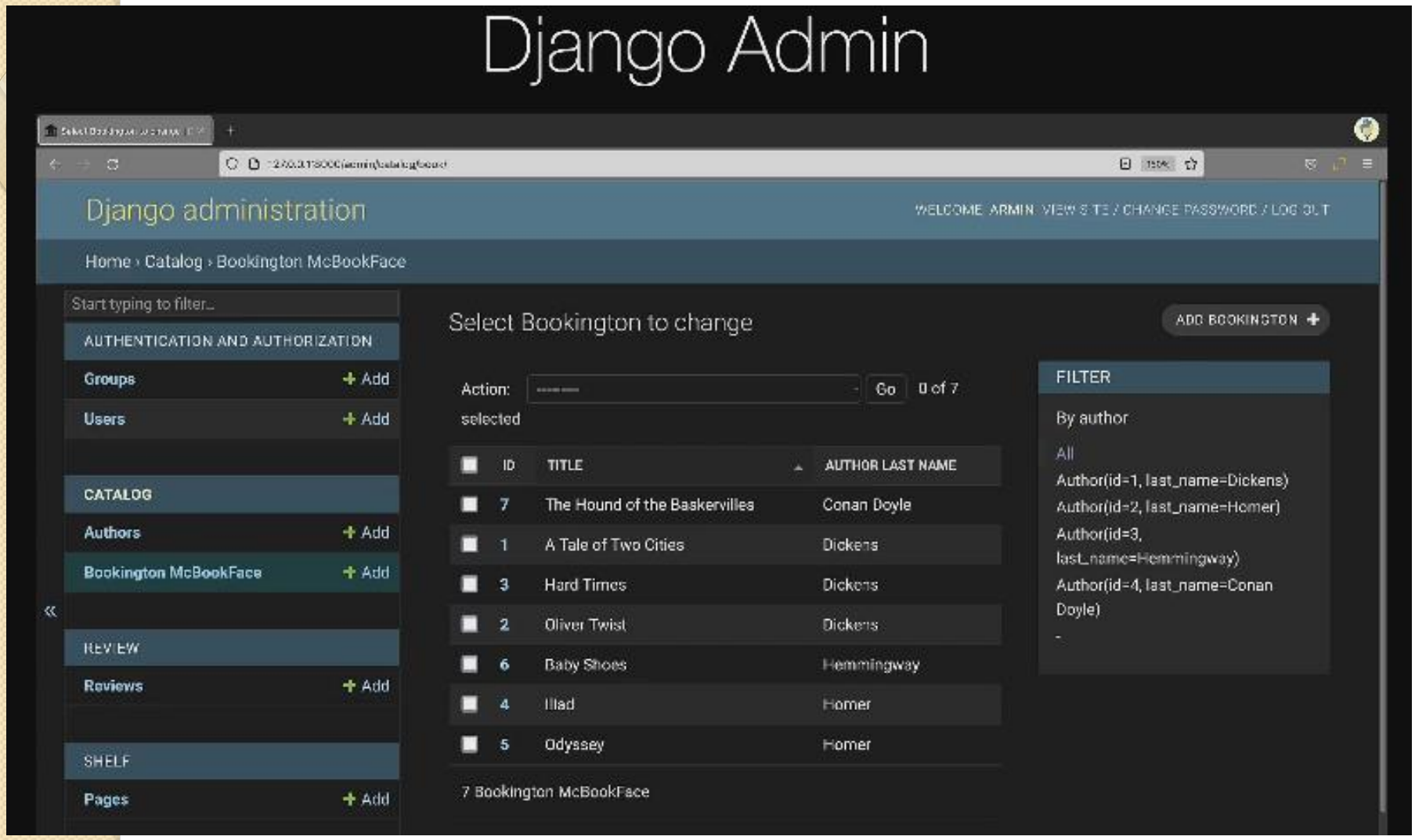
Models

- Abstraction layer for the data base
- Use classes to declare the fields in a data structure
- Django maps the class to a table in the database
- Multiple databases supported
- Mechanisms for managing changes to the models
- Inter-model relationships
- Query manager per class to look-up stored data

Users

- Built-in user management
- Account registration framework
- Authentication
- Authorization
- Control who can access what views through decorators

Django Admin



Pluggable Architecture

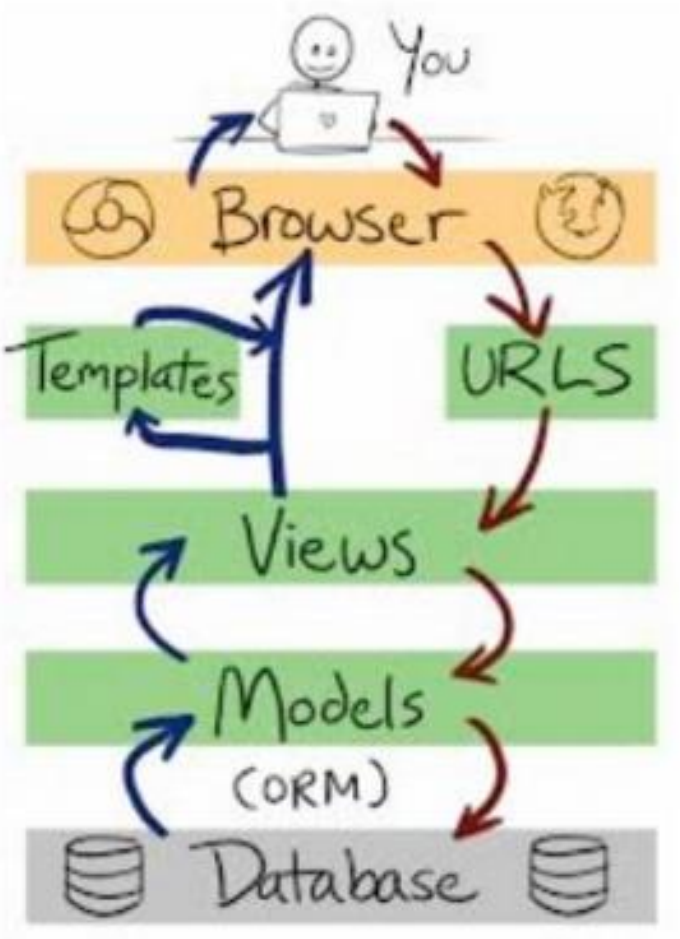
- **Project:** your software
- **App:** a module in your project
  - Apps can be shipped and managed separately
  - Wide variety of third-party libraries

Питання для розгляду

- First Project
- Template Engine
- Models & the ORM
- Django Admin
- User Management
- Using Forms
- Uploading Files
- Management Commands
- Managing Model Changes
- Deployment Considerations

Django MVC

- MVC – Model – View – Controller
- MTV – Model – Template – View



Проект в Django

– являє собою набір конфігураційних файлів і застосунків Django.

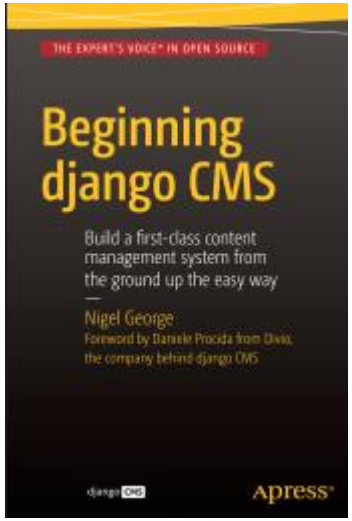
Проект може містити множину застосунків Django.

Застосунок Django – набір файлів, що містять опис моделей та представлень, які є частиною одного пакета Python і являють собою повний застосунок.

Застосунок може міститися у декількох проектах.

Django – готові рішення

- Django CMS
  https://www.django-cms.org/en/
- Mezzanine
  http://mezzanine.jupo.org/
- Oscar
- Pinax
- Djedi-CMS
- Fein-CMS
- Oops-CMS
- Wagtail CMS
- Widgy CMS



- https://www.python.org/
- cmd.exe
- d:
- mkdir DjangoProjects
- cd DjangoProjects
- python –m venv P1Env
- P1Env\Scripts\activate
- pip install django
- django-admin startproject Alexandria

Створення та запуск Django проекта

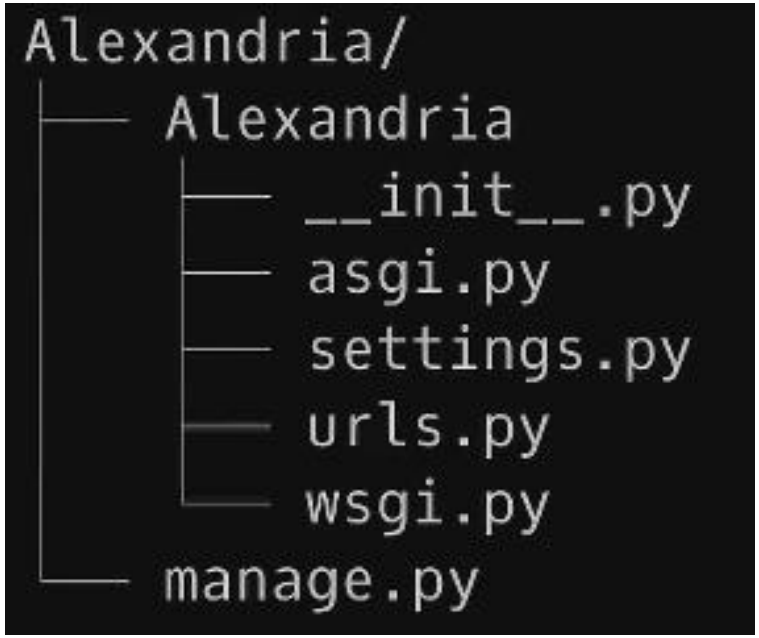
Почати новий проект:

django-admin.py startproject example

Запустити локальний сервер:

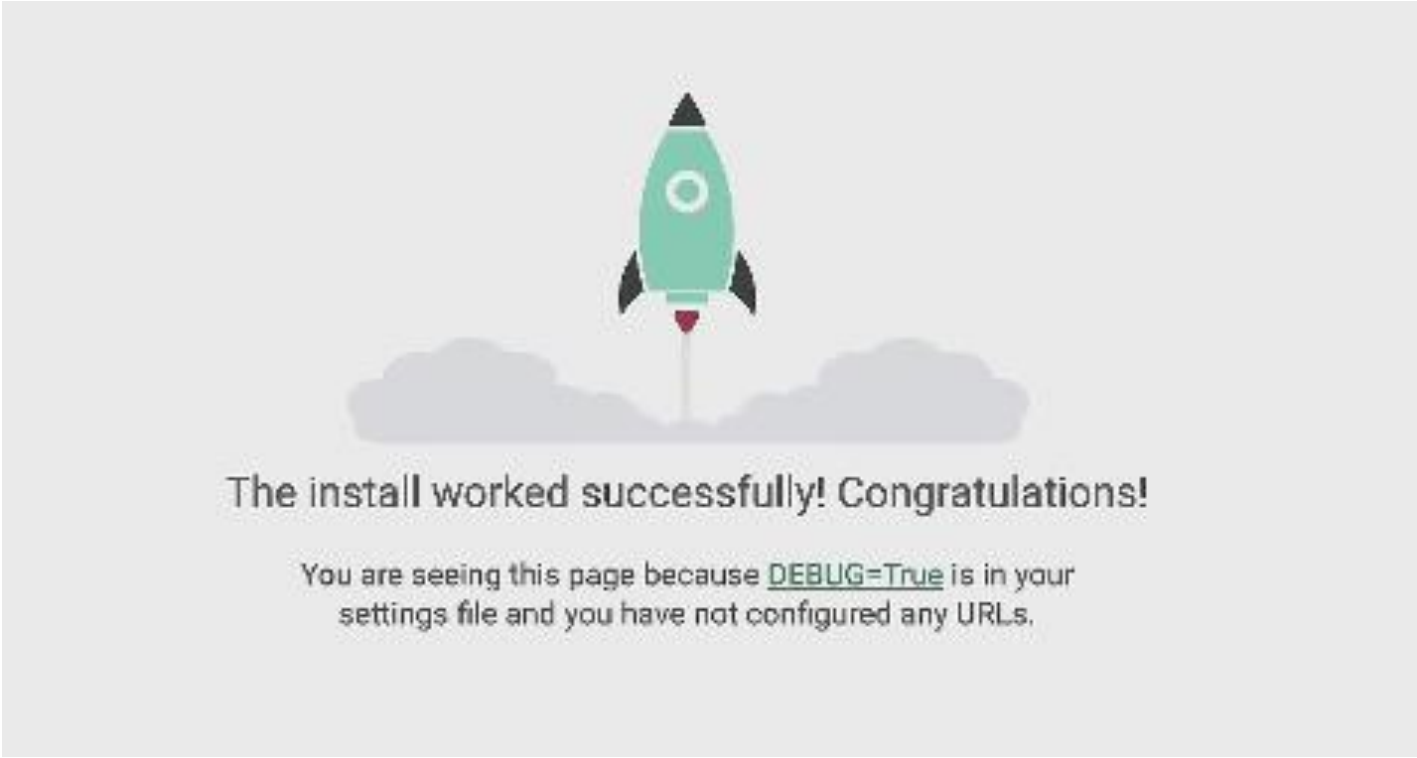
python manage.py runserver

Проєкт Alexandria



- cd Alexandria
- python manage.py runserver

http://localhost:8000



Мінімальна структура проекта Django

```
example/              - каталог проекта
    example/          - пакет проекта
        __init__.py
        settings.py   - налаштування проекта
        urls.py       - конфігурація URL адрес
        wsgi.py       - точка входу для wsgi
    manage.py         - командна утіліта Django
```

WSGI (Web Server Gateway Interface) - стандарт взаємодії між Python-програмою, що виконується на стороні сервера, і самим веб-сервером, наприклад Apache

Middleware

Крім додатків і серверів, стандарт WSGI дає визначення **middleware-компонентів** (компонентів проміжного шару), що надають інтерфейси як додатку, так і серверу.

Тобто для сервера middleware є додатком, а для додатка - сервером.

Це дозволяє складати «ланцюжки» WSGI-сумісних middleware-компонентів.

Middleware можуть брати на себе такі функції (але не обмежуються цим): **обробка сесій**, **аутентифікація** / авторизація, **управління URL** (маршрутизація запитів), **балансування навантаження**, пост-обробка вихідних даних (наприклад, **перевірка на валідність**), **зміна заголовків**.

Створення Django застосунків

python manage.py startup blog

Після створення нового Django застосунку необхідно зареєструвати його в списку застосунків, що належать проекту

settings.py:

```
INSTALLED_APPS = (
    . . . . . .
    ‘example’,
    ‘blog’
)
```

App

- `apps.py`: defines app meta-data, required for discovery
- `view.py`: functions or classes that serve web content
- `tests.py`: unit tests
- `models.py`: classes that map to database tables (ORM)
- `admin.py`: management code for models
- `urls.py`: app specific URL to view mappings
- `migrations/`: folder for database migration scripts

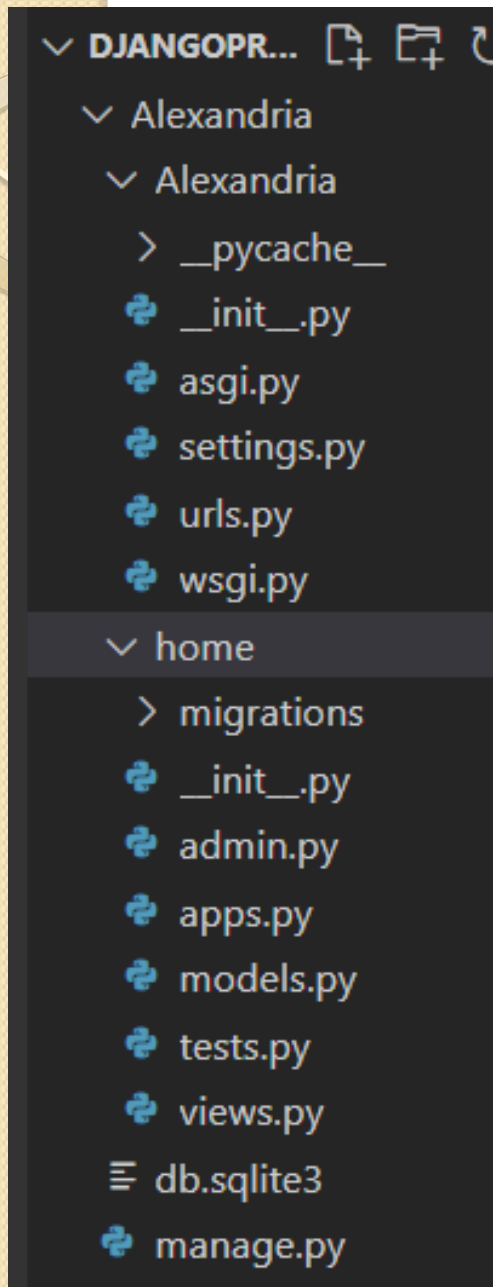
Структура Django застосунку (App)

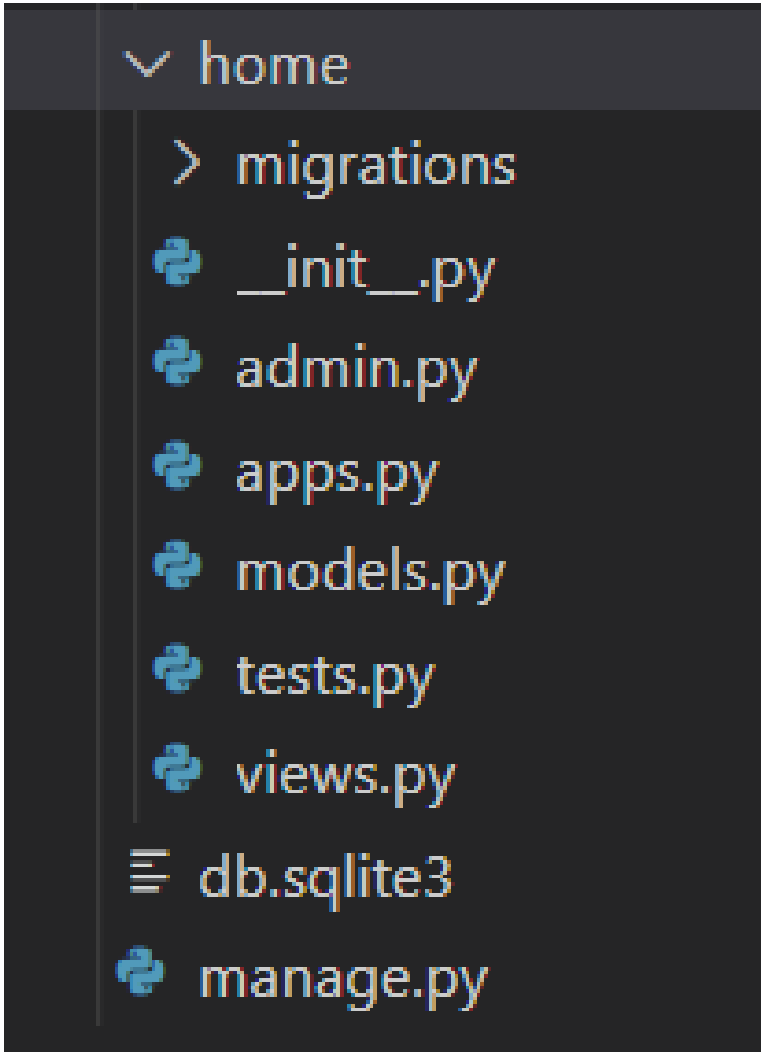
```
blog/                   - пакет застосунку
    __init__.py
    admin.py            - налаштування панелі адміна
    migrations/         - пакет, що містить міграції даних
        __init__.py
    forms.py            - опис форм введення інформації
    models.py           - опис таблиць БД
    tests.py            - unit tests
    views.py            - бізнес-логіка застосунку
```

Приклад

```
C:\Users\UIP>d:

D:\>cd djangoprojects

D:\DjangoProjects>p1env\scripts\activate

(P1Env) D:\DjangoProjects>cd Alexandria

(P1Env) D:\DjangoProjects\Alexandria>python manage.py startapp home

(P1Env) D:\DjangoProjects\Alexandria>
```

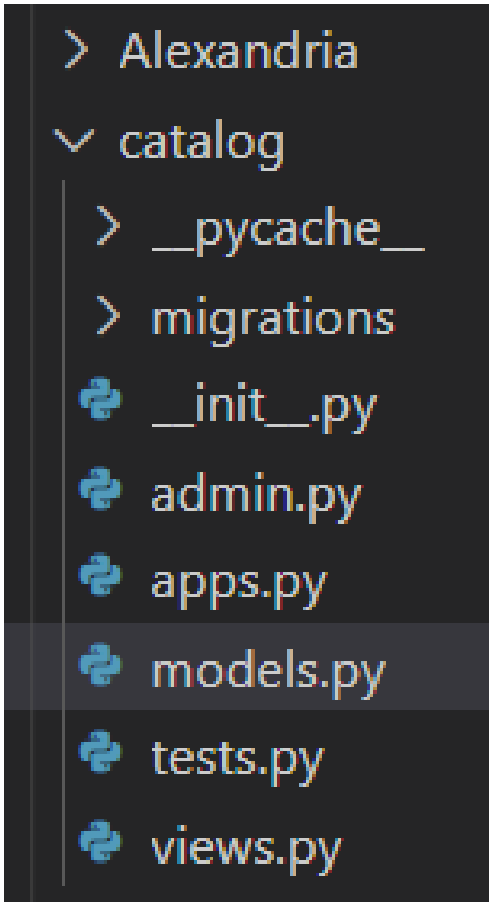
App home


DJANGOPR...
Alexandria
Alexandria
__pycache__
__init__.py
asgi.py
settings.py
urls.py
wsgi.py
home
migrations
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
db.sqlite3
manage.py


home
migrations
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
db.sqlite3
manage.py

App catalog

- python manage.py startapp catalog




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

Функція подання (view)

- Являє собою Python функцію, яка породжує вміст сторінки.

```
from django.http import HttpResponse
```

- def hello(request):

```
    return HttpResponse("Hello, world!")
```

Об’єкти запиту і відповіді (HttpRequest)

Клас HttpRequest являє собою HTTP-запит, отриманий з сторінки клієнта, і містить:

- Інформацію про URL (path і т.д.)
- HTTP-заголовки запиту
- Інформацію про відправлені дані (GET, POST та FILES)
- Інформацію про сесії та cookies (session та COOKIES)
- Інші змінні сервера

Об'єкти запиту і відповіді (HttpResponse)

- Слугує для конструювання відповіді на запит.

  HttpResponse(content = “”, mimetype = None, status = 200, content_type = DEFAULT_CONTENT_TYPE)

Але якщо необхідно додавати вміст поступово, можна використовувати об'єкт response як об'єкт файлу:

```
response = HttpResponse()
response.write(“<p>Here’s my Web Page</p>”)
response.write(“<p>Here’s another paragrath</p>”)
```

Підкласи HttpResponse

Модуль **django.http** містить декілька підкласів HttpResponse, які представляють різні види HTTP-відповідей:

- HttpResponseRedirect (302)
- HttpResponseNotFound (404)
- HttpResponseForbidden (403)
- HttpResponseServerError (500)

та інші

Конфігурування URLs

- Побудовано на принципі слабкої зв'язності.
- Кожна конфігурація повинна зберігатися у вигляді файлу з програмним кодом Python, зазвичай з іменем **urls.py**
- Такі файли повинні визначати об'єкт **urlpatters**, який отримується в результаті виклику функції **django.conf.patterns**
- Шлях до головного конфігуратора визначається в **settings.py** змінною ROOT_URLCONF

Конфігурування URLs (приклад)

```
from django.conf.urls import patterns, include, url
from example.views import hello

urlpatterns = patterns(“,
    #Examples
    (r’^hello/’, hello)
    (r’^$’, ‘example.views.home’, name=‘home’)
    (r’^example/’, include(‘example.blogs.urls’))
)
```

Система шаблонів Django

- Шаблон в Django являє собою рядок тексту в спеціальному форматі, призначений для відділення подання документу (view) від його даних.

- Мова шаблонів в Django не має ніякого відношення до Python, і створена для веб-технологів та дизайнерів інтерфейсів.

Ідеологічні основи шаблонів Django

- Бізнес-логіку треба відділити від логіки подання.
- Систаксис не треба прив'язувати до HTML/XML
- Передбачається, що дизайнери впевнено володіють HTML
- Не передбачається, що дизайнери вміють програмувати на Python
- Не ставилось завдання винайти нову мову програмування

Конфігурування та завантаження Django шаблонів

- Django пропонує зручний та потужний API для завантаження шаблонів з файлової системи.
- Спершу в settings.py треба задати:

```
TEMPLATE_DIRS = (
    "/шлях до/папки що/містить шаблони/"
)
```

Потім можна використовувати:

```
from django.shortcuts import render_to_response

def hello(request):
    return render_to_response('hello.html')
```

Common Template Tags

- `{% for item in container %} ... {% endfor %}`
  - loop variables: `forloop.counter`, `forloop.first`, `forloop.last`, ...
  - looping, supports an `empty` clause
- `{% if condition %} ... {% endif %}`
  - conditional block, supports `elif` and `else` clauses
- `{% comment "optional note" %} ... {% endcomment %}`
  - contents not included when rendered

- `{% url 'url_name' arg1 arg2 %}`
  - inserts named URL from the registry
  - supports both positional and keyword arguments
- `{% verbatim %} ... {% endverbatim %}`
  - template engine ignores contents of block
  - render characters that are template engine specific
  - can name the block

Common Filters

- `{{ data | upper }}`
  - turns `data` into upper case
  - there is also a `lower`
- `{{ data | first }}`
  - returns first item in `data`
  - there is also a `last`
- `{{ data | pluralize }}`
  - returns an “s” if `data` is >1
  - supports arguments to change how plural is done

Django Output Types

- views output content, usually by rendering a template
- HTML references ***static*** content like:
  - images
  - CSS
  - JavaScript
- static content can be served by Django, ***but shouldn't be in production***
- users can upload content: ***media***
- media content can be served by Django, ***but shouldn't be in production***

About Page

- HTML is very repetitive
- Template engine has inheritance tools, embodying DRY principle
- Use Bootstrap for styling

Composition Tags

- `{% extends 'filename' %}`
  - this template inherits from a parent template
- `{% block blockname %} ... {% endblock %}`
  - named section that can be inherited and overridden
  - `block.super` contains parent's content
- `{% include 'filename' %}`
  - insert a template inside of this template
- `{% load packagename %}`
  - loads a template tag set, like Python's `import` statement

Bootstrap Bootcamp

- Bootstrap uses a grid based system for layout
- Based on rows and 12 columns
- Responsive, adjusts to browser size, can specify wrap points
- Widgets
- `<nav>`
  - navigation bar
- `<main class="container">`
  - container for content

Review

- Django templates bring coding-like re-use to HTML
- Tags are actions similar to Python syntax
  - Loops
  - Conditionals
  - Inheritance / Composition
- Filters are data adaptors
  - Modify data items before presenting
- `render()` method is a shortcut for returning a rendered template in a view
  - pass context to the template

Review

- Static files can be presented by the dev server
  - The `{% static %}` tag maps URLs for static files
- Using a redirect to serve favicons
- Home URL
- Bootstrap is a layout and widget toolkit for HTML

ORM

- ***Object Relational Model***
- Object-oriented mechanism for mapping data structures to permanent storage
- Classes represent tables
- Abstracts away the database

Django's ORM

- Supports multiple databases:
  - PostgreSQL
  - MariaDB
  - MySQL
  - Oracle
  - SQLite
- Third-party back-ends
- Table migration management
- Tools for bulk-loading/saving data

Django's ORM

- Declarative, class-based object mapping
- Attributes in a class correspond to columns in the database
  - Over 25 fields supported with configurable options
  - Data fields: integers, characters, dates, ...
  - Relational fields: one-to-one, one-to-many, many-to-many
- Declaration of class defines mapping to tables
- Classes have ***managers*** for doing queries
- Instances of objects correspond to rows in the query result

Модель даних в Django

- Логічний опис структури даних, що зберігаються в БД, надається мовою Python.
- Всі моделі являють собою клас, що спадкує django.db.models.Model

#app/models.py (загальний синтаксис):

```
from django.db import models

class CustomModel(models.Model):
    name = models.CharField(max_length=30)
    body = models.TextField()
    email = models.EmailField(blank=True)
    number = models.IntegerField()
```

Моделі Author і Book

Файл catalog / models.py

```
from django.db import models

class Author(models.Model):
    last_name = models.CharField(max_length=50)
    first_name = models.CharField(max_length=50, blank=True)
    birth_year = models.IntegerField()
```

```
class Book(models.Model):
    title = models.TextField()
    author = models.ForeignKey(Author, on_delete=models.CASCADE, blank=True
        null=True)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
```

Міграції в Django

- Їх використовують для переносу змін в моделях (додавання/видалення поля або моделі і т.д.) на структуру бази даних.
- **Міграції – це дуже потужний механізм, який треба дуже обережно застосовувати в реальних умовах.**

manage.py makemigrations [<app_label>]

The Django REPL

```
ct@talkpython Alexandria$ ./manage.py shell
Python 3.10.0 (v3.10.0:b494f5935c, Oct  4 2021, 14:59:20) [Clang 12.0.5 (
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> from catalog.models import Book, Author
>>> dickens = Author.objects.create(last_name="Dickens", birth_year=1812)
>>> dickens
<Author: Author object (1)>
>>> dickens.last_name
'Dickens'
>>>
```

Query Manager in REPL

```
>>> dickens.last_name
'Dickens'
>>> Author.objects.all()
<QuerySet [<Author: Author object (1)>]>
>>> dickens.first_name
''
>>> dickens.first_name = "Charles"
>>> dickens.save()
>>>
```

Using objects.get()

```
>>> result = Author.objects.get(id=1)
>>> result
<Author: Author object (1)>
>>> result.last_name
'Dickens'
>>> result.id
1
>>> dickens.id
1
>>>
```

Using filter()

```
>>> Book.objects.create(title="A Tale of Two Cities", author=dickens)
<Book: Book object (1)>
>>> Book.objects.create(title="Oliver Twist", author=dickens)
<Book: Book object (2)>
>>> Book.objects.create(title="Hard Times", author=dickens)
<Book: Book object (3)>
>>> Book.objects.all()
<QuerySet [<Book: Book object (1)>, <Book: Book object (2)>, <Book: Book object (3)>]>
>>> Book.objects.filter(title="Hard Times")
<QuerySet [<Book: Book object (3)>]>
>>> Book.objects.filter(author=dickens)
<QuerySet [<Book: Book object (1)>, <Book: Book object (2)>, <Book: Book object (3)>]>
>>> dickens.book_set.all()
<QuerySet [<Book: Book object (1)>, <Book: Book object (2)>, <Book: Book object (3)>]>
>>>
```

Fixtures

- Django provides commands for importing and exporting data
- Supports:
  - JSON
  - JSONL
  - XML
  - YAML (if PyYAML is installed)
- Primary keys are hard-coded

Different database back-ends

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Supported Databases

- SQLite
  - Filename
- PostgresSQL
  - Service file
  - Password file
- MariaDB, MySQL, Oracle
  - Hostname, port, user, password
- Third-party libraries for other databases
  - Non-relational database backends available

More Django ORM

- Default query order
- Field level validations
- Row level validations
- Advanced query capabilities:
  - Grouping
  - Aggregation
  - By value
- Query objects
- Manual querying

Review

- Django comes with an ORM
- Map Python classes & objects to database tables and rows
- Field declarations on model classes represent columns
- Data columns: numbers, strings, booleans, etc.
- Relationship columns
- Migrations manage changes to ORM objects
- Fixtures allow you to export and import data
- Testing creates a special database

Admin Intro


Users
django
Super Users
Admin Interface
Filters
Parameterized URLs

Django Admin

- Most web applications are all about CRUD:
  - Create
  - Read
  - Update
  - Delete
- Django provides a library to map model objects to CRUD operations
- Web interface to the database

Django Users

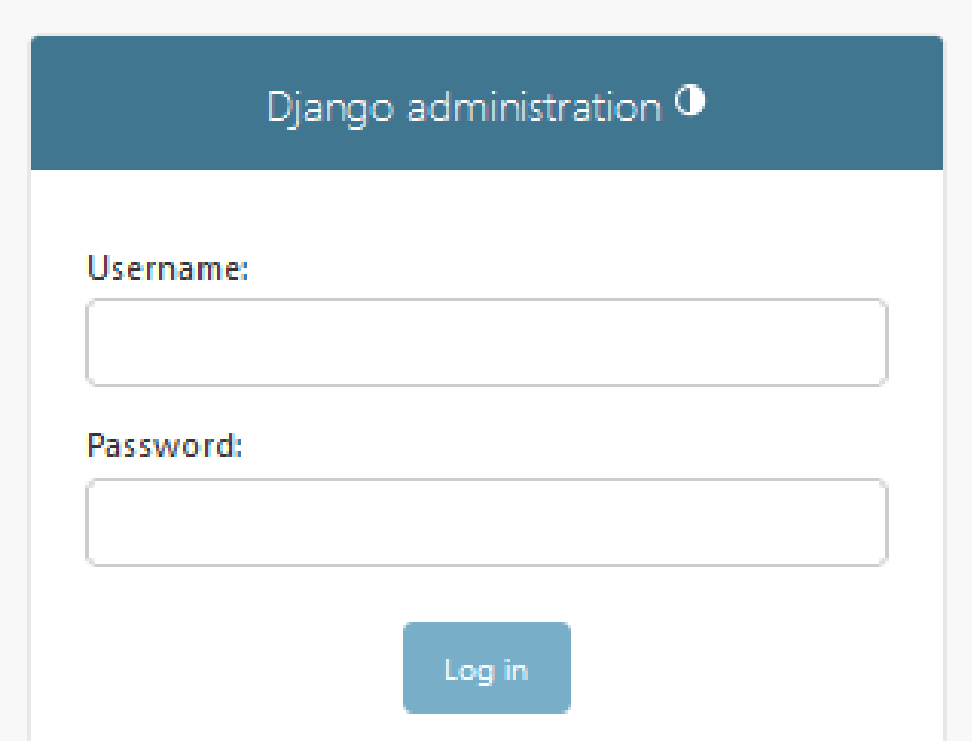
- Three classes of users:
  - Admin / Super-user
  - Staff
  - User
- Admin and staff have access to the Django admin
- Finer grained permissions possible

Створення суперюзера

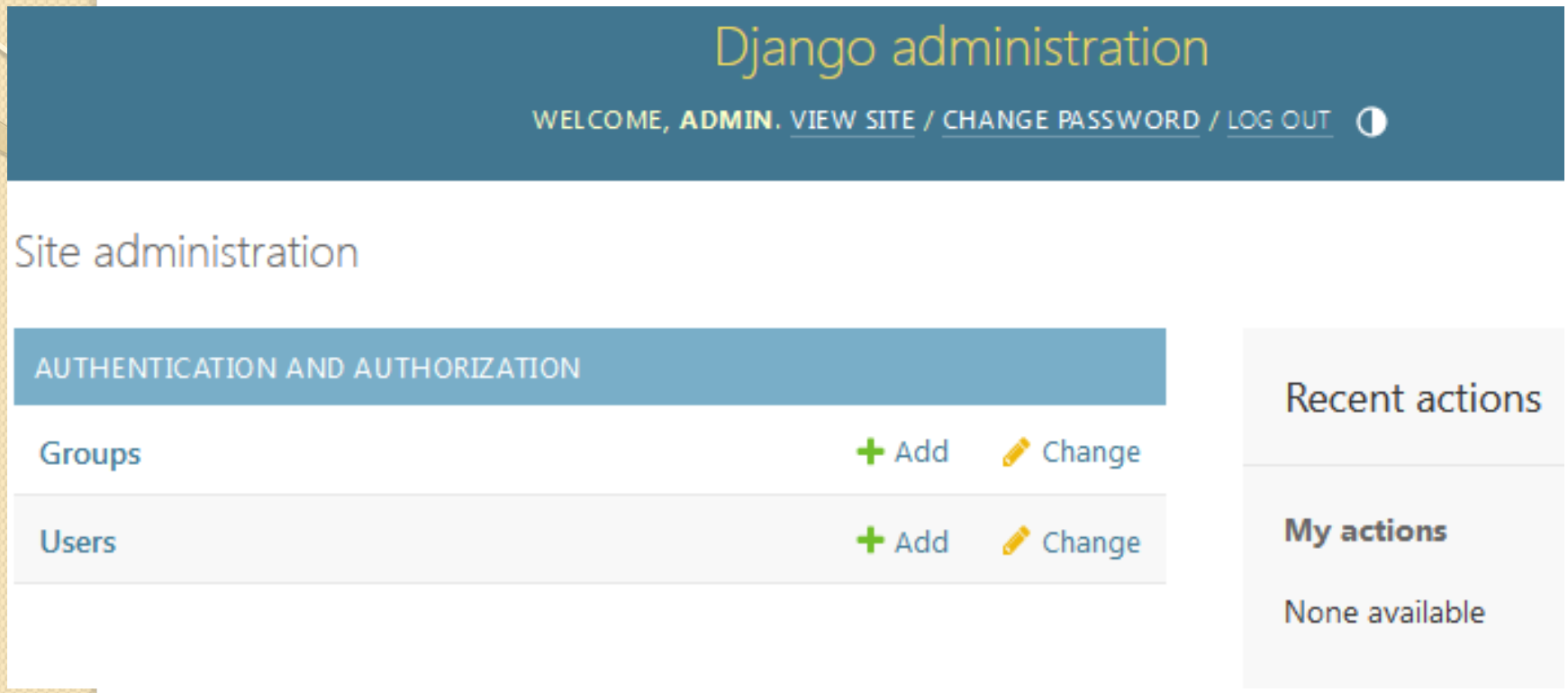
```
(P1Env) D:\DjangoProjects\Alexandria>python manage.py createsuperuser
Username (leave blank to use 'vip'): admin
Email address: admin@example.com
Password:
Password (again):
```

Loggin into the Admin

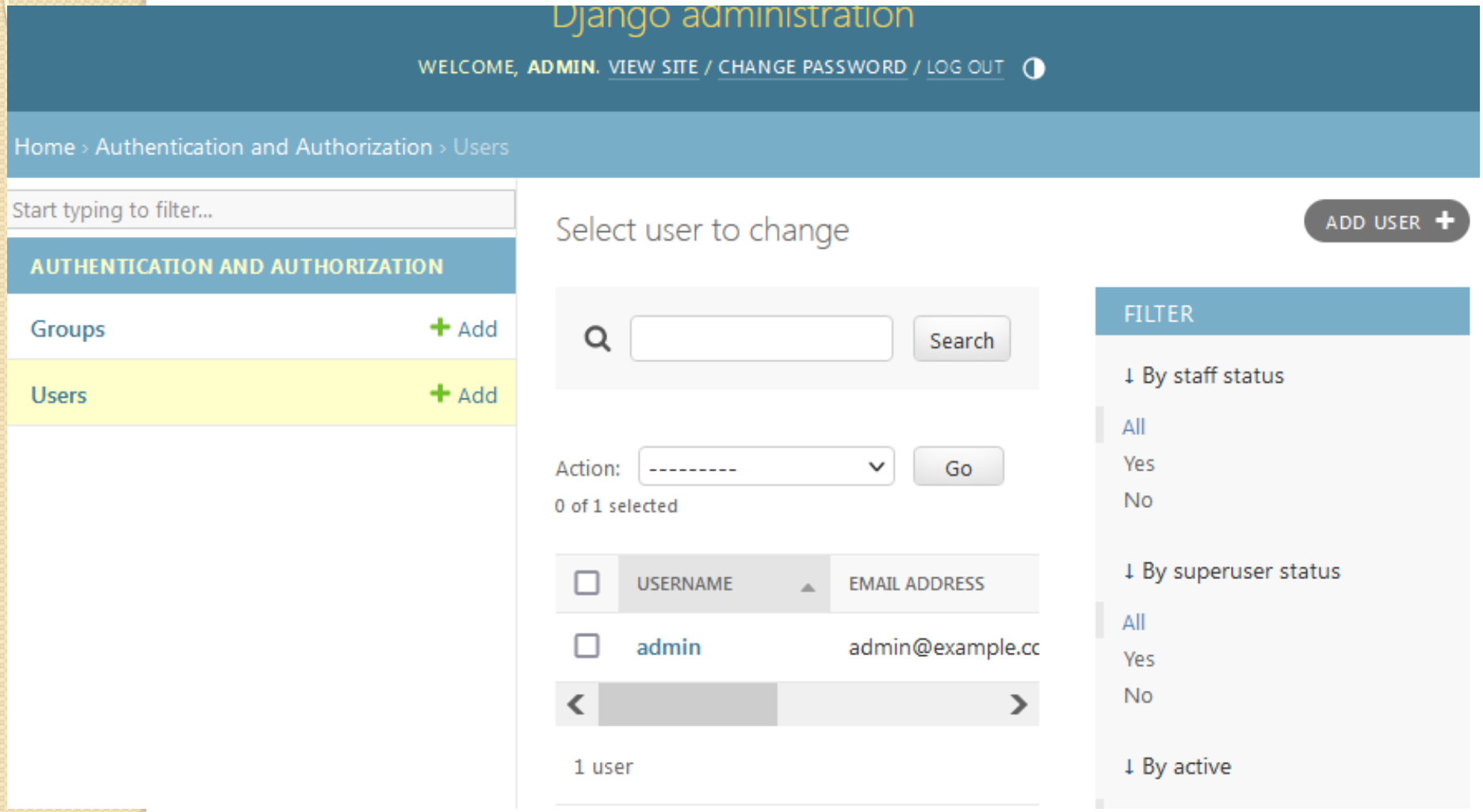
- python manage.py runserver
- http://localhost:8000/admin/login/



Майстерня адміністратора



Після натискання на Users



App's admin.py File

- `ModelAdmin` declares interface
- Register the `ModelAdmin` against a `Model` object
- Customize class to modify the web interface

Model Object Meta Data

- Model objects have a meta data area
- Controls behavior of queries and the admin:
  - Name in admin
  - Default order of queries
  - Database table name control
  - Permissions
  - Indexes
  - Constraints

Referencing Admin URLs

- Django provides a way of naming URLs
- No hard coding!
- Admin page URLs are named
- Cross reference items
- `list_display` supports any callable