Національний університет біоресурсів і
природокористування України

Створення веб-застосунку за допомогою фреймворку Django. Основи роботи з проектом та базою даних.

Частина 1

Лектор: к.ф.-м.н., доцент Кириченко Віктор Вікторович

Вступ до веб-розробки на Python

Веб-розробка: огляд

- Створення динамічних веб-сайтів та веб-застосунків
- Клієнт-серверна архітектура (фронтенд та бекенд)
- Протоколи: HTTP, HTTPS, WebSocket

Python для веб-розробки

- Простота синтаксису та швидкість розробки
- Потужна екосистема бібліотек та фреймворків
- Масштабованість та кросплатформеність

Роль Django

- Високорівневий фреймворк для швидкої розробки
- Реалізує архітектурний патерн MVC (MTV)
- Вбудовані компоненти: ORM, адмін-панель, автентифікація

Що таке Django?

Визначення

Django — це високорівневий Python-фреймворк для веб-розробки, що слідує архітектурному патерну "модель-представлення-шаблон" (MTV).

Ключові особливості

- ORM для роботи з БД
- Вбудована адмін-панель
- Система шаблонів
- Автентифікація та авторизація
- Захист від CSRF, XSS, SQL-ін'єкцій
- Міжнародізація та локалізація

Історія

- Створено у 2003 році
- Названий на честь джазового гітариста Django Reinhardt
- Відкритий код (BSD ліцензія)

Переваги

- Швидкість розробки
- Масштабованість
- Повна документація
- Активна спільнота

Встановлення Django

Передумови

- Python 3.8 або новіший версії
- Пакетний менеджер pip
- Базові знання командного рядка

Методи встановлення

За допомогою pip

```
$ pip install Django
```

У віртуальному середовищі

```
$ python -m venv myenv
$ source myenv/bin/activate
$ pip install Django
```

Перевірка версії

Команда перевірки

```
$ python -m django --version
```

Приклад результату

```
4.2.7
```

Рекомендується використовувати віртуальні середовища для ізоляції залежностей проекту

Основні поняття Django - Частина 1

Проект vs Застосунок

Проект

- Налаштування та конфігурація
- URL-маршрутизація
- Містить один або кілька застосунків

Застосунок

- Виконує конкретну функцію
- Моделі, представлення, шаблони
- Може бути використаний у різних проектах

Архітектура MTV (Model-Template-View)

Model
Структура даних, робота з БД

Template
Представлення даних користувачу

View
Логіка обробки запитів

Цикл "Запит-Відповідь"

Запит → URLconf → View → Template → Відповідь

Основні поняття Django - Частина 2

URL-адреси

Призначення

- Маршрутизація запитів
- Зв'язок URL з view-функціями
- Параметри в URL

Приклад

```
from django.urls import path
from . import views

urlpatterns = [
  path('articles/<int:id>/', views.article_detail),
]
```

Представлення (Views)

Функції представлення

- Приймають HTTP-запит
- Обробляють логіку
- Повертають HTTP-відповідь

Приклад

```
from django.http import HttpResponse

def home(request):
  return HttpResponse("Привіт, Django!")
```

Шаблони (Templates)

Особливості

- Відділення логіки від представлення
- Шаблонні теги та фільтри
- Наслідування шаблонів

Приклад

```
<h1>{{ title }}</h1>
<ul>
  {% for item in items %}
    <li>{{ item.name }}</li>
  {% endfor %}
</ul>
```

Створення нового Django-проекту

Команда для створення проекту

```
$ django-admin startproject project_name
```

Замініть `project_name` на назву вашого проекту

Структура проекту

```
project_name/
├── manage.py
└── project_name/
  ├── __init__.py
  ├── asgi.py
  ├── settings.py
  ├── urls.py
  └── wsgi.py
```

Ключові файли та їх призначення

manage.py

Утиліта командного рядка для взаємодії з проектом

settings.py

Налаштування та конфігурація проекту

urls.py

Оголошення URL-шляхів проекту

wsgi.py/asgi.py

Точки входу для WSGI/ASGI-сумісних веб-серверів

Структура Django-проекту - Детально

Детальна структура файлів

Основна директорія проекту

- `project_name/` - коренева директорія
- `manage.py` - утиліта командного рядка
- `project_name/` - конфігураційна директорія

Файли конфігураційної директорії

- `__init__.py` - позначає пакет Python
- `settings.py` - налаштування проекту
- `urls.py` - URL-маршрутизація
- `wsgi.py/asgi.py` - точки входу сервера

Призначення ключових файлів

settings.py

Основний файл конфігурації, містить налаштування БД, встановлені застосунки, статичні файли, шаблони, middleware тощо

manage.py

Скрипт для взаємодії з проектом: запуск сервера, створення міграцій, виконання команд тощо

urls.py

Визначає URL-шляхи проекту та зв'язує їх з відповідними view-функціями або класами

wsgi.py/asgi.py

Точки входу для веб-серверів, що підтримують WSGI/ASGI протоколи для розгортання проекту

Створення першого Django-застосунку

Команда для створення застосунку

```
$ python manage.py startapp app_name
```

Виконуйте цю команду з кореневої директорії проекту

Структура застосунку

Файлова структура

```
app_name/
├── __init__.py
├── admin.py
├── apps.py
├── migrations/
├── models.py
├── tests.py
└── views.py
```

Призначення файлів

- `models.py` - моделі даних
- `views.py` - логіка обробки запитів
- `admin.py` - налаштування адмін-панелі
- `apps.py` - конфігурація застосунку

Реєстрація застосунку

Файл settings.py

```
# INSTALLED_APPS section
INSTALLED_APPS = [
  'django.contrib.admin',
  'django.contrib.auth',
  'django.contrib.contenttypes',
  'django.contrib.sessions',
  'django.contrib.messages',
  'django.contrib.staticfiles',
  'app_name', # новий застосунок
]
```

Важливість реєстрації

- Дозволяє Django знаходити застосунок
- Увімкнення функціональності застосунку
- Необхідно для роботи з моделями та міграціями

Налаштування URL-адрес

URL-адреси проекту та застосунку

URL-адреси проекту

- Головний файл `urls.py` у кореневій директорії
- Визначає основні маршрути
- Підключає URL-адреси застосунків

URL-адреси застосунку

- Файл `urls.py` у директорії застосунку
- Визначає маршрути конкретного застосунку
- Підключається до головного файлу URL

Шаблони URL (URL patterns)

Приклад проекту

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

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

Приклад застосунку

```
from django.urls import path
from . import views

urlpatterns = [
  path('', views.index, name='index'),
  path('post/<int:id>/', views.post_detail),
]
```

Конвертери шляхів

Типи конвертерів

- `str` - рядок (без слешів)
- `int` - ціле число
- `slug` - slug-рядок
- `uuid` - UUID

Приклад використання

```
# URL: blog/post/2023/
path('post/<int:year>/', views.posts_by_year),

# URL: blog/post/hello-world/
path('post/<slug:post_slug>/', views.post_detail),
```

Створення представлень (Views)

Σ Функціональні представлення

Опис

- Прості функції Python
- Приймають об'єкт request
- Повертають HttpResponse

Приклад

```
from django.http import HttpResponse

def home(request):
  context = {
    'title': 'Головна сторінка'
  }
  return render(request, 'home.html', context)
```

Класові представлення

Переваги

- Об'єктно-орієнтований підхід
- Наслідування та перевизначення
- Вбудовані generic views

Приклад

```
from django.views import View

class HomeView(View):
  def get(self, request):
    context = {
      'title': 'Головна'
    }
    return render(request, 'home.html', context)
```

Повернення HTTP-відповідей

HttpResponse

```
from django.http import HttpResponse

def simple_view(request):
  return HttpResponse('Текст відповіді')
```

JsonResponse

```
from django.http import JsonResponse

def api_view(request):
  data = {'key': 'value'}
  return JsonResponse(data)
```

Redirect

```
from django.shortcuts import redirect

def old_page(request):
  return redirect('new-page')
```

Робота з шаблонами

Структура директорії шаблонів

Рекомендована структура

```
project_name/
├── app_name/
│   ├── templates/
│   │   └── app_name/
│   │       └── index.html
│   └── ... └── project_name/
  └── templates/
      └── base.html
```

Налаштування в settings.py

```
TEMPLATES = [
  {
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [BASE_DIR / 'templates'],
    'APP_DIRS': True,
  },
]
```

Наслідування шаблонів

Базовий шаблон (base.html)

```
<!DOCTYPE html>
<html>
<head>
  <title>{% block title %}{% endblock %}</title>
</head>
<body>
  {% block content %}{% endblock %}
</body>
</html>
```

Дочірній шаблон

```
{% extends 'base.html' %}

{% block title %}
  {{ page_title }}
{% endblock %}

{% block content %}
  <h1>{{ heading }}</h1>
  <p>{{ content }}</p>
{% endblock %}
```

Робота з шаблонами

Змінні та теги шаблонів

Змінні

```
<h1>{{ title }}</h1>
<p>{{ user.username }}</p>
<p>{{ post.date|date:"d.m.Y" }}</p>
```

Фільтри змінних: |upper, |lower, |date, |truncatewords

Теги

```
{% if user.is_authenticated %}
  <p>Вітаємо, {{ user.username }}!</p>
{% else %}
  <p>Будь ласка, увійдіть</p>
{% endif %}

{% for item in items %}
  <li>{{ item.name }}</li>
{% endfor %}
```

Вступ до моделей

Що таке моделі?

Визначення

- Моделі — це Python-класи, що описують структуру даних
- Кожна модель відповідає таблиці в базі даних
- Атрибути моделі — це поля таблиці
- Екземпляр моделі — це рядок у таблиці

Переваги Django ORM

- Абстрагування від SQL
- Незалежність від СУБД
- Захист від SQL-ін'єкцій
- Легке створення зв'язків між таблицями

Поля та опції моделей

Основні типи полів

- `CharField` — рядок обмеженої довжини
- `TextField` — довгий текст
- `IntegerField` — ціле число
- `DateTimeField` — дата і час
- `ForeignKey` — зв'язок один-до-багатьох

Поширені опції

- `null=True` — дозволяє NULL в БД
- `blank=True` — дозволяє пусте значення
- `default=значення` — значення за замовчуванням
- `unique=True` — унікальне значення
- `choices=список` — обмеження вибору

Вступ до моделей

<> Створення моделей в Django

```
from django.db import models

class Post(models.Model):
  title = models.CharField(max_length=200)
  content = models.TextField()
  created_date = models.DateTimeField(auto_now_add=True)
  published_date = models.DateTimeField(blank=True, null=True)
  author = models.ForeignKey('auth.User', on_delete=models.CASCADE)

  def publish(self):
    self.published_date = timezone.now()
    self.save()

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

Налаштування бази даних

Конфігурація бази даних

Файл settings.py

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

Підтримувані СУБД

- SQLite (за замовчуванням)
- PostgreSQL
- MySQL
- Oracle

Міграції

Команди міграцій

```
# Створення файлів міграцій
python manage.py makemigrations

# Застосування міграцій
python manage.py migrate

# Перевірка статусу
python manage.py showmigrations
```

Процес міграцій

- Аналіз змін у моделях
- Генерація файлів міграцій
- Застосування змін до БД
- Відстеження історії змін

Основи Django ORM

Створення записів

```
from .models import Post

# Створення
post = Post.objects.create(
  title='Новий пост',
  content='Текст...'
)
```

Читання записів

```
# Отримання всіх
posts = Post.objects.all()

# Фільтрація
posts = Post.objects.filter(
  title__contains='Django'
)
```

Оновлення та видалення

```
# Оновлення
post.title = 'Новий заголовок'
post.save()

# Видалення
post.delete()
```

Панель адміністратора Django

Створення суперкористувача

Команда створення

```
$ python manage.py createsuperuser
```

Виконуйте цю команду з кореневої директорії проекту

Процес створення

- Введення імені користувача
- Введення email (необов'язково)
- Введення пароля (двічі)

Реєстрація моделей в адмінці

Файл admin.py

```
from django.contrib import admin
from .models import Post

# Проста реєстрація
admin.site.register(Post)

# Розширена реєстрація
class PostAdmin(admin.ModelAdmin):
  list_display = ('title', 'created_date')
  list_filter = ('created_date',)
  search_fields = ('title', 'content')

admin.site.register(Post, PostAdmin)
```

Опції ModelAdmin

- `list_display` — поля для відображення
- `list_filter` — фільтри
- `search_fields` — пошук
- `ordering` — сортування

Використання адмін-інтерфейсу

Доступ до адмінки

- Запустіть сервер розробки
- Перейдіть за адресою `/admin/`
- Увійдіть з даними суперкористувача

Можливості

- CRUD-операції з даними
- Керування користувачами та правами
- Фільтрація та пошук записів

Переваги

- Готовий інтерфейс без кодування
- Гнучкість налаштувань
- Безпека та автентифікація

Практична вправа - Частина 1

Створення застосунку блогу

Кроки створення

- Створення нового застосунку
- Реєстрація в settings.py
- Створення моделі для постів
- Виконання міграцій

Команди

```
# Створення застосунку
python manage.py startapp blog

# Додавання до INSTALLED_APPS
# у файлі settings.py
'blog',

# Створення міграцій
python manage.py makemigrations

# Застосування міграцій
python manage.py migrate
```

Створення моделей для постів блогу

Модель Post

```
from django.db import models
from django.utils import timezone

class Post(models.Model):
  author = models.ForeignKey(
    'auth.User',
    on_delete=models.CASCADE
  )
  title = models.CharField(
    max_length=200
  )
  text = models.TextField()
  created_date = models.DateTimeField(
    default=timezone.now
  )
```

Додаткові поля та методи

```
  published_date = models.DateTimeField(
    blank=True, null=True
  )

  def publish(self):
    self.published_date = timezone.now()
    self.save()

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

Практична вправа - Частина 2

Реєстрація моделі в адмінці

Файл admin.py

```
from django.contrib import admin
from .models import Post

class PostAdmin(admin.ModelAdmin):
  list_display = ('title', 'author', 'created_date')
  list_filter = ('created_date',)
  search_fields = ('title', 'text')

admin.site.register(Post, PostAdmin)
```

Перевірка результату

- Запустіть сервер розробки
- Перейдіть в адмін-панель
- Перевірте наявність моделі Post
- Створіть тестовий запис

Налаштування URL та представлень

Файл blog/urls.py

```
from django.urls import path
from . import views

urlpatterns = [
  path('', views.post_list, name='post_list'),
  path('post/<int:pk>/', views.post_detail, name='post_detail'),
]
```

Підключення в проекті

```
# Файл project_name/urls.py
from django.contrib import admin
from django.urls import path, include

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

Практична вправа - Частина 3

Створення представлень

Файл blog/views.py

```
from django.shortcuts import render, get_object_or_404
from .models import Post

def post_list(request):
  posts = Post.objects.filter(
    published_date__isnull=False
  ).order_by('-published_date')
  return render(request, 'blog/post_list.html', {'posts': posts})
```

Представлення для детального перегляду

```
def post_detail(request, pk):
  post = get_object_or_404(
    Post, pk=pk,
    published_date__isnull=False
  )
  return render(
    request,
    'blog/post_detail.html',
    {'post': post}
  )
```

Практична вправа - Частина 2

Створення шаблонів

Структура шаблонів

```
blog/
└── templates/
    └── blog/
        ├── base.html
        ├── post_list.html
        └── post_detail.html
```

Базовий шаблон (base.html)

```
<!DOCTYPE html>
<html>
<head>
  <title>{% block title %}{% endblock %}</title>
</head>
<body>
  <div class="container">
    {% block content %}{% endblock %}
  </div>
</body>
</html>
```

Шаблон списку постів

```
{% extends 'blog/base.html' %}

{% block title %}Блог{% endblock %}

{% block content %}
  <h1>Блог</h1>
  {% for post in posts %}
    <div>
      <h2><a href="{% url 'post_detail' pk=post.pk %}">{{ post.title }}
</a></h2>
      <p>{{ post.published_date }}</p>
    </div>
  {% endfor %}
{% endblock %}
```

Шаблон детального перегляду

```
{% extends 'blog/base.html' %}

{% block title %}{{ post.title }}{% endblock %}

{% block content %}
  <div>
    <h1>{{ post.title }}</h1>
    <p>{{ post.published_date }}</p>
    <p>{{ post.text|linebreaks }}</p>
  </div>
{% endblock %}
```

Запуск сервера розробки

Запуск сервера

Базова команда

```
$ python manage.py runserver
```

За замовчуванням сервер запускається на порті 8000

Додаткові опції

```
# Зміна порту
python manage.py runserver 8080

# Доступ з інших пристроїв
python manage.py runserver 0.0.0.0:8000
```

Доступ до застосунку

URL-адреси для доступу

- Головна сторінка: `http://127.0.0.1:8000/`
- Адмін-панель: `http://127.0.0.1:8000/admin/`
- Блог: `http://127.0.0.1:8000/blog/`

Приклад роботи застосунку

```
# Вивід у консолі при запуску
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

Django version 4.2, using settings 'myproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Поради з налагодження

Налаштування DEBUG

- Увімкніть `DEBUG = True` у settings.py
- Детальні сторінки помилок
- Вимкніть у продакшені!

Логування

- Використовуйте `print()` для швидкого дебагу
- Налаштуйте логування у settings.py
- Перевіряйте консоль сервера

Поширені проблеми

- URL не знайдено — перевірте urls.py
- Помилка шаблону — перевірте синтаксис
- Помилка БД — запустіть міграції

Найкращі практики

Організація коду

Структура проекту

- Розділяйте логіку за застосунками
- Використовуйте абстрактні моделі
- Створюйте службові файли (utils.py)
- Використовуйте менеджери контексту

Конвенції

- Іменування змінних та функцій
- Документування коду (docstrings)
- Використовуйте версіонування (Git)
- Дотримуйтесь PEP 8

Врахування безпеки

Захист даних

- Використовуйте CSRF-токени
- Захист від XSS-атак
- Валідація вхідних даних

Автентифікація

- Складні паролі
- Двостороння автентифікація
- Обмеження спроб входу

Налаштування

- DEBUG = False у продакшені
- Захист секретних ключів
- Налаштуйте CORS

Оптимізація продуктивності

Оптимізація запитів

- Використовуйте `select_related` та `prefetch_related`
- Уникайте N+1 проблеми
- Індексуйте поля в БД
- Використовуйте `only()` та `defer()`

Кешування

- Кешування шаблонів
- Кешування запитів до БД
- Статичні файли та медіа
- Використовуйте Redis або Memcached

Висновки

- Django — потужний фреймворк для веб-розробки на Python, що забезпечує швидке створення складних застосунків
- Архітектура MTV (Model-Template-View) забезпечує чіткий поділ логіки, даних та представлення
- Моделі Django дозволяють легко працювати з базами даних через ORM, абстрагуючись від SQL
- Система URL-маршрутизації та представлень забезпечує гнучке оброблення HTTP-запитів
- Шаблони Django відокремлюють логіку від представлення, спрощуючи розробку інтерфейсу
- Вбудована адмін-панель дозволяє легко керувати даними без додаткового кодування
- Дотримання найкращих практик забезпечує безпеку, продуктивність та масштабованість застосунків
- Практичне застосування отриманих знань дозволяє створювати функціональні веб-застосунки