УДК 004.056
В.М. Мельник, Н.В. Багнюк, А.І. Нагорнюк
Луцький національний технічний університет

# ОСОБЛИВОСТІ РОБОТИ З БАЗАМИ ДАНИХ У ФРЕЙМВОРКУ DJANGO

**Мельник В.М., Багнюк Н.В., Нагорнюк А.І. Особливості робота з базами даних у фреймворку Django.** В даній роботі проведено дослідження з використання особливостей роботи з базами даних у фреймворку Django. Зокрема, було висвітлено характерні особливості його архітектури, які впливають на роботу з базами даних, а також з'ясовано особливості налаштування серверів для підтримки роботи даного web-фреймворку. Проведені дослідження особливостей з'єднання з локальною та віддаленою базами, таких як PostgreSQL, MySQL, SQLite та Oracle, виявили основні принципи роботи з моделями даних, в яких формується макет бази даних з додатковими метаданими. З'ясовано, що в ході розробки додатків цей процес включає в себе створення моделі, її редагування, впровадження за допомогою міграцій, що може мати характерні особливості у її створенні і застосуванні.
**Ключові слова**: фреймворк Django, бази даних, моделі, міграції, особливості використання, Pyton.

**Мельник В.М., Багнюк Н.В., Нагорнюк А.И. Особенности работы с базами данных в фреймворка Django.** В данной работе проведено исследование по использованию особенностей работы с базами данных в фреймворке Django. В частности, были рассмотрены характерные особенности его архитектуры, которые влияют на работу с базами данных, а также озвучены особенности настройки серверов для поддержки работы данного web-фреймворка. Проведенные исследования особенностей соединения с локальной и удаленной базами, таких как PostgreSQL, MySQL, SQLite и Oracle, выявили основные принципы работы с моделями данных, в которых формируется макет базы данных з дополнительными метаданными. Установлено, что в ходе разработки приложений этот процесс включает в себя создание модели, ее редактирования, внедрения с помощью миграций, которая могут иметь характерные особенности в их создании и применении.
**Ключевые слова:** фреймворк Django, базы данных, модели, миграции, особенности использования, Pyton.

**Melnik V.M., Nataliia Bahniuk., Nagornyuk A.I. Work features with databases in the Django framework.** In this paper were conducted a study to use the working features with databases in the Django framework. In particular, there were considered some its architecture features that affect the work with databases, and were announced the server configuration features to support this web-frame to work. The research of connection features with local and remote databases such as PostgreSQL, MySQL, SQLite and Oracle has revealed the basic principles to work with data models that form a database layout with additional metadata. It is revealed that during the development of applications, this process involves the model creation, its editing, implementation supported by a migration, which may have distinctive features in its creation and application.
**Keywords:** framework Django, databases, models, migration, peculiarities of use, Pyton.

**1. Постановка проблеми.** Вільний фреймворк Django на сьогодні являється перспективним середовищем розробки web-додатків на мові Python. Сукупний сайт на Django будується з одного або декількох додатків, які рекомендується розробляти з можливістю відчуження і під'єднання. Це одна з істотних архітектурних відмінностей цього фреймворка від деяких інших середовищ (наприклад, Ruby on Rails). Один з основних принципів фреймворка – DRY (Do not repeat yourself). То ж, на відміну від інших фреймворків, обробники URL в Django конфігуруються явно за допомогою регулярних виразів, а не виводяться автоматично зі структури моделей контроллерів. Для роботи з базою даних Django використовує власний ORM (Object-Relational Mapping), в якому модель даних описується класами Python і згідно неї генерується схема самої бази даних [1, 6].

Згідно з даними ресурсів db-engines та g2crowd, до найбільш популярних баз даних відносять Oracle, MySQL, Microsoft SQL Server, PostgreSQL, MongoDB, DB2, Mircosoft Access та Redis [8, 9].

Архітектура Django досить схожа на послідовність «модель-представлення-контролер» (Model-View-Controller – MVC). Контроллер класичної моделі MVC приблизно відповідає рівню, який в Django називається "Представлення" (View), а презентаційна логіка його реалізується в Django рівнем шаблонів (Templates). Через це рівневу архітектуру Django часто називають "модель-шаблон-подання" (Model-Templates-View), або скорочено MTV [1, 2, 6].

Первісна розробка Django як засобу для поєднання роботи з новими ресурсами досить сильно відбилася на його архітектурі. Він надає ряд засобів, які допомагають у швидкій розробці веб-сайтів інформаційного характеру. Так, наприклад, розробнику не потрібно створювати контроллери та сторінки для адміністративної частини сайту, тому що в Django є вбудований додаток для керування

вмістом, який можна включити в будь-який сайт, розроблений на Django. Він може управляти відразу декількома сайтами на одному сервері. Адміністративний додаток дозволяє створювати, змінювати і видаляти будь-які об'єкти для наповнення сайту, протоколюючи всі здійснені дії, надаючи інтерфейс для управління користувачами і групами з пооб'єктним призначенням прав доступу. У дистрибутив Django також включені програми для системи коментарів – синдикації RSS і Atom «статичних сторінок», якими можна управляти без необхідності написання контролерів та подання, з перенаправленням URL і інше [3, 6].

**2. Аналіз останніх досліджень.** Django проектувався для роботи під управлінням Apache з модулем mod pyton і з використанням PostgreSQL в якості бази даних [4 ,6]. Із включенням підтримки WSGI Django може працювати під управлінням FastCGI, mod wsgi, або SCGI на Apache і інших серверах (таких як lighttpd, nginx і інших), а також з сервером uWSGI. В наш час, крім бази даних PostgreSQL, Django може співпрацювати з іншими системами управління базами даних (СУБД), такими як MySQL, SQLite, Microsoft SQL Server, DB2, Firebird, SQL Anywhere і Oracle. У складі Django присутній власний веб-сервер, призначений для розробки додатків, який автоматично визначає зміни в файлах вихідного коду проекту і перезапускається, що і прискорює процес розробки на Python. Але при цьому він працює в однопотоковому режимі і придатний тільки для розробки і налагодження програми.

Django володіє деякими можливостями для організації роботи з базами даних:

- ORM, API доступу до БД з підтримкою транзакцій;
- вбудований інтерфейс адміністратора з уже наявними перекладами багатьма мовами;
- диспетчер URL на основі регулярних виразів;
- система розширення шаблонів з тегами і наслідуванням;
- система кешування;
- інтернаціоналізація;
- архітектура з можливістю під'єднання додатків, які можна встановлювати на будь-які Django-сайти;
- «Generic views» – шаблони функцій контроллерів;
- авторизація та аутентифікація з підключенням зовнішніх модулів аутентифікації, таких як LDAP, OpenID та інших;
- система фільтрів (middleware) для побудови додаткових обробників запитів, як наприклад включені в дистрибутив фільтри для кешування, стиснення, нормалізації URL і підтримки анонімних сесій;
- бібліотека для роботи з формами (наслідування, побудова форм за існуючою моделлю БД);
- вбудована автоматична документація за тегами шаблонів і моделями даних, доступна через адміністративний додаток;

Деякі компоненти фреймворка між собою пов'язані слабо, тому їх можна досить просто замінювати на аналогічні. Наприклад, замість вбудованих шаблонів можна використовувати Mako або Jinja [6, 10]. У той же час замінювати ряд компонентів, наприклад, ORM, досить складно.

Крім можливостей, вбудованих в ядро фреймворка, існують пакети, що розширюють його можливості, Можливості, що надаються пакетами, а також повний перелік таких пакетів зручно відстежувати через спеціальний ресурс [7].

Веб-фреймворк Django використовується також в таких великих і відомих сайтах, як Instagram, Disqus, Mozilla, The Washington Times, Pinterest, YouTube, Google та інших [6, 8].

*Метою даної роботи* є дослідження особливостей роботи з базами даних у веб-фреймворку Django, яке включає налаштування під'єднання до бази даних, створення таблиць та записів у базі, створення моделей та створення і запровадження міграцій.

**3. Обговорення особливостей роботи з базами даних у Django.**

Щоб налаштувати з'єднання з базою даних, необхідно ввести зміни у файл settings.py. Даний файл представляє звичайний модуль python з глобальними змінними рівня модуля, що представляють налаштування Django. За замовчуванням в конфігурації використовується база даних SQLite, яку

включено в python і не потрібно інсталювати ніяких додаткових модулів. Якщо необхідно використовувати іншу базу даних, потрібно встановити відповідні прив'язки до неї та змінити наступні ключі в пункті «default» так, щоб вони відповідали налаштуванням з'єднання з базою даних:

- ENGINE – це движок бази даних для її використання. Існують вбудовані бази даних, такі як "django.db.backends.sqlite3", "django.db.backends.postgresql", "django.db.backends.mysql" або "django.db.backends.oracle". На додаток до офіційно підтримуваних баз даних, існують так звані «бекенди», надані третіми сторонами, що дозволяють використовувати інші бази даних з Django, такі як SAP SQL Anywhere, IBM DB2, Microsoft SQL Server, Firebird, ODBC. Версії Django та функції ORM, що підтримуються цими неофіційними бекендами, значно, дещо відрізняються. Запити щодо конкретних можливостей цих неофіційних бекендів, а також будь-які запити підтримки повинні бути спрямовані на їх канали підтримки, надані кожною третьою стороною проекту.
- NAME – назва бази даних. Якщо використовується SQLite, база даних буде зберігатися у вигляді файла. У цьому випадку NAME повинно містити повний абсолютний шлях, включаючи назву файла. За замовчуванням приймається значення os.path.join (BASE_DIR, 'db.sqlite3'), яке зберігатиме файл у каталозі проекту.

Якщо в якості бази даних використовується не SQLite, а інша база, потрібно додати додаткові параметри, такі як USER, PASSWORD та HOST. Найпростіший можливий файл налаштувань для однієї бази даних за допомогою SQLite може мати наступний зміст:

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

Приклад для налаштування з'єднання з базою даних PostgreSQL може мати наступний вигляд:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mydatabase',
        'USER': 'mydatabaseuser',
        'PASSWORD': 'mypassword',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

Після налаштування з'єднання з базою даних необхідно створити моделі: макет бази даних з додатковими метаданими. Модель являє собою єдине, остаточне джерело даних користувача. Вона містить основні поля та поведінку даних, які зберігаються. Django дотримується принципу DRY(Don't repeat yourself), тобто кожна окрема концепція та/або фрагмент даних повинні перебувати в одному, і тільки в одному місці. Надмірність погана. Нормалізація хороша. Отже, мета полягає в тому, щоб визначити модель даних в одному місці та автоматично отримувати від неї як можна більше необхідної корисної інформації. Цей принцип включає в себе міграції, які, на відміну від Ruby On

Rails, наприклад, повністю виведені з файла моделей. Вони, по суті, є лише історією, яку Django може прокручувати, щоб оновити схему бази даних для того, щоб вона відповідала поточним моделям.

Для прикладу зобразимо дві моделі [2, 8]: Question і Choice. Question містить власне запитання і дату його публікації. Модель Choice має два поля: текст вибору і поле підрахунку кількості голосів. Кожен Choice асоційований з відповідним Question. Ці концепти зображені у вигляді звичайних класів python:

```
from django.db import models

class Question(models.Model):
    question_text = models.CharField(max_length = 200)
    pub_date = models.DateTimeField('date published')

class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length = 200)
    votes = models.IntegerField(default = 0)
```

Програмний код є досить простим. Кожна модель представлена класом, який є підкласом django.db.models.Model. Кожна модель має ряд власних класових змінних, кожна з яких являє собою поле бази даних в моделі. Кожне поле представляється екземпляром класу Field. Наприклад, поле CharField призначене для символьних полів, а поле DateTimeField – для часових подій [8]. Ця інформація говорить Django, який тип даних утримує кожне поле. Ім'я кожного екземпляра поля (наприклад, question_text або pub_date) – це назва поля, у машино-дружньому форматі. Це значення буде використовуватися у коді Python, і база даних буде використовувати його як назву стовпця. Деякі поля потребують аргументів. Наприклад, CharField вимагає аргумент max_length – максимальної довжини символьного поля. Подібний підхід використовується не тільки в схемі бази даних, але й при валідації. Поле також може мати різні необов'язкові аргументи, а в даному випадку нами встановлено значення за замовчуванням 0. Зв'язки між моделями визначаються з використанням ForeignKey. Це говорить Django, що кожен вибір пов'язаний з одним питанням. Django підтримує всі можливі відносини з базою даних: "багато-до-одного", "багато-до-багатьох" та "один-до-одного" [8].

Перш ніж використовувати моделі, їх необхідно активувати. Для цього необхідно перш за все включити додаток до проекту. Щоб включити додаток у проект, потрібно додати посилання на його клас конфігурації в налаштуваннях INSTALLED_APPS. Після цього необхідно виконати команду makemigrations, в яку слід передати назву додатка. Під час запуску makemigrations, Django повідомляється, що було внесено деякі зміни до моделей і необхідно, щоб зміни зберіглися як міграція. Міграції містять інформацію, як Django зберігає зміни до моделей, а, отже, і до схеми бази даних, і просто представляються файлами на фізичному носії. Існує команда migrate, яка стартує міграції і автоматично керує схемою бази даних. Дана команда виконує всі незастосовані міграції, а Django відстежує, які з них використовуються за допомогою спеціальної таблиці у базі даних з іменем django_migrations і запускає їх замість бази даних. Тобто, по суті, відбувається синхронізація змін, внесених до моделей, із схемою в базі даних. Міграції є дуже потужним інструментом, що дозволяють змінювати моделі протягом часу, коли проект розробляється без необхідності видаляти базу даних або таблиці та створювати нові. Міграції зазвичай спеціалізується на оновленні бази даних в реальному часі без втрати даних [4, 8].

Отже, щоб ввести зміни до моделі, необхідно:

- Змінити моделі у файлі models.py;
- Виконати python manage.py makemigrations для створення міграції для цих змін;

- Виконати python manage.py migrate, щоб застосувати ці зміни до бази даних.

Причини того, що існують окремі команди для створення та застосування міграцій, пов'язані зі створенням чи модифікацією їх до своєї системи керування власними версіями та надсиланням їх власним додатком. Вони не тільки полегшують розробку, а і можуть бути використані іншими розробниками в ході виробничого програмування [8].

**4. Висновки.** У даній роботі було розглянуто веб-фреймворк Django для роботи з базами даних та, висвітлено особливості його архітектури, а також було з'ясовано особливості налаштування серверів для підтримки роботи даного веб-фреймворку. Проведені дослідження можливостей з'єднання з локальною та віддаленою базами даних PostgreSQL, MySQL, SQLite та Oracle виявили основні принципи та організацію роботи з моделями даних. Відмічається, що цей процес включає в себе створення моделі, її редагування, впровадження за допомогою міграцій, які, в свою чергу, можуть мати характерні особливості у їх створенні і застосуванні.

1. Jacob Kaplan-Moss, Adrian Holovaty. The Definitive Guide to Django: Web Development Done Right.
2. Nigel George. Build Your First Website with Python and Django.
3. Leif Azzopardi, David Maxwell, Tango With Django: A beginner's Guide to Web Development With Python / Django 1.9.
4. Daniel Roy Greenfield, Andrey Roy Greenfeld. Two Scoops of Django 1.11: Best Practices for the Django Web Framework.
5. www.djangopackages.com/
6. https://uk.wikipedia.org/wiki/Django.
7. https://docs.djangoproject.com/en/2.0/.