Лабораторна робота 5. Розробка сайту за технологією Python+Django.

Завдання: За допомогою фреймворку Django розробити сайт простого блогу з наступними можливостями:

- Користувач-адміністратор може створювати, додавати, видаляти, редагувати записи (повідомлення) блогу.
- Звичайним користувач може переглядати список повідомлень блогу та детальну інформацію по кожному повідомленню.
- Передбачити розбиття на сторінки списку повідомлень блогу.
- На сторінці списку повідомлень розмістити посилання на RSS-канал, що відображає 5 останніх повідомлень блогу.

Теоретичні відомості, методичні вказівки та хід роботи.

Установка Django в Windows

1) Інсталяція Python (https://www.python.org/). На березень 2018 маємо реліз Python 3.6.4. Зверніть увагу на другу сторінку "Customize" майстера установки, вам потрібно пролистати вниз і вибрати опцію "Add python.exe to the Path" (додати python.exe в системну змінну Path).

2) Створення віртуального середовища. Для роботи з Django необхідно спочатку створити віртуальне середовище для роботи. Віртуальне середовище являє собою підрозділ системи, в якому ви можете встановлювати пакети в ізоляції від решти пакетів Python. Відділення бібліотек одного проекту від інших проектів принесе користь при розгортанні вашого застосунку на сервері. Створіть для проекту новий каталог, наприклад з ім'ям D:\DjangoSites, перейдіть в цей каталог в термінальному режимі і створіть віртуальне середовище. Якщо ви працюєте в Python 3, то зможете створити віртуальне середовище наступною командою:
   ```
   python -m venv djangosites_env
   ```
   Команда запускає модуль venv і використовує його для створення віртуального середовища з ім'ям djangosites_env.

3) Активізація віртуального середовища. Після того, як віртуальне середовище буде створене, його необхідно активізувати наступною командою (для Windows):
   ```
   djangosites_env\Scripts\activate
   ```
   Команда запускає сценарій activate з каталогу djangosites_env\Scripts. Коли середовище активізується, його ім'я виводиться в круглих

дужках. Тепер ви можете встановлювати пакети в середовищі і використовувати ті пакети, що були встановлені раніше. Пакети, встановлені в djangosites_env, будуть доступні тільки в той час, поки середовище залишається активним. Щоб завершити використання віртуального середовища, введіть команду deactivate.

4) Установка Django. Після того як ви створили своє віртуальне середовище і активізували його, встановіть Django:
   `pip install django==1.11.5`
   Так як ви працюєте в віртуальному середовищі, ця команда виглядає однаково в усіх системах. Версію Django можна не вказувати, тоді буде встановлена остання версія. (Якщо ви отримуєте помилку коли викликаєте pip на Windows, перевірте чи шлях до вашого проекту не містить пробілів, наголосів чи спеціальних символів).

5) Створення проекту mysite в Django. Не виходячи з активного віртуального середовища, введіть наступні команди для створення нового проекту (не забудьте про крапку . в кінці):
   - **`django-admin.py startproject mysite .`**
   - **`dir`** щоб побачити вміст каталогу D:\DjangoSites (там будуть каталоги djangosites_env і mysite та файл manage.py)
   - **`dir mysite`** щоб побачити вміст каталогу D:\DjangoSites\mysite (там будуть файли __init__.py, settings.py, urls.py, wsgi.py).

Файл manage.py – це коротка утиліта командного рядка, яка отримує команди і передає їх відповідній частині Django для виконання. Ми використовуємо ці команди для управління такими завданнями, як робота з базами даних і запуск серверів.

В каталозі mysite знаходяться чотири файли, найважливішими з яких є файли settings.py, urls.py і wsgi.py. Файл settings.py визначає те, як Django взаємодіє з вашою системою і управляє вашим проектом. Ми змінимо деякі з існуючих налаштувань і додамо кілька нових налаштувань в ході розробки проекту. Файл urls.py повідомляє Django, які сторінки слід будувати у відповідь на запити браузера. Файл wsgi.py допомагає Django надавати створені файли (ім'я файлу є скороченням від «Web Server Gateway Interface»).

Оскільки ми використовуємо Python 3.5, який постачається з вбудованою базою SQLite, ми можемо використовувати цю базу даних у нашому проекті для розробки. Якщо ви плануєте розгортати вашу програму у виробничому середовищі, ви повинні використовувати розширену базу даних, таку як MySQL, Oracle або PostgreSQL. Згенерований файл settings.py містить базову конфігурацію для використання бази даних SQLite та список програм

Django, які за замовчуванням працюють у вашому проекті. Нам потрібно створити таблиці в базі даних для початкового застосування.

6) Створення бази даних. Подамо команду
   `cd mysite`
   шоб перейти в каталог mysite. Щоб створити базу даних для проекту mysite, введіть наступну команду
   `python manage.py migrate`
   В терміналі ви побачите процес створення бази даних. Кожна зміна бази даних називається міграцією. Перше виконання команди migrate наказує Django перевірити, що база даних відповідає поточному стану проекту. Під час першого виконання цієї команди в новому проекті з використанням SQLite Django створює нову базу даних за нас. Django повідомляє про створення таблиць бази даних, необхідних для зберігання інформації, використовуваної в проекті, а потім перевіряє, що структура бази даних відповідає поточному коду. Django створює файл з ім'ям db.sqlite3.

7) Запуск Django Development Server.
   Переконаємося в тому, що проект був створений правильно. Введіть команду runserver:
   `python manage.py runserver`
   Django запускає сервер, щоб ви могли переглянути проект в своїй системі і перевірити, як він працює. Коли ви запитуєте сторінку, вводячи URL в браузері, сервер Django відповідає на запит; для цього він будує відповідну сторінку і відправляє сторінку браузеру.
   Тепер відкрийте браузер і введіть URL
   `http://localhost:8000/`
   - або http://127.0.0.1:8000/, якщо перша адреса не працює. Ви побачите щось схоже на Рис.81 - сторінку, яку створює Django, щоб повідомити вам, що все поки працює правильно.

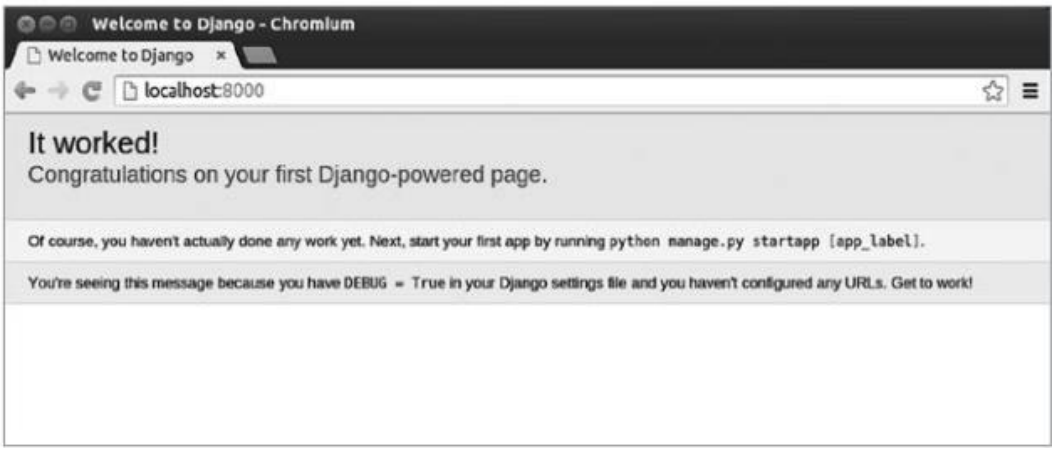

Рис.81. Перша сторінка Django

Ще не завершуйте роботу сервера (але, коли ви захочете перервати його, це можна зробити натисканням клавіш Ctrl + C).

8) Початок роботи над додатком blog. Створення моделей.

Проект Django є групою окремих додатків, спільна робота яких забезпечує роботу проекту в цілому. Поки ми створимо один додаток blog, який буде виконувати більшу частину роботи в нашому проекті. До цього моменту команда runserver повинна продовжувати роботу в термінальному вікні, яке ви відкрили раніше. Відкрийте нове термінальне вікно (або вкладку) і перейдіть в каталог, що містить manage.py. Активізуйте віртуальне середовище і виконайте команду startapp:

python manage.py startapp blog

Команда startapp ім'я_додатку наказує Django створити інфраструктуру, необхідну для побудови додатку. Заглянувши зараз в каталог проекту, ви знайдете в ньому новий підкаталог з ім'ям blog. Відкрийте цей каталог, щоб побачити, які файли були створені Django. Найважливіші файли в цьому каталозі - **models.py**, **admin.py** і **views.py**.

Ми скористаємося файлом models.py для визначення даних, якими потрібно управляти в нашому додатку.

admin.py - Тут ви реєструєте моделі, щоб включити їх в сайт адміністрування Django.

views.py - логіка вашого додатку. Кожне представлення отримує http-запит, обробляє його та повертає відповідь.

Зараз ми створимо схему даних для нашого блогу. Відкриємо каталог проекту в редакторі Sublime Text і відкриємо файл models.py. Модель - це клас Python (підклас django.db.models.Models), в якому кожен атрибут представляє поле бази даних. Django створить таблицю для кожної моделі, яка визначена у файлі models.py. Коли ви створюєте модель, Django пропонує вам практичний API для легкого здійснення запитів до вашої бази даних.

Запишіть у файл blog/**models.py** наступний код:

```
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User

# Our Post Model

class Post(models.Model):
    STATUS_CHOICES = (
```

```
        ('draft', 'Draft'),
        ('published', 'Published'),
    )
    title = models.CharField(max_length=250)
    slug = models.SlugField(max_length=250,
unique_for_date='publish')
    author = models.ForeignKey(User, related_name='blog_posts')
    body = models.TextField()
    publish = models.DateTimeField(default=timezone.now)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    status = models.CharField(max_length=10,
choices=STATUS_CHOICES, default='draft')

    class Meta:
        ordering = ('-publish',)

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

Це наша основна модель для нашого блогу, давайте розглянемо поля, які ми щойно визначили для цієї моделі:

title - це поле для заголовка повідомлення. Це поле CharField, яке перетворюється на varchar у базі даних SQL.

slug - це поле, призначене для використання в URL-адресах. Slug має коротке позначення, містять лише літери, цифри, підкреслення чи дефіс. Ми будемо використовувати це поле slug для створення красивих, дружніх до SEO сторінок для наших публікацій в блозі. Ми додали параметри unique_for_date для цього поля, щоб ми могли створювати URL-адреси для публікації, використовуючи дату та slug публікації. Django дозволить запобігти декілька постів, що мають однакові slug та дату.

author - автор, це поле є зовнішнім ключем. Це поле визначає відносини багато до одного, ми повідомляємо Django, що кожен пост написаний користувачем, і користувач може написати кілька повідомлень. У цьому випадку ми покладаємось на модель користувача системи автентифікації Django.

body – це тіло повідомлення.

publish – це поле date time покаже, коли повідомлення було опубліковане. Ми використовуємо метод Django timezone.now як значення за замовчуванням.

created - це поле date time вказує, коли було створено повідомлення. Оскільки ми використовуємо auto_now_add, дата буде автоматично збережена при створенні об'єкта.

updated - цей поле date time показує, коли останній раз був оновлений певний пост. Оскільки ми тут використовуємо auto_now, дата буде автоматично оновлена при збереженні об'єкта.

status - це поле, яке показує статус публікації. Ми використовуємо параметри вибору (STATUS_CHOICES), тому значення цього поля можна встановити лише на один із зазначених варіантів.

Як ви бачите, Django поставляється з різними типами полів, які можуть бути використані для визначення ваших моделей. Ви можете знайти все про різні поля Django в https://docs.djangoproject.com/en/1.11/ref/models/fields/ .

Клас Meta містить метадані. Ми доручаємо Django сортувати результати за полями публікації в порядку зменшення за замовчуванням, коли ми запитуємо дані. Ми вказуємо спадний порядок, використовуючи знак мінус ('-publish',).

Django викликає метод __str __ () для виведення простого представлення моделі. Ми написали реалізацію __str __ (), яка повертає рядок, що зберігається в атрибуті title.

9) Оскільки ми маємо справу з датою, нам потрібно встановити модуль pytz. Цей модуль забезпечить визначення часового поясу для Python, і він потрібен, щоб SQLite працювала з датами. Отже заходимо в термінал і подаємо команду

```
pip install pytz
```

10) Активізація моделей. Щоб використовувати моделі, необхідно наказати Django включити додаток blog в спільний проект. Відкриємо файл settings.py, знайдемо секцію INSTALLED_APPS = () і додамо додаток blog. Ця секція буде виглядати так:

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

Тепер Django знає, що наш додаток blog активний для цього проекту і зможе самостійно переглядати його моделі. Тепер ми будемо створювати і застосовувати міграції. Давайте створимо в базі даних таблицю для нашої моделі Post.

11) Впевнившись, що в консолі активною є коренева директорія нашого проекту (тобто доступний файл manage.py), подаємо команду

python manage.py makemigrations blog

Django створив файл 0001_initial.py всередині каталогу migrations нашого додатку blog. Ви можете відкрити цей файл, щоб побачити, як виглядає міграція. Виконайте наступну команду, щоб перевірити код:

python manage.py sqlmigrate blog 0001

Для синхронізації нашої бази даних з моделлю Post виконайте таку команду:

python manage.py migrate

База даних тепер відображає поточний стан нашої моделі блогу.

12) Адміністративний сайт Django.

Django дозволяє легко працювати з моделями, визначеними для додатка, через адміністративний сайт. Цей сайт використовується адміністраторами сайту, а не рядовими користувачами. У цьому розділі ми створимо адміністративний сайт і використаємо його для додавання деяких тем через модель Post.

Спочатку ми створимо суперкористувача, а потім додамо модель Post на сайт адміністрації, далі ми налаштуємо те, як моделі відображаються. Щоб створити суперкористувача, запустіть таку команду:

python manage.py createsuperuser

Ваш термінал попросить вас ввести ім'я користувача, адресу електронної пошти та пароль. Ви можете ввести: admin, admin@example.com і залишити поля для пароля пустими.

Тепер ми розглянемо сайт адміністрування Django. Запустіть сервер розробника, запустивши таку команду:

python manage.py runserver

У своєму браузері перейдіть за цим посиланням:

http://localhost:8000/admin/

Ви повинні побачити екран входу адміністратора, як показано нижче (Рис.82):

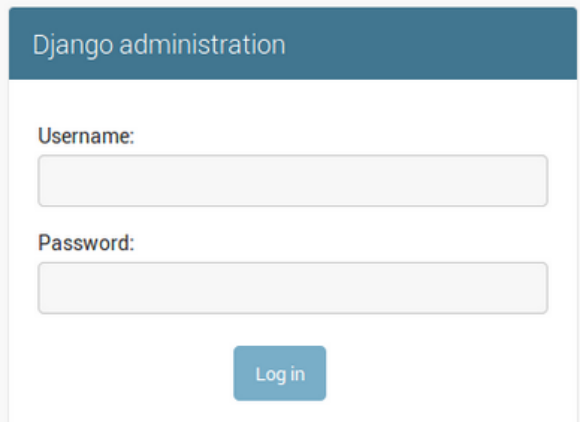

Рис.82. Форма входу адміністратора

Після введення даних ви побачите сторінку адміністратора, як показано тут:

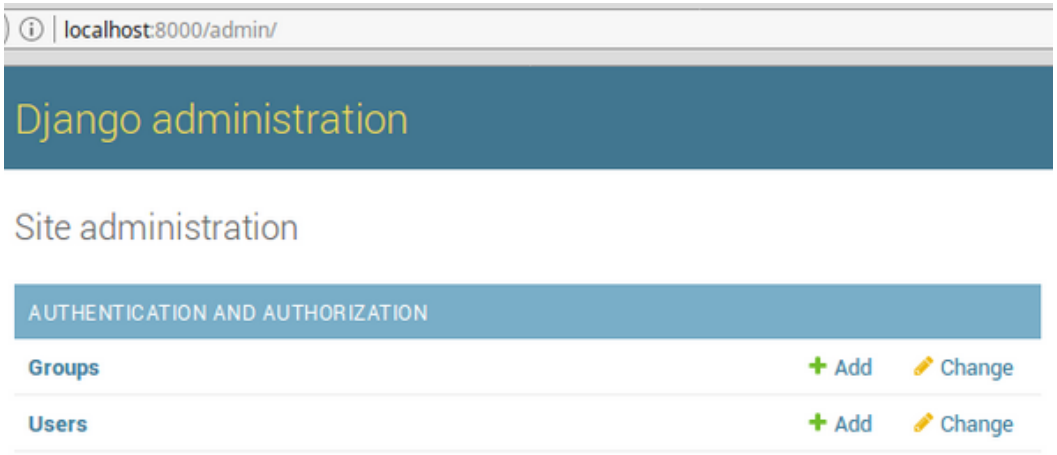

Рис. 83. Сторінка адміністратора

13) Реєстрація моделі на адміністративному сайті.

Django додає деякі моделі (наприклад, User і Group) на адміністративний сайт автоматично, але моделі, які ми створили, доведеться реєструвати вручну.

Щоб додати нашу модель Post на сайт адміністратора, давайте відкриємо файл admin.py і напишемо наступний код:

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

admin.site.register(Post)
```

Тепер перезавантажте адміністративний сайт у своєму браузері, і ви побачите такий екранний знімок:

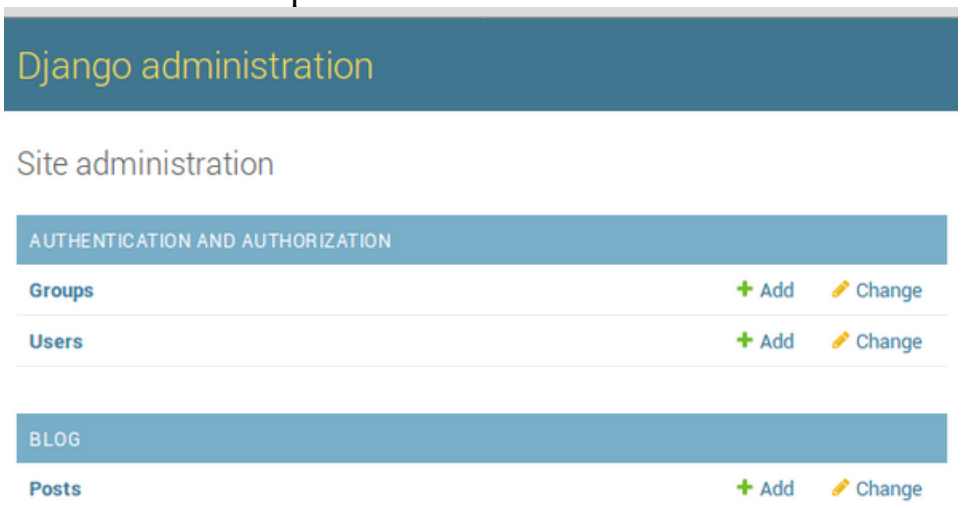

Рис.84. Сторінка адміністратора з моделлю Post

Тепер модель блогу доступна на нашому адміністративному сайті. Це лише невелика частка того, що може зробити адміністратор сайту Django.

Коли ви реєструєте свою модель на сайті адміністратора Django, ви отримуєте зручний інтерфейс користувача для ваших моделей, що дозволяє вам просто переглядати, редагувати, створювати та видаляти об'єкти.

Натисніть посилання Add праворуч від Posts, щоб додати нову публікацію, ви побачите форму створення, яку Django згенерував динамічно на основі моделі Post. Ось екранний знімок:

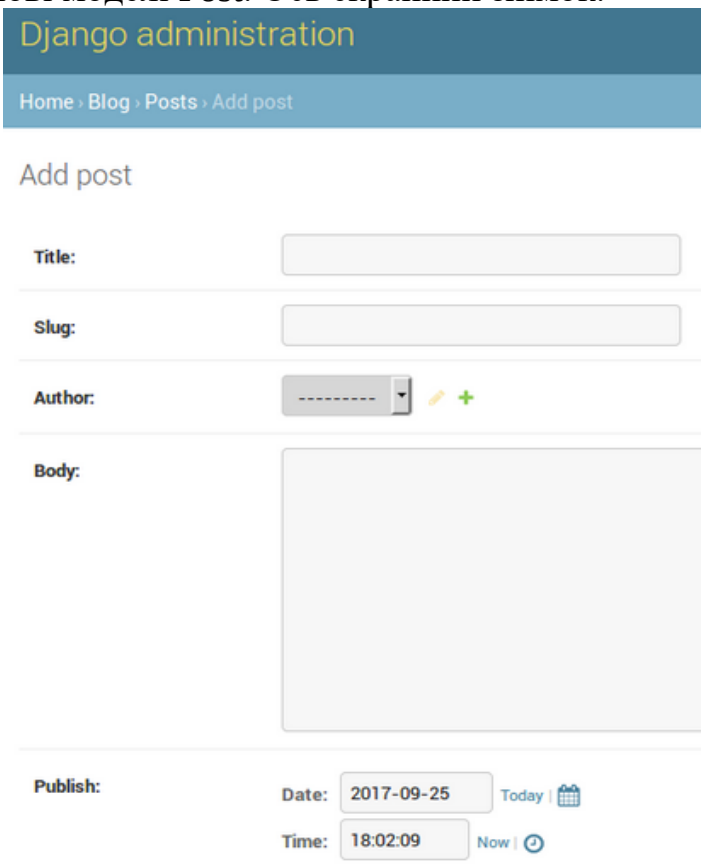

Рис.85. Форма введення данних, згенерована Django

Заповніть форму та натисніть кнопку збереження. Ви будете перенаправлені на сторінку Post з успішним повідомленням про створений вами пост. Django використовує різні формати віджетів для кожного типу полів, навіть складні поля, такі як поля часу дати, які відображаються за допомогою простого інтерфейсу, наприклад, вибору дати JavaScript.

Давайте налаштуємо адміністратора сайту для цього. Додайте наступний код у файл admin.py програми вашого блогу.

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

    class PostAdmin(admin.ModelAdmin):
        list_display                                    =
('title','slug','author','status','created')
        list_filter = ('status','created','publish','author')
```

```
    search_fields = ('title','author')
    prepopulated_fields = {'slug': ('title',)}
    raw_id_fields = ('author',)
    date_hierarchy = 'publish'
    ordering = ['status', 'publish']

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

У клас PostAdmin ми можемо включити інформацію про те, як відображати моделі та взаємодіяти з ними на сайті адміністратора. Атрибут list_display дозволяє вам встановити поля вашої моделі, які ви хочете відобразити на сторінці веб-сайту адміністратора. Щоб дізнатись більше про те, як налаштувати свій сайт адміністратора, відвідайте https://docs.djangoproject.com/en/1.11/ref/contrib/admin/ .

Тепер поверніться до свого браузера та перезавантажте сторінку списку публікацій. Тепер він буде виглядати як екранний знімок нижче:

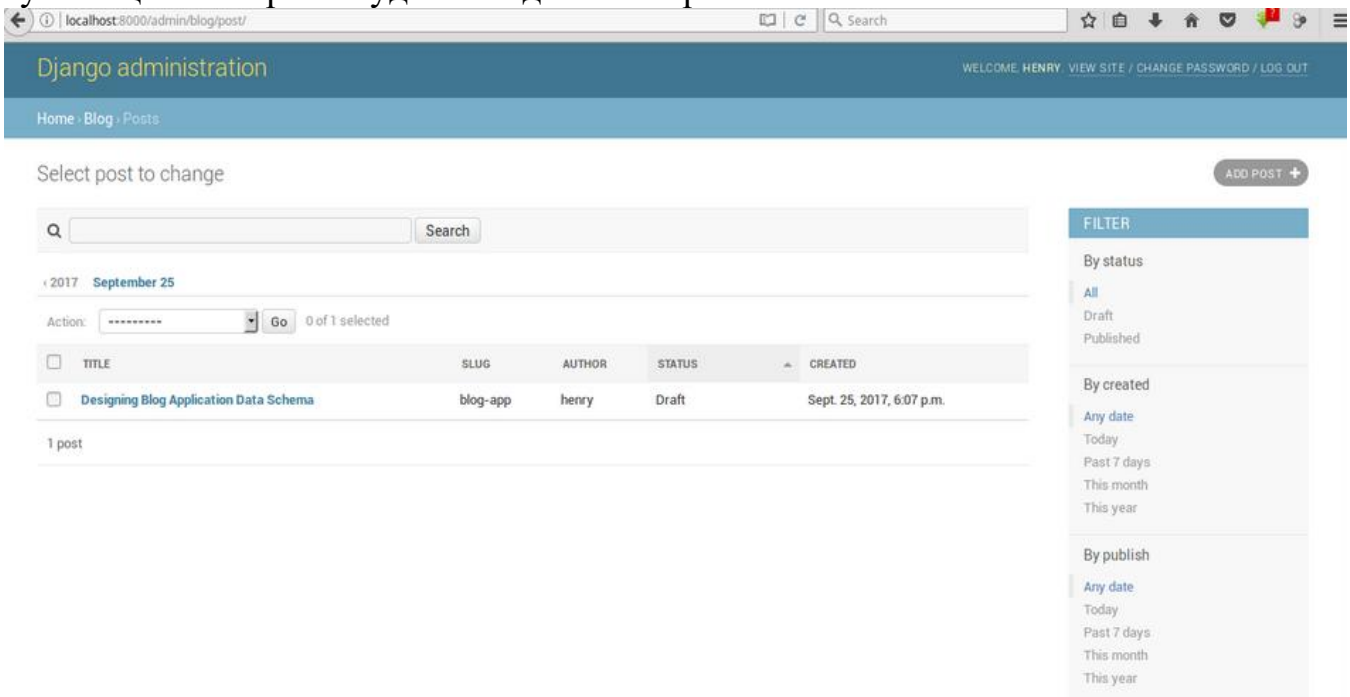

Рис.86. Сторінка списку повідомлень

Сторінка списку тепер включає праву бічну панель, яка дозволяє фільтрувати публікації на основі полів, включених до атрибута list_filter у рядку 6. Панель пошуку з'явилася на сторінці через атрибут search_fields на рядку 7. Трохи нижче поля пошуку присутня панель для навігації через ієрархію дат, це було визначено атрибутом date_hierarchy в рядку 10.

За допомогою декількох рядків коду ми налаштували спосіб відображення моделі нашого повідомлення на сайті адміністратора. Є багато способів налаштування та розширення сайту адміністрування Django. Красиво, ми зробили чудову роботу.

14) Робота з QuerySet & Managers для нашого блогу. Інтерактивна оболонка Django.

Введені дані можна проаналізувати на програмному рівні в інтерактивному термінальному сеансі. Це інтерактивне середовище, зване оболонкою (shell) Django, прекрасно підходить для тестування і діагностики проекту.

У цьому розділі ми спочатку вивчимо, як створювати, оновлювати, завантажувати та видаляти об'єкти в базі даних. Потім ми побачимо, як ми можемо створити модельних менеджерів. Тепер, коли у вас є повністю функціональний сайт адміністрування для керування вмістом вашого блогу, пора навчитися отримувати інформацію з бази даних та взаємодіяти з нею.

Django поставляється з потужною API абстракції баз даних, які дозволяють легко створювати, отримувати, оновлювати та видаляти об'єкти. Mapper Relationship Object Django (ORM) сумісний з такими базами даних, як MySql, SQLite, PostgreSQL та Oracle. Пам'ятайте, що ви можете визначити базу даних свого проекту, редагуючи налаштування бази даних у файлі settings.py вашого проекту. Django може працювати з декількома базами одночасно, і навіть ви можете запрограмувати маршрутизатори баз даних, які обробляють дані будь-яким способом. Створивши свої моделі даних, Django надає вам безкоштовну API для взаємодії з ними. Тепер давайте створювати об'єкти.

Щоб створити свій перший об'єкт, активуйте віртуальне середовище та перейдіть до базової директорії проекту, де знаходиться файл manage.py на вашому терміналі. Запустіть таку команду, щоб відкрити оболонку python:

```
python manage.py shell
```

Три знаки greater than (‘>>>’) показують, що наша інтерактивна консоль python готова до введення. По-перше, ми повинні імпортувати бібліотеку, введіть таку команду:

```
from blog.models import Post
```

і натисніть Enter.

Тепер щоб отримати користувача з бази даних за допомогою інтерактивної консолі python, введіть таку команду:

```
user = User.objects.get(username='admin')
```

і натисніть Enter.

Щоб створити публікацію, введіть таку команду:

```
post = Post.objects.create(title='Django Blog
App',slug='django-blog-app', body='learning how to create
blog',author=user)
```

Створений нами пост об'єкт знаходиться в пам'яті комп'ютера і не зберігається в базі даних. Щоб зберегти об'єкт повідомлення, нам потрібно скористатись методом save, отже запустіть таку команду:

```
post.save()
```

Отримання об'єкта на нашій інтерактивній консолі.

Mapper для зв'язку об'єктів (ORM) Django заснований на QuerySet. QuerySet - це сукупність об'єктів у вашій базі даних, які можуть мати кілька фільтрів для обмеження результатів. Ви вже знаєте, як отримати один об'єкт з бази даних за допомогою методу get. Кожна модель Django має щонайменше одного менеджера, а менеджер за замовчуванням називається **objects**.

Щоб отримати весь наш блог з бази даних, ми можемо використовувати all() метод, отже запустіть таку команду:

```
post_query = Post.objects.all()
```

Зауважте, що post_query = Post.objects.all() ще не виконано, Django QuerySet «ліниві», і вони оцінюються тільки тоді, коли вони змушені це зробити. Ця поведінка робить Django QuerySet дуже ефективним, отже щоб виконати наш запит запустіть таку команду:

```
Post.objects.all()
```

Тепер ми побачимо список всіх наших постів.

Тепер давайте дізнаємося, як ми можемо використовувати метод фільтрації. Розглянемо приклад, де ми фільтруємо публікації, створені автором admin. Щоб виконати це, треба запускати таку команду:

```
Post.objects.filter (author__username = 'admin')
```

І, нарешті, розглянемо приклад, як фільтрувати за допомогою декількох полів. Наприклад, ми хочемо отримати всі пости, написаі admin в 2017 році. Це команда, яку ми буемо використовувати:

```
Post.objects.filter (publish__year = 2017, author__username
= 'admin')
```

Ми створювали запити за допомогою методу пошуку полів із використанням двох підкреслень, наприклад author__username. Наступне, що нам потрібно навчитися, - метод виключення, ви можете виключити певний результат із вашого запиту, використовуючи метод exclude менеджера. Наприклад, ми можемо отримати весь пост, опублікований в 2017 році, і титул якого не починається словом Django. Ми можемо використовувати цю команду:

```
Post.objects.filter(publish__year=2017)\
.exclude(title__startswith='Django')
```

Тепер розглянемо метод order_by. Наприклад, ви можете отримати всі об'єкти, упорядковані за їхнім заголовком. Для цього виконайте таку команду:

```
Post.objects.order_by('title')
```

Видалення об'єктів

Якщо ви хочете видалити об'єкт, ви можете зробити це з екземпляра об'єкта, передаючи ідентифікатор, використовуючи цю команду:

```
post_del = Post.objects.get(id=1)
```

а потім викликати метод видалення за допомогою такої команди:

```
post_del.delete()
```

Менеджери моделей

Як ми вже згадували раніше, objects є менеджером за умовчанням кожної моделі, який використовується для доступу до всіх об'єктів у базі даних, але ми також можемо визначити спеціальні менеджери для наших моделей. Ми збираємося створити спеціальний менеджер, щоб отримати всі публікації з опублікованим статусом. Існує два способи додавання менеджерів до ваших моделей, додавання додаткових менеджерських методів або зміна початкового менеджера QuerySet.

Відредагуйте файл models.py нашої програми для блогу та переконайтеся, що він містить такий код.

```
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User

# Custom Manager
class PublishedManager(models.Manager):
    def get_queryset(self):
        return
super(PublishedManager,self).get_queryset().filter(status='published')

# Our Post Model
class Post(models.Model):
    STATUS_CHOICES = (
        ('draft', 'Draft'),
        ('published', 'Published'),
    )
    title = models.CharField(max_length=250)
    slug = models.SlugField(max_length=250, unique_for_date='publish')
    author = models.ForeignKey(User, related_name='blog_posts')
    body = models.TextField()
    publish = models.DateTimeField(default=timezone.now)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    status = models.CharField(max_length=10, choices=STATUS_CHOICES,
default='draft')

    # The default manager
    objects = models.Manager()

    # Custom made manager
    published = PublishedManager()

    class Meta:
        ordering = ('-publish',)

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

В рядках 6 - 8 ми визначили нашого індивідуального менеджера моделі нашої публікації. На рядку 7 ми визначили метод get_queryset, який повертає queryset для виконання. Після визначення нашого індивідуального менеджера ми повинні використовувати його в нашій моделі Post для виконання запитів, для цього ми називаємо цього менеджера в рядку 29. Щоб дізнатися більше про менеджерів відвідайте https://docs.djangoproject.com/en/1.11/topics/db/managers/ .

15) Створення списку публікацій та детальних переглядів для нашого блогу.

Перегляд (view) Django - це лише функція Python, яка отримує веб-запит і повертає веб-відповідь. Всередині перегляду йде вся логіка, необхідна для повернення бажаної відповіді. Спочатку ми створимо представлення для нашого блогу, потім ми визначимо шаблон URL для кожного представлення даних і, нарешті, ми створимо шаблон HTML, щоб відтворити данні, згенеровані views.

Кожне представлення (view) рендерить шаблон, передавши в нього змінну, і повертає HTTP-відповідь з рендерованим виходом. Почнемо зі створення функції post_list_view, яка буде містити список публікацій, та функції post_detail_view для детального представлення. Відредагуйте файл views.py програми блогу та переконайтеся, що у вас є такий код:

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

def post_list_view(request):
    posts = Post.published.all()
    return render(request, 'blog/post/list.html', {'posts':
posts})

def post_detail_view(request, year, month, day, post):
    post = get_object_or_404(Post, slug=post,
status='published', publish__year=year, publish__month=month,
publish__day=day)
    return render(request, 'blog/post/detail.html', {'post':
post})
```

Ви тільки що створили свої перші Django views. На рядку 4, post_list_view - цей вигляд приймає об'єкт запиту як єдиний параметр, завжди пам'ятайте, що цей параметр потрібен усім переглядам. На рядку 5, posts = Post.published.all () - ми отримуємо всі публікації, використовуючи менеджер публікацій, який ми створили раніше. У рядку 6 ми використовуємо функцію render, надану Django, щоб відобразити список публікацій у даний шаблон Ця функція приймає об'єкт запиту, шлях шаблону та змінні для рендеринга.

На рядку 8, post_detail_view - це наша друга функція, це представлення, яке покаже окреме повідомлення. Цей перегляд приймає запит, рік, місяць, день та post параметри, щоб отримати опублікований допис з даним значенням slug та датою. Пам'ятайте, коли ми створили модель Post, ми додали унікальний параметр дати в поле slug, таким чином, ми гарантуємо, що буде лише одна публікація з унікальним slug за певну дату.

Для кожного перегляду в Django потрібен шаблон url для його відображення в браузері. Шаблон url у Django складається з регулярних виразів Python, представлення та назви, які дозволяють назвати його будь-де у вашому проекті. Django проходить через кожний шаблон URL-адреси і зупиняється на першому, який відповідає запитуваній URL-адресі. Тоді Django імпортує вигляд, що відповідає шаблону url, і виконує його, передаючи екземпляр класу HTTP-запита і аргументів.

У каталозі додатка blog створіть файл **urls.py** та переконайтеся, що він містить такі рядки коду:

```
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^$', views.post_list_view, name='post_list_view'),

url(r'^(?P<year>\d{4})/(?P<month>\d{2})/(?P<day>\d{2})/(?P<post>
[-\w]+)/$', views.post_detail_view, name='post_detail_view'),
]
```

На рядку 5 - це URL-адреса, яка не приймає жодних аргументів і відображається до post_list_view. У рядку 6 - це шаблон url, який приймає чотири аргументи, а саме рік - вимагає чотирьох цифр, місяць - вимагає двох цифр, день - вимагає двох цифр і post - який може складатися з слів і дефісів. Цей URL посилається на post_detail_view.

Створення файлу urls.py для кожного додатку в Django - це найкращий спосіб повторно використовувати ваш додаток іншими проектами Django. Тепер ми маємо включити шаблони url нашого блогу в основні шаблони URL-адрес проектів. Відредагуйте файл urls.py, який знаходиться в директорії mysite, і переконайтеся, що він містить такий код:

```
"""mysite URL Configuration

The `urlpatterns` list routes URLs to views. For more information
please see:
    https://docs.djangoproject.com/en/1.11/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
```

```
    2. Add a URL to urlpatterns:  url(r'^$', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  url(r'^$', Home.as_view(),
name='home')
Including another URLconf
    1. Import the include() function: from django.conf.urls import
url, include
    2. Add a URL to urlpatterns:  url(r'^blog/', include('blog.urls'))
"""
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'blog/', include('blog.urls', namespace='blog',
app_name='blog')),
]
```

У рядку 21 ми повідомляємо Django, що він включає в себе шаблони URL-адрес, визначені у файлі urls.py блогу, ми даємо цим URL-адресам простір імен blog, що полегшує посилання на цю групу URL-адрес.

Канонічні URL-адреси для моделей

Ми можемо використовувати post_detail_view, який ми визначили раніше, для створення канонічної URL-адреси для об'єкта post. Згідно конвенції Django слід додати до моделі метод get_absolute_url, який повертає канонічну URL-адресу об'єктів. Відкрийте файл models.py нашого блогу та переконайтеся, що він містить такий код:

```
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User
from django.core.urlresolvers import reverse

# Custom Manager

class PublishedManager(models.Manager):
    def get_queryset(self):
        return
super(PublishedManager,self).get_queryset().filter(status='published')


# Our Post Model

class Post(models.Model):
    STATUS_CHOICES = (
        ('draft', 'Draft'),
        ('published', 'Published'),
    )
```

```
    title = models.CharField(max_length=250)
    slug = models.SlugField(max_length=250, unique_for_date='publish')
    author = models.ForeignKey(User, related_name='blog_posts')
    body = models.TextField()
    publish = models.DateTimeField(default=timezone.now)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)
    status = models.CharField(max_length=10, choices=STATUS_CHOICES,
default='draft')

    # The default manager
    objects = models.Manager()

    # Custom made manager
    published = PublishedManager()

    class Meta:
        ordering = ('-publish',)

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return reverse('blog:post_detail_view',
args=[self.publish.year,
self.publish.strftime('%m'),self.publish.strftime('%d'),self.slug])
```

У рядку 4 ми імпортуємо бібліотеку для нашого канонічного методу reverse url. На лініях 41 та 42 ми використовуємо метод reverse, який дозволяє створювати URL-адреси за їхніми іменами та опційними параметрами. Зауважте, що ми використовуємо функцію strftime у self.publish.strftime ('% m'), self.publish.strftime ('% d'), щоб побудувати URL-адресу, використовуючи місяць та день з передуючими нулями. Ми будемо використовувати метод get_absolute_url у наших шаблонах.

16) Створення шаблонів (templates) для нашого блогу.

У цьому розділі ми будемо додавати шаблони, щоб відображати наші публікації блогу зручним способом. Ми вже створили перегляди (views) та шаблони URL-адрес (url patterns) для нашого додатку blog, тепер пора додавати шаблони (templates). У нашому каталозі blog ми збираємося створити дві папки, названі **templates** та **static**. Всередині каталогу templates створіть іншу папку та назвіть її **blog**. Всередині каталогу blog створіть інший каталог і назвіть його **post**. Ще в каталозі blog створіть файл HTML і назовіть його **master.html**. Всередині папки post створіть два HTML-файли, **detail.html** та **list.html** Всередині каталогу static створіть іншу папку з назвою **css**. Завантажте bootstrap і розмістіть файл **bootstrap.min.css** в папці css, яку ви щойно створили.

Тепер ваш каталог має виглядати так:

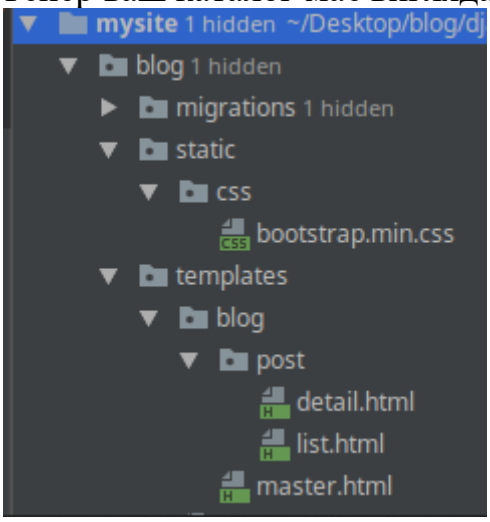

Рис.87. Каталог проекту

Файл master.html включатиме основну HTML-структуру веб-сайту, ми розділимо файл master.html на дві основні області, основну область вмісту та бічну панель. Файли list.html і detail.html успадкують від master.html, щоб відобразити список публікацій блогу та окреме повідомлення відповідно. Django має потужну мову шаблонів, яка дозволяє вказати, яким чином відображаються дані, вона базується на тегах шаблонів (template tags), які виглядають як змінні шаблону тегів (tag template variable) та фільтри (filters). Щоб дізнатись більше про Django template language відвідайте https://docs.djangoproject.com/en/1.11/topics/templates/ .

Відредагуйте файл **master.html** та переконатися, що він має такий код:

```
{% load staticfiles %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock %}</title>
    <link rel="stylesheet" href="{% static "css/bootstrap.min.css"
%}">
</head>
<body>
  <div class="container">
      <div class="row">
          <div class="col-md-8">
              {% block content %}

              {% endblock %}
          </div>
          <div class="col-md-4">
              <h3>Muvalab Blog</h3>
              <p>This is our sidebar</p>
```

```
        </div>
      </div>
  </div>
</body>
</html>
```

Тепер давайте поглянемо на код. На рядку 1, де ми маємо {% load staticfiles%}, ми повідомляємо Django про завантаження статичних файлів тегів шаблонів файлів, які надаються програмою django.contrib.staticfiles. За допомогою цього тегу шаблонів ви можете включити статичні файли. В рядку 7 ми завантажуємо bootstrap.min.css, використовуючи тег статичного фільтра. У рядку 5 ми маємо теги блоків {% block title%} (% endblock%}, блочні теги повідомляють Django, що ми хочемо визначити блок у цій області. Шаблони, які успадковуються з шаблону master.html, можуть заповнювати цю область блоку вмістом. У цьому шаблоні ми визначили два блоки під назвою title та content.

Відредагуйте файл **list.html** та переконатися, що він має такий код:

```
{% extends "blog/master.html" %}

{% block title %}Our Blog{% endblock %}

{% block content %}
   <h2 class="page-header">Recents Posts</h2>
    {% for post in posts %}
       <h3><a href="{{ post.get_absolute_url }}">{{ post.title
}}</a></h3>
        <p class="small">Published {{ post.publish }} by {{
post.author }}</p>
        <p>{{ post.body|truncatewords:25|linebreaks }}</p>
    {% endfor %}

{% endblock %}
```

У рядку 1 ми використовуємо тег extends, щоб успадкувати master.html. У рядку 3 ми заповнюємо блок title нашим заголовком. З рядків від 5 до 13 ми визначили нашу область вмісту (block content), в рядку від 7 до 11 ми проводимо ітерацію по всім повідомленням, і ми показуємо назву повідомлення, дату, тіло повідомлення та автора. У рядку 8 ми включаємо посилання на публікацію, використовуючи get_absolute_url (канонічний URL).

Давайте подивимось, що ми маємо на цей момент. Запустіть свій сервер розробки і завжди пам'ятайте, що треба активувати віртуальне середовище. Щоб запустити сервер, виконайте таку команду:

```
python manage.py runserver
```

Відкрийте наступне посилання у своєму веб-переглядачі:

```
http://localhost:8000/blog/
```

і ви повинні побачити такий екран:

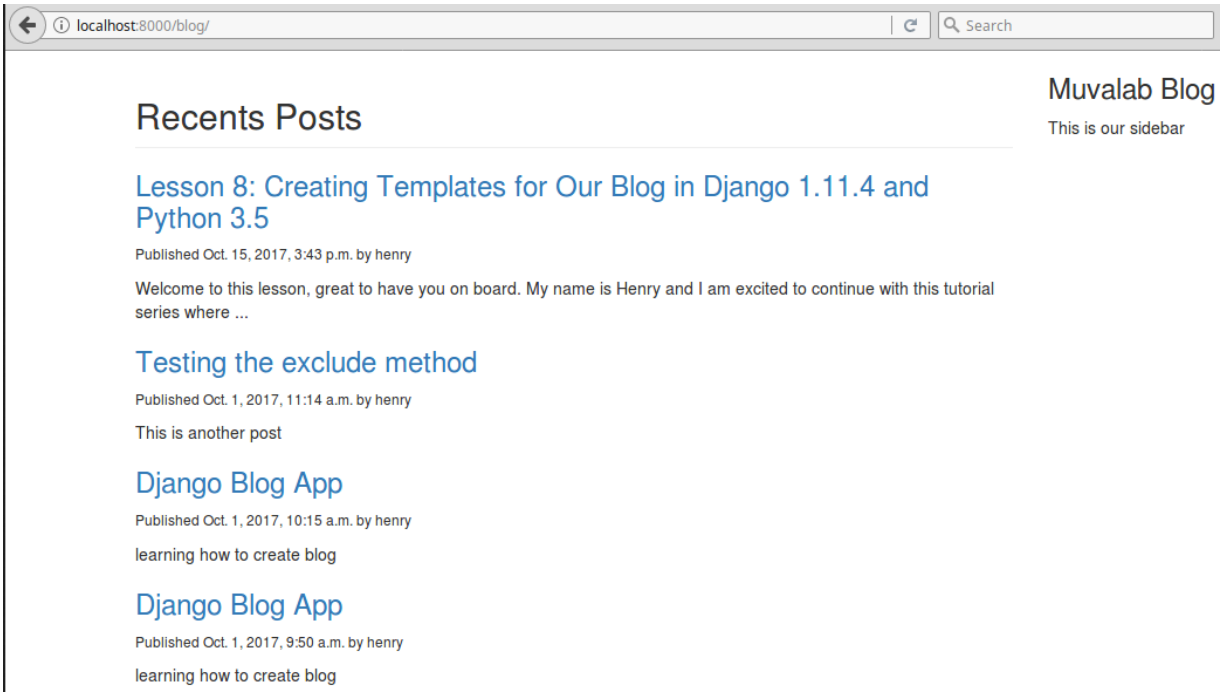

Рис.88. Список повідомлень блогу

Примітка. Вам потрібно мати публікацію, статус якої є published, щоб ви могли переглянути список публікацій.

Тепер відредагуйте файл **detail.html** та переконайтеся, що він має такий код:

```
{% extends "blog/master.html" %}

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

{% block content %}
   <h3>{{ post.title }}</h3>
    <p class="small">Published {{ post.publish }} by {{ post.author
}}</p>
    <p>{{ post.body|linebreaks }}</p>
{% endblock %}
```

Тепер поверніться до свого веб-переглядача та натисніть на одне з заголовків повідомлень, щоб побачити повний допис. Ви повинні побачити щось на зразок цього:

Рис.89. Сторінка окремого повідомлення

Тепер подивіться на URL-адресу свого браузера, ми створили дружню URL-адресу для нашого блогу.

17) Створення розбивки на сторінки (Pagination) для нашого блогу.

Коли кількість публікацій у блозі буде зростати, ви скоро зрозумієте, що вам потрібно розділити список публікацій на кілька сторінок. Django має вбудований клас pagination, який дозволяє розробникам легко керувати розбивкою на сторінки. Тепер відредагуйте файл views.py і переконайтеся, що він має такий код:

```
from django.shortcuts import render, get_object_or_404
from django.core.paginator import Paginator, EmptyPage,
PageNotAnInteger
from .models import Post

def post_list_view(request):
    list_objects = Post.published.all()
    paginator = Paginator(list_objects, 3)
    page = request.GET.get('page')
    try:
        posts = paginator.page(page)
    except PageNotAnInteger:
        posts = paginator.page(1)
    except EmptyPage:
        posts = paginator.page(paginator.num_pages)
    return render(request, 'blog/post/list.html', {'posts':
posts})

def post_detail_view(request, year, month, day, post):
    post = get_object_or_404(Post, slug=post,
status='published', publish__year=year, publish__month=month,
publish__day=day)
```

```
    return render(request, 'blog/post/detail.html', {'post':
post})
```

На рядку 2 ми імпортуємо Django paginated classes. Ми змінили post_list_view, розглянемо як працює pagination у Django:

Рядок 7 – створюється об’єкт класу Paginator з заданням кількості об'єктів, які ви хочете відобразити на кожній сторінці.

Рядок 8 - Ми отримуємо параметр GET сторінки, який вказує поточний номер сторінки.

Рядок 10 - Ми отримуємо об'єкт для потрібної сторінки page, викликаючи метод page об'єкта paginator.

Рядок 12 - Якщо параметр сторінки не є цілим числом, ми отримуємо першу сторінку результатів, якщо цей параметр є більшим, ніж остання сторінка результатів, ми отримуємо останню сторінку.

Рядок 14 - ми передаємо в paginator кількість сторінок та отримуємо об'єкти до шаблону.

Тепер нам потрібно створити шаблон, щоб відобразити paginator таким чином, щоб він міг бути включений у будь-який шаблон, який використовує pagination. У папці **template** додатку blog створіть новий файл та назвіть його **pagination.html**. Переконайтеся, що pagination.html має такий код:

```
<div class="container">
    <div class="row">
    <span class="pagination">
        {% if page.has_previous %}
            <a href="?page={{ page.previous_page_number
}}">previous</a>
        {% endif %}

      <span class="active">
          page {{ page.number }} of {{ page.paginator.num_pages
}}
      </span>
    {% if page.has_next %}
        <a href="?page={{ page.next_page_number }}">next</a>
    {% endif %}
    </span>
</div>
</div>
```

Шаблон сторінок (pagination template) очікує об'єкта page, щоб відобразити попереднє посилання, наступне посилання, поточну сторінку та загальну сторінку. Тепер давайте відкриємо наш файл шаблону **list.html**, розташованого в blog/post/ директорії, і переконайтеся, що він має такий код:

```
{% extends "blog/master.html" %}
```

```
{% block title %}Our Blog{% endblock %}

{% block content %}
   <h2 class="page-header">Recents Posts</h2>
    {% for post in posts %}
       <h3><a href="{{ post.get_absolute_url }}">{{ post.title
}}</a></h3>
        <p class="small" style="color: #777777">Published {{
post.publish }} by {{ post.author }}</p>
        <p>{{ post.body|truncatewords:25|linebreaks }}</p>
    {% endfor %}

    {% include "pagination.html" with page=posts %}
{% endblock %}
```

У рядку 13 ми просто включили нашу сторінку pagination.html у нижню частину блоку вмісту. Оскільки об'єкт page ми передаємо до шаблону list.html, ми у рядку 13 задаємо page = posts. Подібний підхід ви можете використовувати для повторного використання шаблону сторінок у ваших проектах Django, а також для розбивки на сторінки різних моделей.

Переконайтеся, що ви додали більше 5 повідомлень для тестування своїх сторінок. Після завершення додавання повідомлень розгорніть сервер розробки та відкрийте це посилання у своєму веб-переглядачі:

http://localhost:8000/blog/

Ось можливий екранний знімок браузера:

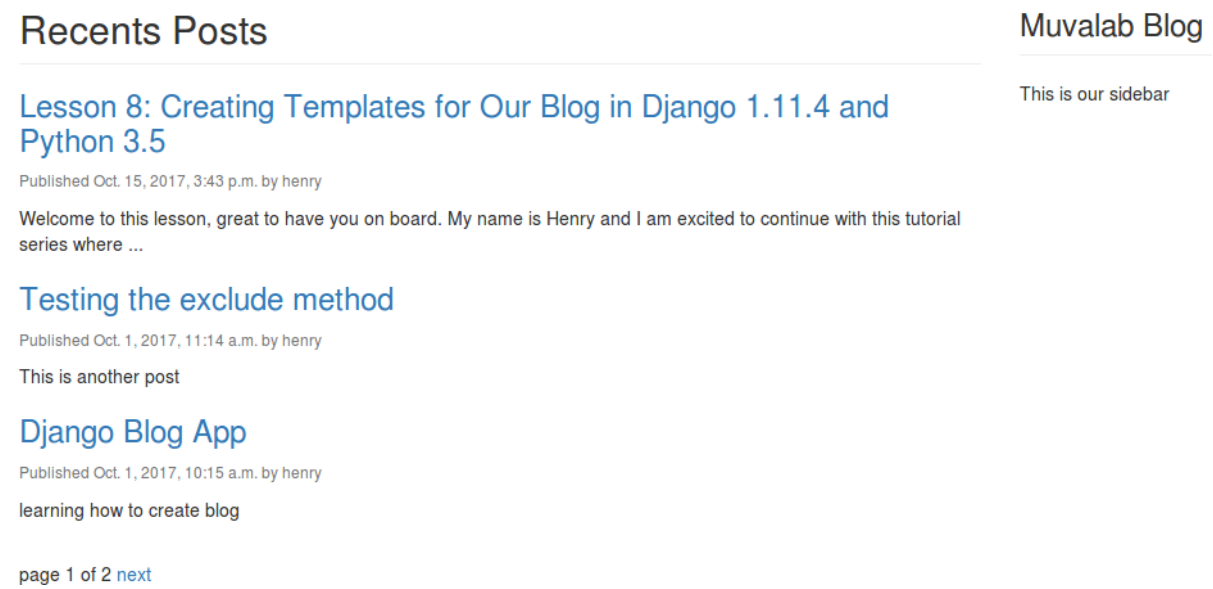

Рис.90. Список повідомлень блогу з розбиттям на сторінки

Ви побачите розбивку на сторінки (pagination) в нижній частині сторінки списку публікацій, і ви матєте змогу переходити по сторінках.

18) Створення RSS-каналів для нашого блогу.

Веб-фреймворк Django має вбудований syndication feed framework, який ви можете використовувати для динамічного створення каналів RSS або atom feeds. У нашому каталозі blog створіть новий файл з назвою **feeds.py** і переконайтеся, що в ньому є такий код:

```
from django.contrib.syndication.views import Feed
from django.template.defaultfilters import truncatewords
from .models import Post

class PostsFeed(Feed):
    title = 'Henry Blog Feeds'
    link = '/blog/'
    description = 'Our latest Posts!'

    def items(self):
        return Post.published.all()[:5]

    def item_title(self, item):
        return item.title

    def item_description(self, item):
        return truncatewords(item.body, 20)
```

Давайте зрозуміємо вищезазначений код. У рядку 1 ми імпортуємо бібліотеку синдикації, яку нам надає Django. У рядку 2 ми імпортуємо фільтр шаблонів, який допоможе нам підсумувати тіло нашого блогу. У рядку 3 ми імпортуємо нашу модель Post з файлу models.py.

У рядках 5 - 8 ми створюємо клас під назвою PostsFeed, і ми передаємо потік синдикації в цей клас, таким чином, у нас є PostFeed (Feed), який стає підкласом класу Feed базової синдикації Django. Атрибути title, link та description відповідають назві, посиланням та опису RSS відповідно.

У рядках 10 - 11 ми створили метод items, який витягує об'єкти з нашої моделі Post, які будуть включені в канал, в цьому випадку ми вказуємо 5 останніх публікацій. У рядках 13 - 14 ми створили метод item_title, який повертає назву отриманого об'єкта. У рядках 16 - 17 ми створили метод item_description, який повертає тіло повідомлення, і ми вирізаємо це тіло до перших 20 слів.

Тепер нам потрібно змінити файл **urls.py** нашої блогової програми, переконайтеся, що файл має такий код:

```
from django.conf.urls import url
from . import views
from .feeds import PostsFeed

urlpatterns = [
```

```
    url(r'^$', views.post_list_view, name='post_list_view'),

url(r'^(?P<year>\d{4})/(?P<month>\d{2})/(?P<day>\d{2})/(?P<post>
[-\w]+)/$', views.post_detail_view, name='post_detail_view'),
    url(r'^feed/$', PostsFeed(), name='post_feed')
]
```

У рядку 3 ми імпортуємо клас PostsFeed, який ми щойно створили в нашому файлі feeds.py. У рядку 8 ми інсталюємо PostsFeed в новому шаблоні URL-адреси. Давайте перевіримо наш код, повертаємо ваш сервер розробки та переходимо до цього URL:

http://localhost:8000/blog/feed/

і ви побачите наступний вивід у своєму веб-переглядачі.

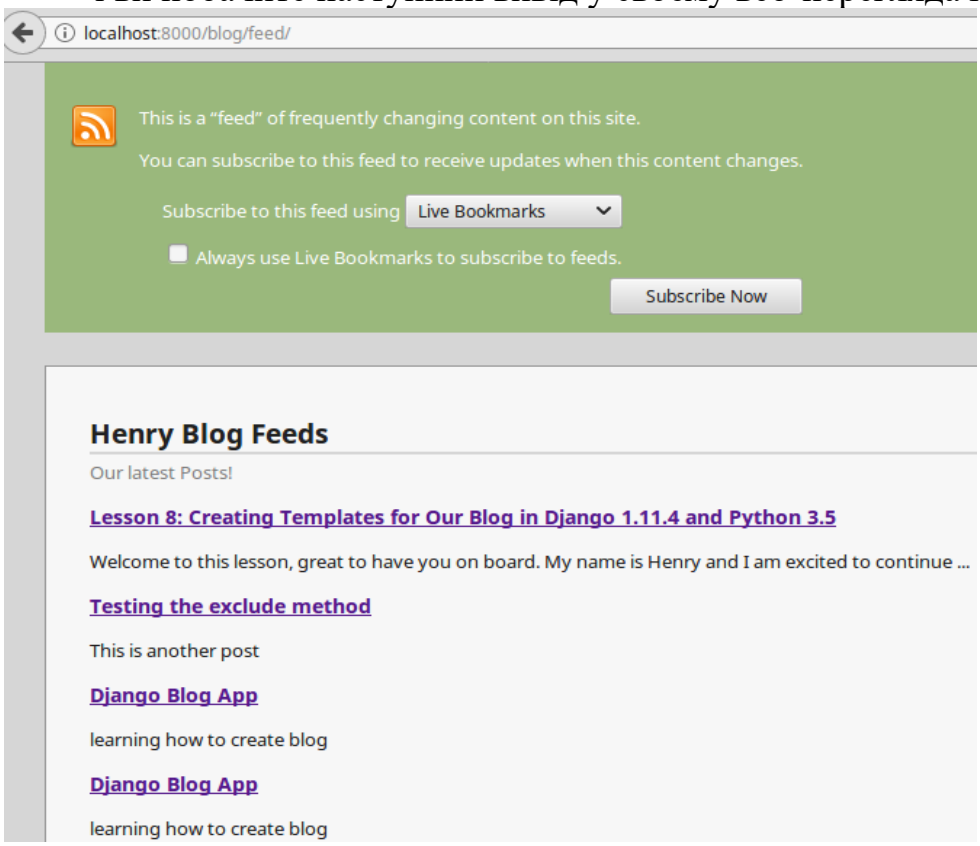

Рис.91. Сторінка RSS-каналів для блогу

Останній крок - створити посилання на передплату RSS на бічній панелі блогу. У нашому шаблоні відкрийте **master.html** і переконайтеся, що він має такий код:

```
{% load staticfiles %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock %}</title>
```

```
    <link rel="stylesheet" href="{% static
"css/bootstrap.min.css" %}">
</head>
<body>
  <div class="container">
      <div class="row">
          <div class="col-md-8">
              {% block content %}

              {% endblock %}
          </div>
          <div class="col-md-4">
              <h3 class="page-header">Muvalab Blog</h3>
              <p><a href="{% url "blog:post_feed" %}"
target="_blank">Subscribe to RSS Feed</a></p>
          </div>
  </div>
</body>
</html>
```

У рядку 19 ми додали цей код: `<p><a href="{% url "blog:post_feed" %}" target="_blank">Subscribe to RSS Feed</a></p>`, які додають посилання для підписки на канал RSS на бічній панелі блогу. Тепер відкрийте наступне посилання у своєму веб-переглядачі:

http://localhost:8000/blog/

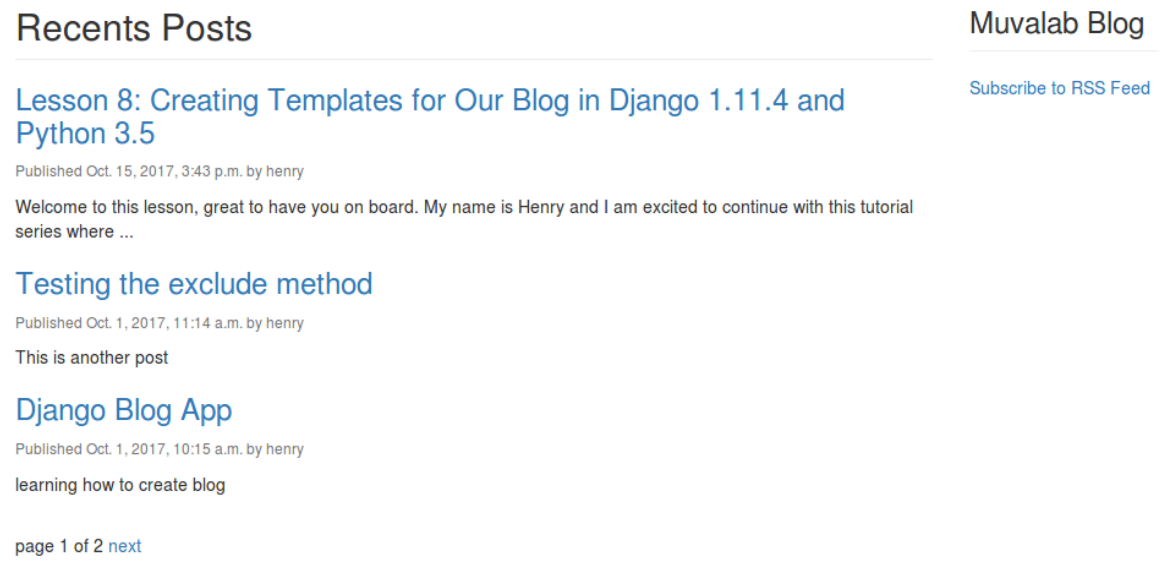

Рис.92. Сторінка з посиланням підписки на RSS-канали

Наведене вище зображення показує, що наше посиланя підписки було успішно додане.