ЛАБОРАТОРНА РОБОТА № 2
СТВОРЕННЯ ПРОЕКТУ DJANGO ТА ЗАПУСК СЕРВЕРА ІЗ NGINX

Мета роботи

Ознайомитися із основними можливостями фреймфорку Django, навчитися створювати та запускати проекти із використанням утиліти Supervisor та Nginx сервера.

Теоретична частина

1.1 Мова програмування Python

Python є високорівневою мовою програмування загального призначення, яка орієнтована на підвищення продуктивності роботи розробника і читабельності коду. Python підтримує декілька парадигм програмування, в тому числі структурне, об'єктно-орієнтоване, функціональне, імперативне і аспектно-орієнтоване. Основні архітектурні риси — динамічна типізація, автоматичне керування пам'яттю, повна інтроспекція, механізм обробки винятків, підтримка багатопоточних обчислень і зручні високорівневі структури даних. Код на Python організовується у функції та класи, які можуть об'єднуватися в модулі (вони в свою чергу можуть бути об'єднані в пакети).

Основні переваги Python:

– швидкість виконання програм, написаних на Python дуже висока: це пов'язано з тим, що основні бібліотеки Python написані на C++ і виконання завдань займає менше часу, ніж на іншими мовами високого рівня;

– також існує можливість створення власних модулів для Python на C або C++;

– мова характеризується чітким і послідовним синтаксисом, продуманою модульністю і масштабованістю, завдяки чому початковий код написаних на Python програм можна легко прочитати;

– у стандартних бібліотеках Python можна знайти засоби для роботи з електронною поштою, протоколами Інтернету, FTP, HTTP, бази даних, тощо;

– скрипти, написані за допомогою Python виконуються на більшості сучасних ОС, така переносимість забезпечує Python популярність в різних областях програмування.

1.2 Django фреймворк

У якості основи для розроблення підсистеми можна використовувати Django — фреймворк (програмний каркас) з багатими можливостями, що підходить для розробки складних сайтів і веб-застосувань, написаний мовою програмування Python.

Один із основних принципів фреймворку — DRY (don’t repeat yourself). Дотримання принципу програмування DRY дозволяє домогтися високого рівня супроводжуваності проекту: простоти внесення змін і якісного тестування. Веб-системи на Django будуються з одного або декількох застосувань, які рекомендується робити відокремленими і такими, що підключаються. Це одне з помітних архітектурних відмінностей цього фреймворку від деяких інших (наприклад, Ruby on Rails). Також, на відміну від багатьох інших фреймворків, обробники URL в Django конфігуруються у явному вигляді (за допомогою регулярних виразів), а не задаються автоматично зі структури контролерів.

Для роботи з базою даних Django використовує власний ORM (англ. Object-Relational Mapping, укр. об'єктно-реляційне відображення), в якому модель даних описується класами Python, і за ним генерується схема бази даних.

Архітектура Django схожа на «Модель-Подання-Контролер» (MVC). Контролер класичної моделі MVC приблизно відповідає рівню, який в Django називається Уявлення (View), а презентаційна логіка Подання реалізується в Django рівнем Шаблонів (Templates). Через цю багаторівневу архітектуру Django часто називають архітектурою типу «Модель-Шаблон-Подання» (MTV). Загальна схема зображена на рис. 1.1.

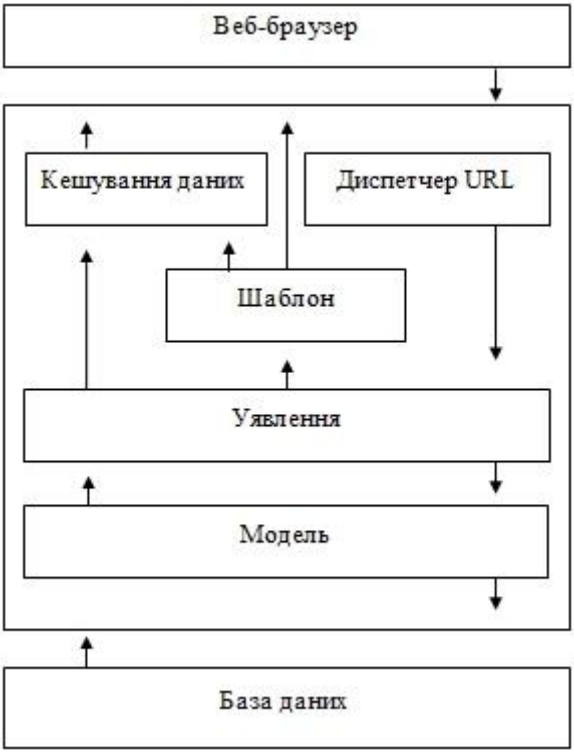

Рисунок 1.1 - Django MTV архітектура

Фреймворк надає ряд засобів, які допомагають у швидкій розробці веб-застосунків. Наприклад, не потрібно створювати контролери та сторінки для адміністративної частини сайту, в Django є вбудований застосунок для управління вмістом, який можна включити в будь-який застосунок, зроблений на Django, і який може керувати відразу декількома web застосуваннями на одному сервері. Адміністративне застосування дозволяє створювати, змінювати і видаляти будь-які об'єкти наповнення web застосунку, протоколюючи всі вчинені дії, і надає інтерфейс для управління користувачами і групами (з пооб'єктним призначенням прав).

Повна схема роботи Django-застосування наведена на рис. 1.2.

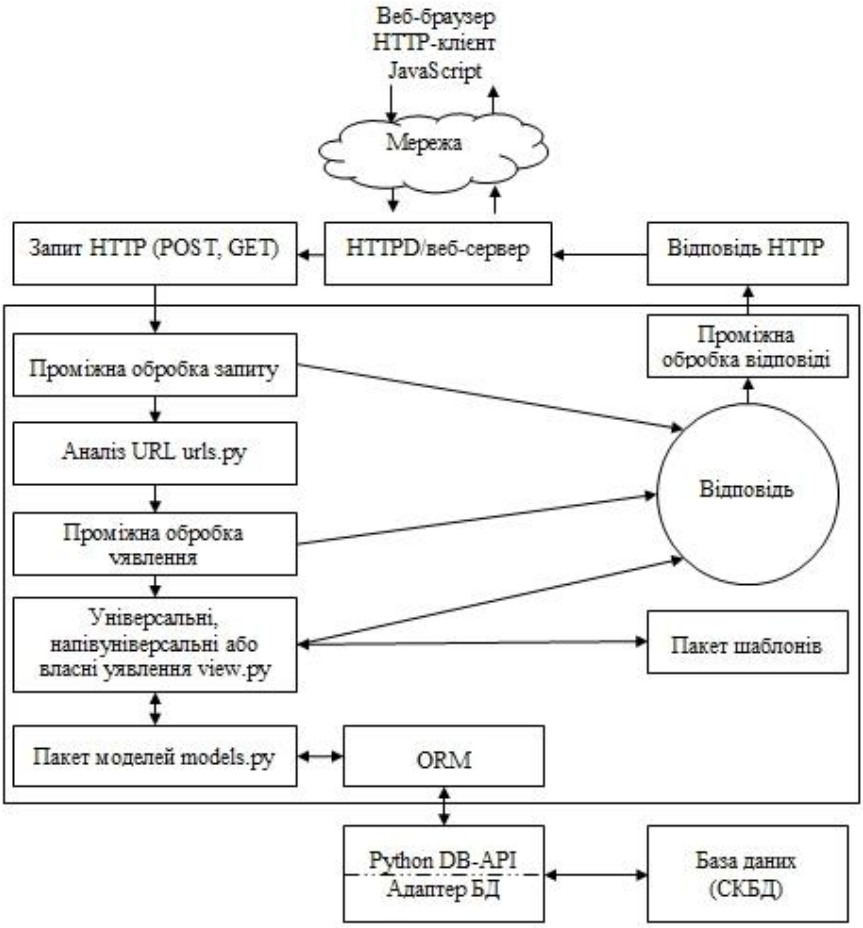

Рисунок 1.2 – Зображення роботи застосування

1.3 Віртуальне середовище

Віртуальне оточення для Python — дуже зручний інструмент при одночасній роботі з декількома проектами. При розробці встановлюються різні бібліотеки, версія самого Python також може відрізнятися.

Використання віртуального середовища дозволяє абстрагуватися від бібліотек, що використовуються в системі.

Створюється віртуальне оточення, активується, і після цього всі модулі Python будуть встановлені лише в даному віртуальному оточенні. Щоб працювати в іншому проекті з іншими версіями бібліотек, досить просто перемкнути віртуальне оточення.

virtualenv — власне утиліта для створення віртуальних оточень.

Для установки досить виконати:

```
sudo apt-get install python-pip
pip install virtualenv
```

Для створення віртуального оточення, виконайте:

```
virtualenv <ім'я віртуального оточення>
```

У поточному каталозі буде створена нова директорія з вказаною вами назвою, куди будуть перенесені python, pip і надалі встановлені інші бібліотеки.

Активація віртуального оточення проводиться командою:

```
<ім'я віртуального оточення>\bin\activate
```

Для деактивації достатньо виконати:

```
deactivate
```

1.4 Nginx

NGINX – програмне забезпечення, написане для UNIX-систем. Основне призначення – самостійний HTTP-сервер, або, як його використовують частіше, фронтенд для високонавантажених проектів. Можливе використання NGINX як SMTP/IMAP/POP3-сервера, а також зворотного TCP проксі-сервера.

Для встановлення:

```
sudo apt-get install nginx
```

Для запуску(start), зупинки(stop) або перегляду стану(status) ngninx служби виконайте:

```
sudo service nginx <action>
```

Для того, щоб nginx застосував нові налаштування, виконайте:

```
sudo service nginx reload
```

Для перевірки правильності конфігурацій, існує команда:

```
sudo nginx -t
```

1.5 WSGI

Веб сервер приймає та обслуговує запити. Він може віддавати файли (HTML, зображення, CSS, і т.д.) безпосередньо зі своєї файлової системи. Однак він не може напряму працювати з Django застосуванням. Необхідний інтерфейс, який буде запускати Django застосування, передавати йому запити від веб клієнта, наприклад браузера, і повертати відповіді.

Для цього був розроблений Web Server Gateway Interface (WSGI) - стандарт взаємодії Python програм і веб-сервера. uWSGI є однією з реалізацій WSGI. Необхідно налаштувати uWSGI для створення Unix-сокету та взаємодії з веб-сервером за допомогою протоколу WSGI. Повний стек компонентів буде виглядати наступним чином:

веб-клієнт ⇔ веб-сервер ⇔ сокет ⇔ uwsgi ⇔ Django.

Встановити uWSGI для Python можна, якщо активувати віртуальне середовище, та виконати команду:

```
pip install uwsgi
```

1.6 Система Supervisor

Supervisor - це утиліта керування процесами в операційній системі. Нею можна скористатися, якщо є програми, які потребують перезапуску за певними правилами. Таким чином не потрібно буде писати підсистему управління (rc-скрипти, систему моніторингу та перезапуску) для таких програм.

Supervisor запускає процеси як свої підпроцеси, тому він має над ними повний контроль і знає їх точний стан.

supervisorctl надає системний контроль, а також веб-інтерфейси для моніторингу та керування процесами. Користувач може бачити стан програм та виконувати дії над ними (start, stop, restart).

Можна групувати програми і здійснювати над ними спільні дії (наприклад, перезавантаження всіх програм). Також є можливість вказувати пріоритет для кожної програми, тим самим організовуючи порядок перезапуску.

Ця утиліта написана на Python, працює на Linux, Mac OS X, Solaris і FreeBSD.

Для встановлення достатньо виконати команду у терміналі:

```
sudo apt-get install supervisor
```

Для перевірки стану програми виконайте команду:

```
sudo service supervisor status
```

Команда запуску утиліти:

```
sudo service supervisor start
```

Практична частина

1.7 Почакові налаштування

Будучи бібліотекою Python, Django вимагає Python. Підходить будь-яка версія Python 2.6, 2.7, 3.2 або 3.3.

Скачайте на Python http://www.python.org. Якщо ви використовуєте Linux або Mac OS X можливо він вже встановлений. Перевірити чи встановлений Python можна виконавши команду `python` у терміналі, ви повинні побачити приблизно наступне:

```
Python 2.7.4 (default, Sep 26 2013, 03:20:26)
[GCC 4.7.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

1.8 Налаштування віртуального середовища

Створимо папку, віртуальне середовище для проекту та активуємо його:

```
mkdir django-project
cd django-project
virtualenv env
source env/bin/activate
```

Після активування, перед ім'ям користувача з'явиться назва віртуального середовища у дужках, наприклад:

```
(env)user@host:~/django-project$
```

Перед встановленням будь-яких залежностей для проекту (бібліотек) середовище повинне бути активоване.

1.9 Створення Django-проекту

1.9.1 Перший проект

Встановимо залежність Django та створимо проект:

```
pip install Django
django-admin.py startproject djangosite
```

Після виконання останньої програми з'явиться нова папка з відповідним ім'ям:

```
cd djangosite/
```

Структура проекту виглядає наступним чином:

```
.
├── djangosite
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── manage.py
```

– djangosite /__init__.py - порожній файл, який вказує Python, що поточний каталог є пакетом Python;

– djangosite /settings.py - налаштування/конфігурація проекту;

– djangosite /urls.py - конфігурація URL для проекту Django;

– djangosite /wsgi.py - точка входу проекту для WSGI-сумісних веб-серверів;

– ./manage.py - скрипт для взаємодії із проектом Django.

Розглянемо детальніше файл з налаштуваннями. Змінна INSTALLED_APPS містить додатки Django, які використовуються в проекті. За замовчуванням, вона містить наступні програми, всі вони включені в Django:

– django.contrib.admin – інтерфейс адміністратора;

– django.contrib.auth – система аутентифікації;

– django.contrib.contenttypes – фреймворк, що надає високорівневий, узагальнений інтерфейс для роботи з моделями;

– django.contrib.sessions – фреймворк сесії;

– django.contrib.messages – фреймворк повідомлень;

– django.contrib.staticfiles – фреймворк для роботи зі статичними файлами.

Ці програми включаються за замовчуванням. Деякі з цих програм можуть використовувати принаймні одну таблицю в базі даних, тому необхідно створити базу даних. Для цього виконайте наступну команду:

```
python manage.py migrate
```

За замовчуванням використовується SQLite база даних. SQLite включений в Python, тому не потрібно встановлювати жодних додаткових бібліотек.

Команда migrate аналізує значення INSTALLED_APPS і створює всі необхідні таблиці в базі даних, використовуючи налаштування бази даних з файлу settings.py та міграції з застосування (про це далі).

Для того, щоб перевірити правильність первинних налаштувань, необхідно перейти до каталогу із файлом manage.py та

виконати команду:

```
python manage.py runserver
```

У терміналі буде наступне:

```
Performing system checks...
System check identified no issues (0 silenced).
October 08, 2016 - 20:15:07
Django version 1.10.2, using settings 'djangosite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Якщо перейти за посиланням http://127.0.0.1:8000/ можна побачити стандартну сторінку Django (рис. 1.3).

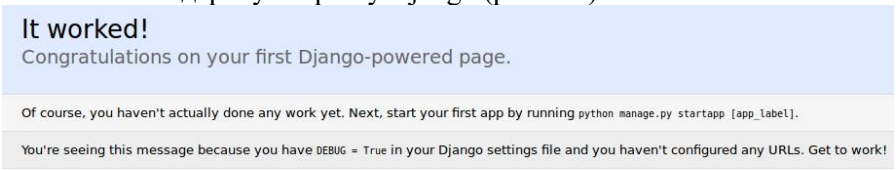

Рисунок 1.3 – Сторінка тестового запуску Django

1.9.2 Створення застосування

Кожний проект Django складається з пакетів Python (застосування). Застосування надає певний функціонал – наприклад, web-блог. Проект – це сукупність програм і конфігурації сайту. Проект може містити кілька застосувань.

Для створення застосування треба виконати:

```
python manage.py startapp djangoapp
```

djangoapp — назва застосування. Після цього змінилась структура проекту:

```
.
├── db.sqlite3
├── djangoapp
│   ├── admin.py
│   ├── apps.py
│   ├── __init__.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── djangosite
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   ├── wsgi.py
└── manage.py
```

Про структуру застосування:

– djangoapp /admin.py — файл для налаштування адмінської частини;

– djangoapp/apps.py — файл керування налаштуваннями застосування;

– djangoapp/migrations — каталог, що містить файли міграцій;

– djangoapp/models.py — файл моделей проекту, що описують структуру бази даних;

– djangoapp /tests.py — файл для тестування;

– djangoapp/views.py — файли, у яких описується логіка програми та обробка запитів за URL.

Створене застосування необхідно додати до змінної INSTALLED_APPS у файлі налаштувань djangosite/settings.py:

```
INSTALLED_APPS = [
    ...,
    'djangoapp'
]
```

1.9.3 Створення моделей та бази даних

Відредагуйте файл djangoapp/models.py, щоб він виглядав наступним чином:

```
from __future__ import unicode_literals
from django.db import models

class Measurement(models.Model):
    value = models.CharField(max_length=50)
    description = models.CharField(
max_length=250, blank=True, null=True)

    def __unicode__(self):
        return self.value
```

Модель представлена класом, успадкованим від django.db.models.Model. Вона містить декілька атрибутів: значення та опис, кожен з яких відображає поле в таблиці бази даних. Опис є необов'язковим полем. Кожне поле є екземпляром класу Field – наприклад, CharField для текстових полів. Це вказує Django які типи даних зберігають ці поля. Метод __unicode__ відповідає за репрезентацію об'єкту, повертає str.

Виконайте команду:

```
./manage.py makemigrations
```

Ви повинні побачити наступне:

```
Migrations for 'djangoapp':
  djangoapp/migrations/0001_initial.py:
    - Create model Measurement
```

Після цього:

```
./manage.py migrate
```

Після виконання `makemigrations`, Django відстежує зміни у ваших моделях та зберігає їх до міграцій. Міграції використовуються Django для збереження змін ваших моделей (і

структури бази даних) - це просто файли на диску. Ви можете вивчити міграцію для створення ваших моделей, вона знаходиться у файлі djangoapp/migrations/0001_initial.py.

Команда migrate виконує міграції і автоматично оновлює базу даних.

1.9.4 HTML шаблони Django

Для використання шаблонів необхідно створити каталог templates в кореневому каталогу проекту поруч із manage.py та внести зміни до файлу налаштувань в змінній TEMPLATES за ключем DIRS додати, щоб це виглядало наступним чином:

```
TEMPLATES = [
    {
        ...,
        'DIRS':          [os.path.join(BASE_DIR,
'templates'), ],
        ...
    },
]
```

Також додайте два шаблони до каталогу templates: measurement_list.html та create_measurement.html

1.9.5 View у Django

Уявлення – це "тип" сторінок вашого застосування, що є функцією для обробки запиту та використовує шаблон для генерації сторінок.

Для створення першого уявлення додамо текстовий файл test_data. Він повинен лежати поряд з manage.py. Нехай його зміст буде наступним:

```
7483728478329
7483784738924
8989438948394
4787438784758
8958439589493
8594389543905
5843958943895
```

Відредагуйте файл djangoapp/views.py:

```
import os
from django.views import generic
from django.conf import settings
from djangoapp.models import Measurement


class MeasurementList(generic.ListView):
    model = Measurement
    context_object_name = 'measurements'
    template_name = 'measurement_list.html'


    def get(self, request, *args, **kwargs):
       try:
          file_path                                    =
os.path.join(settings.BASE_DIR, 'test_data')
          measurement_file    =    open(file_path,
'r')
          for value in measurement_file:

Measurement.objects.get_or_create(value=value)
       except IOError:
          pass

        return             super(MeasurementList,
self).get(request, *args, **kwargs)
```

В Django є безліч класів, що спрощують процес розробки проектів, наприклад, клас ListView призначений для виведення списків об’єктів. Він дозволяє отримати об’єкти, модель яких вказана у атрибуті model класу, та використовувати у HTML шаблоні за допомогою змінної у атрибуті context_object_name. template_name дозволяє вказати назву шаблону, що буде використовуватися.

Відредагуємо шаблон measurement_list.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <title>Виміри</title>
</head>
<body>
 <h3>Виміри</h3>
 {% for measurement in measurements %}
     <p>{{ measurement }}</p>
 {% endfor %}
</body>
</html>
```

В шаблоні використовується ітератор for, який перебирає масив з об'єктів Measurement. Доступ до змінної здійснюється за допомогою «{{ }}». Саме тут можна побачити застосування методу __unicode__ об'єкту, адже об'єкт виглядає як рядок.

Цього було б достатньо для використання уявлення. Але перевизначивши метод get, ми додали більше функціональності до нашого уявлення. Відкривається файл test_data та зчитуються дані, при цьому вони зберігаються у базу даних. Метод get_or_create дозволяє сворити об'єкт в базі даних, тільки якщо такого ще не існує, що дозволяє уникнути дублювання даних.

1.9.6 URL адреса

Для того щоб переглянути результат, додамо URL шаблон до djangosite/urls.py:

```
from django.conf.urls import url
from django.contrib import admin
from djangoapp.views import MeasurementList
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$',         MeasurementList.as_view(),
name='measurement_list')
]
```

URL-шаблон - це загальна форма URL-а. Наприклад: /measurement/<id>/.

Щоб з URL-а отримати уявлення, Django використовується так званий 'URLconf'. URLconf визначає відповідність URL-шаблонів (є регулярними виразами) і уявлень.

Тепер якщо запустити сервер та перейти за адресою http://127.0.0.1:8000/, можна буде побачити список об’єктів Measurement за даними з файлу test_data, вони додані до бази даних. Спробуйте додати рядок із новими даними до файлу та обновити сторінку.

1.9.7 Форми

Якщо ви плануєте створювати сайти і застосування, що приймають і зберігають дані, вам необхідно використовувати форми.

Форма в HTML – це набір елементів <form>...</form>, які дозволяють користувачу вводити текст, вибирати опції, змінювати об'єкти сторінки, і так далі, а потім відправляти цю інформацію на сервер.

Крім <input> елементів форма повинна містити ще дві речі:

– куди: URL, на який будуть відправлені дані;

– як: HTTP метод, який повинна використовувати форма для відправки даних.

GET і POST – єдині HTTP методи, які використовуються для форм. Будь-який запит, який може змінити стан системи - наприклад, який змінює дані в базі даних - повинен використовувати POST. GET повинен використовуватися для запитів, які не впливають на стан системи. Не слід використовувати GET для форми з паролем, тому що пароль з'явиться в URL, а отже в історії браузера і логах сервера. Також він не підходить для відправки великої кількості даних або бінарних даних, наприклад, зображення. Web-застосування, яке використовує GET запити для аутентифікації можна буде легко зламати: не складно підробити форму для запиту на сервер і отримати важливі дані про систему. POST використовує додаткові механізми захисту, наприклад, CSRF захист, і надає більше контролю за доступом до даних. GET є зручним для таких речей, як форма пошуку, тому що URL, який представляє GET запит, можна легко зберегти в обране або відправити поштою.

Django дозволяє:

– підготувати дані для відображення у формі;

– створити HTML форми для даних;

– отримати та обробити надіслані формою дані.

Ви можете написати код, який все це буде робити, але Django може виконати більшу частину самостійно.

HTML <form> лише частина необхідної схеми. Серце всього механізму – клас Form. Як і модель в Django, яка описує структуру об'єкта, його поведінку і уявлення, Form описує форму, як вона працює і відображається для користувача.

В каталог djangoapp додайте файл forms.py з наступним змістом:

```
from django import forms
from djangoapp.models import Measurement

class MeasurementForm(forms.ModelForm):
    class Meta:
        model = Measurement
        fields = '__all__'
```

Таким чином ми створили Django форму для моделі Measurement, вибравши при цьому усі атрибути. Можна було вказати `['value', 'description']`, результат буде однаковий. Також у формах виконується валідація, можна додати власні умови — для цього потрібно перевизначити метод clean.

Для того, щоб форму можна було показати користувачу необхідно створити View, яке буде працювати із формою. Додамо наступний код до файлу djangoapp/views.py:

```
from django.urls import reverse_lazy
from djangoapp.forms import MeasurementForm

class MeasurementCreate(generic.CreateView):
    form_class = MeasurementForm
    template_name = 'create_measurement.html'
    success_url                                    =
reverse_lazy('measurement_list')
```

Зауважте, що всі імпорти повинні знаходитися на початку файлу. CreateView — клас, призначений для обробки запитів на створення нових об’єктів бази даних. Для роботи з формою треба вказати form_class та назву шаблона.

Функція reverse_lazy дозволяє Django отримати url адресу за назвою, що вказана у змінній name URL шаблону, щоб уникнути написання повної адреси та помилок у разі внесення змін до адреси. success_url визначає url, на яку буде перенаправлений користувач у разі успішного створення об'єкту.

Додамо нову URL адресу для створення об'єкту Measurement:

```
from djangoapp.views import  MeasurementCreate

urlpatterns = [
    ...,
    url(r'^create/',MeasurementCreate.as_view(),
name='create_measurement')
]
```

Файл create_measurement.html може виглядати так:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Додати вимір</title>
</head>
<body>
<form  action="{%  url  'create_measurement'  %}"
method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Submit" />
</form>
</body>
</html>
```

До шаблону додається елемент form із атрибутом action, що визначає на яку адресу буде відправлений запит, та method — яким методом. У випадку створення нового об'єкту використовується запрос типу POST. `{% url <url_name> %}` - це шаблонний тег, що дозволяє отримати повну адресу за ім'ям цієї адреси у структурі URL Django.

Django забезпечує захистом проти Cross Site Request Forgeries. Для відправки форми через POST необхідно використовувати шаблонний тег `csrf_token`.

form є контекстною змінною, та містить поля разом із їхніми назвами. Метод `as_p` не є обов'язковим до застосування, він використовується лише для виведення полів форми з використанням тегів `<p></p>`.

Поле input з типом submit є фактично кнопкою, за допомогою якої користувач підтверджує відправлення даних до серверу.

Додайте посилання на нову сторінку до шаблону measurement.html:

```
<a href="{% url 'create_measurement' %}">Додати вимір</a>
```

Відкрийте у браузері головну сторінку застосування. Якщо ви додали посилання на сторінку, перейдіть за ним. Ви повинні побачити два поля для введення даних, спробуйте створити об’єкт. Після успішного додавання об’єкту, ви повинні побачити головну сторінку із списком усіх об’єктів, в тому числі і щойно доданий.

1.10 Налаштування uWSGI взаємодії

Якщо ви ще не встановили uwsgi, виконайте команду у активованому середовищі:

```
pip install uwsgi
```

Далі необхідно створити новий файл uwsgi.ini поруч із manage.py з наступним змістом:

```
[uwsgi]
chdir = /home/user/django-project/djangosite
module=djangosite.wsgi
socket = 127.0.0.1:8080
chmod-socket=666
home=/home/user/django-project/env
```

– chdir використовується для позначення каталогу проекту;

– module — Django wsgi файл;

– socket — вказує на адресу для підключення сокету. Якщо перейти за цією адресою у браузері, то користувач нічого не побачить, оскільки браузер використовує протокол http, а не uWSGI;

– chmod-socket — визначає права доступу до файлу сокета;

– home вказує на віртуальне оточення для вашого проекту.

1.11 Використання Supervisor

Необхідно додати запуск Django проекту до Supervisor. Якщо утілита ще не встановлена, виконайте:

```
sudo apt-get install supervisor
```

Далі додайте файл djangoproject.conf до каталогу /etc/supervisor/conf.d:

```
[program:djangoproject]
directory=/home/user/django-project/djangosite
command=/home/user/django-project/env/bin/uwsgi
--ini uwsgi.ini
stdout_logfile=/home/user/django-
project/logs/supervisor.log
```

– directory вказує на головну директорію вашого проекту;

– command визначає безпосередньо команду яка буде виконуватися та відстежуватися;

– stdout_logfile вказує на файл, куди записується уся інформація щодо роботи вашої програми.

Перейдіть до каталогу django-project та виконайте:

```
mkdir logs
```

Виконайте команду для оновлення конфігурацій:

```
sudo supervisorctl reread
```

У терміналі повинні побачити наступне:

```
djangoproject: available
```

Далі потрібно виконати команду для перезапуску утиліти:

```
sudo supervisorctl reload
```

Виконайте команду:

```
sudo supervisorctl
```

Якщо все виконано правильно, ви побачите приблизно наступне:

```
djangoproject                    RUNNING    pid
13390, uptime 0:00:07
```

Якщо замість цього и бачите помилку або статус FATAL, це означає, що конфігурація виконана неправильно. Переглянути деталі можна в файлі /home/user/django-project/logs.

1.12 Серверні налаштування

Виконайте установку, якщо необхідно:

```
sudo apt-get install nginx
```

Перевірте статус програми, та запустіть якщо необхідно. До каталогу /etc/nginx/sites-available додайте файл djangoproject:

```
upstream django {
    server 127.0.0.1:8080;
}

server {
    listen      9000;
    location / {
        uwsgi_pass  django;
        include     uwsgi_params;
    }
}
```

В даному файлі конфігурацій директива upstream описує сервер, що слухає на UNIX-сокеті локально за портом 8080 (співпадає з параметром socket у файлі uwsgi.ini). Server задає адресу та інші параметри серверу. Директива listen відповідає за порт, за яким сайт буде доступним у браузері, location вказує куди повинен бути перенаправлений запит для оброблення. Таких директив може бути декілька, URL можуть бути задані регулярними виразами. В нашому випадку достатньо «/».

Перейдіть до /etc/nginx/sites-enabled та створіть посилання на щойно створений файл:

```
sudo ln -s /etc/nginx/sites-available/djangoproject ./djangoproject
```

Також у файлі /etc/hosts необхідно відредагувати рядок із локальною адресою та додати туди назву сайту за вашим бажанням, наприклад:

```
127.0.0.1  localhost djangoproject
```

Необхідно перезапустити nginx для застосування нових конфігурацій. Якщо налаштування були виконані правильно, то після переходу за адресою http://djangoproject:9000/ у браузері повинен відобразитися ваш проект.