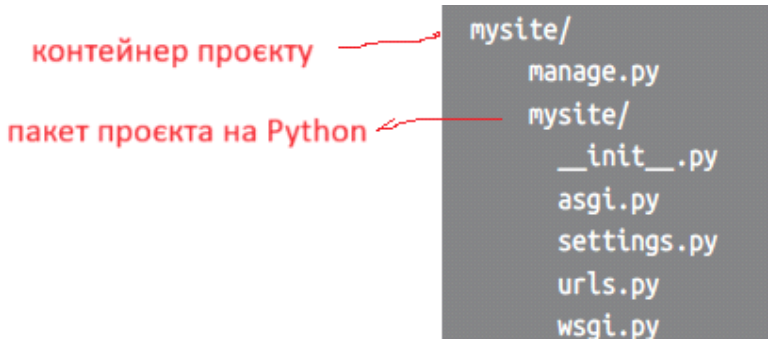
Django

1. СТВОРЕННЯ ПРОЄКТУ

Створення проекта mysite:

```
django-admin startproject mysite
```

Створюється проєкт з назвою mysite



Початкова міграція бази даних:

```
python manage.py migrate
```

Запуск сервера розробки (для перевірки коректного створення проєкту):

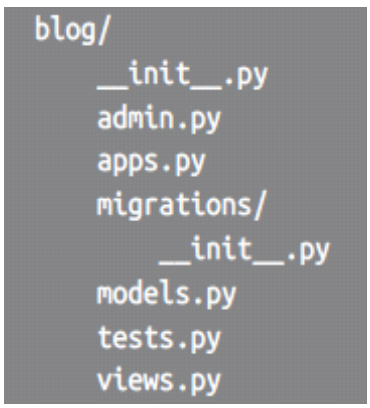
```
python manage.py runserver
```

Файл settings.py параметри:
DEBUG = True *(при розгортанні змінити на Falsee - відключити режим отладки)*
ALLOWED_HOSTS = [] *(при розгортанні додати домен/хост, щоб дозволити йому раздавати ваш сайт Django)*

2. СТВОРЕННЯ ЗАСТОСУНКУ

Створення застосунку blog:

```
python manage.py startapp blog
```



Ниже приведено описание этих файлов:

- __init__.py: пустой файл, который сообщает Python, что каталог blog нужно трактовать как модуль Python;
- **admin.py**: здесь вы регистрируете модели, чтобы включать их в состав сайта администрирования – этот сайт используется опционально, по вашему выбору;
- **apps.py**: содержит главную конфигурацию приложения blog;
- **migrations**: этот каталог будет содержать миграции базы данных приложения. Миграции позволяют Django отслеживать изменения модели и соответствующим образом синхронизировать базу данных. Указанный каталог содержит пустой файл __init__.py;
- models.py: содержит относимые к приложению модели данных; все приложения Django должны иметь файл models.py, но его можно оставлять пустым;
- **tests.py**: здесь можно добавлять относимые к приложению тесты;
-**views.py**: здесь расположена логика приложения; каждое представление получает HTTP-запрос, обрабатывает его и возвращает ответ.
Когда структура приложения готова, можно приступать к разработке моделей данных блога.

3. СТВОРЕННЯ МОДЕЛІ

Створення моделей робиться у файлі models.py застосунку.

Каждой модели ставится в соответствие одна таблица базы данных, где каждый атрибут класса соотносится с полем базы данных.
При применении миграций Django будет создавать таблицу по каждой модели, определенной в файле **models.py** приложения.

```
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=250)
    slug = models.SlugField(max_length=250)
    body = models.TextField()

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

Коли Django або Python потребує **рядкового представлення** вашого об'єкта N, він автоматично викликає метод __str__. Ви, як розробник, маєте визначити, яке саме поле (або комбінація полів) найкраще ідентифікує ваш об'єкт.
Приклади для class Name(models.Model):

- **Якщо Name — це модель User (Користувач):**

```
class User(models.Model):
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    email = models.EmailField(unique=True)

    def __str__(self):
        return f"{self.first_name} {self.last_name}" # Або self.email
```

Тут об'єкт User буде відображатися як "Іван Петров" або "john.doe@example.com".

Django по умолчанию добавляет поле автоматически увеличивающегося первичного ключа в каждую модель. Тип этого поля указывается в конфигурации каждого приложения либо глобально в настроечном параметре DEFAULT_AUTO_FIELD. При создании приложения командой startapp значение параметра **DEFAULT_AUTO_FIELD** по умолчанию имеет тип **BigAutoField**. Это 64-битное целое число, которое увеличивается автоматически в соответствии с доступными идентификаторами. Если не указывать первичный ключ своей модели, то Django будет добавлять это поле автоматически. В качестве первичного ключа можно также определить одно из полей модели, установив для него параметр **primary_key=True**.