

## ЛАБОРАТОРНА РОБОТА № 2

### СТВОРЕННЯ DJANGO-АПЛІКАЦІЙ. РОБОТА З БАЗОЮ ДАНИХ ТА ІНТЕРФЕЙСОМ АДМІНІСТРАТОРА

#### Мета роботи

Ознайомитись з процесом створення Django-аплікації та процесом налаштування проекту, створити базу даних для сайту.

#### Теоретичні відомості

Деякі можливості Django:

- ORM, API доступу до БД з підтримкою транзакцій;
- вбудований інтерфейс адміністратора, з уже наявними перекладами багатьма мовами;
- URL-диспетчер на основі регулярних виразів;
- розширювана система шаблонів з тегами і спадкуванням;
- система кешування;
- підключається архітектура додатків, які можна встановлювати на будь-які Django-сайти;
- авторизація та автентифікація, підключення зовнішніх модулів автентифікації: LDAP, OpenID та інші;
- система фільтрів («проміжного шару») для побудови додаткових обробників запитів, як наприклад включені в дистрибутив фільтри для кешування, стиснення, URL нормалізації і підтримки анонімних сесій;
- бібліотека для роботи з формами (успадкування, побудова форм по існуючій моделі БД);
- вбудована автоматична документація по тегам шаблонів і моделей даних.

**Django Аплікація** – це невелика бібліотека коду, яка представляє один аспект цілого вашого проекту. Зазвичай Django Проект складається із кількох (деколи дуже багатьох) Django Аплікацій. Деякі із цих аплікацій є внутрішніми для вашого конкретного проекту і ніколи не використовуватимуться у інших проектах, в той час як інші є зовнішніми і можуть використовуватися вами у інших Django проектах.

**Сторонні Django Пакети (Додатки, Аплікації)** – python пакети та Django аплікації створені іншими розробниками, які ви використовуєте у ваших власних проектах.

Кожна Django аплікація не повинна виконувати більше, ніж одну задачу у проекті. Тобто заведено тримати аплікації малими.

Важливо відзначити, що Django-додатки слідують парадигмі *Модель - Представлення - Шаблон*. У двох словах, додаток отримує дані від моделі, представлення робить щось з даними, та створюється шаблон, що містить оброблену інформацію. Таким чином, шаблони Django відповідають представленню у традиційному MVC і представлення Django можуть бути прирівнені до контролерів в традиційному MVC.

Додаток Python входить до складу проекту і реалізує функціональність одного з розділів сайту і всіх його підрозділів. Кількість додатків в проекті не

обмежена. Фізично додаток являє собою пакет, папка якого знаходиться в папці проекту. Ім'я цього пакета стане ім'ям програми, а сам пакет називається пакетом додатку. Пакет додатку формується самою Django при створенні програми.

Спочатку він містить наступні модулі:

- **migrations/**: тут Django зберігає деякі файли для відстеження змін, створених у файлі **models.py**, щоб підтримувати синхронізацію бази даних та моделей **models.py**.
- **admin.py**: файл конфігурації для вбудованого застосунку Django під назвою **Django Admin**.
- **apps.py**: файл конфігурації самого застосунку.
- **models.py**: тут ми визначаємо об'єкти нашого веб-застосунку. Django автоматично переводить моделі в таблиці бази даних.
- **tests.py**: файл використовується для написання модульних тестів для застосунку.
- **views.py**: файл, в якому ми обробляємо цикл запитів/відповідей нашого веб-застосунку.

Зрозуміло, ми можемо, якщо виникне така необхідність, створити в пакеті додатки інші модулі, що зберігають код моделей, контролерів або додаткових функцій і класів, що ми задіємо в коді сайту. Django в цьому плані ніяк нас не обмежує. Ось тільки всі шаблони, які застосовуються в сайті, треба створити вручну. Навіть якщо додаток входить до складу проекту, це ще не говорить про те, що він буде задіяний в сайті. Щоб додаток успішно працював, слід, по-перше, виконати його прив'язку до інтернет-адреси, а по-друге, вказати його в списку активних додатків, що знаходиться в модулі **settings** пакета проекту. Тільки після цього додаток стане активним. Потрібно сказати, що частина допоміжних модулів Django реалізована також у вигляді додатків (вбудовані додатки). Таким вбудованим додатком є, зокрема, підсистема, що реалізує розмежування доступу.

Адміністративний сайт, за допомогою якого ми можемо працювати зі збереженими в базі даними, також є додатком подібного роду.

Вбудовані додатки або також прив'язуються до інтернет-адреси та, тим самим, формують новий розділ сайту, або працюють постійно, забезпечуючи допоміжну функціональність. У будь-якому випадку їх також потрібно вказати в списку активних додатків, інакше вони не будуть задіяні. Прив'язка інтернет-адрес. Ми знаємо, що кожен додаток сайту, запускається у відповідь на звернення до певної інтернет-адреси, до якої він був прив'язаний. Єдиний виняток тут – вбудовані додатки, з якими ми тільки що познайомилися і які працюють постійно і не вимагають такої прив'язки.

У бібліотеці Django прив'язка інтернет-адреси до додатка виконується в модулі **urls** пакета проекту. Або, кажучи іншими словами, прив'язка адрес до додатків виконується на рівні проекту. Але в реальності один додаток може виконувати відразу кілька дій. Скажімо, додаток списку товарів може виводити як і сам цей список, так і відомості про обраний товар, а додаток гостьової книги – як виводити гостьову книгу, так і додавати в неї новий запис. Як це реалізувати? Дуже просто. Окремим контролерам додатків ставляться у відповідність віртуальні папки згаданих раніше папок, що формують підрозділи

даних розділів сайту. При прив'язці інтернет-адреси до контролера ми можемо вказати, що останній повинен приймати будь-які дані. Ці дані будуть передані в складі інтернет-адреси із застосуванням методу GET. Наприклад, оскільки ми збираємося реалізувати висновок відомостей про обраний товар, нам доведеться передавати відповідним контролеру ідентифікатор цього товару.

**База даних** – впорядкований набір логічно взаємопов'язаних даних, що використовуються спільно та призначені для задоволення інформаційних потреб користувачів.

Реляційна база даних - це набір таблиць та зв'язків між цими таблицями. Таблиця, в свою чергу, складається із стовпців та рядків. *Стовпці* (поля) - це структура таблиці. *Рядки* - це дані таблиці.

Один рядок в таблиці представляє одну одиницю даних і містить значення (або порожнє значення) для кожного із полів таблиці. Також рядок ще називають “записом” таблиці (англ. record). Кожне поле (англ. field) таблиці має тип подібно до того, як мова програмування Python має набір своїх типів даних.

Типи даних поділяються на три категорії: числові, текстові і типи дати та часу.

Найчастіше використовуваними типами в проекті будуть:

- VARCHAR;
- TEXT;
- ENUM;
- INT;
- DATE;
- DATETIME;

Таким чином поля таблиці визначають її структуру. А записи (рядки) таблиці містять самі дані.

### **Програма роботи**

1. Налаштувати інтегроване середовище для розробки.
2. Створити Django app та ознайомитись із принципами архітектури, роботою з представленнями.
3. Налаштувати базу даних, ознайомитись з інтерфейсом вбудованої панелі адміністратора.
4. Створити першу сторінку сайту.

### **Обладнання та програмне забезпечення**

1. Персональний комп'ютер.
2. Інтерпретатор Python встановлений на ПК
3. Середовище розробки програмного забезпечення PyCharm Community Edition (чи інше IDE, наприклад Eclipse, Sublime Text, Geany).
4. Web-фреймворк Django.

### **Порядок виконання роботи і опрацювання результатів**

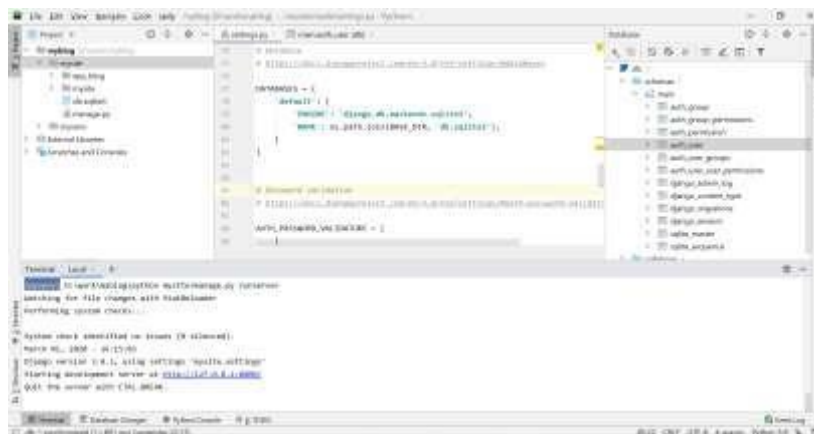
#### **Налаштування IDE**

Для прискорення процесу написання програм використовують зручно використовувати інтегроване середовище розробки, яке включає в себе різні

інструменти для роботи з кодом: засіб для написання коду (текстовий редактор), інтерактивний інтерпретатор, відлагоджувач тощо.

Веб-розробку на Django підтримує інтегроване середовище розробки для мови програмування Python PyCharm. Присутня безкоштовна версія Community. PyCharm працює під операційними системами Windows, MacOS і Linux. Ось так виглядає проект myblog, відкритий в PyCharm.

Зліва присутня панель навігації по проекту, справа власне відкритий для редагування один із файлів проекту. Зручною можливістю є робота в терміналі (в нижньому вікні клікнувши на вкладці Terminal). Детальніше про налаштування PyCharm <https://py-charm.blogspot.com/2017/09/blog-post.htm>.



Якщо ви встановили PyCharm та відкрили в ньому проект, активувати віртуальне (команда `myenv\Scripts\activate`) оточення можна тепер не за допомогою командного рядка Windows, а за допомогою терміналу PyCharm (слідкуйте, щоб при роботі з проектом віртуальне оточення було завжди активоване). Тут же вводимо команду запуску сервера (`python mysite/manage.py runserver`) і переконуємось, що проект працює перейшовши в браузері за адресою `http://127.0.0.1:8000/`. Ще однією перевагою є зручність в роботі з базою даних.

PyCharm дозволяє працювати з Git не з командного рядка, а засобами IDE (вкладка VCS). При додаванні нових файлів IDE запитуватиме, чи потрібно додавати файли до проекту. Файли, в яких внесені зміни та які не додані до репозиторію, виділяються іншим кольором для зручного керування проектом.

Після виконання лабораторної роботи, щоб закомітити внесені в проект зміни, натискаємо Commit та уважно переглядаємо змінені файли, впевнюємось що всі нові файли додано, пишемо коментар про те, що було змінено. Після цього натискаємо Push.

### Створення власного додатку

Для створення Django-аплікації. Зробіть перехід у першу папку mysite (в консолі за допомогою команди `cd`) слідкуючи при цьому, щоб віртуальне (myenv) оточення було активоване

```
(myenv) D:\work\myblog>cd mysite
```

```
(myenv) D:\work\myblog\mysite>
```

і введіть:

## **python manage.py startapp app\_blog**

Виконання цієї команди створить додаток під назвою app\_blog.

Щоб Django впізнав наш новий додаток, нам потрібно додати його назву в список Installed Apps в файлі settings.py.

```
# mysite/settings.py
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app_blog'
]
```

## **Додавання URL та Шаблонів**

Коли ми запустили сервер, то побачили дефолтну сторінку Django. Нам потрібно, щоб Django отримувала доступ до нашого додатку app\_blog, коли хтось відвідує URL головної сторінки /. Для цього нам потрібно визначити URL, який повідомить Django, де шукати шаблон головної сторінки.

Відкрийте файл urls.py у внутрішній папці mysite. Це має виглядати наступним чином.

```
"""mysite URL Configuration
```

```
The `urlpatterns` list routes URLs to views. For more information please see:
https://docs.djangoproject.com/en/3.0/topics/http/urls/
```

```
Examples:
```

```
Function views
```

1. Add an import: from my\_app import views
2. Add a URL to urlpatterns: path('', views.home, name='home')

```
Class-based views
```

1. Add an import: from other\_app.views import Home
2. Add a URL to urlpatterns: path('', Home.as\_view(), name='home')

```
Including another URLconf
```

1. Import the include() function: from django.urls import include, path
2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))

```
"""
```

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

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

Як ви бачите, є існуючий патерн URL для адмін-сайту

Django, який створено по дефолту з Django. Додамо наш власний URL, щоб вказати на наш app\_blog додаток. Відредагуйте цей файл наступним чином.

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

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

Зверніть увагу, що ми додали імпорт для include з `django.conf.urls` та додали патерн URL для порожнього маршруту. Коли хтось заходить на головну сторінку, (в нашому випадку `http://localhost:8000`), Django буде шукати більше URL в додатку `app_blog`. Так як там немає жодного, запустивши додаток ми отримаємо величезне трасування стеку через `ImportError`.

`ImportError: No module named 'app_blog.urls'`

Для того, щоб виправити це, перейдіть в папку `app_blog` і створіть файл під назвою `urls.py`. Папка `app_blog` тепер має виглядати наступним чином.

```
├── app_blog
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   ├── urls.py
│   └── views.py
```

Всередині нового файлу `urls.py`, напишіть наступне.

```
# app_blog/urls.py
from django.urls import path
from app_blog import views
```

```
urlpatterns = [
    path(r'', views.HomePageView.as_view()),
]
```

Цей код імпортує представлення з нашого `app_blog` додатку та очікуватиме визначення представлення, яке називається `HomePageView`. Так як у нас немає ні одного, відкрийте файл `views.py` в `app_blog` і введіть цей код.

```
# app_blog /views.py
from django.shortcuts import render
from django.views.generic import TemplateView
```

```
# Створіть свої представлення тут.
class HomePageView(TemplateView):
    def get(self, request, **kwargs):
```

```
return render(request, 'index.html', context=None)
```

Цей файл визначає представлення під назвою `HomePageView`. Представлення `Django` приймають `request` і повертають `response`. У нашому випадку метод `get` очікує запит `HTTP GET` до `URL`, визначеного в нашому файлі `urls.py`.

Після того, як запит HTTP GET був отриманий, метод надає шаблон під назвою `index.html`, який представляє собою звичайний HTML-файл, який може мати спеціальні теги Django шаблонів, написані зі звичайними HTML-тегами.



Якщо запустити сервер зараз, ви побачите наступну error сторінку:

Це відбувається тому, що у нас взагалі немає шаблонів. Django шукає шаблони в папці `templates` всередині вашого додатку, тому створіть її у вашій папці `app blog`.

Перейдіть до папки для шаблонів, яку ви створили та створіть файл під назвою `index.html`

В index.html вставьте цей код:

&lt;!DOCTYPE html&gt;

&lt;html lang="uk"&gt;

<head>

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

&lt;title&gt;Blog&lt;/title&gt;

&lt;/head&gt;

&lt;body&gt;

Home page

&lt;/body&gt;

Тепер запустіть ваш сервер. **python manage.py runserver** Ви побачите ваш шаблон.

## Налаштування бази даних

Відкрийте файл `mysite/settings.py`. Це звичайний модуль Python з набором змінних, які представляють настройки Django та знайдіть рядки

*# Database*

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',          'NAME':  
os.path.join(BASE_DIR, 'db.sqlite3'),  
    }  
}
```

За замовчуванням в налаштуваннях зазначено використання бази даних SQLite, це найпростіший вибір. SQLite вже включений в Python, і не потрібно додатково щось встановлювати, проте для реальних робочих проектів ця база даних не підходить. Зазвичай використовується більш «серйозна» база даних, наприклад MySQL або PostgreSQL.

Якщо ви використовуєте SQLite, вам нічого не потрібно створювати заздалегідь - файл бази даних (за замовчуванням db.sqlite3) створений



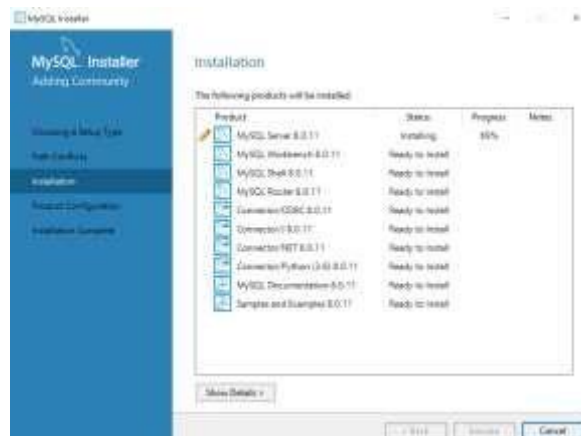
автоматично при запуску першої команди міграції.

Як уже зазначалось, SQLite підходить для навчальних цілей і не використовується на продакшені.

Для того щоб використовувати іншу «робочу» базу даних  
*Цей крок можна пропустити і використовувати SQLite.*

Наприклад, установка і налагодження MySQL під Django включає наступні кроки:

1. (якщо не встановлено) Встановіть MySQL або PostgreSQL сервер, при цьому обов'язково запам'ятайте root пароль. Цей етап при роботі під ОС Windows може викликати певні труднощі.





2. Створіть базу даних для нового сайту та надайте доступ до неї. Знаходячись в директорії, куди проінстальований MySQL Server введіть команду `mysql -u root -p` і після введення рут-пароля повинні побачити запрошення до вводу sql команд:

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 8.0.11 MySQL Community Server - GPL

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Створіть базу даних *mysite\_db*:

**create database mysite\_db;**

та надайте доступ до неї користувачу, якого вкажете в налаштуваннях (settings.py) сайту:

**create user 'mysite\_usr'@'localhost' identified by 'mysite\_pass';**

**grant all privileges on mysite\_db.\* to 'mysite\_usr'@'localhost' with grant option;**

Поміняйте наступні ключі в елементі 'default' настройки DATABASES, щоб вони відповідали налаштуванням підключення до вашої бази даних:

ENGINE - один з

'django.db.backends.sqlite3' 'django.db.backends.postgresql'  
'django.db.backends.mysql' 'django.db.backends.oracle'. також доступні інші.

NAME - назва вашої бази даних. Якщо ви використовуєте SQLite, база даних буде файлів на вашому комп'ютері, в цьому випадку NAME містить абсолютний шлях, включаючи ім'я, до цього файлу. Значення за замовчуванням, `os.path.join(BASE_DIR, 'db.sqlite3')`, створить файл в каталозі вашого проекту.

Якщо ви використовуєте не SQLite, вам необхідно вказати додатково USER, PASSWORD.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql', 'NAME': 'mysite_db',
        'USER': 'mysite_usr', 'PASSWORD': 'mysite_pass',
    }
}
```

Також зверніть увагу, що користувач бази даних з *mysite / settings.py* має права створювати базу даних («create database»). Це дозволить автоматично створювати тестову базу даних.

Також зверніть увагу на налаштування `INSTALLED_APPS` на початку файлу. Вона містить назви всіх додатків Django, які активовані у вашому проєкті. Додатки можуть використовуватися на різних проєктах, ви можете створити пакет, поширити його і дозволити іншим використовувати його на своїх проєктах.

За замовчуванням, `INSTALLED_APPS` містить наступні додаток, які надаються Django:

`django.contrib.admin` - інтерфейс адміністратора. `django.contrib.auth` - система авторизації. `django.contrib.contenttypes` - фреймверк типів даних. `django.contrib.sessions` - фреймверк сесії. `django.contrib.messages` - фреймверк повідомлень. `django.contrib.staticfiles` - фреймверк для роботи зі статичними файлами.

Ці додатки включені за замовчуванням для покриття основних завдань.

Деякі додатки використовують мінімум одну таблицю в базі даних, тому їх створити перед тим, як використовувати, так само створюються автоматично при запуску першої команди міграції. Якщо ви змінили базу даних з SQLite на MySQL, цю команду потрібно запустити знову для створення схеми, та суперюзера:

*(якщо деактивували віртуальне оточення чи закривали термінал, активуйте знову знаходячись в кореновому каталозі `D:\work\myblog>myenv\Scripts\activate`)*

**`python manage.py migrate`**

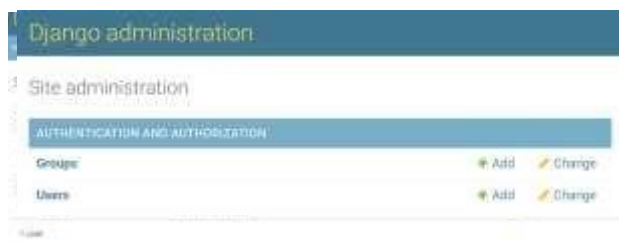
**`python manage.py createsuperuser`**

В браузері перейдіть за адресою <http://127.0.0.1:8000/admin/>, повинно з'явитися вікно входу в адмін-панель. Сюди необхідно ввести дані, які ви вказали при створенні суперюзера. Коли ви успішно увійдете до системи, перед вами відкриється головна сторінка адміністративної панелі, через яку ви можете управляти вашими додатками, редагуючи існуючі записи в базі даних або генеруючи нові.

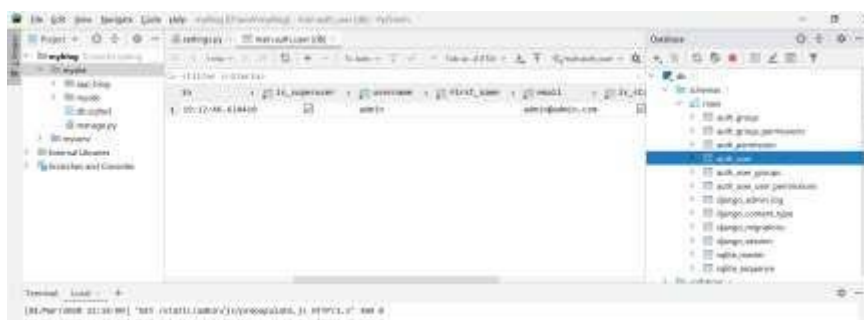


Зверніть увагу, що, незважаючи на те, що вами ще не було створено ніяких моделей і записів в базі даних, в адмінці вже є розділ аутентифікації, це одна з найбільш зручних особливостей Django.

Розкривши вкладку Users, побачите запис з даними користувача, який був створений з консолі як суперюзер.




Запис про цього користувача можна побачити також в базі даних.

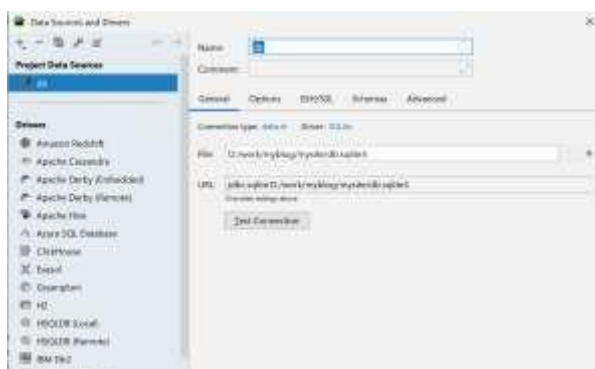


Крім того записи тут можна редагувати та фільтрувати використовуючи рядочок <filter criteria>:

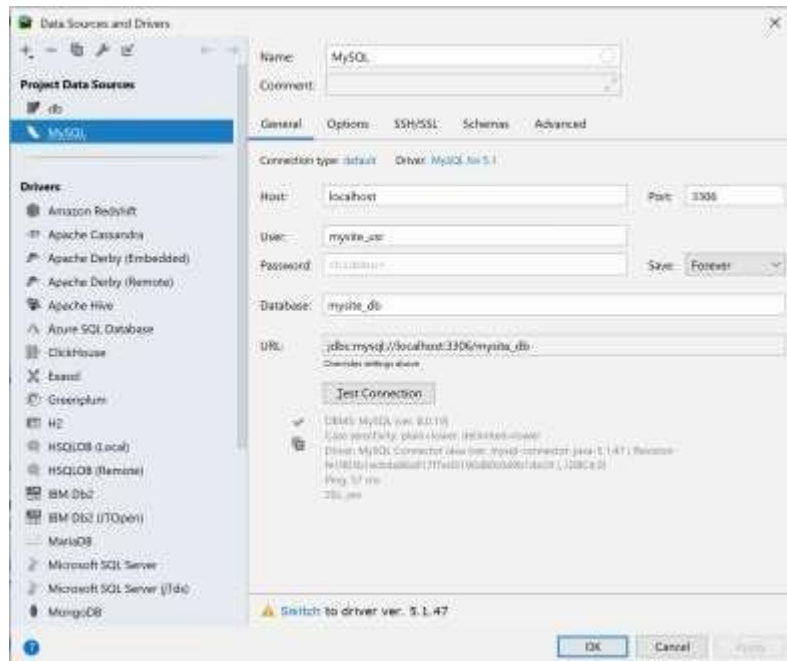


Для налаштування підключення до бази даних в PyCharm відкрийте вкладку Database, натисніть  та підключіть в налаштуваннях свою (SQLite3 або MySQL) базу даних, як показано нижче:

SQLite:



MySQL



У разі успішного підключення в правій частині (вкладка DataBase) побачите схему бази даних.



### Завдання:

Додайте декілька користувачів за допомогою панелі адміністратора та переконайтесь у тому, що записи створені в базі даних. Дослідіть схему бази даних, проаналізуйте наявні таблиці. Закомітьте зміни в репозиторій на GitHub та додайте посилання на репозиторій до звіту.

### Контрольні запитання.

1. Що таке аплікація?
2. Яка архітектура використовується в Django Framework?
3. Для чого призначений файл urls?
4. Для чого призначений файл views?
5. Де необхідно вказувати налаштування бази даних для проекту?