**Модуль 10: Основи роботи з Django**

**Заняття 1: Основи роботи з Django.**

**Вебфреймворки**

Набір інструментів на вирішення конкретного класу завдань називають **фреймворк**. Саме для реалізації веб-застосунків — вебфреймворк.

Веб-фреймворк — це програмне забезпечення (пакет), каркас, який робить розробку вебзастосунків швидше та простіше. Вебфреймворк призначений для створення динамічних вебсайтів, мережевих застосунків, сервісів або ресурсів. Він спрощує розробку та позбавляє необхідності написання рутинного коду.

WIKI

[Фреймворк](https://www.goodfirms.co/glossary/web-framework/) (https://www.goodfirms.co/glossary/web-framework/) відрізняється від поняття бібліотеки тим, що використовуючи бібліотеку (requests, NumPy тощо), програміст самостійно відповідає за потік застосунку. Тільки він вирішує, коли залучати до роботи сторонню функціональність. Фреймворк сам відповідає за потік. Він надає кілька місць для розміщення вашого коду, але викликати його або ні вирішує сам фреймворк (Django, FastAPI тощо).

У світі вебзастосунків переважна частина коду не залежить від проекту та реалізує підтримку стандартів HTTP, HTML, JSON, XML та інших. Звичайно, завжди можна написати свою власну реалізацію допоміжних інструментів, які реалізують зазначені протоколи, як вам потрібно.

NOTE

Ми починали це робити в модулі "Основи Web", коли будували простий сервер HTTP.

Але, найчастіше, на це немає сенсу витрачати час та гроші. Набагато простіше взяти одне з доступних, вільних рішень, що реалізують потрібний протокол або інструментарій.

Щоб глибоко розібратися в темі, ви можете написати свій веб-фреймворк, скориставшись [подібною інструкцією](http://mattscodecave.com/posts/simple-python-framework-from-scratch.html) (https://mattscodecave.com/posts/simple-python-framework-from-scratch.html). Але будьте готові до того, що написання власного веб-фреймворку — це дуже об'ємне і непросте завдання, яке потребує глибокого розуміння принципів роботи цілого набору технологій, що утворює сучасний Інтернет.

Типовий набір завдань, які вирішує веб-фреймворк:

* дозвіл URL адрес (перенаправлення запиту потрібної функції-обробнику, залежно від URL);
* валідація даних запиту (форм, полів, адрес);
* генерація відповіді у форматі HTML, XML, JSON;
* збереження даних та маніпуляція з даними у базі даних, налаштування з'єднань з базою;
* налаштування безпеки, захист від атак;
* збереження користувальницьких сесій.

Не всі веб-фреймворки вирішують усі ці завдання. Часто веб-фреймворк є набором вузькоспеціалізованих інструментів для вирішення одного завдання зі списку вище та "тонкого" прошарку допоміжних утиліт, що спрощують інтеграцію.

Більшість популярних сучасних фреймворків (не обов'язково із префіксом "веб") реалізують "Принцип Голівуду" (**Не телефонуйте нам, ми самі вам зателефонуємо**).

Цей принцип полягає у передачі контролю над потоком виконання застосунку фреймворку. Користувач повинен тільки модифікувати обмежений набір функцій (методів, класів), які відповідають за "бізнес-логіку" застосунку. Увесь контроль за тим, як саме обробляється подія, лежить на фреймворку.

Ви звикли писати код, який робить щось корисне і те, як саме це "щось" робиться, знаходиться під вашим контролем. Використання фреймворку передбачає інший підхід. Ви повинні віддати контроль над виконанням запиту від користувача на сторону фреймворку та реалізувати потрібний функціонал, абстрагувавшись від деталей реалізації протоколів обміну.

Це важливий аспект і його порушення перетворює будь-який фреймворк з "друга" на "ворога". Замість того, щоб спростити розробку, спроба взяти контроль над потоком виконання застосунку ускладнить розробку, і вебфреймворк буде швидше заважати, ніж допомагати.

**Класифікація web frameworks.**

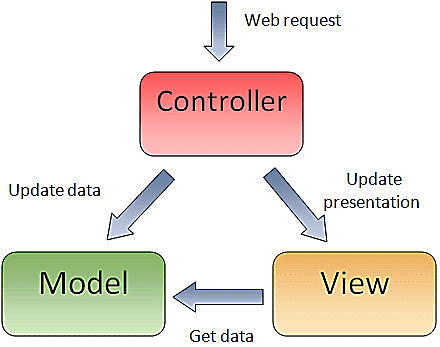
[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-10/module-10-01/intro#%D0%BA%D0%BB%D0%B0%D1%81%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%86%D1%96%D1%8F-web-frameworks)Архітектурно вебфреймворки можна розділити на:

* Model-view-controller (MVC).
* Багаторівнева організація.

**Model–view–controller**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-10/module-10-01/intro#modelviewcontroller)

[«Модель-Представлення-Контролер»](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) (https://en.wikipedia.org/wiki/Model –view–controller) — схема поділу даних застосунку, інтерфейсу користувача і керуючої логіки на три окремих компоненти: модель, представлення і контролер таким чином, що модифікація кожного компонента може здійснюватися незалежно.



**Model–view–controller**

* Модель (Model) надає дані та реагує на команди контролера, змінюючи свій стан.
* Представлення (View) відповідає за відображення даних моделі користувача, реагуючи на зміни моделі.
* Контролер (Controller) інтерпретує дії користувача, сповіщаючи модель про необхідність змін.

Такий підхід базується на тому, що формат зберігання даних не залежить від того, в якому форматі вони передаються користувачеві, а формат запиту не впливає на те, як цей запит обробляється. Така парадигма дозволяє для однієї і тієї самої бізнес-логіки зробити кілька варіантів використання (вебзастосунок, консольний застосунок, GUI-застосунок, чат-бот тощо). Для кожного такого варіанта буде своє представлення та контролер, але модель може бути спільна.

**Багаторівнева організація**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-10/module-10-01/intro#%D0%B1%D0%B0%D0%B3%D0%B0%D1%82%D0%BE%D1%80%D1%96%D0%B2%D0%BD%D0%B5%D0%B2%D0%B0-%D0%BE%D1%80%D0%B3%D0%B0%D0%BD%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F)Архітектурний шаблон «Шари» допомагає структурувати застосунки розкладанням на групи підзавдань, що знаходяться на певних рівнях абстракції.

Поширені шари:

* представлення, інтерфейс користувача відповідає за відображення даних у форматі, зручному клієнту, і перетворення клієнтських команд у формат шару застосунку;
* застосунок відповідає за обробку користувальницьких команд і працює з шаром бізнес-логіки;
* бізнес-логіка — шар предметної області, який не залежить від представлення та формату зберігання даних, реалізує набір ключових правил застосунку і є ядром застосунку;
* шар даних відповідає за зберігання даних і надає дані у визначеному на рівні бізнес-логіки форматі.

Такий шаблон підходить для великих застосунків. Дозволяє максимально абстрагуватися від деталей реалізації зберігання даних, протоколів обміну командами та способу взаємодії з користувачем. Дуже гнучкий шаблон, але може бути надлишковим для невеликих простих застосунків.

**Веб-фреймворк Django**

[**Django**](https://www.djangoproject.com/) (<https://www.djangoproject.com/>) - це один із найпопулярніших і найпотужніших фреймворків для створення web-застосунків.

INFO

Веб-фреймворк Django використовується у таких великих та відомих сайтах, як Instagram, Disqus, Mozilla, The Washington Times, Pinterest, Lamoda та ін.

Це швейцарський ніж у web-розробці на Python. Django "з коробки" надає набір інструментів для вирішення практично будь-якого завдання у web-стеку. А що не йде в стандартному постачанні, швидше за все, можна знайти у вигляді окремого плагіна.

У Django реалізовано принцип DRY (don't repeat yourself). Завдяки цьому скорочується час створення сайтів. Тобто, при використанні Django вам не потрібно кілька разів переписувати один і той самий код. Фреймворк дозволяє створювати сайт із компонентів. Це можна порівняти з будівництвом фортеці за допомогою Lego.

Спочатку розробка Django велася для забезпечення зручнішої роботи з ресурсами новин, що досить сильно позначилося на архітектурі: фреймворк надає ряд засобів, які допомагають у швидкій розробці веб-сайтів інформаційного характеру. Наприклад, розробнику не потрібно створювати контролери та сторінки для адміністративної частини сайту. У Django є вбудований застосунок для управління вмістом, який можна включити в будь-який сайт, зроблений на Django. Він може управляти одразу кількома сайтами на одному сервері. Адміністративний застосунок дозволяє створювати, змінювати та видаляти будь-які об'єкти наповнення сайту, протоколюючи всі вчинені дії, та надає інтерфейс для управління користувачами та групами (з пооб'єктним призначенням прав).

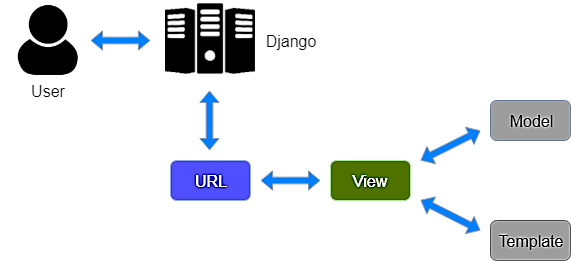
Деякі можливості Django:

* ORM, API доступи до БД з підтримкою транзакцій;
* вбудований інтерфейс адміністратора з уже наявними перекладами багатьма мовами;
* диспетчер URL на основі регулярних виразів;
* розширювана система шаблонів з тегами та наслідуванням;
* система кешування;
* інтернаціоналізація;
* архітектура застосунків, що підключається, які можна встановлювати на будь-які Django-сайти;
* "generic views" - шаблони функцій контролерів;
* авторизація та аутентифікація, підключення зовнішніх модулів аутентифікації: LDAP, OpenID та інші;
* система фільтрів (middleware) для побудови додаткових обробників запитів, як наприклад включені в дистрибутив фільтри для кешування, стиснення, нормалізації URL та підтримки анонімних сесій;
* бібліотека для роботи з формами (наслідування, побудова форм за існуючою моделлю БД);
* вбудована автоматична документація за тегами шаблонів та моделями даних, доступна через адміністративний застосунок.

**Model-View-Template**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-10/module-10-01/intro_django_02#model-view-template)Творці реалізували в Django патерн MVC, і він застосовується у поточній версії фреймворку.

Архітектура MVC дозволяє розробнику працювати з візуальним представленням та бізнес-логікою застосунку окремо. До речі, під час роботи з Django спеціалісти частіше використовують термін MVT — Model-View-Template або модель-представлення-шаблон.



**Model-View-Template**

Компоненти MVT можна використовувати незалежно один від одного.

**Model**[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-10/module-10-01/intro_django_02#model)

Документація Django визначає модель (model) як «джерело інформації про дані, в яких містяться ключові поля та поведінка даних». Зазвичай одна модель вказує на одну таблицю у базі даних.

Моделі містять інформацію про дані. Ці дані представлені атрибутами або полями. Оскільки модель є простим класом, вона нічого не знає про інші рівні Django. Взаємодія між рівнями відбувається через API.

Модель відповідає за бізнес-логіку, методи, властивості та інші елементи, пов'язані з маніпуляцією даними. Також моделі дозволяють розробникам створювати, читати, оновлювати та видаляти об'єкти в базі даних.

**View**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-10/module-10-01/intro_django_02#view)Представлення вирішує три завдання: приймає HTTP-запити, реалізує бізнес-логіку, визначену методами та властивостями, відправляє HTTP-відповідь у відповідь на запити. Тобто, представлення отримує дані від моделі та надає шаблонам (templates) доступ до цих даних або попередньо обробляє дані, а потім надає до них доступ шаблонам.

**Template**[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-10/module-10-01/intro_django_02#template)

У Django реалізований потужний двигун шаблонів та власна мова розмітки. Шаблони представляють файли HTML, за допомогою яких відображаються дані. Вміст файлів може бути статичним або динамічним. Шаблони не містять бізнес-логіку. Тому вони лише відображають дані.

**Сфера застосування**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-10/module-10-01/intro_django_02#%D1%81%D1%84%D0%B5%D1%80%D0%B0-%D0%B7%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)Фреймворк Django справляється з великою кількістю завдань та підвищеними навантаженнями. Його застосовують для створення:

* CRM-систем (Customer relationship management).
* CMS (Content management system).
* Комунікаційних платформ.
* Сервісів бронювання номерів.
* Платформ управління документообігом.

Також Django підходить для створення алгоритмічних генераторів, платформ для електронних розсилок, систем верифікації, систем фільтрації з динамічними правилами та складними параметрами, платформ для аналізу даних та складних обчислень, машинного навчання.

Тисячі сайтів у різних країнах світу створено на Django. Цей фреймворк чудово підходить для розробки веб-застосунків.

**Встановлення**

Встановлення Django може відбуватися з публічного репозиторію PyPi командою:

pip install django

Щоб перевірити, яка версію встановлена, можна виконати python -m django --version.

CAUTION

Зверніть увагу, що Django не гарантує зворотну сумісність між версіями. Застосунок, написаний для Django 3.0, може не працювати з версією 3.2.

**Створення проекту**[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-10/module-10-01/install_03#%D1%81%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F-%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D1%83)

Django надає набір консольних скриптів (команд), які автоматизують розробку. Для створення проекту достатньо виконати в папці з проектом django-admin startproject mysite.

Ця команда створить папку mysite у поточному розташуванні.

mysite/

manage.py

mysite/

\_\_init\_\_.py

settings.py

urls.py

asgi.py

wsgi.py

Зверніть увагу, ім'я проекту не повинно перетинатися з іменами, які використовує Django, наприклад django, test тощо.

* manage.py — відповідає за взаємодію із застосунком через термінал;
* mysite/settings.py — містить налаштування застосунку;
* mysite/urls.py — модуль, де описані всі URL-адреси і як вони повинні вирішуватися, відповідає за роутинг застосунку;
* mysite/asgi.py та mysite/wsgi.py — є точками входу для [ASGI](https://en.wikipedia.org/wiki/Asynchronous_Server_Gateway_Interface) (https://en.wikipedia.org/wiki/Asynchronous\_Server\_Gateway\_Interface) та  [WSGI](https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface) (https://en.wikipedia.org/wiki/Web\_Server\_Gateway\_Interface) веб -серверів для роботи з застосунком Django.

Щоб запустити локально сервер для розробки, достатньо виконати команду в консолі python manage.py runserver, знаходячись всередині папки проекту mysite.

INFO

Цей сервер не підходить для використання в реальних застосунках, але дуже корисний на етапі розробки.

**Створення web-застосунку**

Застосунок — це сутність, яка робить щось конкретне (отримує дані від користувача, відображає інформацію з бази даних тощо). Кожен проект може складатися з кількох застосунків. Застосунки також можуть бути одночасно частиною кількох проектів.

Тепер, коли у нас є проект, ми можемо створити перший застосунок:

python manage.py startapp polls

Ім'я для застосунку може бути будь-яким, у нашому випадку ми вибрали polls.

Ця команда створить пакет із наступною структурою:

polls/

\_\_init\_\_.py

admin.py

apps.py

migrations/

\_\_init\_\_.py

models.py

tests.py

views.py

Почнемо з відображення, додамо наступний код у файл polls/views.py:

**polls/views.py**

from django.http import HttpResponse

def index(request):

return HttpResponse("Hello, world. You're at the polls index.")

Ми імпортували спеціальний клас HttpResponse, який формує відповідь для клієнта. У нашому випадку це текстовий рядок `Hello, world. You're at the polls index.'

Далі, щоб представлення стало видимим, його потрібно зареєструвати. Для цього створимо модуль polls/urls.py

ВАЖЛИВО!

Імена модулів мають значення. Так, Django автоматично зрозуміє, що містить urls.py і як з ним працювати.

**polls/urls.py**

from django.urls import path

from . import views

urlpatterns = [

path('', views.index, name='index'),

]

Ми визначаємо список urlpatterns, в якому будуть маршрути для нашого застосунку. Поки що тут тільки один маршрут під час звернення до застосунку polls і він оброблятиметься функцією index з файлу views.py. Цьому маршруту ми даємо ім'я index. Тепер кореневий каталог програми polls буде дозволений за допомогою views.index.

Щоб зареєструвати сам застосунок polls, у проекті потрібно модифікувати mysite/urls.py:

**mysite/urls.py**

from django.contrib import admin

from django.urls import include, path

urlpatterns = [

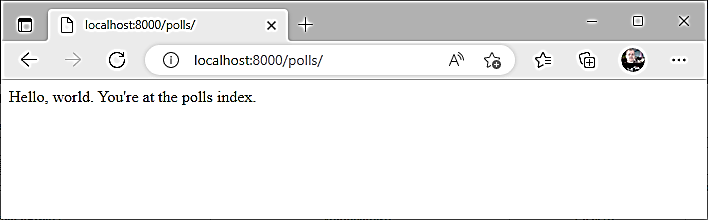
path('polls/', include('polls.urls')),

path('admin/', admin.site.urls),

]

Тут вже є шлях до адмін-панелі нашого проекту — admin/, і ми додали шлях до створеного застосунку: polls/. Функція include потрібна, щоб повідомити Django, що всі маршрути, що починаються з polls/, повинні оброблятися застосунком polls. Сам маршрут втрачає префікс polls (відбудеться заміна polls/ на /) і відправляється для обробки вже в сам застосунок.

Щоб переглянути результат роботи нашого застосунку, можна виконати python manage.py runserver та перейти на http://localhost:8000/polls/.



**http://localhost:8000/polls/**

**Робота з базами даних в Django**

Django містить дуже потужну ORM з інструментом міграцій. Щоб синхронізувати стан моделей у Python та таблиць у базі даних, достатньо виконати команду python manage.py migrate. Команда migrate "дивиться"

в INSTALLED\_APPS в mysite/settings.py і створює всі необхідні таблиці в базі даних для всіх застосунків.

Давайте створимо моделі для нашого застосунку polls. Для цього перепишемо модуль polls/models.py:

**polls/models.py**

from django.db import models

class Question(models.Model):

question\_text = models.CharField(max\_length=200)

pub\_date = models.DateTimeField('date published')

class Choice(models.Model):

question = models.ForeignKey(Question, on\_delete=models.CASCADE)

choice\_text = models.CharField(max\_length=200)

votes = models.IntegerField(default=0)

Кожен клас в polls/models.py отримає своє представлення у базі даних. Атрибути цього класу стануть полями, а об'єкт класу — записом у таблиці.

Ми створили дві моделі для питань Question та для відповідей користувачів Choice.

Тепер потрібно зареєструвати застосунок у проекті, щоб інструмент міграцій знав про ці моделі. Для цього потрібно додати наш застосунок polls до списку застосунків - константа INSTALLED\_APPS у файлі mysite/settings.py. Наша константа INSTALLED\_APPS виглядатиме:

INSTALLED\_APPS = [

"django.contrib.admin",

"django.contrib.auth",

"django.contrib.contenttypes",

"django.contrib.sessions",

"django.contrib.messages",

"django.contrib.staticfiles",

"polls",

]

Для створення міграції для нашого застосунку: python manage.py makemigrations polls.

В результаті буде створено файл міграції polls/migrations/0001\_initial.py з описом таблиць, які потрібно створити у базі даних.

Migrations for 'polls':

polls\migrations\0001\_initial.py

- Create model Question

- Create model Choice

Міграції ще не застосовувалися, ви можете подивитися, що повинно бути зроблено в міграціях, та змінити цей файл, якщо потрібно.

Щоб застосувати всі міграції: python manage.py migrate.

Щоб змінити налаштування роботи з базою даних, потрібно змінити значення константи DATABASES у файлі mysite/settings.py. Django налаштований на роботу з SQLite за замовчуванням.

DATABASES = {

"default": {

"ENGINE": "django.db.backends.sqlite3",

"NAME": BASE\_DIR / "db.sqlite3",

}

}

INFO

Щоб змінити, як саме Django відображатиме об'єкт моделі, достатньо змінити метод \_\_str\_\_ для класу моделі.

Щоб показати, як використовувати моделі у представленні, змінимо файл polls/views.py:

**polls/views.py**

from django.http import HttpResponse

from .models import Question

def index(request):

latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]

output = ', '.join([q.question\_text for q in latest\_question\_list])

return HttpResponse(output)

Тепер застосунок polls відображає клієнту останні 5 питань при запиті на кореневий ресурс застосунку /.

ПРИМІТКА

Але самі дані в базі даних поки що відсутні, і щоб їх додати туди, потрібна адмін-панель.

**Admin панель в Django**

Щоб користуватися адміністративною панеллю Django, потрібно спочатку створити користувача, який зможе зайти в адмін-панель:

python manage.py createsuperuser

Після того, як користувач успішно створений, ви можете запустити сервер і зайти на http://127.0.0.1:8000/admin/, використовуючи ім'я та пароль для нового користувача.

Поки що в панелі адміністратора немає нічого для роботи із застосунком polls. Давайте зареєструємо таблиці Question та Choice в адмін-панелі. Для цього у файлі polls/admin.py додамо наступні рядки:

from django.contrib import admin

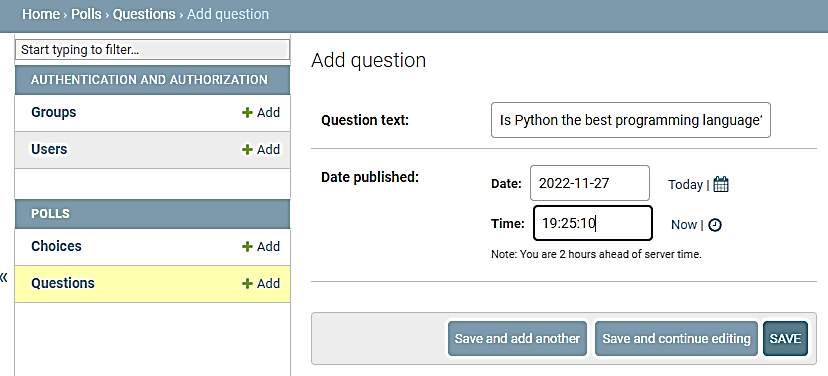
from .models import Question, Choice

admin.site.register(Question)

admin.site.register(Choice)

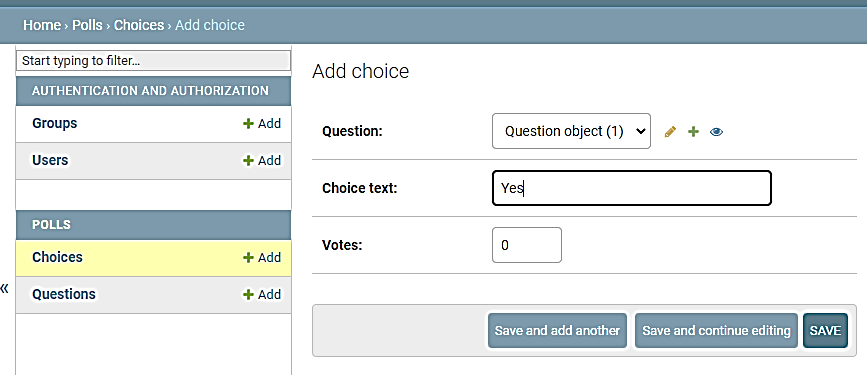
Тепер ви можете додавати питання для застосунку polls, використовуючи адмін-панель.

Додамо перше запитання "Is Python the best programming language?":



**Додавання питання**

А потім додамо дві відповіді **Yes**та**No**



**Додавання відповідей**

**Аргументи запиту**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-10/module-10-01/admin_06#%D0%B0%D1%80%D0%B3%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B8-%D0%B7%D0%B0%D0%BF%D0%B8%D1%82%D1%83)Давайте додамо ще представлень у файл polls/views.py:

def detail(request, question\_id):

return HttpResponse("You're looking at question %s." % question\_id)

def results(request, question\_id):

response = "You're looking at the results of question %s."

return HttpResponse(response % question\_id)

def vote(request, question\_id):

return HttpResponse("You're voting on question %s." % question\_id)

І зареєструємо ці представлення у файлі polls/urls.py:

from django.urls import path

from . import views

app\_name = 'polls'

urlpatterns = [

# ex: /polls/

path('', views.index, name='index'),

# ex: /polls/5/

path('<int:question\_id>/', views.detail, name='detail'),

# ex: /polls/5/results/

path('<int:question\_id>/results/', views.results, name='results'),

# ex: /polls/5/vote/

path('<int:question\_id>/vote/', views.vote, name='vote'),

]

Зверніть увагу на аргументи запиту, визначені у форматі <int:question\_id>/.

Тепер шлях /polls/34/ буде оброблений представленням views.detail, куди як аргумент question\_id буде передано число 34. Також ми додали ім'я застосунку app\_name = 'polls'. Це необхідно, щоб працювали конструкції виду <a href="{% url 'polls:detail' question.id %}">{{ question.question\_text }}</a>

**Django шаблони**

Django містить свій інструмент для генерації документів із шаблонів. Синтаксис створення шаблонів дуже схожий на такий у Jinja, з яким ви вже знайомі, але API відрізняється.

Щоб Django автоматично виявив шаблони застосунку, потрібно в пакеті застосунку створити папку polls/templates/polls. Усі HTML документи у цій папці будуть сприйняті як шаблони. Давайте створимо файл polls/templates/polls/index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Polls</title>

<!-- CSS only -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet"

integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65" crossorigin="anonymous">

</head>

<body>

{% if latest\_question\_list %}

<ul>

{% for question in latest\_question\_list %}

<li><a href="/polls/{{ question.id }}/">{{ question.question\_text }}</a></li>

{% endfor %}

</ul>

{% else %}

<p>No polls are available.</p>

{% endif %}

</body>

</html>

INFO

Для стилізації шаблонів ми використовуємо популярний фреймворк [Bootstrap](https://getbootstrap.com/) (https://getbootstrap.com/)

І оновимо файл polls/views.py таким чином, щоб використати цей шаблон:

from django.http import HttpResponse

from django.template import loader

from .models import Question

def index(request):

latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]

template = loader.get\_template('polls/index.html')

context = {

'latest\_question\_list': latest\_question\_list,

}

return HttpResponse(template.render(context, request))

Або з використанням синтаксичного цукру:

from django.shortcuts import render

from .models import Question

def index(request):

latest\_question\_list = Question.objects.order\_by('-pub\_date')[:5]

context = {'latest\_question\_list': latest\_question\_list}

return render(request, 'polls/index.html', context)

Для спрощення створення посилань у шаблонах, щоб не писати їх хардкодом, можна переписати шаблон:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Polls</title>

<!-- CSS only -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet"

integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65" crossorigin="anonymous">

</head>

<body>

<div class="container">

{% if latest\_question\_list %}

<h1>List of questions</h1>

<ul>

{% for question in latest\_question\_list %}

<li>

<a href="{% url 'polls:detail' question.id %}">{{ question.question\_text }}</a>

</li>

{% endfor %}

</ul>

{% else %}

<p>No polls are available.</p>

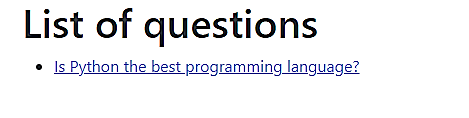
{% endif %}

</div>

</body>

</html>

Вигляд головної сторінки повинен відображати перші 5 запитів, але в нашій базі зараз лише один:



**Вигляд головної сторінки**

**Обробка форм**

Давайте створимо просту форму для вибору відповіді на запитання. Для цього створимо шаблон форми в polls/templates/polls/detail.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Detail</title>

<!-- CSS only -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet"

integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65" crossorigin="anonymous">

</head>

<body>

<div class="container">

<form action="{% url 'polls:vote' question.id %}" method="post" class="mb-3">

{% csrf\_token %}

<fieldset>

<legend>

<h1>{{ question.question\_text }}</h1>

</legend>

{% if error\_message %}

<p><strong>{{ error\_message }}</strong></p>

{% endif %}

{% for choice in question.choice\_set.all %}

<div class="form-check">

<input

class="form-check-input"

type="radio"

name="choice"

id="choice{{ forloop.counter }}"

value="{{ choice.id }}"

/>

<label for="choice{{ forloop.counter }}" class="form-check-label">{{ choice.choice\_text }}</label>

</div>

<br/>

{% endfor %}

</fieldset>

<button type="submit" class="btn btn-primary"> Vote</button>

</form>

<a href="{% url 'polls:index' %}">Go back to main page</a>

</div>

</body>

</html>

Цей шаблон при виборі відповіді та натисканні кнопки Submit надішле **POST** запит на polls/question.id/vote. Django вміє працювати з [Cross Site Request Forgeries](https://en.wikipedia.org/wiki/Cross-site_request_forgery) (https://en.wikipedia.org/wiki/Cross-site\_request\_forgery) і вам не обов'язково налаштовувати цей механізм самостійно, достатньо додати {% csrf\_token %} у шаблон, і токен буде автоматично згенерований та доданий.

Додамо тепер обробник форм у polls/urls.py:

path('<int:question\_id>/vote/', views.vote, name='vote'),

І створимо функцію обробник vote у polls/views.py:

from django.http import HttpResponse, HttpResponseRedirect

from django.shortcuts import render, get\_object\_or\_404

from django.urls import reverse

from .models import Question, Choice

...

def vote(request, question\_id):

question = get\_object\_or\_404(Question, pk=question\_id)

try:

selected\_choice = question.choice\_set.get(pk=request.POST['choice'])

except (KeyError, Choice.DoesNotExist):

# Redisplay the question voting form.

return render(request, 'polls/detail.html', {

'question': question,

'error\_message': "You didn't select a choice.",

})

else:

selected\_choice.votes += 1

selected\_choice.save()

# Always return an HttpResponseRedirect after successfully dealing

# with POST data. This prevents data from being posted twice if a

# user hits the Back button.

return HttpResponseRedirect(reverse('polls:results', args=(question.id,)))

Об'єкт request містить атрибут POST, який поводиться як словник, і ви можете отримати значення полів форми: request.POST['choice']. Оновлення значення поля виконує метод choice\_set, цей метод автоматично створений Django і встановлює зв'язок між Question і Choice об'єктами.

Після оновлення інформації в базі ми можемо надіслати redirect відповідь, щоб браузер автоматично перенаправив користувача на іншу сторінку. Для цього використовується HttpResponseRedirect, який перенаправить на URL, що відповідає за відображення результатів polls:results з аргументом у запиті, що дорівнює question.id. Щоб згенерувати цю URL-адресу, викликаємо функцію reverse.

Додамо обробник результатів у polls/views.py:

def results(request, question\_id):

question = get\_object\_or\_404(Question, pk=question\_id)

return render(request, 'polls/results.html', {'question': question})

І відповідний йому шаблон у polls/templates/polls/results.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Results</title>

<!-- CSS only -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/css/bootstrap.min.css" rel="stylesheet"

integrity="sha384-rbsA2VBKQhggwzxH7pPCaAqO46MgnOM80zW1RWuH61DGLwZJEdK2Kadq2F9CUG65" crossorigin="anonymous">

</head>

<body>

<div class="container">

<h1>{{ question.question\_text }}</h1>

<ul class="list-unstyled">

{% for choice in question.choice\_set.all %}

<li class="mb-3">

{{ choice.choice\_text }} &mdash;

<mark>{{ choice.votes }}</mark>

vote{{ choice.votes|pluralize }}

</li>

{% endfor %}

</ul>

<a href="{% url 'polls:detail' question.id %}" class="btn btn-info">Vote again?</a>

</div>

</body>

</html>

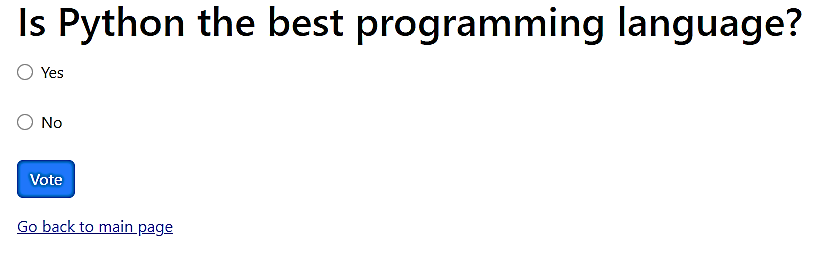
Також потрібно оновити представлення pools/views.py, функцію detail:

def detail(request, question\_id):

question = get\_object\_or\_404(Question, pk=question\_id)

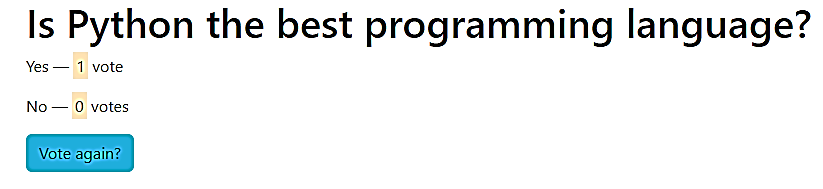
return render(request, 'polls/detail.html', {'question': question})

Тепер, перейшовши на /polls/1/ у браузері, ви можете проголосувати



**/polls/1/**

та побачити сторінку з результатами після голосування



/**polls/1/results/**

Готовий результат повинен бути як у цьому живому прикладі, запустіть та подивіться код:

Щоб побачити питання, після запуску прикладу, натисніть

[посилання](https://django-intro.krabaton.repl.co/polls/) (https://replit.com/@Krabaton/Django-intro).

Ми закінчили вступ до фреймворку Django і розглянули приклад створення базового застосунку. Тепер переходимо до побудови складнішого застосунку.