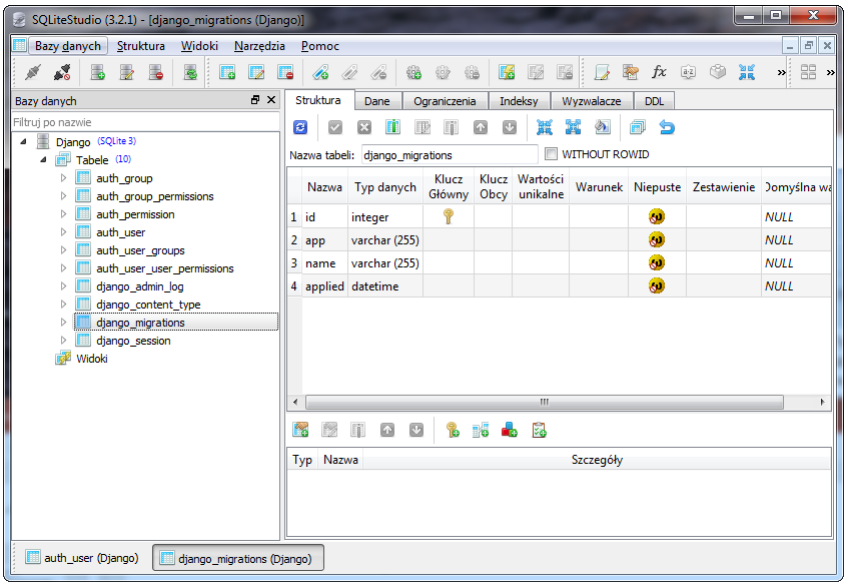
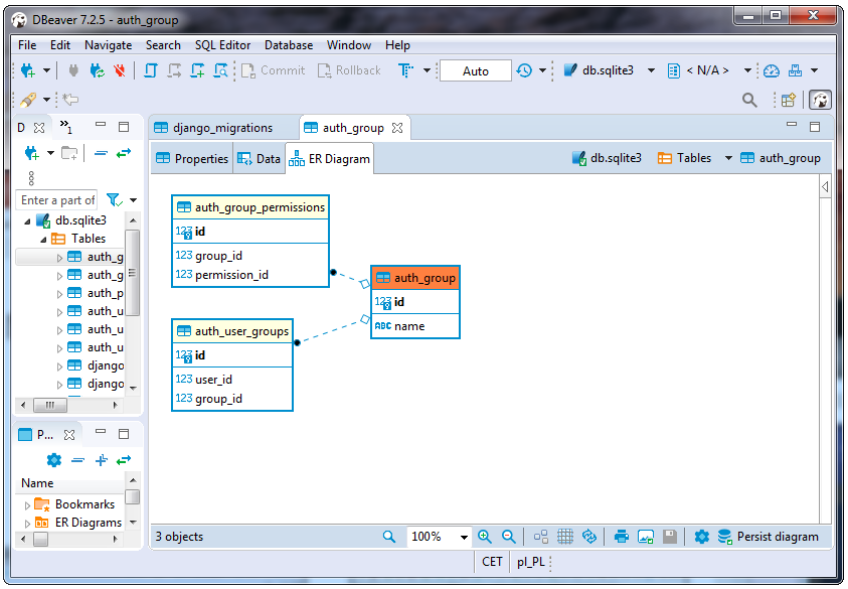
Django

Narzędzia SQLite

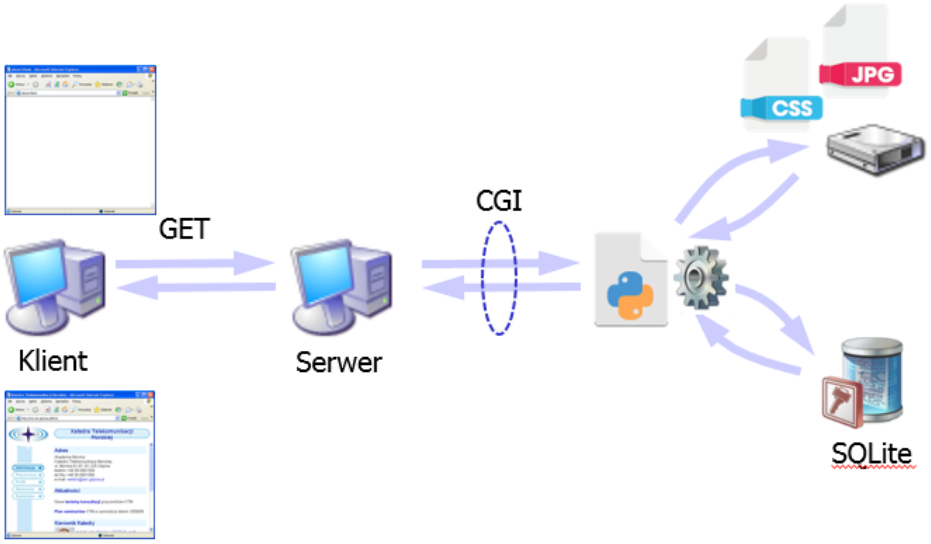
- SQLiteStudio
  Darmowy program do zarządzania bazami danych SQLite;
  Ma narzędzia exportu/importu baz danych do różnych formatów danych (XML, JSON, …); Bez wersji instalacyjnej ("portable"):
  https://www.sqlitetutorial.net/download-install-sqlite/



- DBeaver Community
  Darmowa wersja rozbudowanego programu do zarządzania różnymi typami baz danych, w tym SQLite; Jest dostępny zarówno w wersji "portable", jak i z instalatorem
  https://dbeaver.io/

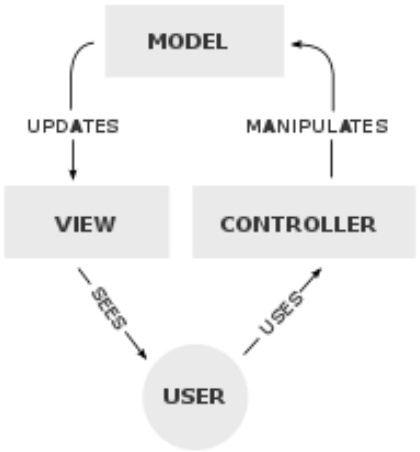


Dynamiczne strony WWW – CGI



Model MVC

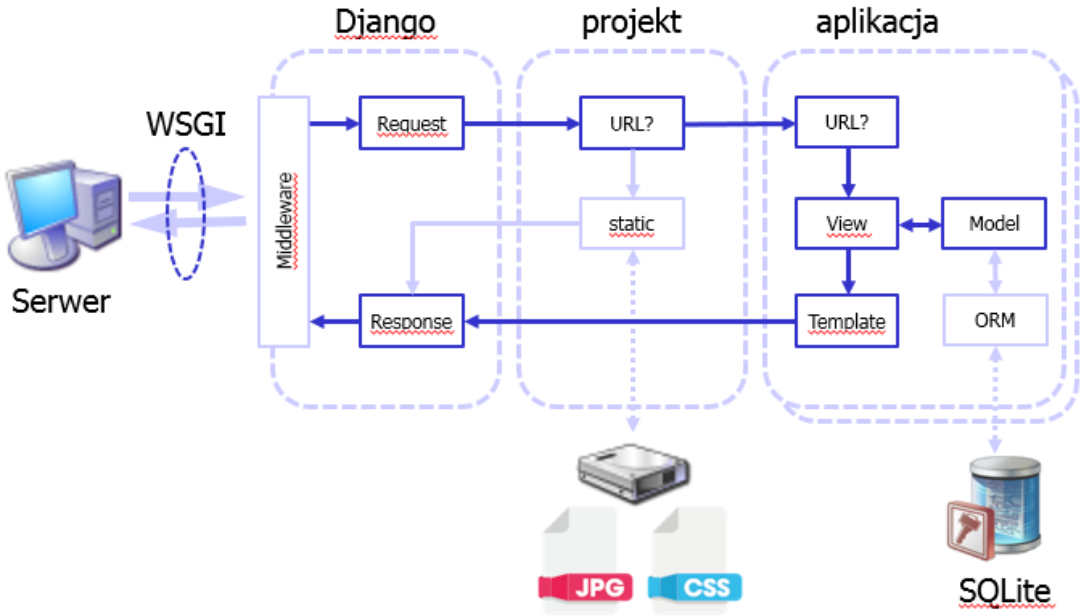
- Django jest zbudowany w oparciu o wariant modelu MVC (Model-View-Controller):
  – Model (model) – logika biznesowa
  – View (widok) – sposób prezentacji danych w GUI
  – Controller (kontroler) – pośredniczy między modelem a widokiem (aktualizacja modelu oraz odświeżanie widoków)
- Model MVC separuje logikę aplikacji od interfejsu użytkownika – umożliwia zmianę części modelu (np. bazy danych) bez zmian w widoku i na odwrót



Warianty modelu MVC

- MVP (model-view-presenter)
- MVVC (model-view-view model)
- Django: MVT (model-view-template)
  Sugestia z dokumentacji (formalnie takiego modelu nie ma):
  – Model (model) – klasy z ORM i logiką biznesową,
  – View (widok) – wybiera dane z modelu (wg adresu URL)
  – Template (szablon) – sposób prezentacji danych
  Model MVT Django separuje również dane (view) od sposobu ich prezentacji (template); Kontrolerem z modelu MVC w Django jest sam Django

Dynamiczne strony WWW – Django



Generowanie strony WWW przez Django:

- Serwer WWW odbiera żądanie HTTP;
  Jeżeli z konfiguracji serwera i URL żądania wynika, że żądanie ma być obsłużone przez skrypt WSGI, serwer przygotowuje dane o żądaniu w odpowiednim formacie i przekazuje do WSGI
- Django odbiera dane z WSGI i tworzy obiekt HttpRequest;
  Dane z odebrane WSGI są przekształcane, np. tworzone są oddzielne struktury na dane z formularzy oraz ciasteczka – dzięki temu dostęp do potrzebnych danych jest wygodniejszy;
  HttpRequest jest słownikiem języka Python
- Django "dekoduje" URL żądania i przeszukuje pliki urls.py projektu oraz aplikacji – na tej podstawie wybiera widok (tj. znajduje funkcję widoku)
- Django wywołuje funkcję widoku, przekazując jej HttpRequest;
  Dodatkowo, jeżeli we wzorcu URL widoku były zmienne (np. /archiwum/<rok>), funkcja widoku otrzymuje te zmienne; Widok jest odpowiedzialny za wygenerowanie obiektu HttpResponse – Django przekształci go zgodnie z wymogami interfejsu WSGI i przekaże do serwera WWW; Obiekt HttpResponse można utworzyć bezpośrednio w funkcji widoku, jednak typowo wykorzystuje się do tego szablony, natomiast dane pobiera się z bazy danych przez modele
- Funkcja widoku tworzy potrzebny model (albo modele) oraz wywołuje odpowiednie funkcje modelu, zgodnie z żądaniem oraz ewentualnie danymi z formularzy. Model aktualizuje swój stan (jeżeli to potrzebne) i dostarcza potrzebne dane.
- Funkcja widoku tworzy kontekst – zbiór wszystkich danych potrzebnych do wygenerowania strony WWW i używając szablonu renderuje (tworzy) dokument HTML; Następnie tworzy obiekt HttpResponse (zawierający dokument HTML oraz typ zawartości i kod odpowiedzi), który zwraca jako swój rezultat;
  Najczęściej wykorzystuje się do tego funkcję "skrótu" render
- Django przekształca HttpResponse zgodnie z wymaganiami WSGI i zwraca do serwera WWW, który odsyła go przeglądarce
- W generowaniu odpowiedzi może być użyty Middleware;
  Klasy Middleware są wskazane w konfiguracji i są aktywowane na różnych etapach

generowania odpowiedzi (zależnie od funkcji, które mają zdefiniowane);
Django ma kilka użytecznych Middleware, ale można też definiować własne

- Obiekty statyczne (pliki CSS, obrazki, skrypty Javascript, …) powinny być umieszczone w folderze aplikacji wskazanym
w konfiguracji, domyślnie "static", i jego podfolderach;
Są odsyłane do serwera bez udziału widoków;

Struktura projektu Django:

```
<projekt>
 - settings.py
 - urls.py
 - wsgi.py
 - <aplikacja>
 - <aplikacja>
```

- Settings – konfiguracja projektu
- Urls – plik startowy dekodowania adresów URL
(ciąg dalszy znajduje się w poszczególnych aplikacjach)
- Foldery aplikacji – każda aplikacja ma własny folder

Struktura aplikacji (bardzo prostej) Django:

```
<aplikacja>
 - <static>
 - urls.py
 - views.py
 - models.py
 - <templates>
    - szablon.html
```

- Folder <static> - pliki statyczne
- Urls – plik dekodowania adresów URL aplikacji
- Views.py – plik widoków
- Models.py – plik modeli
- Folder <templates> - pliki szablonów

Konfiguracja Django (<projekt>/settings.py):

- Włączenie trybu debugowania – powoduje wyświetlanie szczegółowych raportów o błędach w oknie przeglądarki;
Należy wyłączyć w wersji produkcyjnej projektu

```
DEBUG = True
```

- Zainstalowane aplikacje – należy **dopisać** nazwy własnych aplikacji – bez tego Django nie znajdzie np. plików szablonów

```
INSTALLED_APPS = [
    […]
    'nazwa-aplikacji',
]
```

- Plik URL – startowy plik dekodowania adresów żądań

```
ROOT_URLCONF = 'project.urls'
```

- Baza danych – lokalizacja i ew. dane uwierzytelniania połączenia z bazą danych, domyślnie SQLite,

```
DATABASES = {
  'default':
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': BASE_DIR / 'db.sqlite3',
  }
}
```

- Folder plików statycznych – domyślnie "static", można zmienić

```
STATIC_URL = '/static/'
```

- Ustawienia regionalne

```
LANGUAGE_CODE = 'pl-pl'
TIME_ZONE = 'Europe/Warsaw'
USE_I18N = True
USE_L10N = True
USE_TZ = True
```

Dekodowanie adresów URL żądań:

Plik urls.py projektu

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

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

- Początkowo zawiera tylko ścieżkę do panelu zarządzania, "admin/" – aplikacji generycznej Django
- Funkcja include dołącza plik urls aplikacji
- Powinien uwzględniać stronę główną serwisu ("pusty" URL)
- Ważne! Adres powinien kończyć się znakiem "/"

Plik urls.py aplikacji

```
from django.urls import path
from . import views

urlpatterns = [
  path('', views.blog, name='blog'),
  path('all/', views.blog, name='blog'),
  path('arch/<int:year>/', views.year),
]
```

- Rozpoznawane adresy URL są połączeniem adresów z plików urls projektu i aplikacji, np. "blog/all/" = "blog/" + "all/"
- W adresach można używać zmiennych i wyrażeń regularnych
- Można stosować aliasy

Typy zmiennych w adresach URL

- Typy zmiennych są zdeterminowane przez konwertery:
  – str – dowolny niepusty łańcuch znaków, bez separatora "/"
  – int – liczba całkowita nieujemna
  – slug – łańcuch znaków z myślnikami, "litery-i-cyfry-np-13"
  (popularny w portalach informacyjnych i systemach CMS)

```
'arch/<int:year>/'
'arch/<int:year>/<int:month>'
'notes/<slug:title>/'

r'^arch/(?P<year>[0-9]{4})/$'
```

- Można też używać wyrażeń regularnych (r'…') oraz definiować własne konwertery

Nazwy adresów URL i funkcja reverse()

- Plik urls.py aplikacji

```
urlpatterns = [
  path('', views.blog, name='blog'),
  path('arch/<int:year>/', views.year),
]
```

- Drugi argument wskazuje plik widoku, który zostanie użyty do wygenerowania strony
- Trzeci opcjonalny argument nazwany (name='<nazwa>') rejestruje nazwę adresu URL – taki adres można uzyskać przy pomocy funkcji reverse i użyć np. w odnośnikach menu

```
from django.urls import reverse
loginPage = reverse('login')
```

Funkcja widoku

- Po zdekodowaniu adresu URL żądania, Django wywołuje funkcję widoku, przekazując jej HttpRequest:

```
from django.shortcuts import render
from .models import Blog, Archive

def blog(request):
  return render(request, 'blog.html',
    {'data': Blog.GetAllData()}
  )

def arch(request, year):
  return render(request, 'arch.html',
    {'data': Archive.getYear(year)}
  )
```

- Funkcja widoku zwykle tworzy obiekt modelu i poprzez odpowiednie funkcje modelu aktualizuje bazę jego stan oraz pobiera dane potrzebne do wyświetlenia strony
- Dodatkowo, jeżeli we wzorcu URL widoku były zmienne (np. /archiwum/<int:year>), funkcja widoku otrzymuje te zmienne w postaci dodatkowych argumentów;
- Funkcja widoku jest odpowiedzialna za wygenerowanie kompletnej strony WWW, którą zwraca jako rezultat (obiekt typu HttpResponse)
- Funkcja widoku może zrobić to sama, ale zwykle używa się do tego celu szablonów.
- Za aktywowanie właściwego szablonu odpowiada obiekt klasy Dispatcher ("dyspozytor"). Oczywiście można tworzyć własne obiekty dispatcherów albo używać systemu szablonów spoza Django
- Wbudowane szablony Django najlepiej jest używać poprzez funkcję skrótu render, która znajduje i renderuje szablon, po czym zwraca gotowy obiekt HttpResponse; Funkcji render trzeba przekazać m.in. kontekst – wszystkie dane potrzebne do wygenerowania strony, w słowniku języka Python

Zadania

Proszę utworzyć model i widoki w projekcie Django, zgodnie z instrukcjami zamieszczonymi w poradniku "Django Girls", a w tym celu:

1. Dodać model Post i wykonać migrację
   https://tutorial.djangogirls.org/pl/django_models/
2. Utworzyć konto administratora, zalogować się do panelu administracyjnego i utworzyć kilku postów https://tutorial.djangogirls.org/pl/django_admin/

   Efekty wykonania punktów 1 i 2 sprawdzić za pomocą programu do zarządzania bazami danych SQLite, najlepiej DBeaver Community (https://dbeaver.io/), który wyświetla diagram powiązań między tabelami
3. Dodać URL
   https://tutorial.djangogirls.org/pl/django_urls/
4. Dodać funkcję widoku, pobierającą dane z modelu
   https://tutorial.djangogirls.org/pl/django_views/
   https://tutorial.djangogirls.org/pl/dynamic_data_in_templates/

   Można pominąć publikowanie postów przez interaktywny interpreter Pythona, należy tylko zmienić sposób pobierania danych z modelu, tj. zamiast funkcji filter, `posts = Post.objects.filter(…)`, użyć funkcji all, `posts = Post.objects.all()`
5. Dodać szablon:
   https://tutorial.djangogirls.org/pl/django_templates/

   Należy pominąć Bootstrap i zamiast tego dodać arkusz stylów z ćwiczenia HTML/CSS
6. Rozbudować szablon przy użyciu znaczników block i extend:
   https://tutorial.djangogirls.org/pl/template_extending/
7. Dodać podstrony szczegółów wpisu:
   https://tutorial.djangogirls.org/pl/extend_your_application/