

Web

приложения

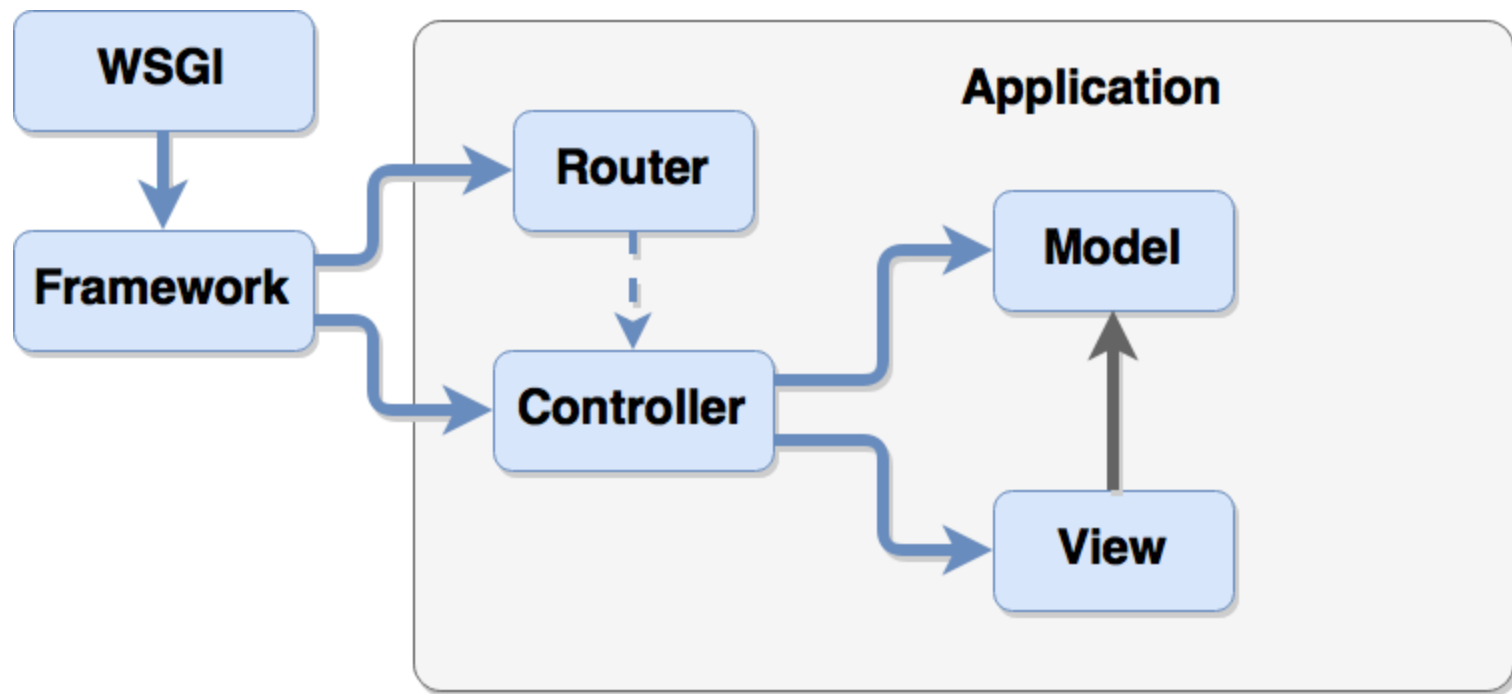
Основные типы запросов

- Запросы статических документов
- Запросы динамических документов
- Отправка данных форм
- AJAX - запросы
- Запросы к API сайта
- Персистентные соединения

Основные задачи

- Маршрутизация URL
- Парсинг заголовков и параметров запроса
- Хранение состояния (сессии) пользователя
- Выполнение бизнес-логики
- Работа с базами данных
- Генерация HTML страницы или JSON ответа

MVC



Роли компонентов MVC

- Router - выбор конкретного controller по URL
- Model - реализация бизнес-логики приложения
- Controller - работа с HTTP, связь model и view
- View - генерация HTML или другого представления

django



yiiframework



Плюсы фреймворков

- + Готовая архитектура
- + Повторное использование кода
- + Экономия ресурсов
- + Участие в Open Source
- + Проще найти программистов
- + Проще обучать программистов

Django

Соглашение о именовании

MVC

Django

Model

Model

Router

urls.py

Controller

views

View

templates

Структура проекта

`django-admin startproject project` - создание проекта.

`./manage.py startapp crm` - создание приложения.

```
project
├── crm
│   ├── models.py
│   ├── urls.py
│   └── views.py
├── manage.py
└── project
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

Основные файлы проекта

- `manage.py` - скрипт управления проектом
- `project/settings.py` - настройки
- `project/urls.py` - router, список URL проекта
- `project/wsgi.py` - WSGI приложение, точка входа
- `crm` - Django - приложение

Структура не-open-source проекта

```
project
├── crm
│   └── ...
├── blog
│   └── ...
├── manage.py
├── project
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── templates
└── static
```

Django приложения

Django приложения

Приложения - способ распространения кода в Django инфраструктуре. В случае, если вы не планируете публиковать ваш код, приложения - это просто способ логической организации кода.

`./manage.py startapp crm` - создание нового приложения с именем `crm`. Нужно вызывать из директории проекта.

Структура приложения

```
|— templates
|— static
|— templatetags
|— management
|   └— commands
|— migrations
|— models.py
|— tests.py
|— urls.py
└— views.py
```


Основные файлы приложения

- `models.py` - файл с моделями, бизнес-логика
- `views.py` - контроллеры
- `urls.py` - URL роутер данного приложения
- `templates` - директория с шаблонами
- `management/commands` - консольные команды приложения
- `static` - CSS, JavaScript, картинки
- `migrations` - миграции для обновления базы данных

Конфигурация Django

Конфиг - просто python модуль

```
# project/project/settings.py
ROOT_URLCONF = 'project.urls'
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions'
]
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Пути в конфиге

Проблемы:

- Проект может быть развернут в любой директории
- Несколько копий проекта на одном сервере

Решения:

- Абсолютные пути в каждом конфиге
- Переменные окружения, `$PROJECT_PATH`
- Относительные пути

Относительные пути

```
from pathlib import Path
```

```
# Build paths inside the project like this: BASE_DIR / 'subdir'.
```

```
BASE_DIR = Path(__file__).resolve().parent.parent
```

```
STATIC_ROOT = BASE_DIR / 'static'
```

Паттерн local_settings.py

```
# В конце project/settings.py  
try:  
    from project.local_settings import *  
except ImportError:  
    pass
```

Файл local_settings.py добавляется в .gitignore

Настройка шаблонов

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [  
            BASE_DIR / "templates"  
        ],  
        'APP_DIRS': True,  
        'OPTIONS': {  
            'context_processors': [  
                'django.template.context_processors.debug',  
                'django.template.context_processors.request',  
                'django.contrib.auth.context_processors.auth',  
                'django.contrib.messages.context_processors.messages',  
            ],  
        },  
    },  
]
```

Маршрутизация URL

Порядок поиска контроллера

- Django начинает поиск с файла `ROOT_URLCONF` из настроек
- Загрузив файл, Django использует переменную `urlpatterns`
- Django проходит по всем паттернам до первого совпадения
- Если совпадения не найдено - будет возвращен код

`404 Not Found`

Маршрутизация в проекте

```
# project/project/urls.py
```

```
urlpatterns = [  
    path('', blog.views.home, name='home'),  
    re_path(r'^blog/', include('blog.urls')),  
    re_path(r'^admin/', include('admin.site.urls')),  
]
```

Маршрутизация в приложении

```
# project/blog/urls.py
from blog import views

urlpatterns = [
    path('', views.post_list, name='post-list'),
    path('profile/<int:id>/', views.profile, name='profile-by-id'),
    path('profile/<username>/', views.profile, name='profile'),
    re_path('^category/(\d+)/$', views.category_view,
            name='post-list-by-category'),
    re_path('^(?P<pk>\d+)/$', views.post_detail,
            name='post-detail'),
]
```

Используемые функции

- `urlpatterns` - особая переменная list of path
- `path` - простые именованные параметры
- `re_path` - именованные параметры по регулярному выражению
- `include` - включение одного urls.py внутрь другого

Особенности маршрутизации в Django

- Слеш (/) в начале роутов не указывается
- Роуты описываются с помощью регулярных выражений и паттернов
- Можно и нужно разносить роуты по приложениям
- Можно и нужно создавать именованные роуты
- Одно действие – один роут – один контроллер

Reverse routing

В python коде:

```
from django.core.urlresolvers import reverse
reverse('home')
reverse('post-list-by-category', args=(10,))
reverse('post-detail', kwargs={'pk': 7})
```

В шаблоне:

```
{% url 'profile-by-id' user.id %}
```