ТЕХНОАТОМ

Веб-программирование Python

Лекция 4. Интернет и веб-приложения

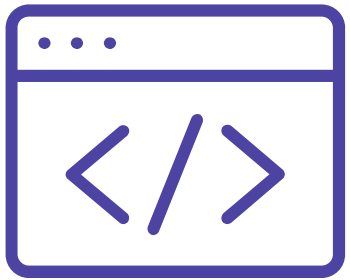
Михалев Олег

Сегодня

- WSGI-приложения и middleware-компоненты

- Трехуровневая архитектура веб-приложений
- Шаблон разработки MVC

- Фреймворки Pyramid, Flask и Falcon
- Фреймворк Django
- Первое веб-приложение на Django



ТЕХНОАТОМ

WSGI-приложения

Вызываемый объект, который принимает два аргумента:

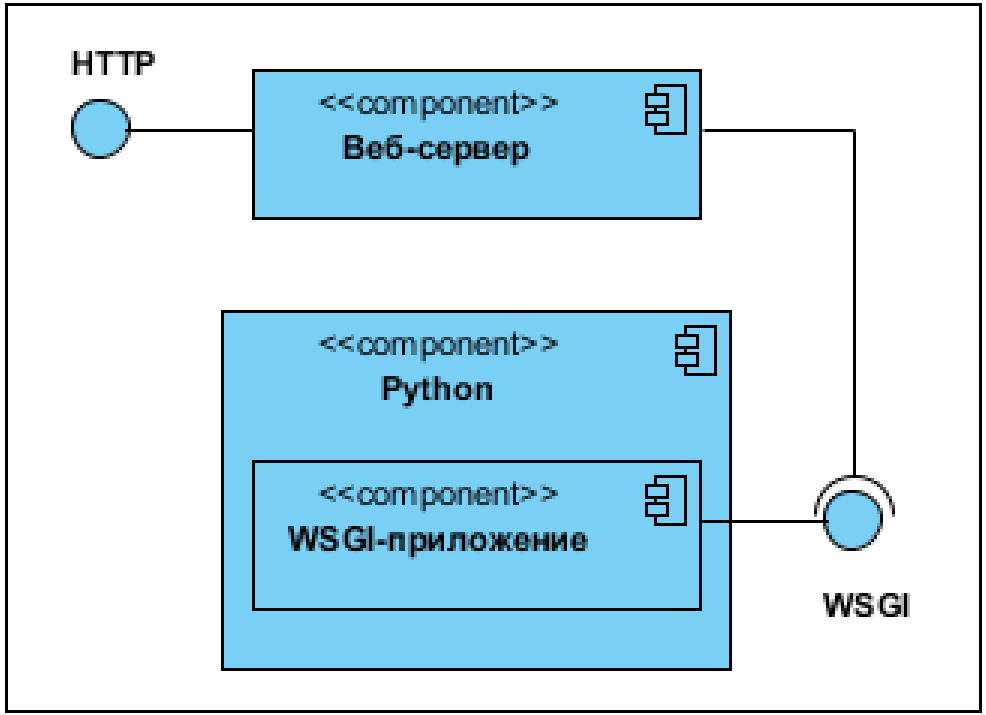
- словарь переменных окружения **environ**
- обработчик запроса **start_response**

Вызывает обработчик запроса **start_response**
Возвращает итерируемый объект с бинарными строками

Простое WSGI-приложение

```
1. def simple_wsgi_application(environ, start_response):
2.     response_status = '200 OK'
3.     response_headers = [('Content-type', 'text/plain')]
4.     start_response(response_status, response_headers)
5.     yield b'Hello, World!/n'
```

Введение


HTTP
<<component>>
Веб-сервер
<<component>>
Python
<<component>>
WSGI-приложение
WSGI

Специфичные переменные окружения

'wsgi.url_scheme'	Схема (протокол) веб-сервера
'wsgi.input'	Входящий поток запроса
'wsgi.errors'	Поток ошибок
'wsgi.version'	Версия интерфейса

Специфичные переменные окружения

'wsgi.multithread'
Флаг, информирующий о запуске приложения в несколько потоков

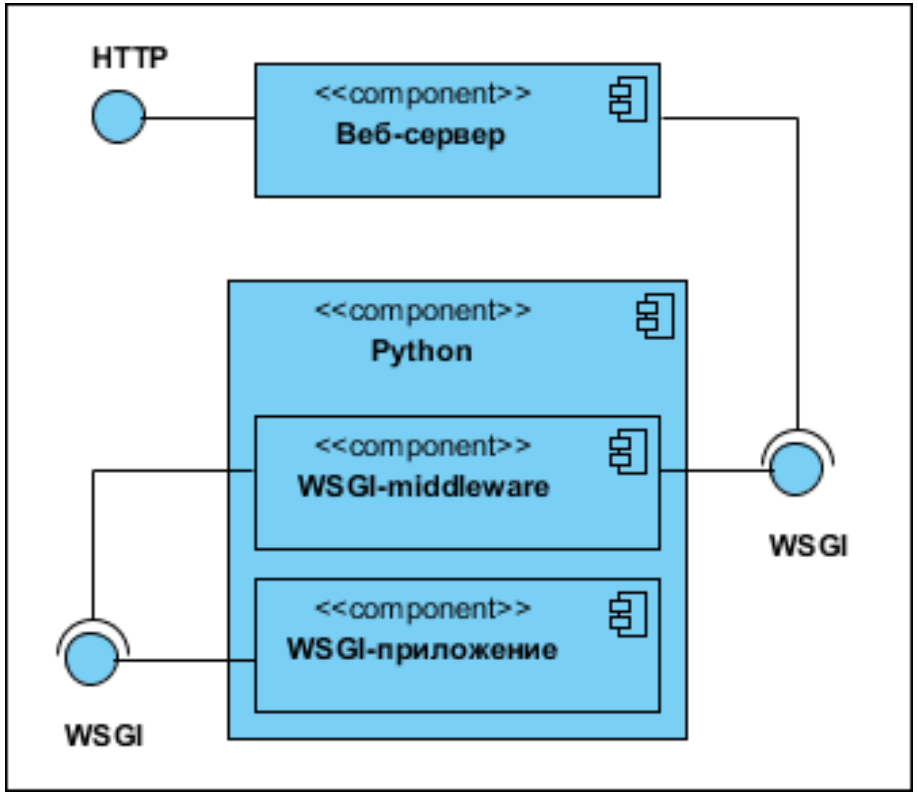
'wsgi.multiprocess'
Флаг, информирующий о запуске приложения в несколько процессов

Семинар №2

- Потоки и процессы
- Python GIL
- Библиотека threading
- Библиотека multiprocessing
- Параллельная обработка данных в WSGI-приложениях

Дополнительную обработку входящих и исходящих от WSGI-приложения данных можно включать посредством middleware-компонентов, включая их в цепочку между веб-сервером и приложением.

WSGI-middleware



WSGI-middleware

TEXHOATOM

```
1.  def wsgi_middleware(wsgi_application):
2.      def application_wrapper(environ, start_response):
3.          print('Call application')
4.          print('Pre-process input arguments (skip)')
5.          print('Wrap start_response')
6.          start_response = wsgi_response_middleware(start_response)
7.          results = wsgi_application(environ, start_response)
8.          if results:
9.              results = list(results)
10.             print('Post-process results (skip)')
11.             yield from results
12.         print('Done application')
13.     return application_wrapper
```

WSGI-middleware

TEXHOATOM

```
1. def wsgi_response_middleware(start_response):
2.     def start_response_wrapper(status, response_headers, exc_info=None):
3.         print('Call start_response')
4.         print('Post-process response status and headers (skip)')
5.         if exc_info:
6.             print('Handle errors (skip)')
7.         start_response(status, response_headers, exc_info)
8.         print('Done start_response')
9.     return start_response_wrapper
```

Применение middleware-компонентов

- Обработка ошибок
- Профилирование (сбор статистики)
- Аутентификация/авторизация и сеанс пользователя
- Балансировка нагрузки
- Обработка данных (сжатие, шифрование)
- Кэширование
- Маршрутизация

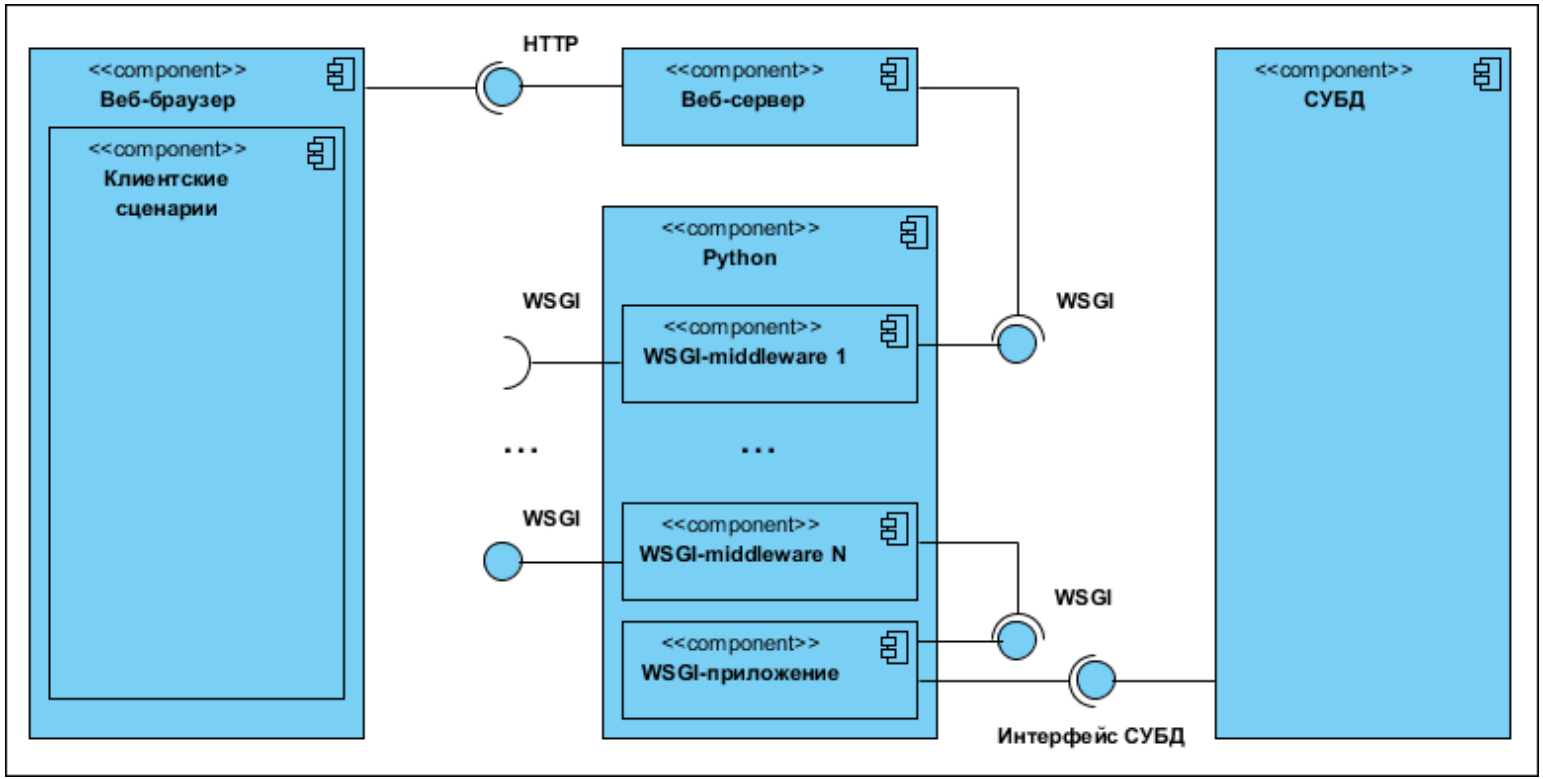
WSGI-middleware

TEXHOATOM

```
1.  def wsgi_stats_middleware(wsgi_application):
2.      def application_wrapper(environ, start_response):
3.          start_time = time()
4.          results = wsgi_application(environ, start_response)
5.          if results:
6.              yield from results
7.          execution_time = time() - start_time
8.          print('Execution time: %s' % execution_time)
9.      return application_wrapper

10. @wsgi_stats_middleware
11. def simple_wsgi_application(environ, start_response):
12.     start_response('200 OK', [('Content-type', 'text/plain; charset=utf-8')])
13.     sleep(randint(0, 10))
14.     yield 'This is wrapped!/n'.encode('utf-8')
```

WSGI-middleware



ТЕХНОАТОМ

Системная архитектура - организация и структура распределения элементов информационной системы.

Трехуровневая архитектура

ТЕХНОАТОМ

Трехуровневая архитектура предполагает наличие в программном комплексе трех уровней:

- Слой представления
- Слой логики
- Слой данных

Слой представления

Интерфейс и клиентские сценарии
(взаимодействие с пользователем)

ТЕХНОАТОМ

Слой логики

Веб-приложение
(правила и алгоритмы обработки данных)

ТЕХНОАТОМ

Слой данных

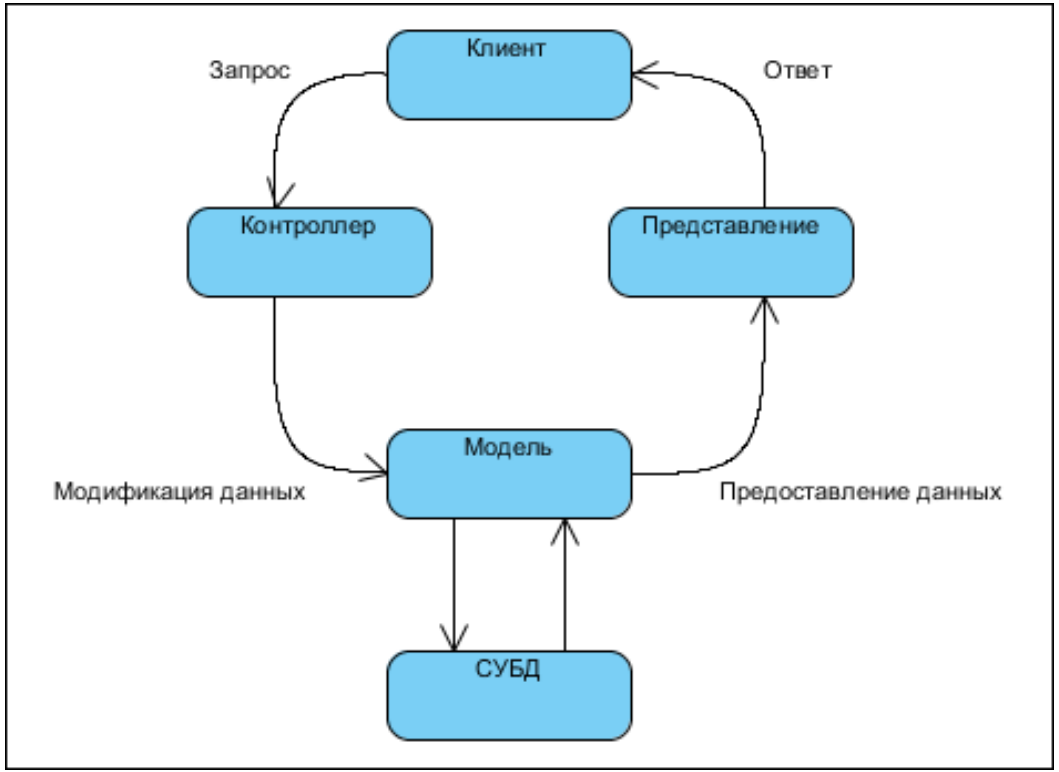
СУБД (хранение, выборка и изменение данных)

Слой логики имеет связи как со слоем представления
(WSGI-интерфейс, HTTP-протокол),
так и со слоем данных
(протоколы взаимодействия с СУБД)

Model-View-Controller (MVC)

модель-представление-контроллер

MVC


Клиент
Запрос
Ответ
Контроллер
Представление
Модель
Модификация данных
Предоставление данных
СУБД

Контроллер несет ответственность за обработку запросов пользователя и вызов соответствующих ресурсов. Его основная функция — вызывать и координировать действия.

Представление формирует ответ в конкретном формате. Его основная функция - наглядно отобразить данные модели для конечного пользователя.

Модель отвечает за данные и правила их обработки.
Его функция - интеграция со слоем данных.

Модель предоставляет контроллеру программные интерфейсы для чтения и модификации данных.

Данные модели используются представлениями при формировании веб-страницы.

Модель «Студент»

- Поле «Имя»
- Поле «Оценка»

Представление «Профиль студента»

- Поля модели «Студент»

Контроллер «Студенты»

- Действие «Просмотреть профиль»
  - Прочитать модель «Студент»
  - Вернуть представление «Профиль студента»
- Действие «Поставить оценку»
  - Прочитать модель «Студент»
  - Изменить поле «Оценка» модели «Студент»
  - Вернуть представление «Профиль студента»

Давайте придумаем свой пример

Итого

- Веб-сервер
- Python
  - Middleware-стек
  - Приложение
    - Контроллеры
    - Модели и представления
- СУБД

Программный комплекс не так уж прост

Фреймворки по сути - программная платформа, но не просто набор готовых компонент, а нечто диктующее правила построения архитектуры приложения. Частная логика при этом встраивается в общую структуру.

Pyramid
http://www.pylonsproject.org/
http://docs.pylonsproject.org/projects/pyramid/en/latest/

Flask
http://flask.pocoo.org/
http://flask.pocoo.org/docs/

Falcon
https://falconframework.org/
https://falcon.readthedocs.io/en/stable/

За бортом остались:

Tornado
http://www.tornadoweb.org/en/stable/

Twisted
https://twistedmatrix.com/trac/

Asyncio
https://docs.python.org/dev/library/asyncio.html

Pyramid - минималистичный веб-фреймворк, построенный на принципах WSGI и middleware-компонент.

Pyramid отрицает MVC в принципе, но является очень гибким инструментом.

```
1.  from wsgiref.simple_server import make_server
2.  from pyramid.config import Configurator
3.  from pyramid.response import Response

4.  def hello_world(request):
5.      return Response('<h1>Hello, World!</h1>')

6.  if __name__ == '__main__':
7.      config = Configurator()
8.      config.add_route('hello', '')
9.      config.add_view(hello_world, route_name='hello')
10.     application = config.make_wsgi_app()
11.     server = make_server('', 80, application)
12.     server.serve_forever()
```

Flask - очень минималистичный веб-фреймворк без моделей (они предоставляются сторонними компонентами фреймворка).

Flask идеально подходит для разработки простых сайтов.

```
1. from flask import Flask
2. app = Flask(__name__)

3. @app.route('/')
4. def hello_world():
5.     return '<h1>Hello, World!</h1>'

6. if __name__ == '__main__':
7.     app.run('', 80)
```

ТЕХНОАТОМ

Falcon - высокопроизводительный веб-фреймворк для API.

```
1.  from wsgiref.simple_server import make_server
2.  import falcon

3.  class HelloWorldResource(object):
4.      def on_get(self, request, response):
5.          response.status = falcon.HTTP_200
6.          response.append_header('Content-Type', 'text/html')
7.          response.body = '<h1>Hello, World!</h1>'

8.  if __name__ == '__main__':
9.      application = falcon.API()
10.     application.add_route('/', HelloWorldResource())
11.     server = make_server('', 80, application)
12.     server.serve_forever()
```

Django

https://www.djangoproject.com/
https://docs.djangoproject.com/en/
http://djangobook.com/

Фреймворк **Django** соответствует шаблону MVC (с оговорками), написан целиком на Python.

Модели Django = Модели MVC
Шаблоны Django = Представления MVC
Представления Django = Контроллеры MVC

Почему мы выбираем Django?

- Содержит все необходимое (совсем все и еще чуть чуть)
- Распространен и часто применяем
- Прост в обращении

Создание Django-проекта

1. django-admin startproject myfinance

Внешняя директория **myfinance** - контейнер вашего проекта.

manage.py	скрипт для управления проектом
myfinance	внутренняя директория проекта

Внутренняя директория **myfinance** - модуль вашего проекта.

settings.py	настройки проекта
urls.py	маршруты проекта
wsgi.py	точка входа для WSGI-интерфейса

Создание Django-приложения

1. python manage.py startapp finance

Директория **finance** - модуль приложения вашего приложения.

apps.py	настройки приложения
admin.py	модуль для подключения к интерфейсу admin
tests.py	модуль для тестирования приложения
models.py	модуль для моделей приложения
views.py	модуль для представлений приложения

Опишем представление

```
1. from django.http import HttpResponse

2. def hello_world(request):
3.     return HttpResponse('<h1>Hello, World!</h1>')
```

Команда **startapp** может не создать в приложении **urls.py**, однако зачастую удобно создавать отдельные маршруты для приложения.

Опишем маршруты Django-приложения

```
1. from django.conf.urls import url
2. from finance.views import hello_world

3. urlpatterns = [
4.     url(r'.*', hello_world),
5. ]
```

Проинсталлируем приложение в **settings.py** проекта

```
1. INSTALLED_APPS = [
2.     'django.contrib.admin',
3.     'django.contrib.auth',
4.     'django.contrib.contenttypes',
5.     'django.contrib.sessions',
6.     'django.contrib.messages',
7.     'django.contrib.staticfiles',
8.     'finance',
9. ]
```

Проинсталлируем маршруты в **urls.py** проекта

```
1. from django.conf.urls import url, include
2. from django.contrib import admin

3. urlpatterns = [
4.     url(r'^admin/', admin.site.urls),
5.     url(r'^', include('hello_world.urls')),
6. ]
```

Запустим проект

1. python manage.py runserver 127.0.0.1:80

Спасибо за внимание!

Михалев Олег
mailto:mhalairt@gmail.com

Домашнее задание

ТЕХНОАТОМ

Создать Django-проект **myfinance** и Django-приложение **finance**.

Опеределить два маршрута:

- Главная страница (корень r'^$')
- Выписка по счету (например, r'^/charges/$')

Главная страница должна содержать приветствие и ссылку на страницу выписки по счету. Страница выписки по счету должна содержать произвольную таблицу и ссылку на главную страницу.

Полезные ссылки

ТЕХНОАТОМ

Примеры из лекции доступны по ссылке
https://drive.google.com/open?id=0B3RxNTHeHOTKdEpxQ1ZyS3lRVVE

HTML

http://htmlbook.ru/samhtml
http://www.w3schools.com/html/
https://www.w3.org/TR/html5/