Application Server

Теоретическая часть

- Паттерн MVC;
- Фреймворк Flask;
- Запросы, обработчики, маршруты;
- Обработка параметров запроса;
- Генерация ответов;
- Работа с заголовками;
- GET, POST и другие методы запросов.

Практическая часть

- Простой Flask проект;
- Реализация заглушек API.

Flashback


Web Server Vs Application Server

Backend (application) сервер

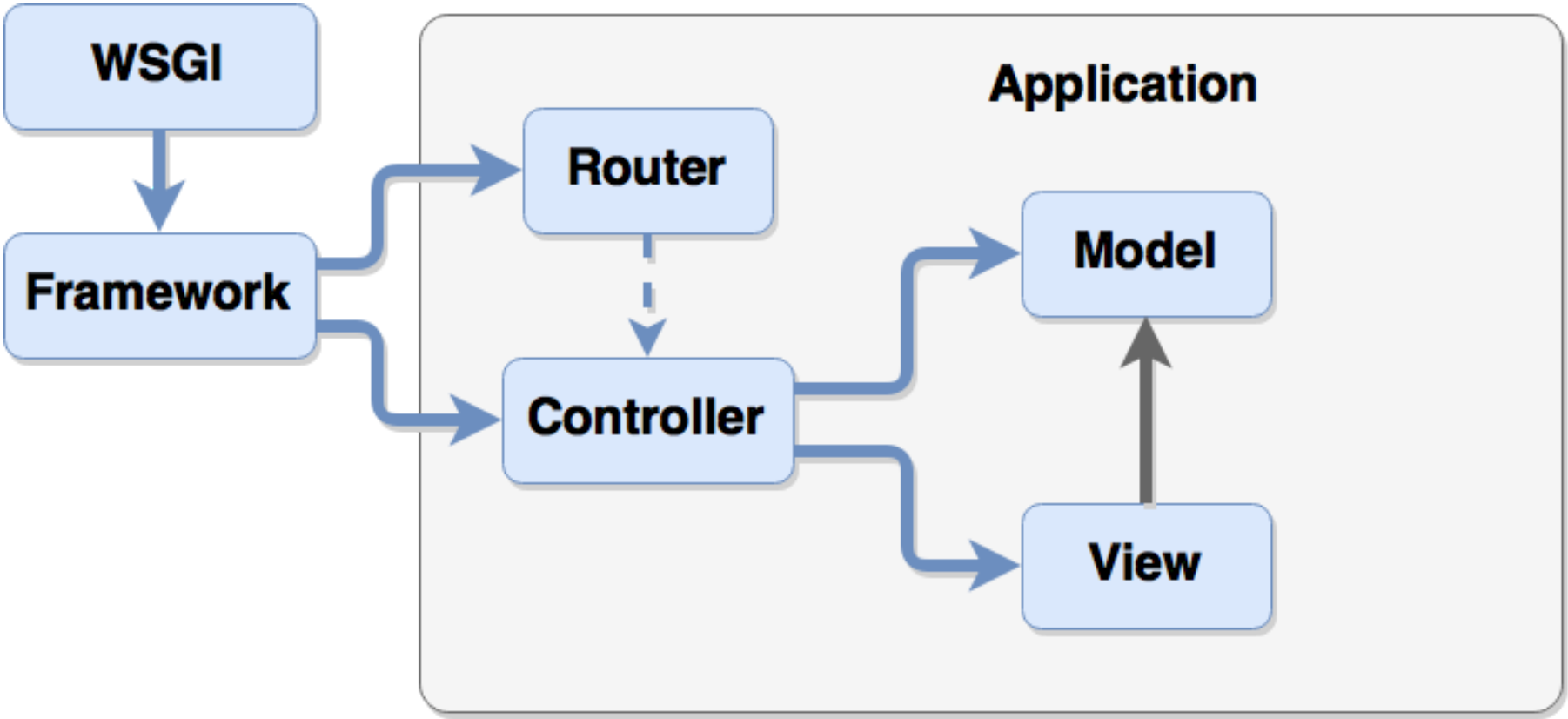
- Роль application сервера заключается в исполнении бизнес-логики приложения и генерации динамических документов;
- На каждый HTTP запрос application сервер запускает некоторый обработчик в приложении. Это может быть функция, класс или программа, в зависимости от технологии.

Паттерн MVC

Паттерн MVC

Схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента:

- Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние;
- Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели;
- Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений.


WSGI
Framework
Application
Router
Controller
Model
View

Соглашение о именовании

MVC	**Flask**
Model	Model
Router	Декоратор flask.Flask.route()
Controller	views
View	templates

Фреймворк Flask

Flask

Набор минималистичных каркасов веб-приложений, сознательно предоставляющих лишь самые базовые возможности.

- Разработан в 2010 году австрийским разработчиком Армином Ронахером;
- Язык программирования Python;
- Набор инструментов Werkzeug;
- SQLAlchemy;
- Шаблонизатор Jinja2.

Flask vs. Django

- Flask реализуется с минимальными надстройками, которые всецело предоставлены аддонам или разработчику;
- Django следует философии «все включено», и даёт большой ассортимент для работы;
- У Flask нет собственной ORM, поэтому обычно подключается библиотека SQLAlchemy;
- У Django чёткая стурктура проекта, у Flask'а --- нет.

SQLAlchemy

- Библиотека для работы с базами данных при помощи языка SQL;
- Реализует технологию программирования ORM;

Blueprint

Концепция для создания компонентов приложений и поддержки общих шаблонов внутри приложения или между приложениями.

```
from flask import Blueprint, render_template, abort
from jinja2 import TemplateNotFound

simple_page = Blueprint('simple_page', __name__,
                        template_folder='templates')

@simple_page.route('/<page>')
def show(page):
    return render_template('pages/%s.html' % page)
```

Jinja2

```
<title>{% block title %}{% endblock %}</title>
<ul>
{% for user in users %}
    <li><a href="{{ user.url }}">{{ user.username }}</a></li>
{% endfor %}
</ul>
```

Werkzeug

Набор инструментов WSGI, стандартного интерфейса Python для развертывания веб-приложений и взаимодействия между ними и различными серверами разработки.

Структура Flask проекта

```
config.py
requirements.txt
run.py
instance/
    config.py  # Добавляем в .gitignore
app/
    __init__.py
    views.py
    models.py
    forms.py
    static/
    templates/
```

Запросы, обработчики, маршруты

Контекст запроса

- **request** - данные поступившего запроса;

```
from flask import request

# Инициализация приложения

@app.before_request
def before_request():
    g.db = connect_db()

@app.teardown_request
def teardown_request(exception):
```

Маршрутизация

Три способа определить правила для маршрутизации:

- Декоратор flask.Flask.route();
- Функция flask.Flask.add_url_rule();
- Напрямую обратиться к Werkzeug через flask.Flask.url_map

Маршрутизация

```
from flask import Flask
app = Flask(__name__)
# ...
def index():
    if request.method == 'OPTIONS':
        # custom options handling here
        ...
    return 'Hello World!'
index.provide_automatic_options = False
index.methods = ['GET', 'OPTIONS']

app.add_url_rule('/', index)
```

Маршрутизация

Декоратор flask.Flask.route() используется для связывания функций с URL

```
from flask import Flask
app = Flask(__name__)
# ...
@app.route('/')
def index():
    return "Index page"

@app.route('/hello')
def hello():
    return 'Hello, World'
```

Использование переменных

Для добавления переменной части в URL вы можете пометить эти разделы, как **<variable_name>**

```
@app.route('/user/<username>')
def show_user_profile(username):
    # show the user profile for that user
    return 'User {}'.format( username )

@app.route('/post/<int:post_id>')
def show_post(post_id):
    # show the post with the given id, the id is an integer
    return 'Post {}'.format( post_id )
```

Типы переменных

- **string** (по умолчанию) - принимает любой текст без слешей;
- **int** - принимает целый числа;
- **float** - принимает вещественные числа;
- **path** - как и string, но принимает слеши;
- **any** - любой аргумент из списка;
- **uuid** - принимает UUID строки.

```
from flask import Flask, url_for, redirect

# Инициализация приложения app.
@app.route('/')
@app.route(/<string:name>)
def index(name="Gustavo"):
    return "Index page, {}".format(name)

@app.route('/blog/'):
def blog():
    redirect( url_for('index', name="Mike") )
```

Обработка параметров запроса

Параметры запроса хранятся в request, но в разных переменных:

- **request.args** - параметры строки запроса (всё что после ? name1=value1&name2=value2);
- **request.form** - параметры POST запроса формы (тип multipart/form-data);
- **request.get_json()** - параметры POST запроса (использование application/json);

```
# Идём по урлу /?name=Jack
@app.route('/')
def index():
    name = request.args.get('name', "Default")
    return name
```

Генерация ответов

```python
from flask import Response, jsonify, json

@app.route('/hello', methods = ['GET'])
def api_hello():
    data = {
        'hello'  : 'world',
        'number' : 3
    }
    js = json.dumps(data)

    resp = Response(js, status=200, mimetype='application/json')
    # или использовать этот код
    # resp = jsonify(data)
    # resp.status_code = 200
    resp.headers['Link'] = 'http://luisrei.com'

    return resp
```

Работа с заголовками

Заголовки запроса можно найти в **request.headers**

```
@app.route('/api/messages', methods = ['POST'])
def api_message():

    if request.headers['Content-Type'] == 'text/plain':
        return "Text Message: " + request.data

    elif request.headers['Content-Type'] == 'application/json':
        return "JSON Message: " + json.dumps(request.json)

    # Обрабатываем другие заголовки

    else:
        abort(415)
```

GET, POST и другие методы запросов

HTTP методы

- **GET** - получение документа;
- **HEAD** - получение только заголовков;
- **POST** - отправка данных на сервер;
- PUT - отправка документа на сервер.

Передаём методы в параметре methods для декоратора route.

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        return do_the_login()
    else:
        return show_the_login_form()
```

Материалы

- Документация Flask

Домашнее задание

- Создать и запустить Flask проект (3 балла)
- Реализовать "заглушки" для всех методов API, используя json.dumps (4 балла)
- Обрабтывать только нужные методы (GET/POST) (1 балл)
- Написать тесты (pytest,pytest-flask) проверяющие работу (достаточно проверить) (3 балла)

Срок сдачи: следующее занятие.

Спасибо за внимание!