ЛАБОРАТОРНАЯ РАБОТА №5
MVC. FLASK. JINJA2

Цель работы: получить навык разработки веб-приложений с помощью микрофреймворка Flask

Задачи:

1. Разработать серверную часть веб-приложения с помощью микрофреймворка Flask.
2. Разработать шаблоны страниц с помощью шаблонизатора Jinja2.
3. Реализовать динамическую загрузку данных при помощи AJAX

Результатами работы являются:

1. Разработанный сайт, содержащий python, JS и CSS файлы.
2. Подготовленный отчет.

MVC

Шаблон проектирования MVC предполагает разделение данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: Модель, Представление и Контроллер – таким образом, что модификация каждого компонента может осуществляться независимо:

1. Модель — этот компонент отвечает за данные, а также определяет структуру приложения. Например, если вы создаете To-Do приложение, код компонента model будет определять список задач и отдельные задачи.

2. Представление — этот компонент отвечает за взаимодействие с пользователем. То есть код компонента view определяет внешний вид приложения и способы его использования.

3. Контроллер — этот компонент отвечает за связь между model и view. Код компонента controller определяет, как сайт реагирует на действия пользователя.

Существует множеств MVC-фреймворков (и микрофреймворков), одним из которых является Python-микрофреймворк Flask.

FLASK

В самом простом виде (без модели) пример Flask-приложения имеет следующий вид:

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return 'index'

app.run(debug = True,  host='127.0.0.1', port='5050')
```

После импорта необходимых модулей необходимо создать экземпляр Flask-приложения. После этого в приложении можно регистрировать контроллеры, т.е. те функции, которые необходимо будет выполнить приложению по соответствующим URL запросов клиента. Контроллеры могут возвращать как конкретные значения, так и HTML-разметку страниц. После ригастрации всех контроллеров необходимо запустить приложение, указав параметрами диапазон IP-адресов, с которых приложение будет прослушивать запросы и номер порта приложения (по умолчанию 5000). Хотя, разумеется, контроллеры нужно выносить в отдельные файлы и, как будет рассмотрено далее, разбивать приложение на blueprint'ы.

В качестве URL контроллерам можно указывать изменяемые значения:

```
@app.route('/user/<username>')
def show_user_profile(username):
    return 'User %s' % escape(username)

@app.route('/post/<int:post_id>')
def show_post(post_id):
    return 'Post %d' % post_id

@app.route('/path/<path:subpath>')
def show_subpath(subpath):
    return 'Subpath %s' % escape(subpath)
```

Значения будут подставлены «на лету» по запросу клиента, к полученным от клиента данным можно обращаться как обычным переменным внутри контроллера. Для более гибкой обработки запросов клиентов можно перегрузить контроллерв, указав ожидаемый тип входного знчения окончания URL. Применяемый типы: string, int, float, path, uuid.

Контроллер может вернуть до 3 значений: данные, код HTTP ответа, HTTP-сообщение с заголовками (2 последних можно опустить и Flask вернеь значения по умолчанию). Для возвращения данных в формате json (например, для AJAX-запросов) нужно воспользоваться функцией

jsonify – эта функция создаст json из данных переданных параметром (данные должны быть сериализуемы!), кроме того, при формировании HTTP-ответа, добавит заголовок mimetype='application/json'. Для управления заголовками также можно вручную создать экземпляр HTTP-ответа с помощью метода make_response, установить необходимые заголовки, задать возвращаемое значение, код ответа и вернуть этот объект контроллером.

Если необходимо вернуть HTTP-ошибку, то в любом месте контроллера можно вызвать метод abort с параметром – кодом ошибки.

Контроллер может обрабатывать несколько типов HTTP-запросов (по умолчанию GET-запрос), для этого при создании контроллера необходимо указать список доступных HTTP-методов, в теле контроллера можно получить тип текущего запроса через объект request:

```
from flask import request

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

Помимо HTTP-метода, используя объект request, можно получить и другую полезную информацию:

- method – HTTP-метод
- form – словарь значений, переданных в теле POST-запроса
- args – словарь параметров GET-запроса
- files – список вложенных файлов
- url – полный URL запроса
- cookies – список cookies, переданных вместе с запросом

По умолчанию Flask хранит значения сессий на клиенте в зашифрованном виде (для альтернативных решений необходимо использовать модуль Flask-Session). Для шифрования данных

приложению до запуска необходимо указать секретный ключ шифрования: app.secret_key = b'_5#y2L"F4Q8z\n\xec]/’

Для работы с сессиями во Flask нужно использовать словать session:

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
          session['username'] =
                request.form['username']
          return redirect(url_for('index'))
    return '''<form method="post" action="login">
                <input type=text name=username>
                <input type="submit" value="Submit">
          </form>'''

@app.route('/logout')
def logout():
    session.pop('username', None)
    return redirect(url_for('index'))
```

В данном примере на GET-запрос по URL 'login' будет возвращен HTML-содержащий форму с полем для ввода имени и кнопкой отправки формы. После отправки формы по URL 'login' придет POST-запрос, обработчик которого установит значению username в сессии полученное значение и с помощью функции redirect перенаправит запрос контроллеру, отрабатывающему URL 'index'. Для удаления сессии достаточно удалить значения из словаря session с помощью метода pop. Для работы с пользователями можно использовать дополнительный модуль Flask-Login.

В данном примере использовалась функция url_for, позволяющая построить полный URL для текущего сервера, что часто используется для обработки статических файлов. Для этого первым параметром указывается ключевое значение 'static', а затем относительный путь к файлу в папке 'static' (по умолчанию в корне проекта, но при запуске приложения можно указать любую доступную директорию парметром static).

Jinja2 во Flask

Jinja2 – это шаблонизатор, позволяющий на основе подготовленных шаблонов формировать текстовые данные. Данный шаблонизатор широко применяется в микрофреймворке Flask для генерации HTML-страниц контроллерами на основе заранее подготовленных шаблонов, для этого нужно воспользоваться функцией render_template. В качестве параметров указав имя шаблона и значениядля переменных, применяемых в шаблоне. Все шаблоны будут искаться в директории templates (по умолчанию в корне проекта, но при запуске приложения можно изменить на произвольную доступную директорию параметром templates).

Возможные конструкции в шаблоне:

- {{ }} – переменные, выражения и вызовы функций
- {# комментарий #} – комментарии
- {% set fruit = 'apple' %} – объявление переменных
- {% if user.newbie %}
  Display newbie stages
  {% elif user.pro %}
  Display pro stages
  {% elif user.ninja %}
  Display ninja stages
  {% else %}
  You have completed all stages
  {% endif %} – условный оператор
- {% for user in user_list %}
  {{ user }}
  {% else %}
  user_list is empty
  {% endfor %} – цикл. Цикл for предоставляет специальную переменную loop для отслеживания прогресса цикла

Пример:

```
@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    return render_template('hello.html', name=name)
```

```
//hello.html
<!doctype html>
<title>Hello from Flask</title>
{% if name %}
    <h1>Hello {{ name }}!</h1>
{% else %}
    <h1>Hello, World!</h1>
{% endif %}
```

В примере контроллер будет возвращать HTML, сформированный на основе шаблона 'hello.html'. В случае, если name не передан параметром (None по умолчанию) по шаблону сформируется страница, содержащая заголовок 'Hello, World!', иначе контроллер передаст шаблонизатору значение name и сформируется страница, содержащая заголовок с приветствие конкртеного пользователя.

В шаблонах можно применять фильтры, изменяющие переменные до процесса рендеринга. Синтаксис использования фильтров следующий: variable_or_value|filter_name.

- upper –делает все символы заглавными
- lower – приводит все символы к нижнему регистру
- capitalize – делает заглавной первую букву и приводит остальные к нижнему регистру
- escape – экранирует значение
- safe – предотвращает экранирование
- length – возвращает количество элементов в последовательности
- trim – удаляет пустые символы в начале и в конце
- random – возвращает случайный элемент последовательности

Для повышения повторного использования кода, можно использовать вложенные шаблоны:

```
<nav>
    <a href="/home">Home</a>
    <a href="/blog">Blog</a>
    <a href="/contact">Contact</a>
</nav>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    {% include 'nav.html' %}
</body>
</html>
```

Или наследовать шаблоны, для этого в родительском шаблоне определяются блоки, которые могут быть переопределены в шаблонах-наследниках:

```
<body>
    {% block content %}
    {% endblock %}
</body>

{% extends 'base.html' %}
{% block content %}
    {% for bookmark in bookmarks %}
        <p>{{ bookmark.title }}</p>
    {% endfor %}
{% endblock %}
```

Если необходимо дописать что-либо в блок, определенный родителем, можно переобпределить блок и вызвать содержимое родителя с помощью метода {{ super() }}.

Blueprints

Blueprints — способ организации Flask-приложений. Эскиз может иметь собственные функции представления, шаблоны и статические файлы, для них можно выбрать собственные URI.

Основная концепция blueprint’ов заключается в том, что они записывают операции для выполнения при регистрации в приложении. Flask связывает функции представлений с blueprint’ами при обработке запросов и генерировании URL’ов от одной конечной точки к другой.

Пример использования blueprint’ов:

```
//создание blueprint’a со своими шаблонами
//и контроллерами
from flask import Blueprint, render_template, abort
from jinja2 import TemplateNotFound

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

@auth_bp.route('/', defaults={'page': 'index'})
@auth_bp.route('/<page>')
def show(page):
    try:
        return render_template('pages/%s.html' % page)
    except TemplateNotFound:
        abort(404)

//регистрация blueprint’a в приложении
from flask import Flask, render_template
from flask import Blueprint, render_template, abort
from auth.auth import auth_bp

app = Flask(__name__)
app.register_blueprint(auth_bp)
app.run(debug = True,  host='127.0.0.1', port='5050')
```

Фабрика приложений

В ситуациях, требующих создания различных инстансов приложения с различными конфигурациями (например, при тестировании), полезным будет создание фабрики приложений – функции, создающей и возвращающий экземпляр приложения нужной конфигурации. Пример:

```
def create_app(config):
    app = Flask(__name__)
    app.config.from_object(config)
    mail.init_app(app)

    from .main import main as main_blueprint
    app.register_blueprint(main_blueprint)

    from .admin import main as admin_blueprint
    app.register_blueprint(admin_blueprint)

    return app
```

Таким образом, несколько файлов (например, run.py или test.py) могут создавать свой конфигурационный объект каждый и, вызывая фабрику приложений, получать свой уникальный экземпляр приложения. Обобщая все вышеуказанное, получим следующую структуру приложения:

```
/
    app/
        blueprint1/
            templates/
            static/
            __init__.py
            blueprint1.py
        static/
        templates/
        model.py
        __init__.py
    run.py
    test.py
```

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Разработать сайт тематики, согласно полученному варианту, используя паттерн MVC и микрофреймворк Flask.

Python-приложение должно быть запущенно в индивидуальном виртуальном окружении.

При разработке шаблонов необходимо использовать наследование: на страницах должны быть header, footer и панель навигации.

Как минимум на одной из страниц должна быть отображена таблица, содержащая информацию, хранимую на сервере в отдельном текстовом файле (возможно использование формата xml или json). На сайте должна быть предусмотрена возможность добавления новых данных (с сохранением в файл).

Загрузка и передача данных должна осуществляться с помощью AJAX.

Возможно применение CSS-фреймворка Bootstrap

ВАРИАНТЫ ЗАДАНИЙ

1. Сайт школы
2. Сайт фитнес-центра
3. Новостной сайт
4. Метеорологический сайт
5. Сайт музыкального исполнителя
6. Сайт кинотеатра
7. Сайт отеля
8. Сайт аэропорта
9. Сайт телеканала
10. Сайт радиостанции
11. Сайт автосалона
12. Сайт ресторана
13. Сайт университета
14. Сайт библиотеки
15. Сайт театра

16. Сайт ветеренарной клиники
17. Сайт туристической фирмы
18. Сайт агентства по продаже недвижимости
19. Сайт букмекерской компании
20. Сайт биржи криптовалют

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Раскройте суть паттерна MVC.
2. Раскройте назначение метода make_response.
3. Перечислите информацию, содержащуюся в объекте request.
4. Опишите механизм использования шаблонов Jinja2
5. Приведите способы генерирования HTTP-ответа с кодом ошибки.
6. Раскройте суть использования фильтров в Jinja2.
7. Опишите механизм наследования шаблонов.
8. Опишите механизм применения вложенных шаблонов.
9. Раскройте суть применения blueprint’ов.
10. Раскройте суть фабрики приложений.

ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

На выполнение лабораторной работы отводится 5 часов: 4 часа на выполнение работы, 1 час на подготовку и защиту отчета по лабораторной работе.

Структура отчета: титульный лист, цель и задачи, формулировка задания (вариант), этапы выполнения работы, исходный код разработанного сайта, результаты выполнения работы (скриншоты), выводы.