1 Знакомство с документацией REST API ө https://starkovden.github.io/intro-rest-api.html

REST API: REST (Representational State Transfer) - это веб-сервисы, которые позволяют отправлять запросы к ресурсам по URL-путям. Указывается операция, которую необходимо выполнить, с помощью пути (например, GET, CREATE, DELETE). Как и в случае с другими API веб-служб, запросы и ответы передаются через HTTP через Интернет, и серверы, принимающие запросы, не зависят от языка запроса (необязательно, чтобы он был определенным языком программирования). Ответы обычно возвращаются в формате JSON или XML. API REST имеют много разных путей (конечных точек) с различными параметрами, которые можно настраивать для определения желаемых результатов.

Free APIs
https://apilist.fun/collection/free-apis

2 Flask Web Development

Links
Documentation: https://flask.palletsprojects.com/
Changes: https://flask.palletsprojects.com/changes/
PyPI Releases: https://pypi.org/project/Flask/

Install and update using pip:

```
$ pip install -U Flask
```

A Simple Example

```
# save this as app.py
from flask import Flask

app = Flask(__name__)

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

```
$ flask run
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

2.1 Инициализация

Все программы Flask должны **Создать экземпляр программы**, Экземпляр программы является объектом класса Flask.

Веб-сервер использует протокол, называемый интерфейсом веб-сервера (WSGI), для пересылки всех запросов, полученных от клиента, к этому объекту для обработки. Он часто создается с использованием следующего кода:

```
from flask import Flask
app = Flask(__name__)
```

Конструктор класса Flask имеет только один параметр, который должен быть указан, а именно имя основного модуля или пакета программы __name__. Если вы не передадите целочисленное значение, вы получите сообщение об ошибке и не сможете ввести стандартное имя модуля: это повлияет на доступ к статическим файлам и не повлияет на доступ к функциям просмотра.

2.2 Маршрутизация и просмотр функций

Клиент (например, веб-браузер) отправляет запрос веб-серверу, а веб-сервер отправляет запрос экземпляру программы Flask. Экземпляр программы должен знать, какой код запускать для каждого запроса URL-адреса, поэтому он сохраняет отношение отображения URL-адреса к функции Python.

Маршрутизация: Программа, которая обрабатывает отношения между URL и функциями.

В программе Flask используйте декоратор app.route, предоставленный экземпляром программы, чтобы определить маршрут, и зарегистрируйте декорированную функцию как маршрут. В следующем примере показано, как использовать этот декоратор для объявления маршрута:

```
@app.route('/')
def index():
return '<h1>Hello World!</h1>'
```

Просмотр функций и ответов: Предыдущий пример регистрирует функцию index () в качестве обработчика корневого адреса программы. Когда браузер обращается к серверу, на котором развернута программа, он запускает сервер для выполнения функции index (). Возвращаемое значение этой функции называется**ответ**Является ли контент, полученный клиентом. Если клиент является веб-браузером, ответом является документ, отображаемый пользователю для просмотра. Функции как index () называются**Функция просмотра**(Просмотр функции). Ответ, возвращаемый функцией представления, может быть простой строкой, содержащей HTML, или сложной формой.

Некоторые форматы URL содержат переменные части. Содержимое в угловых скобках является динамической частью, и любой URL, соответствующий статической части, будет сопоставлен с этим маршрутом. При вызове функции просмотра Flask передает динамическую часть в качестве параметра функции.

Динамическая часть маршрута по умолчанию использует строки, Но определения типов также могут быть использованы. Например, маршрут / пользователь / <int:id> Совпадают только те URL, чей динамический идентификатор клипа является целым Flask поддерживает типы int, float и path в маршрутизации. Код выглядит следующим образом:

тип пути: Является строкой, но косая черта не рассматривается как разделитель, а как часть динамического сегмента.

```
@app.route('/user/<name>')
def user(name):
        return '<h1>Hello, %s!</h1>' % name
```

2.3 Запустить сервер

Экземпляр программы использует метод run для запуска веб-сервера разработки, интегрированного с Flask (сервер, предоставленный flask, не подходит для использования в режиме разработки): функция app.run () может получать параметры, устанавливающие режим работы веб-сервера: режим отладки-> debug = True:

```
if __name__ == '__main__':
        app.run(debug=True)
```

name==’__main__' : используется для оценки входа в программу, чтобы гарантировать, что веб-сервер разработки запускается только при непосредственном выполнении этого сценария. Если этот скрипт введен другим скриптом, значение, отображаемое __name__, является именем модуля.

2.4 Полная программа

Программный код показан в Примере 2-1.

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def index():
        return '<h1>Hello World!</h1>'

@app.route('/')
def user('/user/<name>):
        return '<h1>Hello, %s!</h1>' % name
if __name__ == '__main__':
app.run(debug=True)
```

2.5 Цикл запроса-ответа

Теперь, когда вы разработали простую программу Flask, в следующих разделах будут представлены некоторые концепции проектирования этой платформы.

2.5.1 Процедура и контекст запроса

Когда Flask получает запрос от клиента, функция просмотра должна иметь доступ к некоторым объектам, чтобы запрос мог быть обработан. Хороший пример - объект запроса, он инкапсулирует HTTP-запрос, отправленный клиентом. Flask использует контекст, чтобы временно сделать определенные объекты глобально доступными. С помощью контекста вы можете написать следующую функцию просмотра:

```
from flask import request
@app.route('/')
def index():
        user_agent = request.headers.get('User-Agent')
        return '<p>Your browser is %s</p>' % user_agent
```

Обратите внимание: Как мы используем запрос в качестве глобальной переменной в этой функции представления. На самом деле запрос не может быть глобальной переменной. Представьте себе, что на многопоточном сервере, когда несколько потоков одновременно обрабатывают разные запросы, отправленные разными клиентами, объект запроса, видимый каждым потоком, должен быть разным.Falsk использует контекст, чтобы сделать определенные переменные глобально доступными в потоке.

контекст: Эквивалентно контейнеру, который сохраняет некоторую информацию во время работы программы Flask.

Запросите контекстные переменные, как показано в следующей таблице:

переменная	содержание
request	Содержимое HTTP-запроса инкапсулируется, а HTTP-запрос является целевым. user = request.args.get ('user'), получить параметры запроса get.
session	Он используется для записи информации в сеансе запроса и реализации настройки поддержания состояния, которая предназначена для информации пользователя. session.get ('name') получает информацию о пользователе.

Переменные контекста программы, как показано в следующей таблице:

переменная	содержание
g	Объект используется как временное хранилище при обработке запроса. Эта переменная сбрасывается при каждом запросе
current_app	Экземпляр программы активированной в данный момент программы, используемый для записи информации о конфигурации во время работы программы, такой как журнал проекта для записи проекта

Обратите внимание: Flask активирует (или толкает) программу и контекст запроса перед распространением запроса и удаляет его после обработки запроса. После нажатия на контекст программы в потоке можно использовать переменные current_app и g. переменные запроса и сеанса. Если мы не активируем программный контекст или контекст запроса при использовании этих переменных, это вызовет ошибку. Вызовите app.app_context () для экземпляра программы, чтобы получить контекст программы.

2.5.2 Планирование запроса

Когда программа получает запрос от клиента, ей нужно найти функцию представления, которая обрабатывает запрос. Для выполнения этой задачи Flask найдет запрошенный URL-адрес в отображении URL-адреса программы.**Flask использует app.route decorator или не-decorator app.add_url_rule () для генерации отображения.** Чтобы увидеть, как выглядит сопоставление URL в программе Flask, мы можем проверить сопоставление, сгенерированное для hello.py в оболочке Python.

Сопоставление URL: Соответствие между URL и функцией просмотра.

```
print（app.url_map）
----------------------------------------------------------
Map([<Rule '/' (HEAD, OPTIONS, GET) -> index>,
<Rule '/static/<filename>' (HEAD, OPTIONS, GET) -> static>,
<Rule '/user/<name>' (HEAD, OPTIONS, GET) -> user>])
```

Обратите внимание: HEAD, Options и GET в отображении URL являются методами запроса, которые обрабатываются маршрутом. Flask определяет метод запроса для каждого маршрута, поэтому, когда разные методы запроса отправляются на один и тот же URL, для обработки используются разные функции представления. Методы HEAD и OPTIONS автоматически обрабатываются Flask, и три маршрута используют метод GET по умолчанию. Вы можете добавить метод запроса в декоратор: @ app.route ('/', [«GET», «POST»])

2.5.3 Запрос хука

Иногда полезно выполнить код до или после обработки запроса. Например, в начале запроса нам может потребоваться создать соединение с базой данных или аутентифицировать пользователя, который инициировал запрос. Flask предоставляет функцию регистрации общих функций.Зарегистрированные функции можно вызывать до или после того, как запрос передается в функцию просмотра. Хуки запросов реализованы с использованием декораторов. Колба поддерживает следующие 4 вида крючков.

крюк	содержание
before_first_request	Зарегистрируйте функцию для запуска перед обработкой первого запроса.
before_request	Зарегистрируйте функцию для запуска перед каждым запросом.
after_request	Зарегистрируйте функцию и запускайте ее после каждого запроса, если не выдается необработанное исключение.
teardown_request	Зарегистрируйте функцию, которая будет запускаться после каждого запроса, даже если выдается необработанное исключение.

Обратите внимание: Обмен данными между функцией ловушки запроса и функцией представления обычно использует глобальную переменную контекста g. Например, обработчик before_request может загрузить зарегистрированного пользователя из базы данных и сохранить его в g.user. Когда впоследствии вызывается функция view, функция view использует g.user для получения пользователя.

2.5.4 ответ

После того, как Flask вызовет функцию представления, ее возвращаемое значение будет использоваться в качестве содержимого ответа. В большинстве случаев ответ представляет собой простую строку, которая отправляется обратно клиенту в виде HTML-страницы. Но протокол HTTP требует больше, чем строка в ответ на запрос. Важной частью ответа HTTP является код состояния.

(1) Вернуть разные коды состояния в соответствии с требованиями: Если в ответе, возвращаемом функцией просмотра, нужно использовать другой код состояния, вы можете использовать числовой код в качестве второго возвращаемого значения и добавить его в текст ответа. Пример программы выглядит следующим образом:

```
@app.route('/')
def index():
        return '<h1>Bad Request</h1>', 400
```

(2) Функция просмотра Flask также может возвращать объект Response: Функция make_response () может принимать 1, 2 или 3 параметра (аналогично возвращаемому значению функции представления) и возвращает объект Response. Иногда нам нужно выполнить это преобразование в функции представления, а затем вызвать различные методы для объекта ответа, чтобы дополнительно установить ответ. В следующем примере создается объект ответа, а затем устанавливается файл cookie:

```
from flask import make_response
@app.route('/')
def index():
        response = make_response('<h1>This document carries a cookie!</h1>')
        response.set_cookie('answer', '42')
        return response
```

(3) Перенаправлено как тип возврата функции просмотра, Для этого ответа нет страницы документа, только браузеру сообщается новый адрес для загрузки новой страницы. Перенаправление часто используется в веб-формах. Перенаправление часто обозначается кодом состояния 302, а указанный адрес предоставляется заголовком Location. Flask предоставляет вспомогательную функцию redirect () для генерации этого ответа:

```
from flask import redirect
@app.route('/')
def index():
        return redirect('http://www.example.com')
```

Ответ, генерируемый функцией прерывания, Прервать функцию: оператор обработки исключений в колбе. Функция прерывания может генерировать только ненормальный код состояния, соответствующий протоколу http.В следующем примере, если пользователь, соответствующий идентификатору динамического параметра в URL-адресе, не существует, он возвращает код состояния 404:

Функция: функция прерывания обычно используется для реализации специального сообщения об ошибке, чтобы сделать код более масштабируемым и улучшить взаимодействие с пользователем.

```
from flask import abort
@app.route('/user/<id>')
def get_user(id):
        user = load_user(id)
        if not user:
                abort(404)
        return '<h1>Hello, %s</h1>' % user.name
```

Примечание: abort не возвращает управление функции, которая его вызвала, но выдает исключение и передает управление веб-серверу. Пока код, следующий за прерыванием, не выполняется.

2.6 Расширение колбы

Flask предназначен для расширения, поэтому он не обеспечивает некоторые важные функции, такие как проверка подлинности базы данных и пользователя, поэтому разработчики могут свободно выбирать наиболее подходящий пакет для программы или разрабатывать в соответствии со своими потребностями. Далее мы можемhello.py В качестве примера добавлено расширение для использования параметров командной строки для улучшения функции программы. Используя Flask-Script для поддержки параметров командной строки, веб-сервер разработки Flask поддерживает множество параметров

настройки запуска, но может передаваться только в качестве параметров в сценарии в функцию app.run ().

Flask-Script: Расширение Flask, которое добавляет анализатор командной строки в программу Flask. Flask-Script поставляется с набором часто используемых параметров, а также поддерживает пользовательские команды. Расширение Flask-Script устанавливается с помощью pip: pip install flask-script.

Пример программы:

```
from flask import Flask
from flask_script import Manager

app = Flask(__name__)
manager = Manager(app)

@app.route('/')
def index():
        return '<h1>Hello World!</h1>'

if __name__ == '__main__':
        manager.run()
```

Просмотр команд терминала в командной строке：python hello.py runserver --help

Терминальная команда для запуска сервера：python3 hello.py runserver -p 5000

Вызовите метод запуска вместо приложения: Runserver должен быть добавлен в параметры скрипта конфигурации интерпретатора Python

эффект: Вы можете вручную ввести IP и порт в терминале, вы можете настроить параметры скрипта, вы можете добиться миграции базы данных
Параметр –host - очень полезный параметр, он сообщает веб-серверу, на каком сетевом интерфейсе прослушивать клиент.
подключиться.По умолчанию веб-сервер разработки Flask прослушивает соединения на локальном хосте, поэтому он принимает сервисы только от Соединение, инициированное компьютером, на котором расположен сервер.

Следующая команда позволяет веб-серверу прослушивать соединение через общедоступный сетевой интерфейс, что позволяет
Другие компьютеры в сети подключаются к серверу:

（1）python hello.py runserver --host 0.0.0.0
（2）Running on http://0.0.0.0:5000/
（3）Restarting with reloader

Веб-сервер теперь доступен http://a.b.c.d:5000/ Доступ с любого компьютера в сети, где "a.b.c.d" - это внешний IP-адрес компьютера, на котором расположен сервер.

Task- Отправка данных из формы в базу данных

1- **Проверить наличие библиотек!**

pip install Flask-MySQLdb

Исходной код App.py:

```
from flask import Flask, render_template, request
from flask_mysqldb import MySQL
import yaml

app = Flask(__name__)

# DB configuration
db = yaml.load(open('db.yaml'))
app.config['MYSQL_HOST'] = db['mysql_host']
app.config['MYSQL_USER'] = db['mysql_user']
app.config['MYSQL_PASSWORD'] = db['mysql_password']
app.config['MYSQL_DB'] = db['mysql_db']
app.config['MYSQL_CURSORCLASS'] = 'DictCursor'
mysql = MySQL(app)

@app.route('/', methods=['GET','POST'])
def index():
    if request.method == 'POST':
        form = request.form
        name = form['name']
        age = form['age']
        cursor = mysql.connection.cursor()
        cursor.execute("INSERT INTO employee(name, age) VALUES(%s, %s)", (name,
age))
        mysql.connection.commit()
    return render_template('index.html')

@app.route('/employees')
def employees():
    cursor = mysql.connection.cursor()
    result_value = cursor.execute("SELECT * FROM employee")
    if result_value > 0:
        employees = cursor.fetchall()
        return render_template('employees.html', employees=employees)



if __name__ == '__main__':
    app.run(debug=True)
```

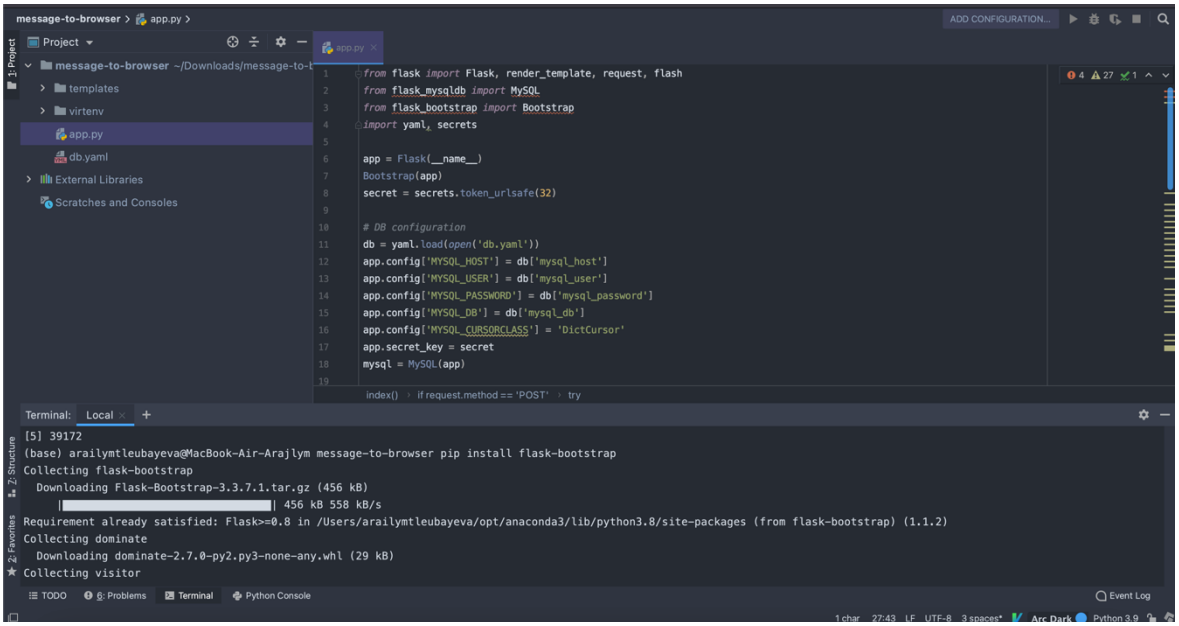
db.yaml:

```
mysql_host: 'localhost'
mysql_user: 'root'
mysql_password: '11111111'
mysql_db: 'employee_data'
```

Task 2 – Сообщения в браузере

2- Проверить наличие библиотек!

pip install Flask-MySQLdb
pip install Flask-bootstrap



Исходной код: app.py

```
from flask import Flask, render_template, request, session
from flask_mysqldb import MySQL
import yaml
import os

app = Flask(__name__)

# DB configuration
db = yaml.load(open('db.yaml'))
app.config['MYSQL_HOST'] = db['mysql_host']
app.config['MYSQL_USER'] = db['mysql_user']
app.config['MYSQL_PASSWORD'] = db['mysql_password']
app.config['MYSQL_DB'] = db['mysql_db']
app.config['MYSQL_CURSORCLASS'] = 'DictCursor'
mysql = MySQL(app)

app.config['SECRET_KEY'] = os.urandom(24)

@app.route('/', methods=['GET','POST'])
def index():
   if request.method == 'POST':
      form = request.form
      name = form['name']
      age = form['age']
      cursor = mysql.connection.cursor()
      cursor.execute("INSERT INTO employee(name, age) VALUES(%s, %s)", (name,
age))
      mysql.connection.commit()
   return render_template('index.html')
```

```python
@app.route('/employees')
def employees():
    cursor = mysql.connection.cursor()
    result_value = cursor.execute("SELECT * FROM employee")
    if result_value > 0:
        employees = cursor.fetchall()
        session['username'] = employees[0]['name']
        return render_template('employees.html', employees=employees)



if __name__ == '__main__':
    app.run(debug=True)
```

db.yaml:

```yaml
mysql_host: 'localhost'
mysql_user: 'root'
mysql_password: '11111111'
mysql_db: 'employee_data'
```