Что такое REST?

REST расшифровывается как REpresentational State Transfer. Это был термин, первоначально введен Роем Филдингом (Roy Fielding), который также был одним из создателей протокола HTTP. Отличительной особенностью сервисов REST является то, что они позволяют наилучшим образом использовать протокол HTTP. Теперь давайте кратко рассмотрим HTTP.

URI ресурса

Когда вы разрабатываете RESTful сервисы, вы должны сосредоточить свое внимание на ресурсах приложения. Способ, которым мы идентифицируем ресурс для предоставления, состоит в том, чтобы назначить ему URI — универсальный идентификатор ресурса. Например:

Создать пользователя: POST /users

Удалить пользователя: DELETE /users/1

Получить всех пользователей: GET /users

Получить одного пользователя: GET /users/1

В этой работе я использовала две библиотеки: flask и flask\_restful.

Во второй довольно удобно через ООП описать все нужные методы

Код работы:

from flask import Flask, \

json, Response

from flask\_restful import reqparse, \

abort, Api, Resource

app = Flask(\_\_name\_\_)

api = Api(app)

events = {}

def abort\_404(todo\_id):

if todo\_id not in events:

abort(404, message="promo {} doesn't exist".format(todo\_id))

parser = reqparse.RequestParser()

parser.add\_argument('name')

parser.add\_argument('description')

parser.add\_argument('participant')

parser.add\_argument('prize')

class promo(Resource):

def get(self):

dict\_get\_all\_promos = {}

for id\_promo in events.keys():

id\_promo = int(id\_promo)

task = {'id': events[id\_promo]['id'], 'name': events[id\_promo]['name'],

'description': events[id\_promo]['description']}

dict\_get\_all\_promos[id\_promo] = task

return dict\_get\_all\_promos

def post(self):

args = parser.parse\_args()

id = int(max(events.keys())) + 1

if args['name'] is None:

abort(404, message="name is not defined")

elif args['description'] is None:

task = {'id': id, 'name': args['name'], 'description': None, 'participant': [],

'prizes': []}

else:

task = {'id': id, 'name': args['name'], 'description': args['description'], 'participant': [],

'prizes': []}

events[int(id)] = task

return id, 201

class promo\_id(Resource):

def get(self, id\_promo: int):

abort\_404(id\_promo)

return events[id\_promo]

def delete(self, id\_promo: int):

abort\_404(id\_promo)

del events[id\_promo]

return '', 204

def put(self, id\_promo: int):

args = parser.parse\_args()

if id\_promo in events.keys() and args['name'] is not None and args['description'] is not None:

task = ({

'id': id\_promo,

'name': args['name'],

'description': args['description'],

'participant': events[id\_promo]['participant'],

'prize': events[id\_promo]['prize']

})

elif id\_promo in events.keys() and args['name'] is not None:

task = ({

'id': id\_promo,

'name': args['name'],

'description': None,

'participant': events[id\_promo]['participant'],

'prize': events[id\_promo]['prize']

})

else:

abort(404, message="name is not defined")

events[id\_promo] = task

return task, 201

class participant(Resource):

def post(self, id\_promo: int):

args = parser.parse\_args()

if id\_promo in events.keys():

if len(events[id\_promo]['participant']) >= 1:

id\_participant = int(events[id\_promo]['participant'][-1]['id']) + 1

else:

id\_participant = 1

events[id\_promo]['participant'].append({'id': id\_participant, 'name': args['name']})

return id, 201

else:

abort(404, message="name is not defined")

class participant\_delete(Resource):

def delete(self, id\_promo: int, id\_participant: int):

abort\_404(id\_promo)

for i, participant in enumerate(events[id\_promo]['participant']):

if id\_participant == participant['id']:

del events[id\_promo]['participant'][i]

return '', 204

class prize(Resource):

def post(self, id\_promo):

args = parser.parse\_args()

id\_promo = int(id\_promo)

if id\_promo in events.keys():

if len(events[id\_promo]['prize']) >= 1:

id\_prize = int(events[id\_promo]['prize'][-1]['id']) + 1

else:

id\_prize = 1

events[id\_promo]['prize'].append({'id': id\_prize, 'description': args['description']})

return id\_prize

else:

abort(404, message="name is not defined")

class prize\_delete(Resource):

def delete(self, id\_promo: int, id\_prize: int):

abort\_404(id\_promo)

for i, prize in enumerate(events[id\_promo]['prize']):

if id\_prize == prize['id']:

del events[id\_promo]['prize'][i]

return '', 204

class raffle(Resource):

def get(self, id\_promo: int):

args = parser.parse\_args()

final\_list = []

if id\_promo in events.keys():

if len(events[id\_promo]['prize']) == len(events[id\_promo]['participant']):

for i in range(len(events[id\_promo]['prize'])):

final\_list.append(

{

"winner": {

"id": events[id\_promo]['participant'][i]['id'],

"name": events[id\_promo]['participant'][i]['name']

},

"prize": {

"id": events[id\_promo]['prize'][i]['id'],

"description": events[id\_promo]['prize'][i]['description']

}

}

)

return final\_list

else:

abort(409, message="409 (Conflict)")

else:

abort(404, message="name is not defined")

api.add\_resource(promo, '/promo')

api.add\_resource(promo\_id, '/promo/<id\_promo>')

api.add\_resource(prize, '/promo/<id\_promo>/prize')

api.add\_resource(participant, '/promo/<id\_promo>/participant')

api.add\_resource(prize\_delete, '/promo/<id\_promo>/prize/<id\_prize>')

api.add\_resource(participant\_delete, '/promo/<id\_promo>/participant/<id\_participant>')

api.add\_resource(raffle, '/promo/<id\_promo>/raffle')

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=False, port=8080)