

# Tinkoff Python

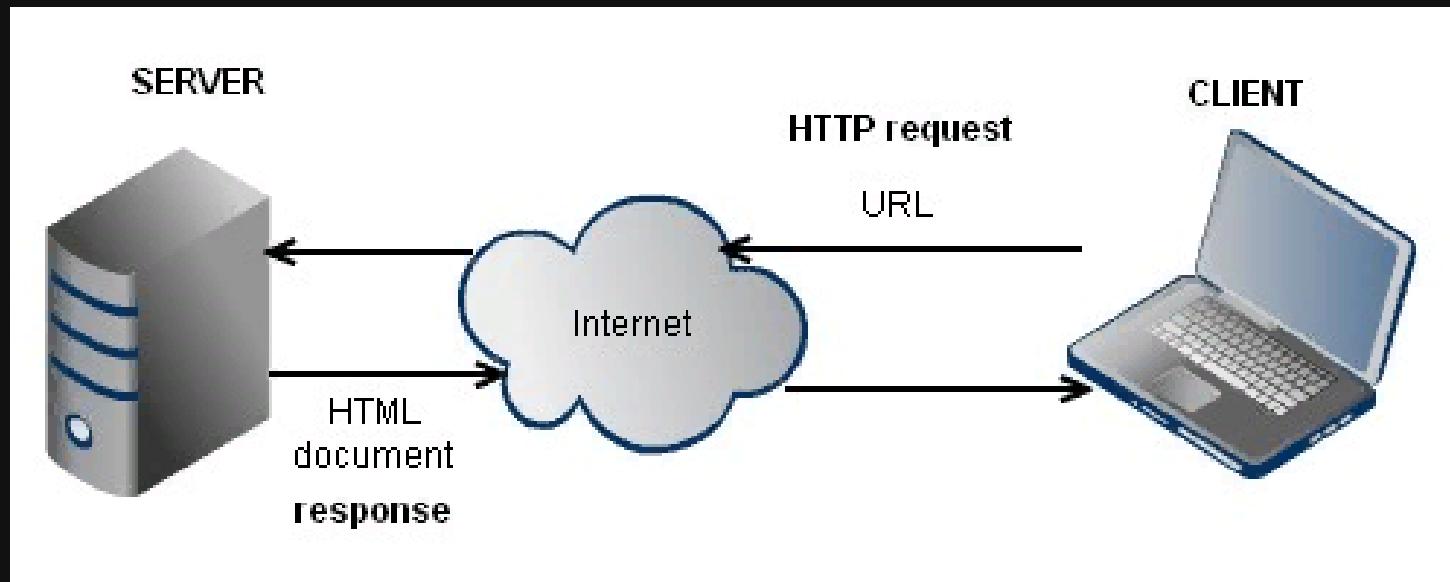
## Лекция 4

# Проектирование API. REST



Афонасьев Евгений

# Серверный рендеринг



# Редко используется в крупных проектах

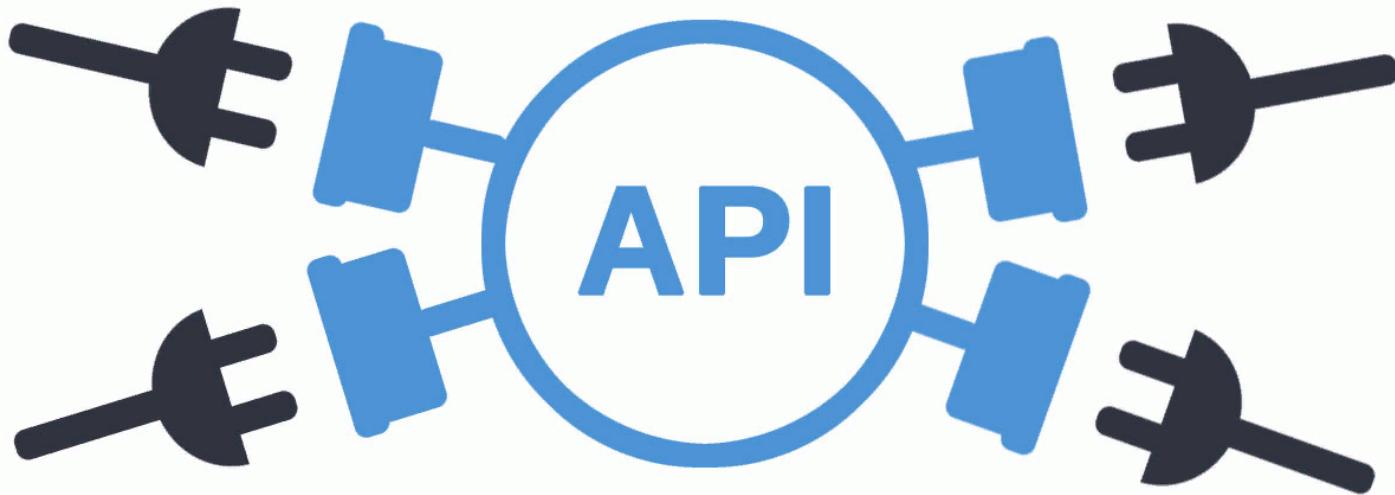
Обычно только во внутренних системах,  
например в административных панелях



# Почему?

- Интерфейс требует перезагрузки страницы и запросов на сервер на каждое действие
- ограниченность в разработке интерактивных интерфейсов
- нельзя написать нормальное мобильное приложение
- неудобно работать с ajax

Чтобы работать одновременно и с мобильными приложениями и с браузером, сервер должен оперировать данными, а не представлением

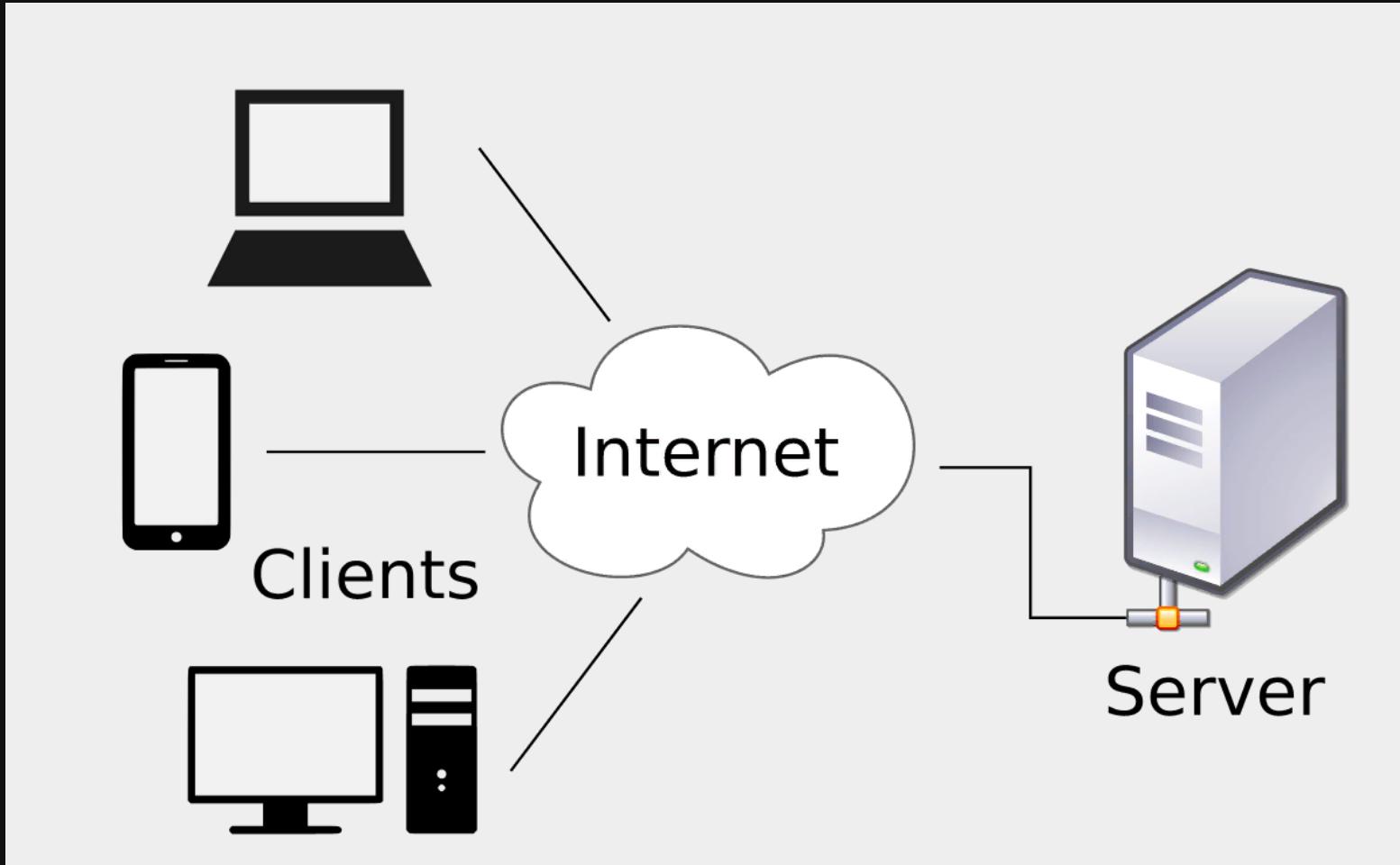


# Эндпоинты API

- GET /api/v1/get\_user
- GET /api/v1/get\_user\_list
- POST /api/v1/create\_user

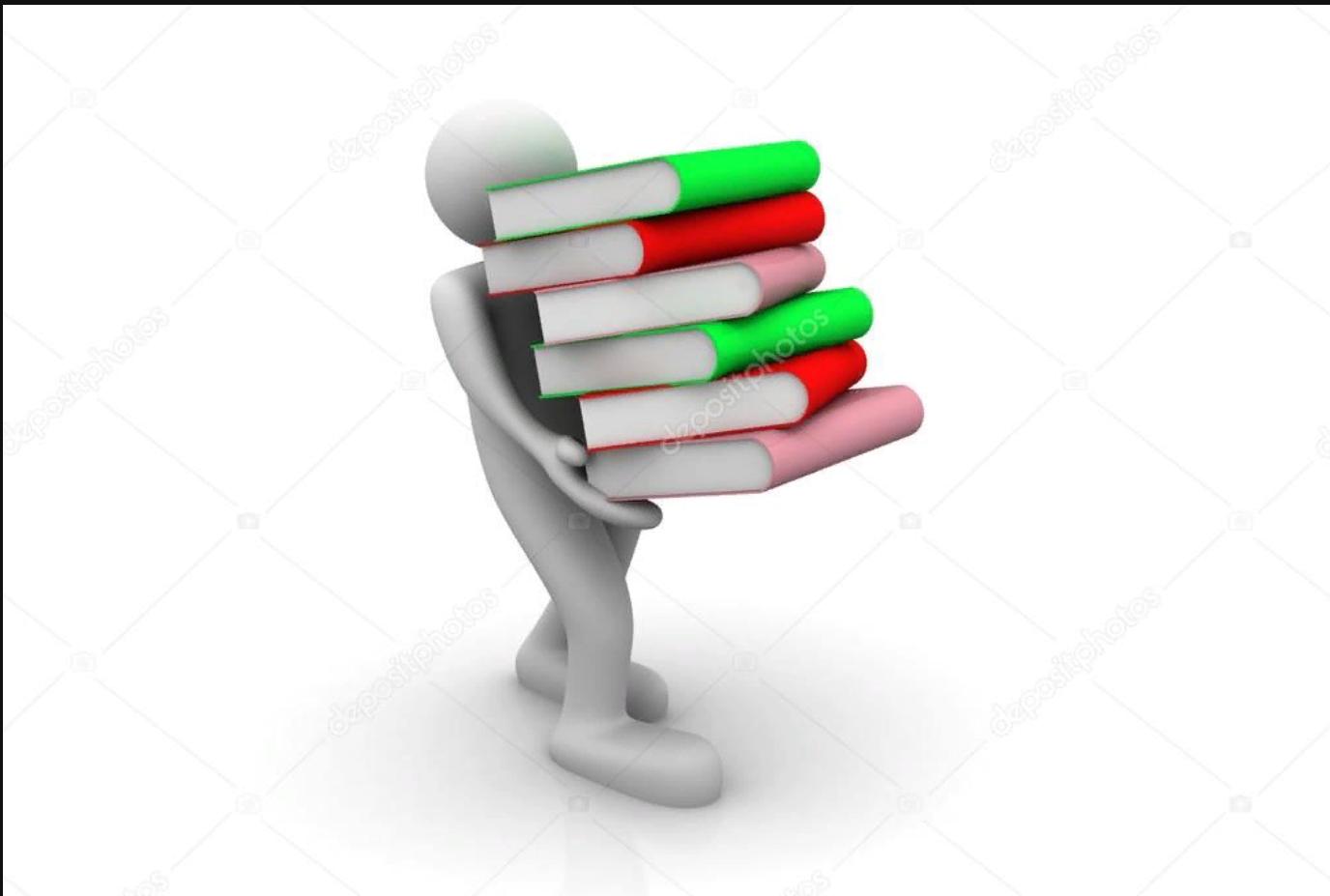
Грамотно написанное API  
позволяет использовать себя  
для разных платформ

# Фронты разные, бэк один



Server side rendering  
обрел второе дыхание  
на фронте

# Форматы передачи данных



# Content-type header

[https://developer.mozilla.org/ru/docs/Web  
/HTTP/Заголовки/Content-Type](https://developer.mozilla.org/ru/docs/Web/HTTP/Заголовки/Content-Type)

# Accept header

[https://developer.mozilla.org/ru/docs/Web  
/HTTP/Заголовки/Accept](https://developer.mozilla.org/ru/docs/Web/HTTP/Заголовки/Accept)

# Text

4:19

GALATIANS

146

with good motives and sincere hearts, especially if they aren't doing it just when I am with you!

19 Oh my children, how you are hurting me. I am once again suffering for you the pains of a mother waiting for her child to be born—longing for the time when you will finally be filled with Christ.

20 How I wish I could be there with you right now and not have to reason with you like this, for at this distance I frankly don't know what to do.

21 Listen to me, you friends who think you have to obey the Jewish laws to be saved: Why don't you find out what those laws really mean?

22 For it is written that Abraham had two sons, one from his slave-wife and one from his freeborn wife.

23 There was nothing unusual about the birth of the slave-wife's baby. But the baby of the freeborn wife was born only after God had especially promised he would come.

24, 25 Now this true story is an illustration of God's two ways of helping people. One way was by giving them His laws to obey. He did this on Mount Sinai, when He gave the Ten Commandments to Moses. Mount Sinai, by the way, is called "Mount Hagar" by the Arabs—and in my illustration Abraham's slave-wife Hagar represents Jerusalem, the mother-city of the Jews, the center of that system of trying to please God by obeying God's laws; and the Jews, who try to follow that system, are her slave children.

26 But our mother-city is the heavenly Jerusalem, and she is not a slave to Jewish laws.

27 That is what Isaiah meant when he prophesied,

147

GALATIANS

4:28

"Now you can rejoice, oh childless woman; you can shout with joy though you never before had a child. For I am going to give you many children—more children than the slave-wife has."

28 You and I, dear brothers, are the children that God promised, just as Isaac was.

29 And so we who are born of the Holy Spirit are persecuted now by those who want us to keep the Jewish laws, just as Isaac the child of promise was persecuted by Ishmael the slave-girl's son.

30 But the Scriptures say that God told Abraham to send away the slave-wife and her son, for the slave-wife's son could not inherit Abraham's home and lands along with the free woman's son.

31 Dear brothers, we are not slave children, obligated to the Jewish laws, but children of the free woman, acceptable to God because of our faith.

## CHAPTER 5

So Christ has made us free. Now make sure that you stay free and don't get all tied up again in the chains of slavery to Jewish laws and ceremonies.

2 Listen to me, for I really mean it: if you are counting on circumcision and keeping the Jewish laws to make you right with God, then Christ cannot save you.

3 I'll say it again. Anyone trying to find favor with God by being circumcised must always obey every other Jewish law or perish.

4 Christ is useless to you if you are counting on clearing your debt to God by keeping those laws; you are lost from God's grace.

# Text

```
@app.route('/user/<user_id>/name')
def user_name(user_id):
    return 'name'

@app.route('/user/<user_id>/desc')
def user_desc(user_id):
    return 'long text about user...'
```

# CSV

```
@app.route('/users')
def users():
    return (
        'user_id;name;surname;money\n'
        'user1;igor;volkov;13;2000\n'
        'user2;vanya;batkov;83;8000\n'
    )
```

<https://ru.wikipedia.org/wiki/CSV>

# xml

```
@app.route('/data')
def data():
    return """
        <note>
            <to>Tove</to>
            <from>Jani</from>
            <heading>Reminder</heading>
            <body>Don 't forget me this weekend!</body>
        </note>
    """
    """
```

# xml



Когда-то был самым распространенным  
форматом

# json

```
{ Object Starts
  "Title": "The Cuckoo's Calling"
  "Author": "Robert Galbraith",
  "Genre": "classic crime novel",
  "Detail": { Object Starts
    "Publisher": "Little Brown" Value string
    "Publication_Year": 2013, Value number
    "ISBN-13": 9781408704004,
    "Language": "English",
    "Pages": 494
  } Object ends
  "Price": [ Array starts
    {
      "type": "Hardcover",
      "price": 16.65,
    } Object ends
    {
      "type": "Kindle Edition",
      "price": 7.03,
    } Object ends
  ] Array ends
}
```

Object ends

# json

```
import json

@app.route('/data')
def data():
    return json.dumps({ 'id': 1, 'name': 'name' })
```

<https://ru.wikipedia.org/wiki/JSON>

Обычно работа с json  
встроена во  
фреймворки

# json

```
from flask import jsonify

@app.route('/_get_current_user')
def get_current_user():
    return jsonify(
        username=g.user.username,
        email=g.user.email,
        id=g.user.id,
    )
```

# json

- Современный стандарт
- Человеко-читаемые данные
- Компактнее чем xml
- Будет достаточен в 99% случаев

<https://ru.wikipedia.org/wiki/JSON>

# ujson etc.

Работает быстрее встроенного json модуля

```
import ujson

@app.route('/data')
def data():
    return ujson.dumps({'id': 1, 'name': 'name'})
```

<https://pypi.org/project/ujson/>



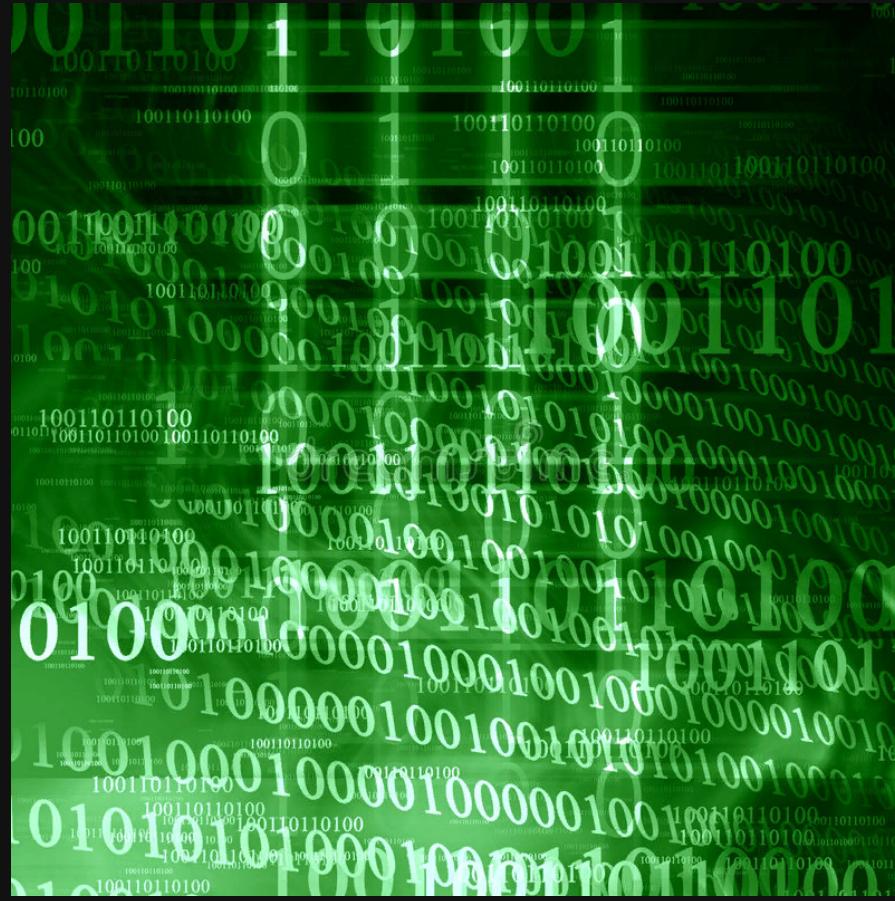
# ujson

Часто является опциональной зависимостью для библиотек (но его поведение не на 100% совпадает со стандартным модулем!)

```
try:  
    import ujson as json  
except ImportError:  
    import json
```

<https://pypi.org/project/ujson/>

# Бинарные форматы



# msgpack

JSON 27 bytes

```
{ "compact": true, "schema": 0 }
```

MessagePack 18 bytes



# msgpack

```
import msgpack

@app.route('/data')
def data():
    user = {'id': 1, 'name': 'name'}
    return msgpack.packb(user, use_bin_type=True)
```

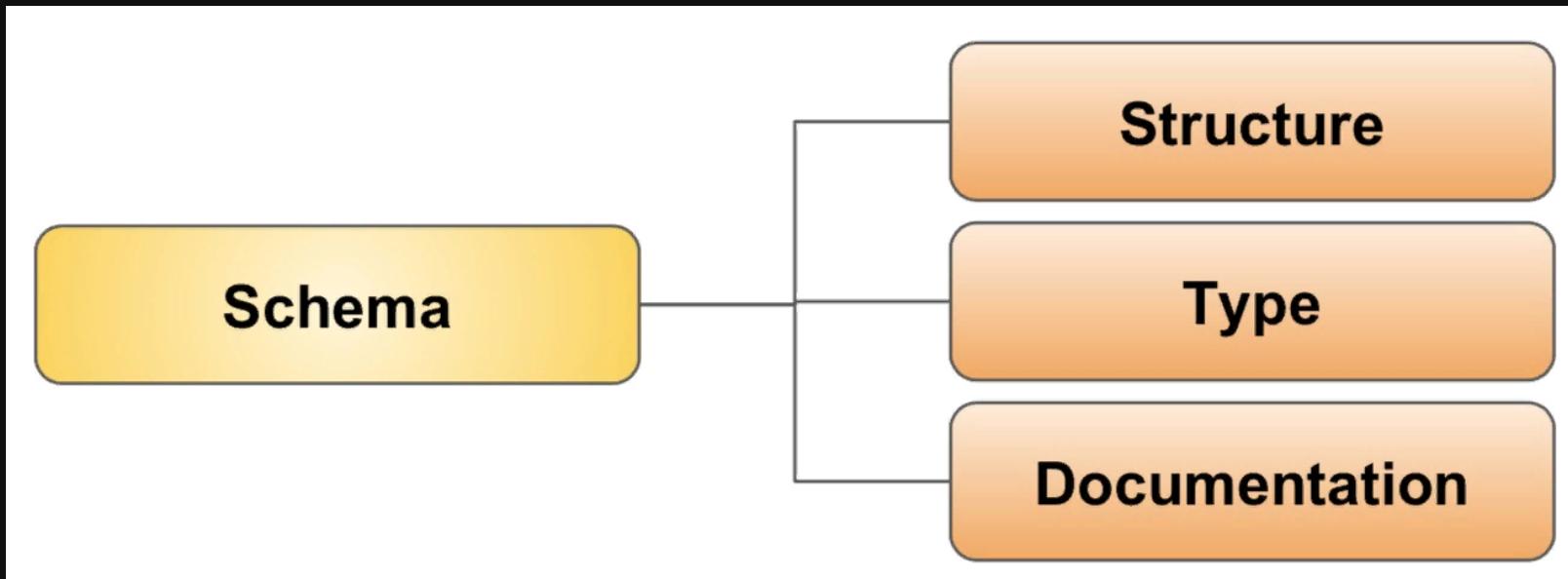
<https://msgpack.org>

# Бинарные форматы

- + Объем передаваемых данных значительно меньше
- Больше нагрузка на CPU (обычно не критично)
- Все клиенты должны уметь распаковывать данные из используемого формата
- Нельзя понять, что лежит в сообщении, без декодирования

Зачастую можно добиться  
схожих результатов с помощью  
**gzip (brotli)**

# Бинарные форматы со схемами



# Protobuf

```
message Car {
    required string model = 1;

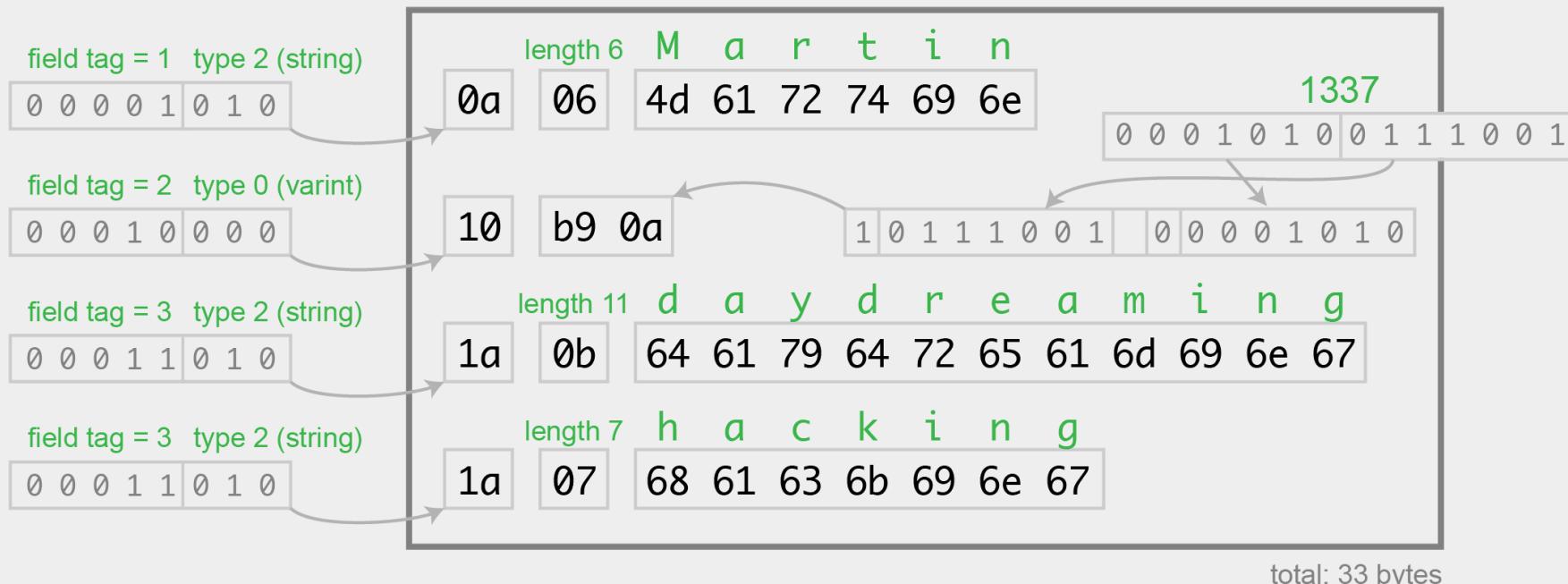
    enum BodyType {
        sedan = 0;
        hatchback = 1;
        SUV = 2;
    }

    required BodyType type = 2 [default = sedan];
    optional string color = 3;
    required int32 year = 4;

    message Owner {
        required string name = 1;
        required string lastName = 2;
        required int64 driverLicense = 3;
    }

    repeated Owner previousOwner = 5;
}
```

# Protocol Buffers



# Бинарные форматы со схемами

- + Объем передаваемых данных значительно меньше
- + Встроенная валидация данных по схеме
- + Обычно позволяет изменять схему с сохранением обратной совместимости (версионирование)

# Бинарные форматы со схемами

- Больше нагрузка на CPU
- Нельзя понять, что лежит в сообщении, без декодирования
- Нужно передавать схемы всем системам, которые должны будут работать с данными!

# Шаблоны построения API



<https://nest.scriptrapps.io/thermostats/12>

<https://nest.scriptrapps.io/getThermostat?Id=12>

# Велосипеды



# Велосипеды

```
@app.route('/get_users')
def get_users():
    return json.dumps([
        {'id': 1, 'name': 'name'},
        {'id': 2, 'name': 'name'},
    ])

@app.route('/user/<user_id>')
def get_user(user_id):
    user = db.get_user(user_id)
    return json.dumps(user)

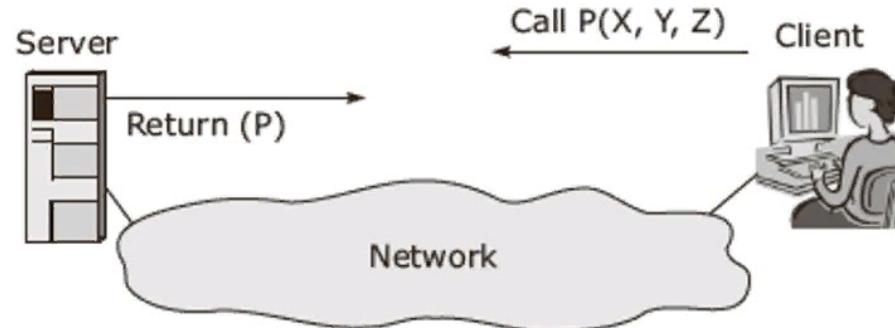
@app.route('/create_user')
def create_user(user_id):
    user = db.create_user(request.data)
    return 'ok'
```

# Плюсы/минусы

- + Легко начать
- Сложно объяснить пользователям как использовать апи
- Нет готовых стандартных инструментов

# Remote Procedure Call

- Basic RPC operation



**Figure 4-3** Basic RPC model

<https://ru.wikipedia.org/wiki/JSON-RPC>

Все запросы попадают на  
один http эндпоинт

POST /api/grpc

Определяет методами

User.create\_user

User.add\_friend

# Можно написать что-то свое

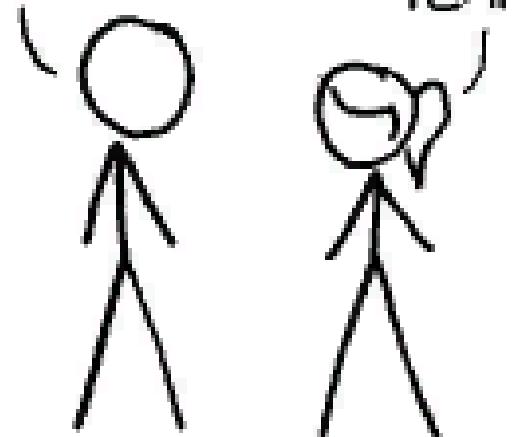
```
@app.route('/api', methods=['POST'])
def handle_request():
    content = request.json
    if content['method'] == 'User.add_friend':
        User.add_friend(**content['kwargs'])
    if content['method'] == 'User.get_friends':
        friends = User.get_friends(**content['kwargs'])
        return friends, 200
    return 200
```

# HOW STANDARDS PROLIFERATE:

(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)

SITUATION:  
THERE ARE  
14 COMPETING  
STANDARDS.

14?! RIDICULOUS!  
WE NEED TO DEVELOP  
ONE UNIVERSAL STANDARD  
THAT COVERS EVERYONE'S  
USE CASES.



SOON:

SITUATION:  
THERE ARE  
15 COMPETING  
STANDARDS.

# XML-RPC - устарел



# JSON RPC

```
# normal request
--> {
    "jsonrpc": "2.0",
    "method": "subtract",
    "params": {"subtrahend": 23},
    "id": 3
}
<-- {
    "jsonrpc": "2.0",
    "result": 19,
    "id": 3
}

# error
--> [ ]
<-- {
    "jsonrpc": "2.0",
    "error": {"code": -32, "message": "Invalid Request"},
    "id": null
}
```

<https://www.jsonrpc.org/specification>

# JSON RPC

```
from flask import Flask, request, Response
from jsonrpcserver import method, dispatch

app = Flask(__name__)

@method
def ping():
    return "pong"

@app.route("/", methods=[ "POST" ])
def index():
    req = request.get_data().decode()
    response = dispatch(req)
    return Response(
        str(response),
        response.http_status,
        mimetype="application/json",
    )

if __name__ == "__main__":
    app.run()
```

Какие методы мы бы  
написали для todo  
приложения?

# UserBucket.get\_items(user\_id)

UserBucket.add\_item(user\_id, item\_id)

UserBucket.buy(user\_id)

Item.get\_info(item\_id)

# Плюсы

Легко проектировать арі, методы из кода напрямую  
ложатся на методы RPC

# Плюсы

Можно использовать в качестве транспорта не только  
http (ws) без больших изменений в коде

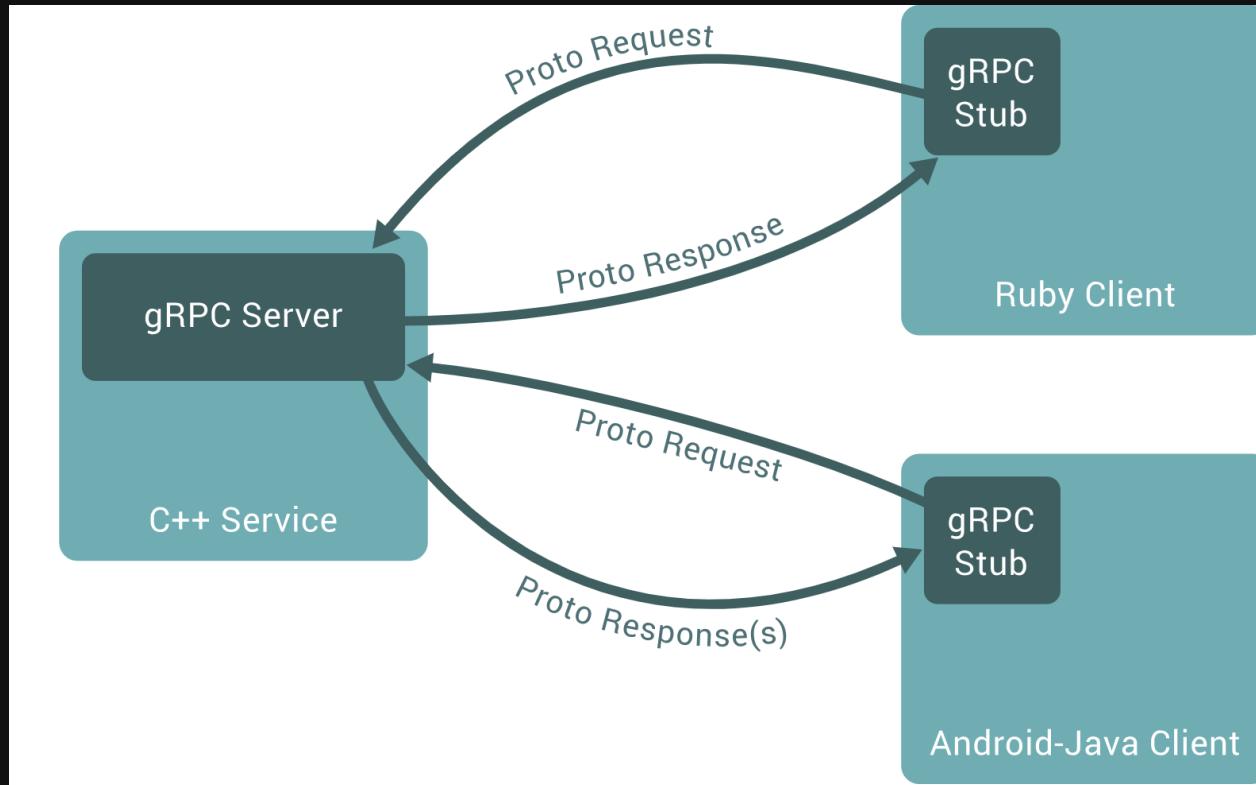
# Плюсы

Хорошо подходит для внутренних API внутри одной системы для взаимодействия между сервисами

# Минусы

Описывать и документировать API все еще сложно,  
нужно пояснить смысл методов и придумывать им  
понятные названия

# gRPC



<https://grpc.io>

# gRPC

Экосистема - свои сервера, свои  
клиенты, свой формат сериализации.  
Кодогенерация сервера и клиента.

<https://grpc.io>



*Representational State Transfer*

<https://ru.wikipedia.org/wiki/REST>

# Оперирует ресурсами

*users, pages, operations, actions*

Ресурсом может быть как одна сущность, так и список сущностей

/users

/users?older=30

/users/<user\_id>

Каждый ресурс имеет  
свой уникальный адрес

/users/<user\_id>/friends/<friend\_id>

Операции кодируются  
HTTP методами

GET, POST, DELETE (PUT, PATCH)

Ошибки задаются http  
статусами

400, 401, 403 etc.

# GET

GET /users - список пользователей

GET /users/<user\_id> - конкретный пользователь

GET /users?page=5 - страница списка пользователей

# GET

Безопасный метод, никогда не должен приводить к изменениям данных на бэке!

# POST

POST /users {} - создание пользователя

POST /users/<user\_id>/friends {} - добавление друга

# POST

Каждый вызов создает новую сущность!

# POST

Ожидает код 201

# DELETE

DELETE /users/<user\_id> - удаление пользователя

# DELETE

Ожидает код 204 или 404

# PUT

PUT /users/<user\_id> {} - перетирание данных  
новыми

# PUT

Нужно передать все поля сущности!

# PATCH

PUT /users/<user\_id> {} - обновление данных

# PATCH

Можно передать только часть данных

# Добавление друга на RPC

```
@app.route('/api', methods=['POST'])
def handle_request():
    content = request.json
    if content['method'] == 'User.add_friend':
        User.add_friend(**content['kwargs'])
    if content['method'] == 'User.get_friends':
        return User.get_friends(**content['kwargs'])
    return 200
```

# Добавление друга на REST

```
@app.route(  
    '/api/users/<user_id>/friends',  
    methods=[ 'GET' , 'POST' ]  
)  
def handle_request(user_id):  
    if request.method == 'POST':  
        User.add_friend(user_id, friend=request.json)  
        return friend, 201 # Created  
    if request.method == 'GET':  
        friends = User.get_friends(user_id)  
        return friends, 200 # Ok  
    return 405 # Method not allowed
```

Какими ресурсами мы бы  
оперировали в todo  
приложении?

**GET /users/<user\_id>/tasks**

**POST /users/<user\_id>/tasks**

**DELETE /users/<user\_id>/tasks/<task\_id>**

**GET /users/<user\_id>/tasks/<task\_id>**

**PUT /users/<user\_id>/tasks/<task\_id>**

# Retry

Важно понимать какие запросы  
можно повторять при ошибках!

# Retry

Повторять можно 5\*\* коды и  
ошибки соединения

# Идемпотентность

Свойство объекта или операции  
при повторном применении  
операции к объекту давать тот же  
результат, что и при первом.

# Идемпотентность

В контексте веб обычно подразумевается, что на один и тот же запрос будет одно и то же действие на сервере, и такой запрос можно безопасно повторять

POST не  
идемпотентен!!!

# Идемпотентность

Идемпотентности можно достигнуть, добавив ко всем запросам ключ идемпотентности с дальнейшем валидацией на стороне сервера

<https://habr.com/ru/company/yandex/blog/442762/>

# Stateless

Это протокол передачи данных,  
который относит каждый запрос к  
независимой транзакции, которая не  
связана с предыдущим запросом

# Stateless

Такой подход позволяет строить  
хорошо масштабируемые/надежные  
системы

# Плюсы

Удобно документировать апи, методы и ошибки HTTP достаточно понятны и очевидны.

# Плюсы

Хорошо подходит для публичных API для  
сторонних разработчиков

# Плюсы

Хорошо поддается дополнительной обработке на промежуточных узлах и кэшированию

# Минусы

Сложнее проектировать API, нужно  
придумывать как описать свою систему в  
виде набора ресурсов

# Минусы

Нет единого стандарта (как передавать ошибки, в каком формате передавать данные)

# Перерыв?

# Pagination

# Limit, offset

GET /users?limit=100&offset=200

GET /users?size=100&page=200

**OFFSET на больших значениях  
в SQL СУБД медленный!**

# Keyset

GET /users?id=20&limit=200

GET /users?ts=1635346346&page=200

Данные должны быть  
отсортированы по  
выбранному ключу

# Формат ответа

```
{  
    "page": 10,  
    "count": 1000,  
    "result": [  
        {  
            "user_id": 1  
        },  
        ...  
    ]  
}
```

# Формат ответа

```
[  
  {  
    "user_id": 1  
  },  
  ...  
]  
# служебные данные в http headers
```

TradeshowD... | SamplePetstore... | 1.0.0 ▾ OAS3

PRIVATE

UNPUBLISHED



Editor

Split

UI

Last Saved: 12:46:55 pm May 4, 2018



VALID

Save



https://virtserver.swaggerhub.com/TradeshowDemos/SamplePetstoreAPI/1.0.0 ▾

Show Comments



pet Everything about your Pets

Find out more: <http://swagger.io> ▾

POST /pet Add a new pet to the store



PUT /pet Update an existing pet



GET /pet/findByStatus Finds Pets by status



GET /pet/findByTags Finds Pets by tags



GET /pet/{petId} Find pet by ID



POST /pet/{petId} Updates a pet in the store with form data



DELETE /pet/{petId} Deletes a pet



POST /pet/{petId}/uploadImage uploads an image



store Access to Petstore orders

9 . 1

# OpenAPI

```
{  
  "swagger": "2.0",  
  "basePath": "/api",  
  "paths": {  
    "/bindings": {  
      "parameters": [  
        {  
          "in": "header",  
          "description": "Correlation ID",  
          "name": "X-Correlation-Id",  
          "type": "string"  
        },  
        {  
          "in": "header",  
          "description": "Connection ID",  
          "name": "X-Connection-Id",  
          "type": "string"  
        },  
        {  
          "in": "header",  
          "description": "Trace ID",  
          "name": "X-TraceId-Id",  
          "type": "string"  
        }  
      ],  
      "get": {  
        "responses": {  
          "200": {  
            "description": "Success",  
            "schema": {  
              "type": "array",  
              "items": {  
                "$ref": "#/definitions/Binding.v1"  
              }  
            }  
          },  
          "headers": {  
            "X-Correlation-Id": {  
              "description": "Correlation ID",  
              "type": "string"  
            }  
          }  
        }  
      }  
    }  
  }  
}
```

# http://editor.swagger.io/

Swagger Editor    File ▾    Edit ▾    Generate Server ▾    Generate Client ▾    Supported by SMARTBEAR

INFO:

```
3   description: "This is a sample server Petstore server. You can
    find out more about Swagger at [http://swagger.io](http://swagger.io) or on [irc.freenode.net, #swagger](http://swagger.io/irc/). For this sample, you can use the api key
    `special-key` to test the authorization filters."
4   version: "1.0.0"
5   title: "Swagger Petstore"
6   termsOfService: "http://swagger.io/terms/"
7   contact:
8     email: "apiteam@swagger.io"
9   license:
10    name: "Apache 2.0"
11    url: "http://www.apache.org/licenses/LICENSE-2.0.html"
12 host: "petstore.swagger.io"
13 basePath: "/v2"
14 tags:
15  - name: "pet"
16    description: "Everything about your Pets"
17  - externalDocs:
18    description: "Find out more"
19    url: "http://swagger.io"
20  - name: "store"
21    description: "Access to Petstore orders"
22  - name: "user"
23    description: "Operations about user"
24  - externalDocs:
25    description: "Find out more about our store"
26    url: "http://swagger.io"
27 schemes:
28  - "https"
29  - "http"
30 paths:
31  /pet:
32    post:
```

**pet** Everything about your Pets    Find out more: <http://swagger.io>

<b>POST</b>	/pet	Add a new pet to the store	🔒
<b>PUT</b>	/pet	Update an existing pet	🔒
<b>GET</b>	/pet/findByStatus	Finds Pets by status	🔒
<b>GET</b>	/pet/findByTags	Finds Pets by tags	🔒
<b>GET</b>	/pet/{petId}	Find pet by ID	🔒
<b>POST</b>	/pet/{petId}	Updates a pet in the store with form data	🔒
<b>DELETE</b>	/pet/{petId}	Deletes a pet	🔒
<b>POST</b>	/pet/{petId}/uploadImage	uploads an image	🔒

**store** Access to Petstore orders

9 . 3

Можно генерировать  
спеку из кода (**flask-  
rest-plus**)

Можно часть кода  
вынести в спеку  
(specification)

Можно генерировать  
клиентские  
библиотеки по спеке

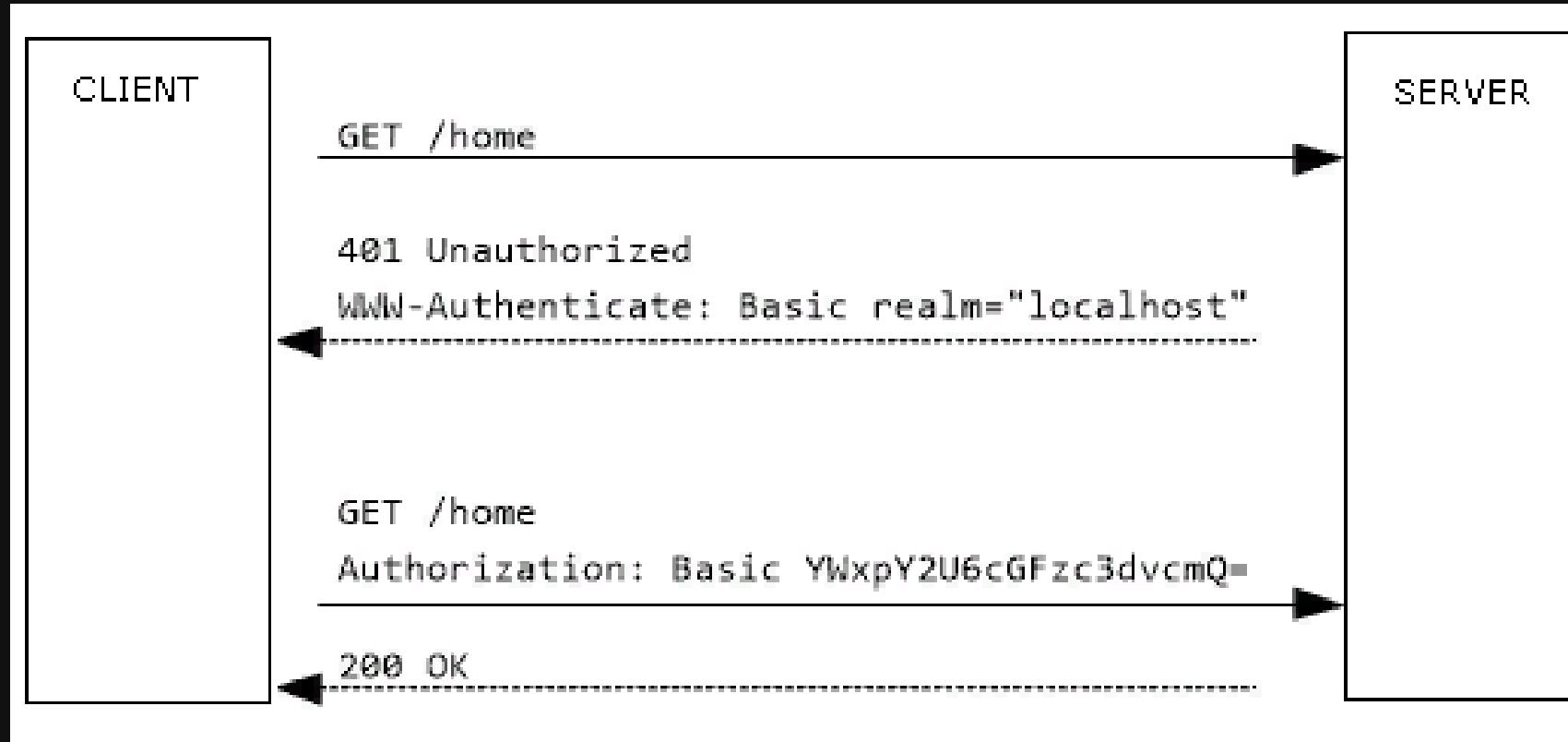
# Аутентификация



# ТОЛЬКО HTTPS!



# Аутентификация через заголовки Basic Auth



# Проблемы

- Нужно везде таскать логин пароль в открытом виде (BASE64)
- Нужно хранить логин/пароль на стороне клиента
- Смена пароля потребует повторной аутентификации во всех системах

# MyPy

Инструмент статического  
анализа типов в python  
коде

<http://mypy-lang.org/>



prototyping.py •

```
1  from typing import List, Type
2
3  class A:
4      def __init__(self):
5          pass
6
7  class B(A):
8      def __init__(self, parent: Type[A]):
9          super().__init__()
10         self.parent = parent
11
12 class C(A):
13     def __init__(self, parent: Type[A]):
14         super().__init__()
15         self.parent = parent
16
17
18 o_a: A = A()
19 o_b: B = B(parent=o_a)
20 o_c: C = C(parent=o_b)
21
22 things: List[Type[A]] = []
23
24 things.append(o_a)
25 things.append(o_b)
26 things.append(o_c)
27
28 |
```



PROBLEMS 5

OUTPUT



Filter. Eg: text, \*\*/\*.ts... ⚙



prototyping.py 5

- ✖ Argument "parent" to "B" has incompatible type "A"; expected "Type[A]" mypy(error)
- ✖ Argument "parent" to "C" has incompatible type "B"; expected "Type[A]" mypy(error)
- ✖ Argument 1 to "append" of "list" has incompatible type "A"; expected "Type[A]" mypy
- ✖ Argument 1 to "append" of "list" has incompatible type "B"; expected "Type[A]" mypy
- ✖ Argument 1 to "append" of "list" has incompatible type "C"; expected "Type[A]" mypy



Ловит ошибки, даже  
если аннотаций нет

Стандартная библиотека и  
многие сторонние уже  
аннотированы

# Type hints

```
def greeting(name: str) -> str:  
    return 'Hello ' + name
```

# Type hints

```
user: User = get_user(...)
```

# Type hints

```
class Container:  
    users: List[User]  
    ...
```

Python не как не  
проверяет!

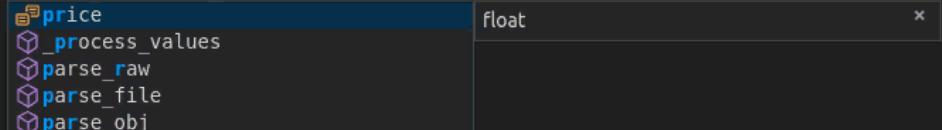
```
In [1]: def x(i: int) -> int:  
...:     pass  
...:
```

```
In [2]: x.__annotations__  
Out[2]: {'i': int, 'return': int}
```

# Документация

# Автодополнение

```
1  from fastapi import FastAPI
2  from pydantic import BaseModel
3
4  app = FastAPI()
5
6
7  class Item(BaseModel):
8      name: str
9      price: float
10     is_offer: bool = None
11
12
13 @app.get("/")
14 def read_root():
15     return {"Hello": "World"}
16
17
18 @app.get("/items/{item_id}")
19 def read_item(item_id: int, q: str = None):
20     return {"item_id": item_id, "q": q}
21
22
23 @app.put("/items/{item_id}")
24 def save_item(item_id: int, item: Item):
25     return {"item_name": item.pr, "item_id": item_id}
```



`int, str, bytes, float,  
bool`

# Union (один из)

```
Union[int, str, float]
```

# Optional (или None)

```
from typing import Optional

def greeting(name: str) -> Optional[str]:
    if name:
        return 'Hello ' + name
    else:
        return None
```

List[ int ]

Dict[ str , int ]

# NoReturn

```
from typing import NoReturn

def inf_loop() -> NoReturn:
    while True:
        pass
```

```
user = cast(User, user)
```

# Aliases

```
from typing import List, Union
Vector = List[Union[int, float]]\n\ndef scale(scalar: float, vector: Vector) -> Vector:
    return [scalar * num for num in vector]
```

# Callable

```
from typing import Callable

def feeder(get_next_item: Callable[[], str]) -> None
    # Body
```

# Generic

```
from typing import Sequence, TypeVar

# Declare type variable
T = TypeVar('T')

# Generic function
def first(l: Sequence[T]) -> T:
    return l[0]
```

```
from typing import Any
```

# Type

```
class User: ...
class BasicUser(User): ...
class ProUser(User): ...
class TeamUser(User): ...

# Accepts User, BasicUser, ProUser, TeamUser, ...
def make_new_user(user_class: Type[User]) -> User:
    ...
    return user_class()
```

# Generator

```
# Generator[YieldType, SendType, ReturnType]
def echo_round() -> Generator[int, float, str]:
    sent = yield 0
    while sent >= 0:
        sent = yield round(sent)
    return 'Done'
```

# Generator

```
def infinite_stream(  
    start: int  
) -> Generator[int, None, None]:  
    while True:  
        yield start  
        start += 1
```

# Iterator

```
def infinite_stream(  
    start: int  
) -> Iterator[int]:  
    while True:  
        yield start  
        start += 1
```

# NamedTuple

```
class Employee(NamedTuple):  
    name: str  
    id: int = 3  
  
employee = Employee('Guido')  
assert employee.id == 3
```

# TYPE\_CHECKING

```
if TYPE_CHECKING:  
    import expensive_mod  
  
def fun(arg: 'expensive_mod.SomeType') -> None:  
    pass
```

# Валидация

Никогда нельзя  
доверять входным  
данным!

# Pydantic

```
from datetime import datetime
from typing import List
from pydantic import BaseModel

class User(BaseModel):
    id: int
    name: str
    signup_ts: datetime
    friends: List[int]
```

# Pydantic

```
external_data = {  
    'id': '123',  
    'signup_ts': '2019-06-01 12:22',  
    'friends': [1, 2, '3']  
}  
  
user = User(**external_data)  
print(user.id)  
print(repr(user.signup_ts))  
print(user.friends)  
print(user.dict())
```

# Pydantic

```
from pydantic import ValidationError

try:
    User(
        signup_ts='broken',
        friends=[1, 2, 'not number'
    )
except ValidationError as e:
    print(e.json())
```

# Модели можно вкладывать друг в друга

```
class Friend(BaseModel):
    id: int
    name: str

class User(BaseModel):
    id: int
    name: str
    signup_ts: datetime
    friends: List[Friend]
```

# User.parse\_raw(json\_str)

# User(...).json()

# Зачем?

Простой, читаемый код

Автодополнение

Домашняя работа

Сервис кино отзывов

# Вопросы?