

# NodeJS

## środowisko i technologia ServerSide

---

PAWEŁ ŁUKASZUK





Knižovny SSSR  
Knihovny Vostoku

18

Knihovny v dělnictví  
Knihovny - dělníci

19

Knihovny - dělníci  
Knihovny - dělníci

20

Knihovny - dělníci  
Knihovny - dělníci

21

Knihovny - dělníci  
Knihovny - dělníci

22

Knihovny - dělníci  
Knihovny - dělníci

23

Knižovny v dělnictví  
Knihovny v dělnictví

24

Knižovny v dělnictví  
Knihovny v dělnictví

25

Knižovny v dělnictví  
Knihovny v dělnictví

26

Knižovny v dělnictví  
Knihovny v dělnictví

27

Knižovny v dělnictví  
Knihovny v dělnictví

28

Knižovny v dělnictví  
Knihovny v dělnictví

29

Knižovny v dělnictví  
Knihovny v dělnictví

30

Knižovny v dělnictví  
Knihovny v dělnictví

31

Knižovny v dělnictví  
Knihovny v dělnictví

32

Knižovny v dělnictví  
Knihovny v dělnictví

33

Knižovny v dělnictví  
Knihovny v dělnictví

34

Knižovny v dělnictví  
Knihovny v dělnictví

35

Knižovny v dělnictví  
Knihovny v dělnictví

36

Knižovny v dělnictví  
Knihovny v dělnictví

37

Knižovny v dělnictví  
Knihovny v dělnictví

38

Knižovny v dělnictví  
Knihovny v dělnictví

39

Knižovny v dělnictví  
Knihovny v dělnictví

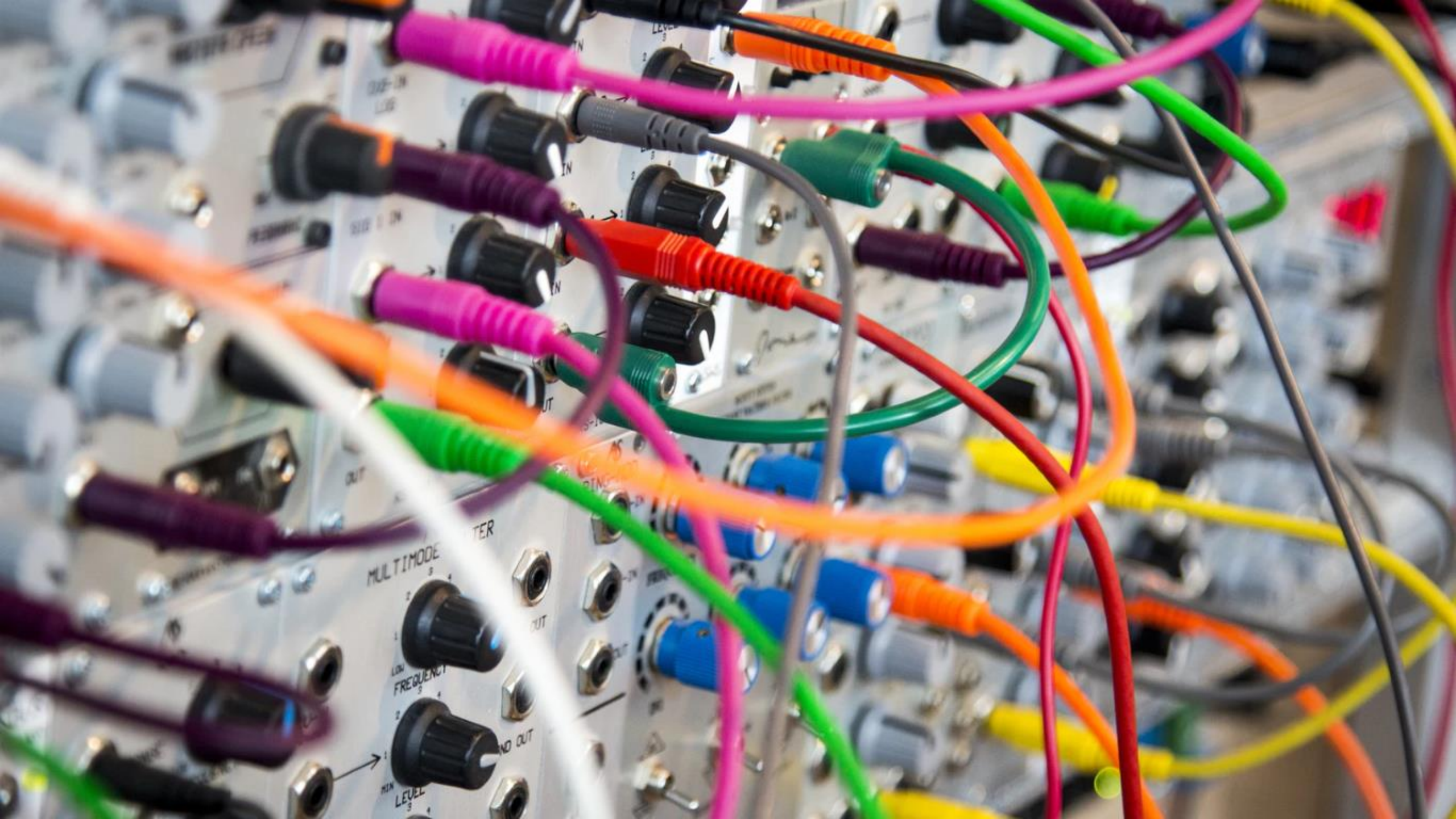
40

Knižovny v dělnictví  
Knihovny v dělnictví

41







# API



# API

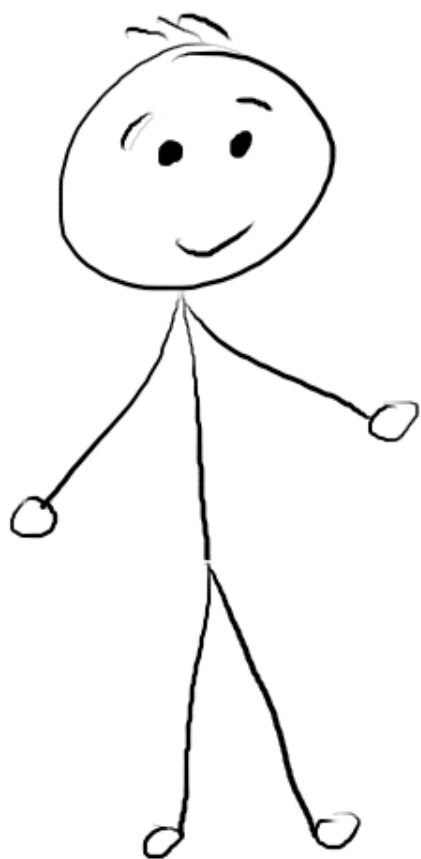
---

## Application Programming Interface

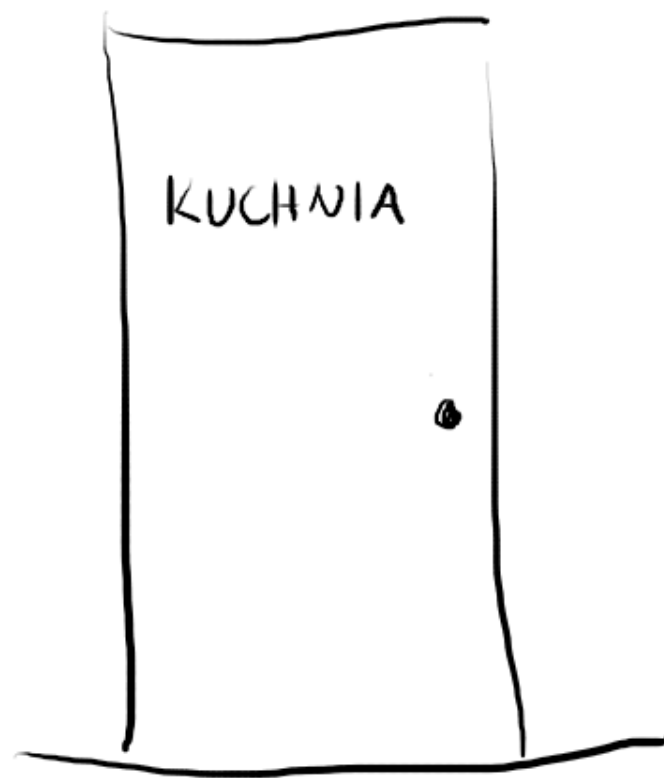
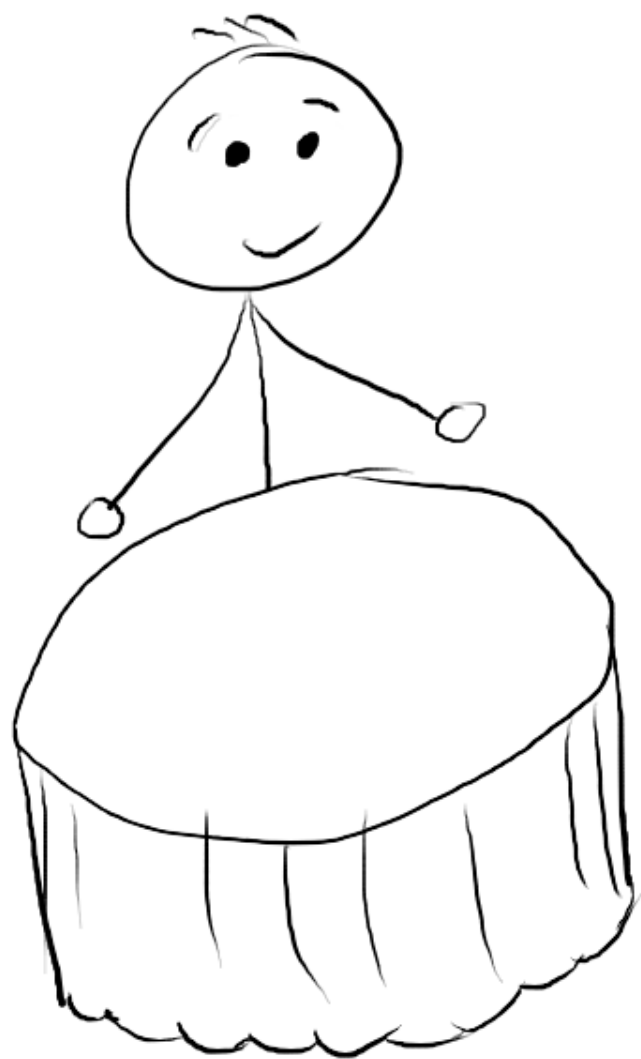
A set of rules and their description of how computer programs communicate with each other.

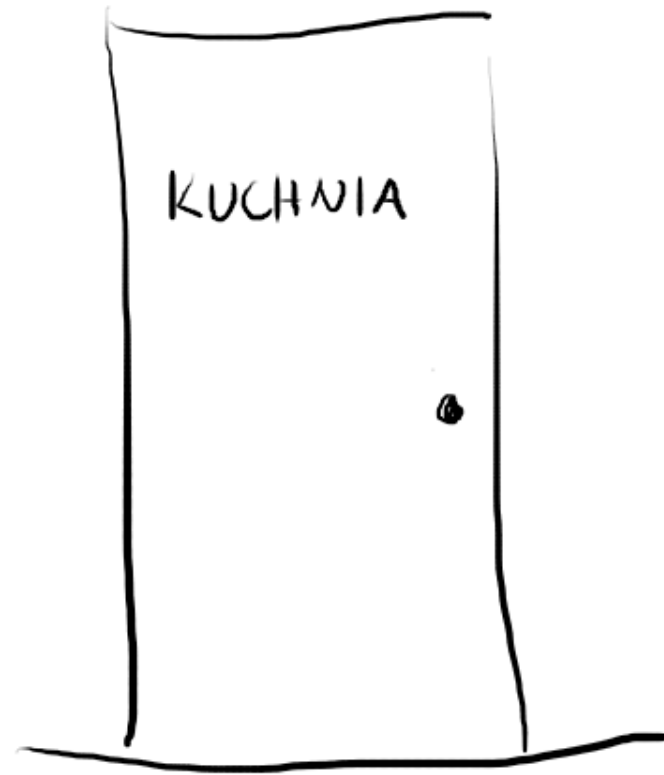
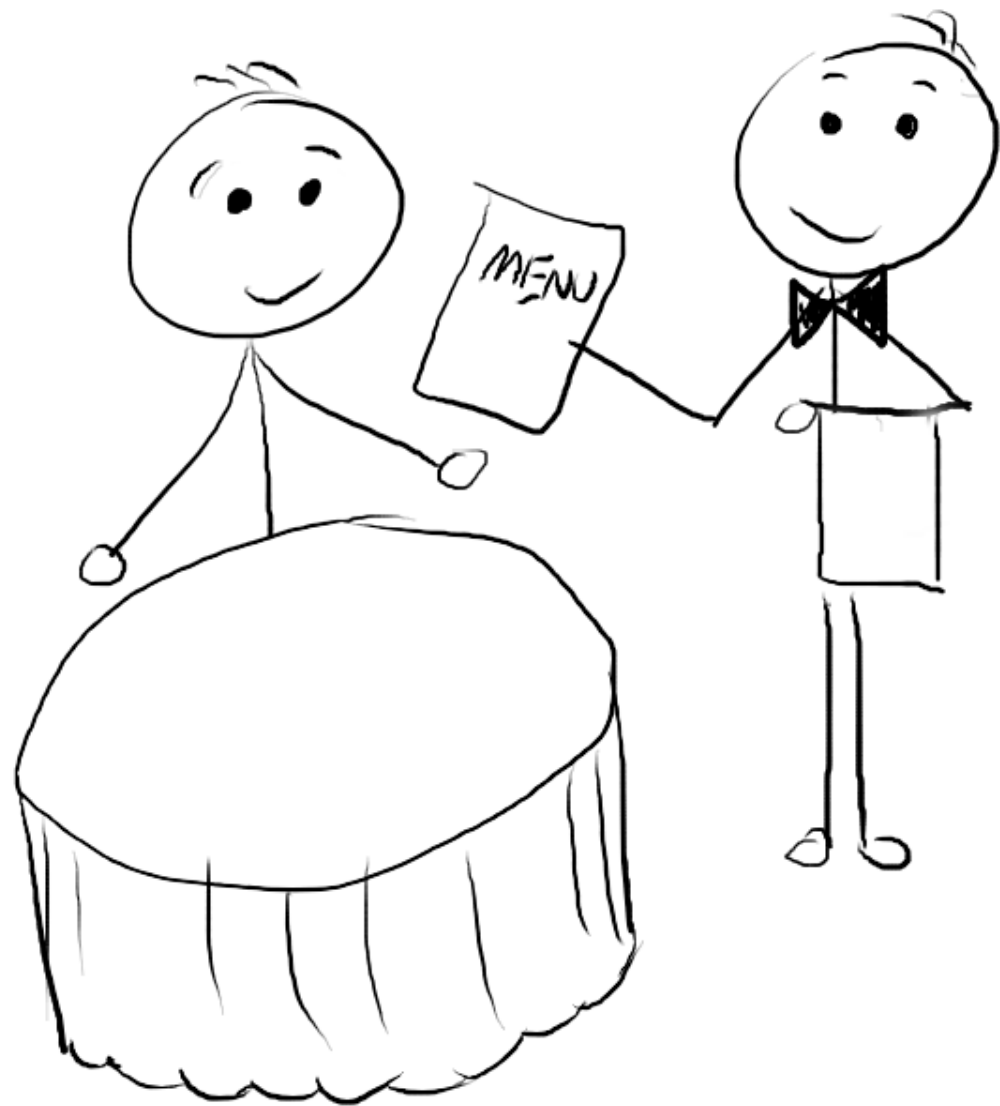
The goal of an application programming interface is to provide the appropriate specifications of subroutines, data structures, object classes, and required communication protocols.



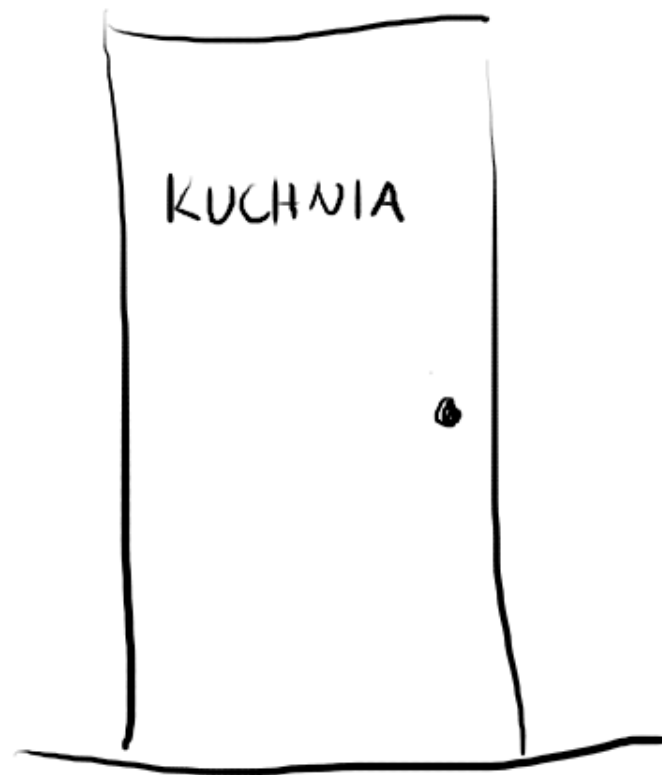
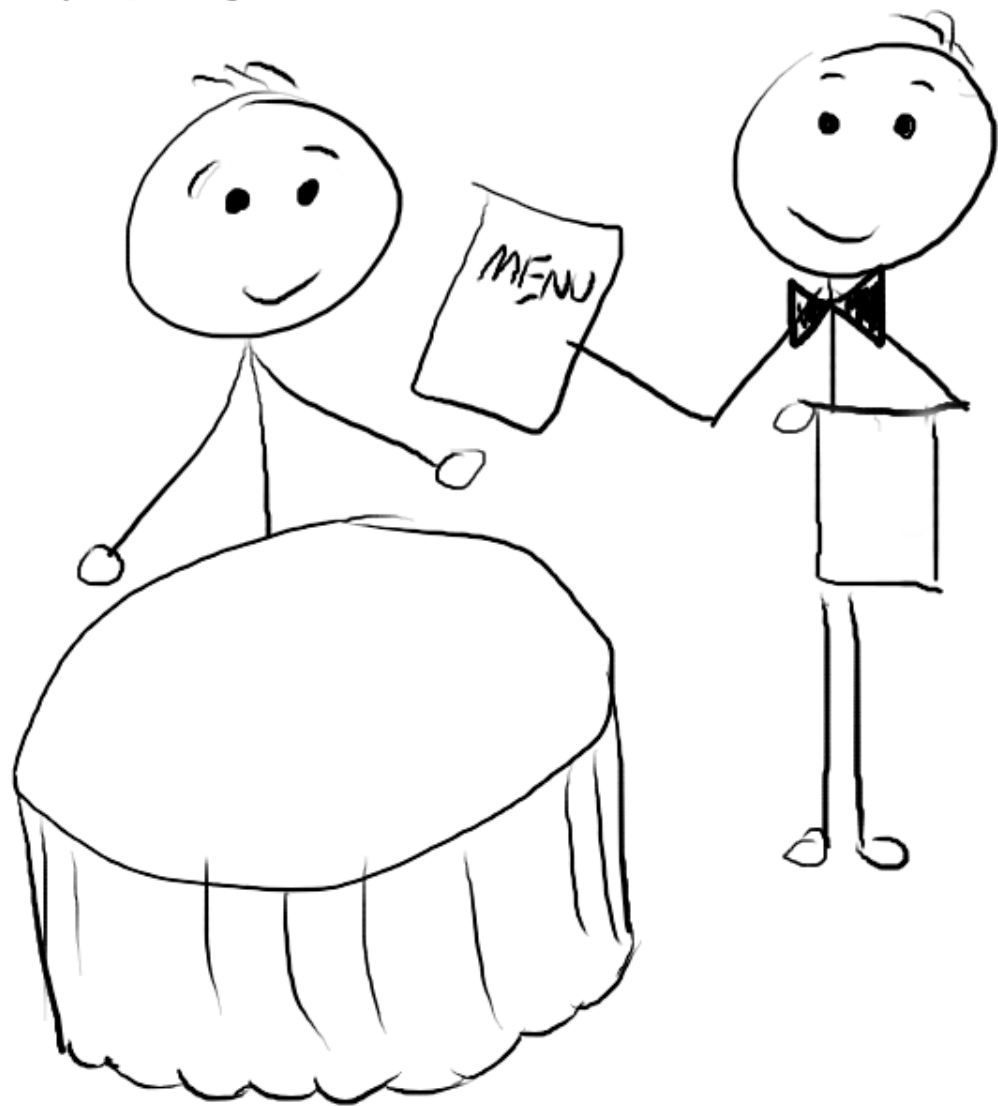








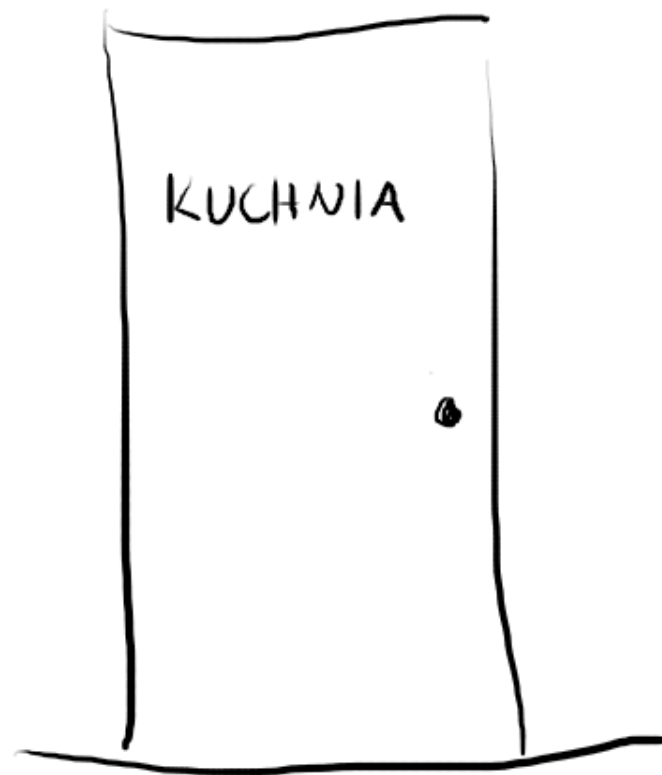
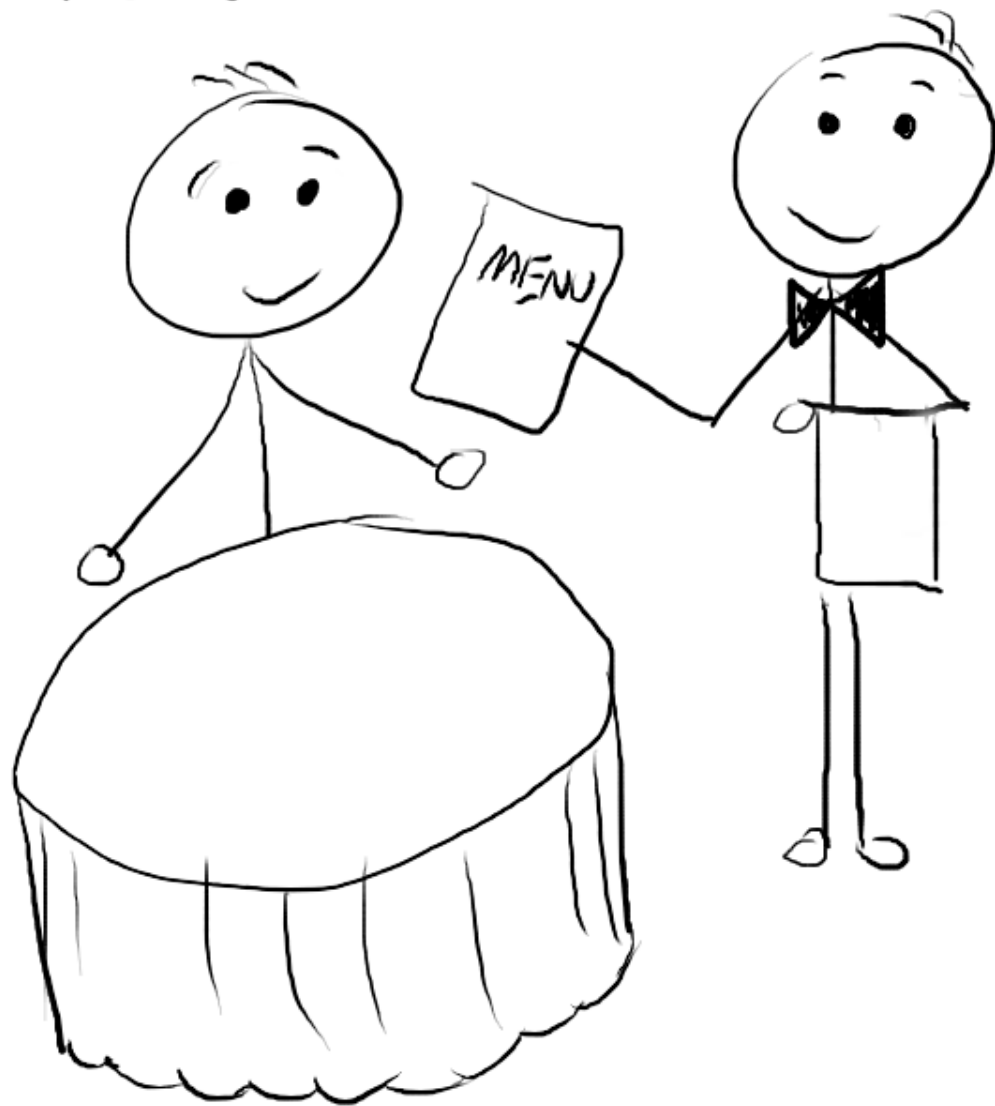
KLIENT





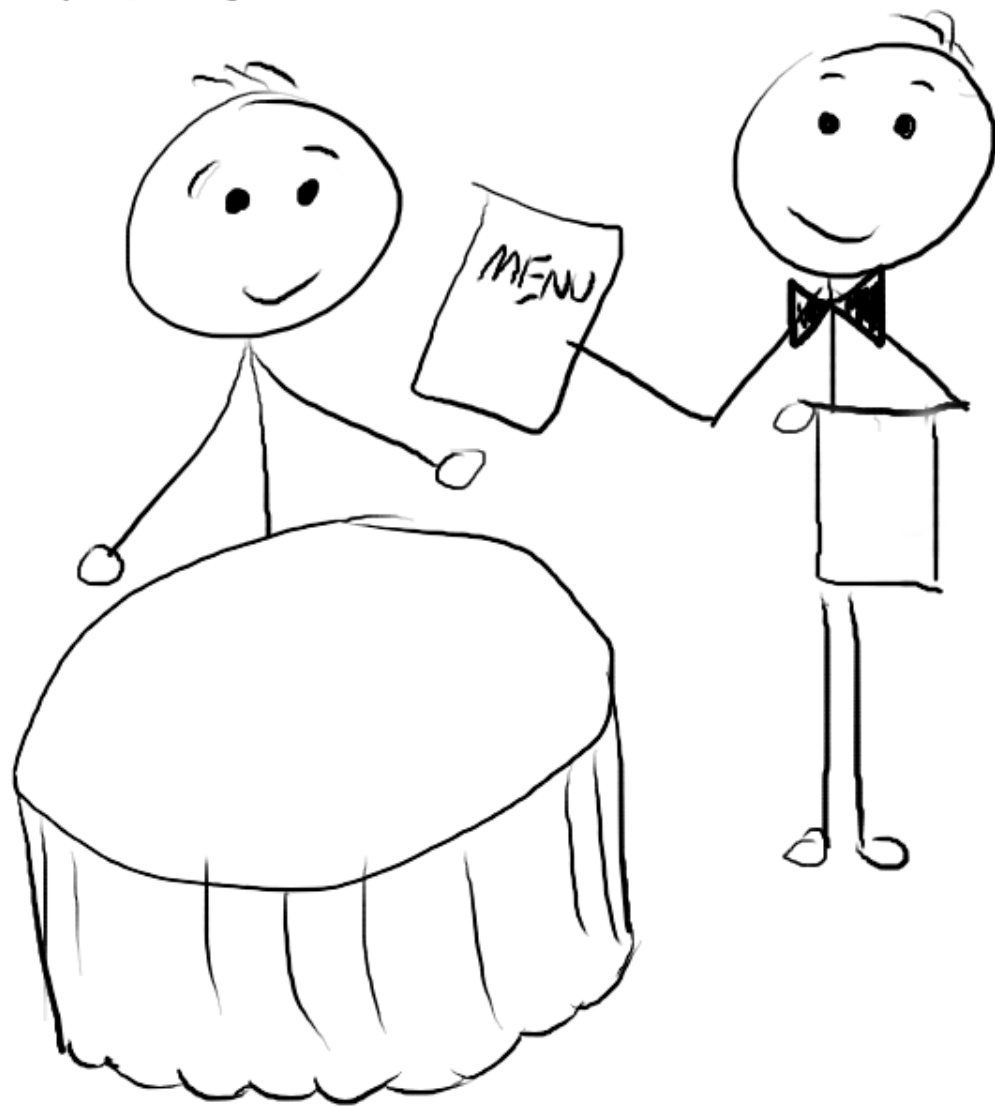
KLIENT

API

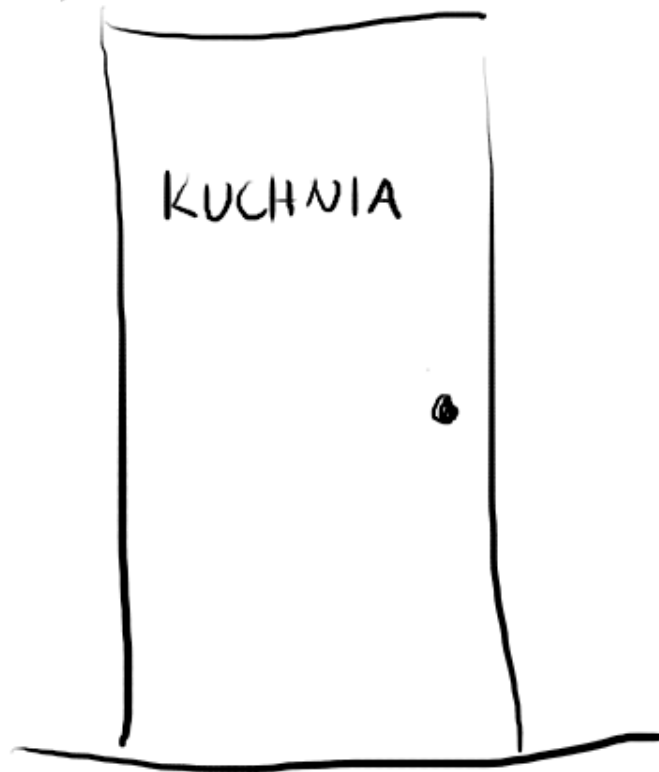


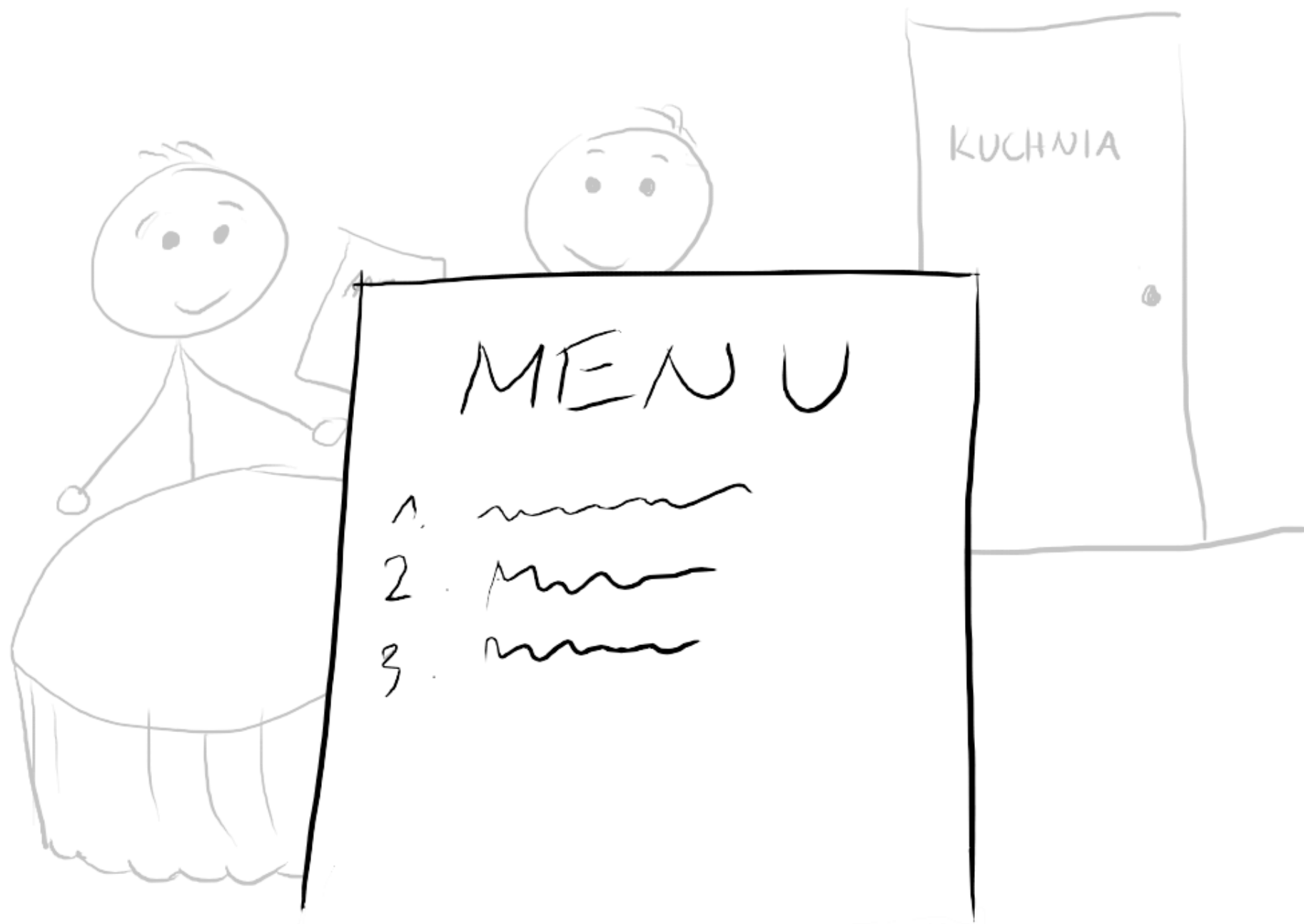
KLIENT

API





SERWER

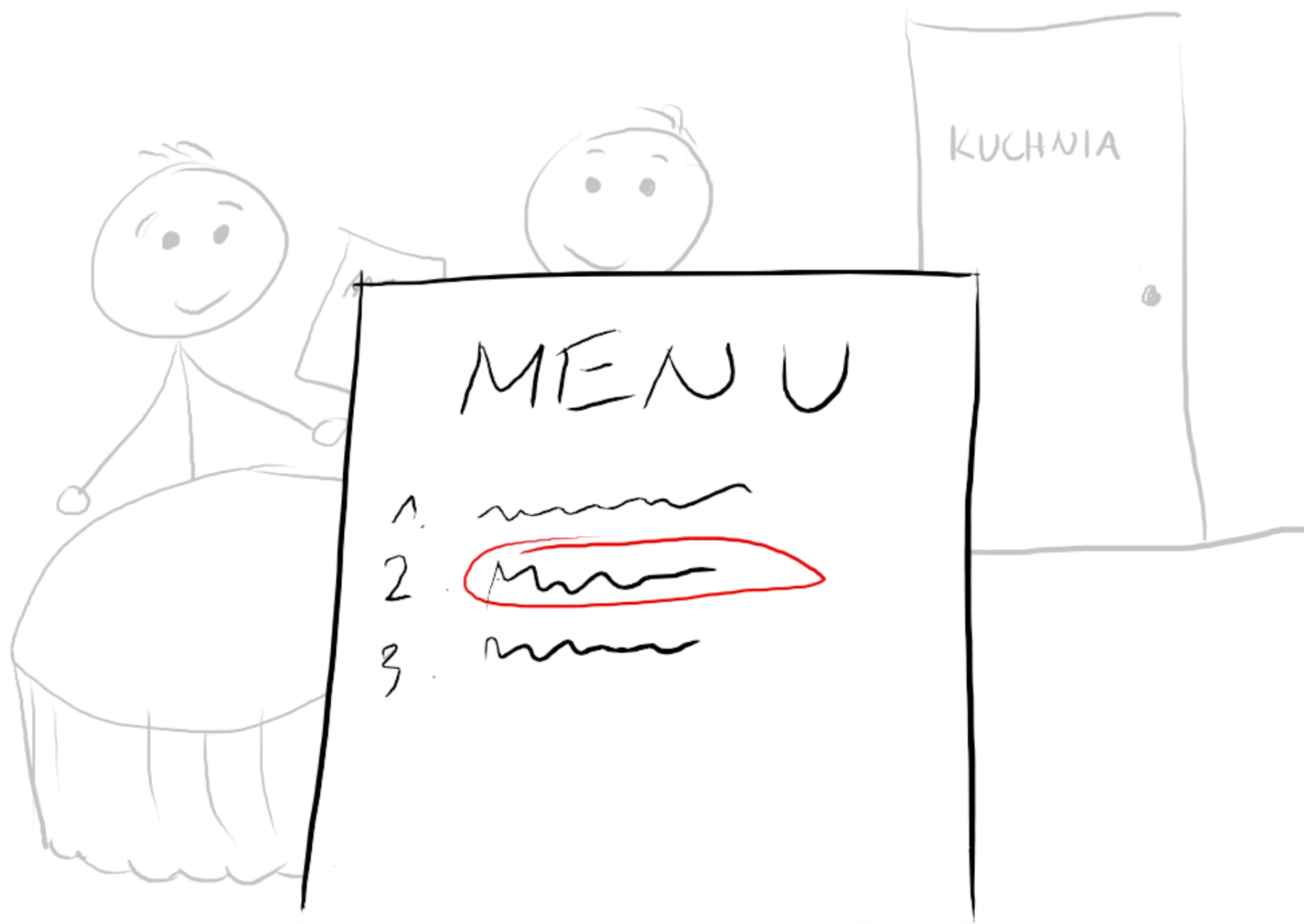


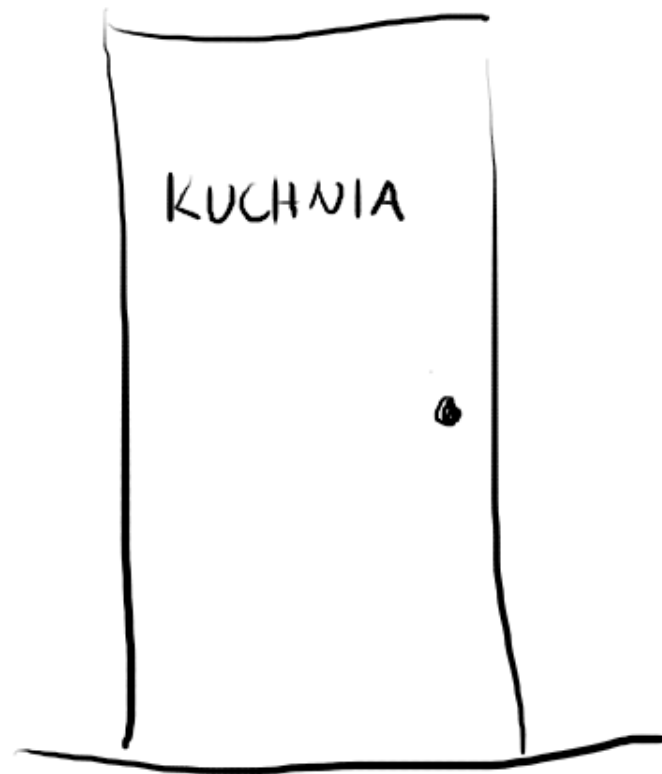
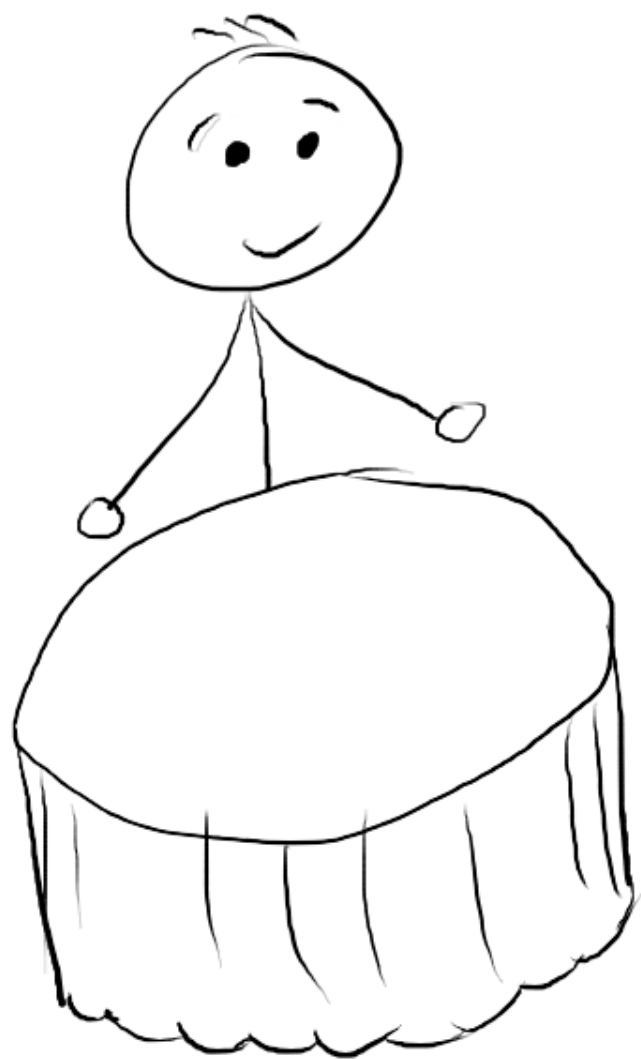


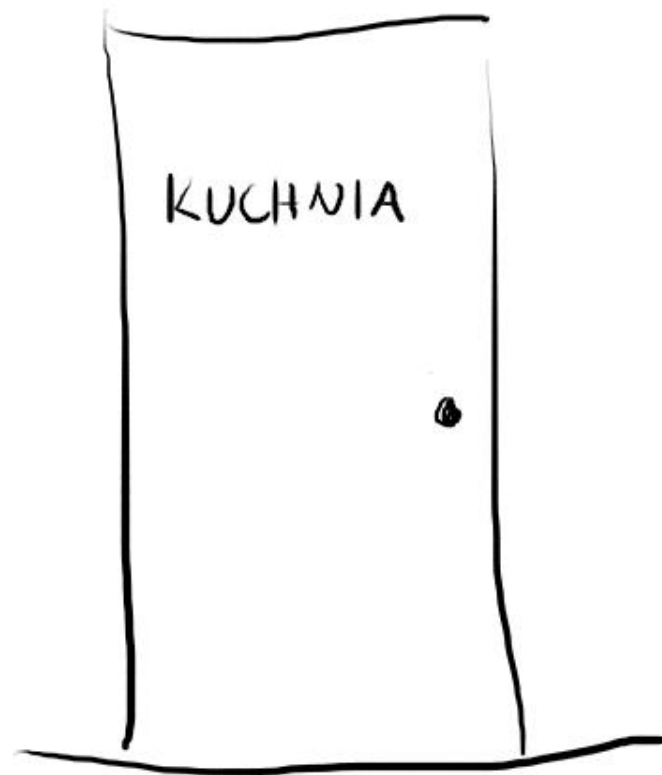
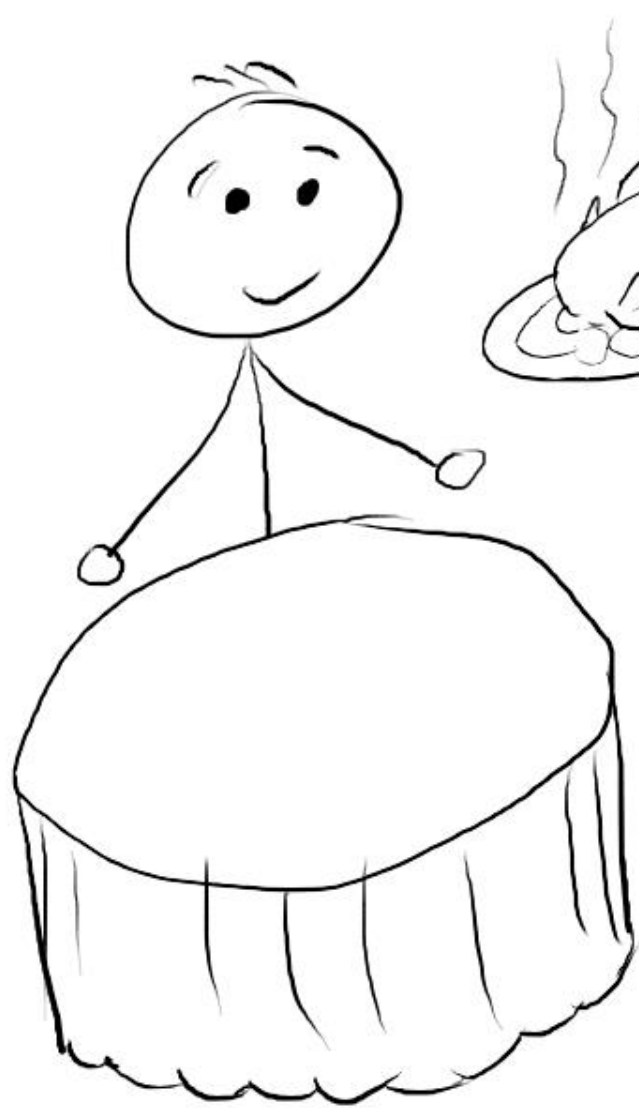
# MENU

1. 
2. 
3. 

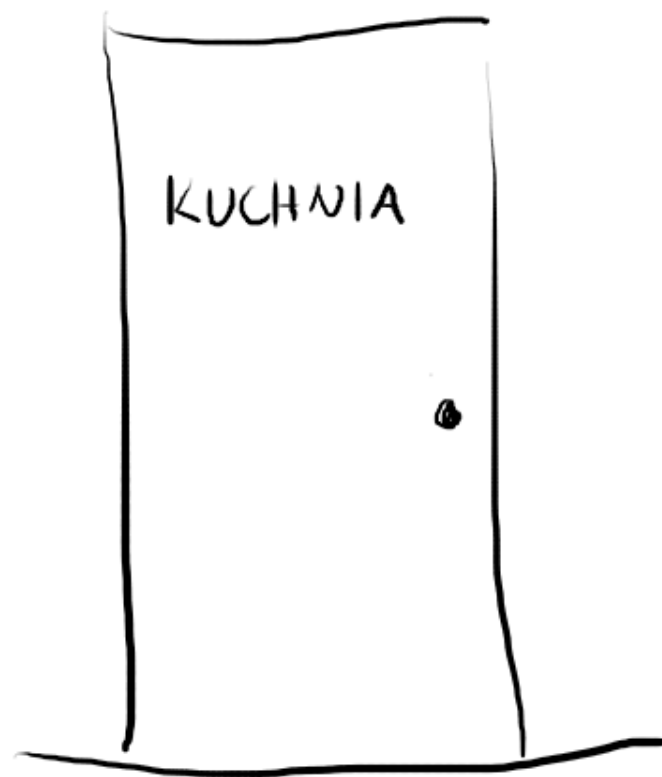
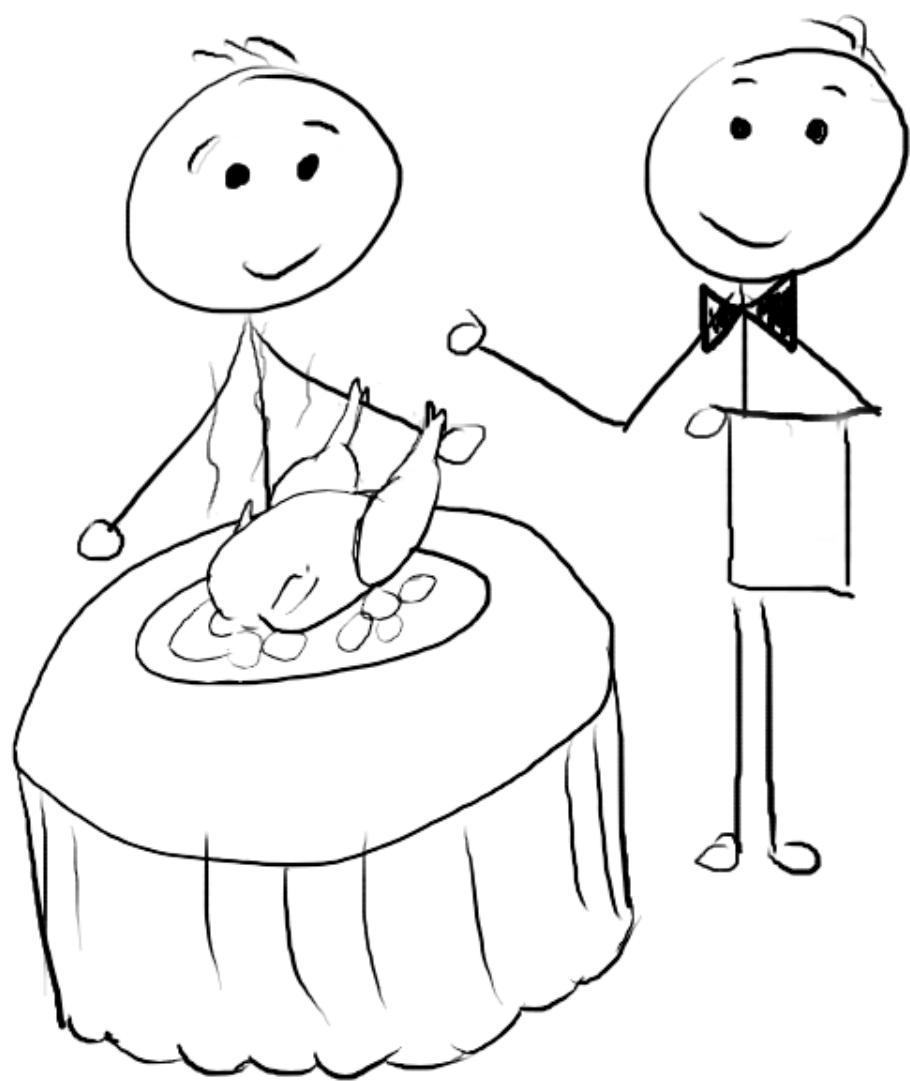


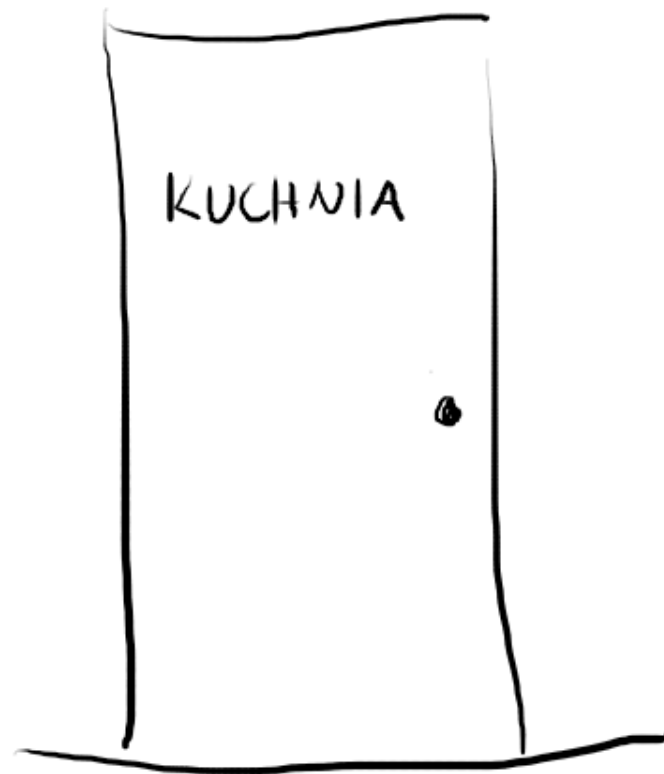












# API Styles

# RPC

---

Remote Procedure Call approach has many meanings and many forms.

In the case of the web, an RPC call allows you to manipulate data over the HTTP protocol.

RPC in the web server sense: WYGOPIAO(What You GET Or POST Is An Operation).





# RPC

---

**The structure of RPC communication is not predetermined.**

Endpoint should contain the name of the procedure you want to call on the remote server.

Standard accepts only two communication methods, GET and POST.



# RPC - example

---

GET /getUsers?someType=abc

POST /saveNewUser

```
{  
  "name": "Jan",  
  "lastName": "Nowak"  
}
```

```
attachEvent("o  
boolean Number  
_={};function  
t[1])===!1&&e.  
?o=u.length:r&  
nction(){retur  
re:function(){  
ending",r={sta  
romise)?e.prom  
dd(function(){  
=0,n=h.call(ar  
(r),l=Array(r)  
</table></tab  
/TagName("input  
test(r.getAttr
```

# RPC

---

Standard based on RPC

- JSON-RPC
- JSON-XML
- SOAP



# gRPC

---

Modern open source high performance Remote Procedure Call (RPC) framework that can run in any environment.

It can efficiently connect services in and across data centers with pluggable support for load balancing, tracing, health checking and authentication.

It is also applicable in last mile of distributed computing to connect devices, mobile applications and browsers to backend services.

<https://grpc.io>

<https://grpc.io/docs/languages/node/>





# gRPC

---

The main usage scenarios

- Efficiently connecting polyglot services in microservices style architecture
- Connecting mobile devices, browser clients to backend services
- Generating efficient client libraries



# gRPC

---

Core features that make it awesome

- Idiomatic client libraries in 10 languages
- Highly efficient on wire and with a simple service definition framework
- Bi-directional streaming with http/2 based transport
- Pluggable auth, tracing, load balancing and health checking



# GraphQL

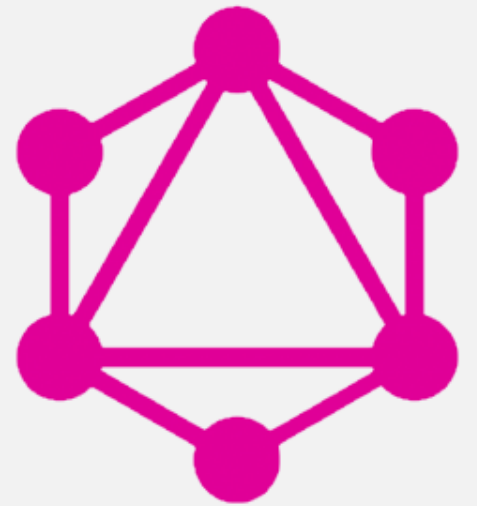
---

GraphQL is a query language for APIs and an execution environment for completing queries with existing data.

It provides a complete and understandable description of the data in the API.

Allows you to retrieve multiple resources in a single query.

<https://graphql.org>

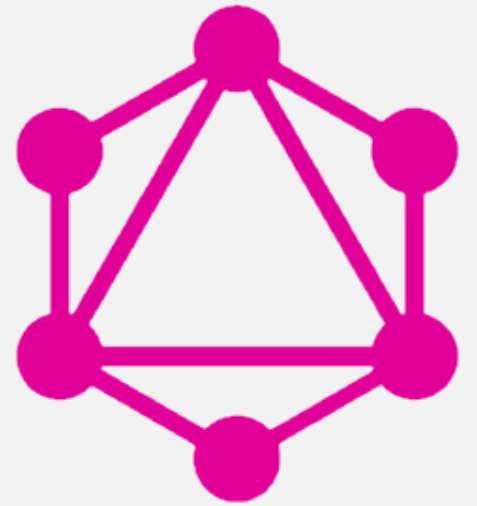


# GraphQL

---

GraphQL gives you the ability to ask for what a particular client needs, without any additional data.

It connects to a single address to which it sends the appropriate query, which is processed by the server.



# GraphQL example

---

Type	Query	Result
<pre>type Query {   me: User }</pre>	<pre>{   me {     name   } }</pre>	<pre>{   "me": {     "name": "Luke Skywalker"   } }</pre>
<pre>type User {   id: ID   name: String }</pre>	<pre>}</pre>	<pre>}</pre>

# REST

---

Representational State Transfer

REST - changing state through representations

A standard that defines the principles of API design.

It is based on the HTTP protocol.

With a REST API, we can expose data as resources, which we manipulate using appropriate HTTP protocol methods.





# REST

---

A style of application design in which the client queries a particular resource and the server responds only with a representation of the resource (or information about it) formatted in an appropriate way, such as JSON.

When API is designed with REST principles we call it RESTful



# REST

---

Methods provided by HTTP protocol can be easily assigned to operations CRUD(Create/Read/Update/Delete) on the object.

C => Create => POST

R => Read => GET

U => Update => PUT/PATCH

D => Delete => DELETE



# REST example

---

GET /users

GET /users/:id

POST /users

PUT /users/:id

DELETE /users/:id



# REST principles

---

Uniform interface - The interface should provide standardized communication between the client and the server

Client-server - A clear separation between client-side and server-side applications

Stateless - Each request should contain a complete set of information to properly handle the request





# REST principles

---

Cacheable - API should support data caching to improve performance

Layered system - client connecting to the server should not know what is happening on the other side

Code on demand (optional) - API provides a possibility to send a piece of code that can be executed on the client side



# REST - resources

---

Resource is any information that has a name can be a resource if it:

- is a noun, e.g.: user, post, comment
- is unique and points to a specific thing
- can be represented as data
- it has at least one URI address under which it is available





# REST - naming

Resources should be created in such a way that they represent an object. This allows multiple actions to be performed on a single resource.

## GET /users

## POST /users

# PATCH /users

DELETE /users/:id



# REST - naming

---

`example.org/api/notes` – all notes

`example.org/api/notes/10` – note with `Id=10`

`example.org/api/notes/titles` – titles of all notes

`example.org/api/users/1/notes` – notes created by user with `Id=1`

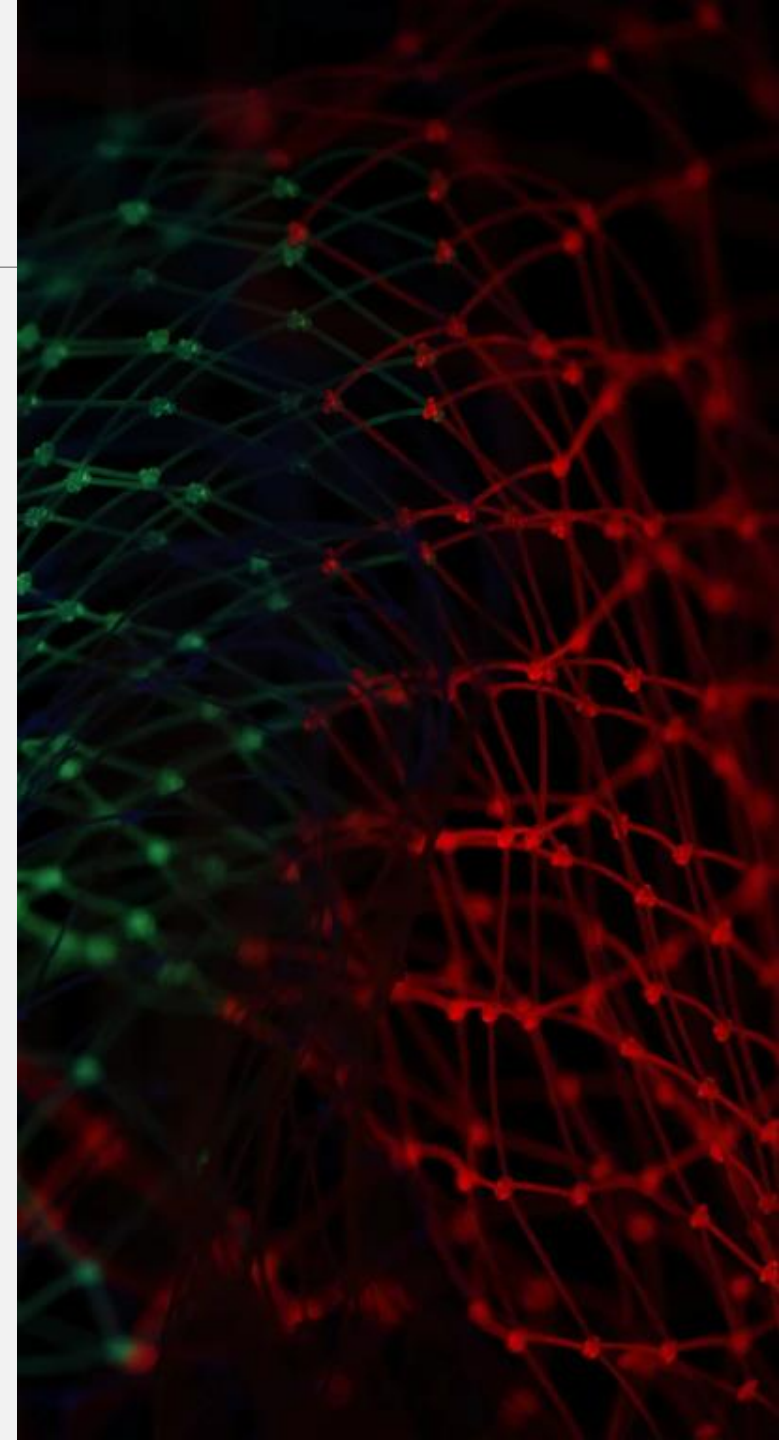
`example.org/api/users/1/notes/10` – note with `Id=10` created by user with `Id = 1`



# REST - data types

---

- JSON
- YAML
- XML
- ...



# REST - versioning

---

Since we cannot change address of resource we must consider other ways:

- URI prefix
- URI parameter
- Headers



# REST - versioning

---

URI prefix (most used)	URI parameter	Headers (most RESTful way)
/v1/users /v2/users /v3/users	/users?api=version1 /users?api=version2 /users?api=version3	GET /users HTTP/1.1 Accept: application/vnd.username.v1+json  GET /users HTTP/1.1 Accept: application/vnd.username.v2+json



# REST - HATEOAS

---

HATEOAS - Hypertext As The Engine Of Application State

It provides us with the knowledge of navigating the API without knowing the specific addresses (in addition to the result, links are sent to resources that we can access by the way).



# REST - HATEOAS

---

The client should really only know what it wants to do, not how to do it.

Self-documenting

No need to harrcode endpoint addresses, the client itself can retrieve information from the body of the response.





# REST - HATEOAS - example

```
{  
  "title": "C_tech",  
  "message": "TDD!",  
  "_links": {  
    "self": {  
      "href": "http://localhost:8080/posts/3"  
    },  
    "posts": {  
      "href": "http://localhost:8080/posts"  
    },  
    „comments": {  
      "href": "http://localhost:8080/posts/3/comments"  
    },  
  }  
},  
}
```



# REST - good practices

---

- Documentation – you won't be the only one using your API
- URI – resource hierarchy
- Consistency – you can't use PUT and POST interchangeably
- Use CRUD
- Response codes – should mean what they are supposed to mean
- Versioning – changes that effect using must be introduced in new version



# gRPC vs GraphQL vs REST



grpc rest graphql



[Wszystko](#)

[Grafika](#)

[Wideo](#)

[Wiadomości](#)

[Zakupy](#)

[Więcej](#)

[Ustawienia](#)

[Narzędzia](#)

Około 677 000 wyników (0,44 s)

<https://www.redhat.com> › architect ▼ [Tłumaczenie strony](#)

## An Architect's guide to APIs: SOAP, REST, GraphQL, and gRPC

2 paź 2020 — **gRPC** is a data exchange technology developed by Google and then later made open-source. Like **GraphQL**, it's a specification that's ...

<https://nordicapis.com> › when-to-us... ▼ [Tłumaczenie strony](#)

## When to Use What: REST, GraphQL, Webhooks, & gRPC ...

21 sie 2018 — Comparing Use Cases For **REST**, **GraphQL**, Webhooks, and **gRPC** · **REST**: A stateless architecture for data transfer that is dependent on ...

<https://medium.com> › devops-dudes ▼ [Tłumaczenie strony](#)

## GraphQL vs REST vs gRPC. Choosing the right protocol for ...

20 lip 2020 — **REST** defines interactions through standardized terms in its requests, **GraphQL** runs requests against a created schema to fetch exactly what is ...

<https://technologyrivers.com> › blog ▼ [Tłumaczenie strony](#)

## REST Vs GRPC Vs GraphQL - Technology Rivers

# gRPC, GraphQL, REST

---

## gRPC

- exact and wicked fast

## GraphQL

- flexible

## REST

- most popular one
- easy to start (but hard to master)

