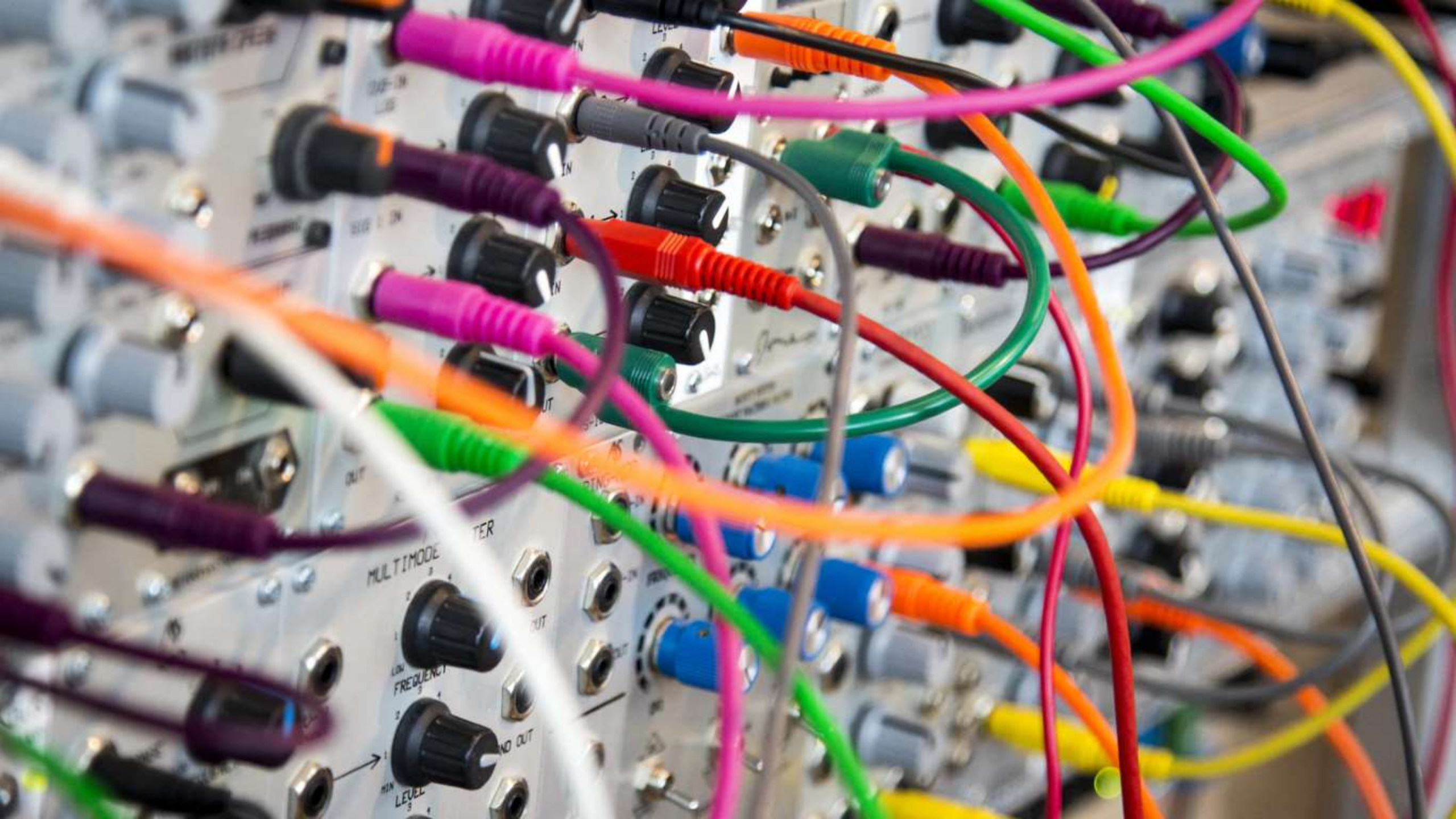
NodeJS

środowisko i technologia ServerSide

PAWEŁ ŁUKASZUK

Rok akademicki: 2023/2024, Semestr: 2, Wykład: 5


mongoDB.


MULTIMODE
FREQUENCY
LEVEL
OUT

API

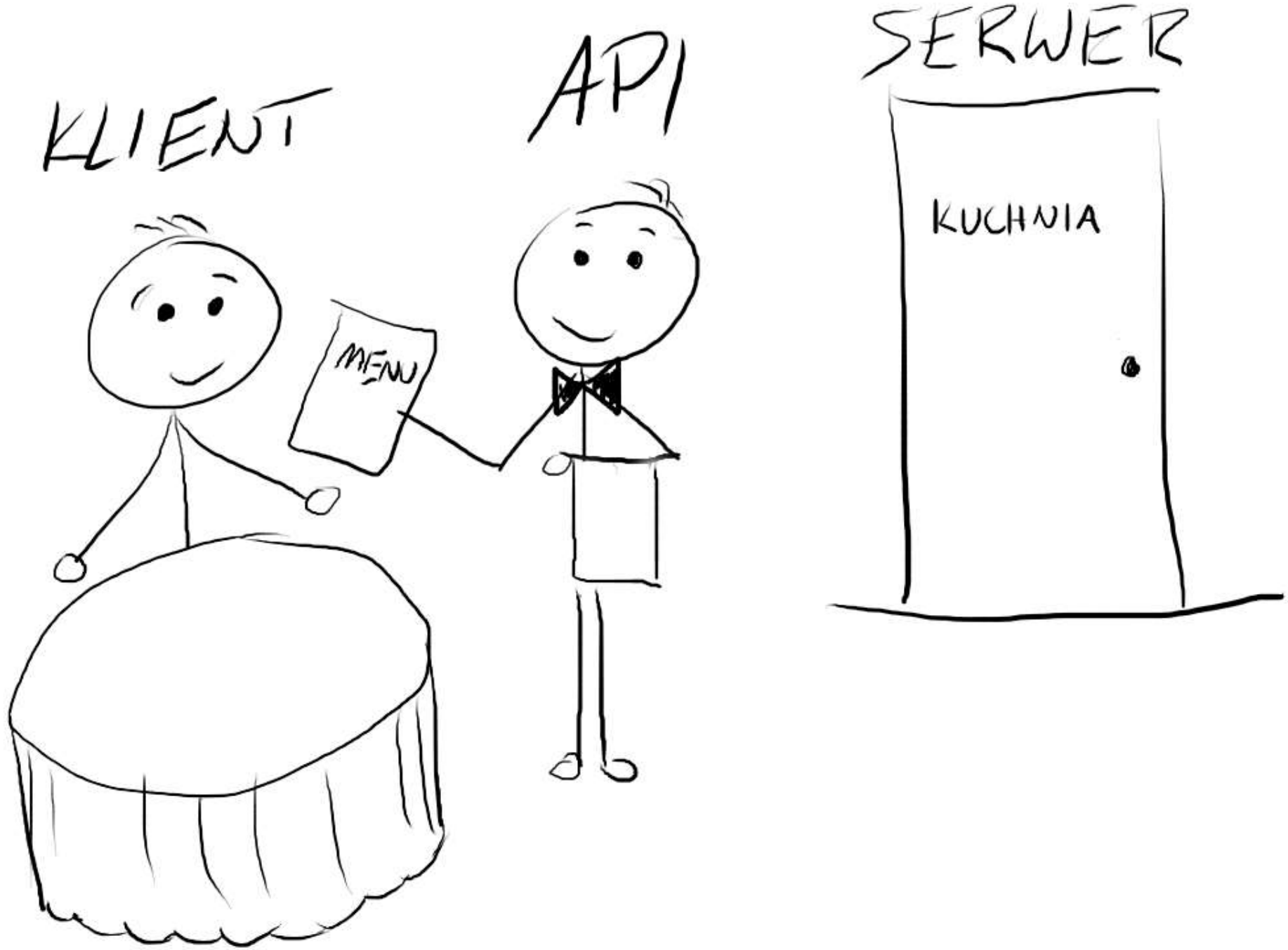
API

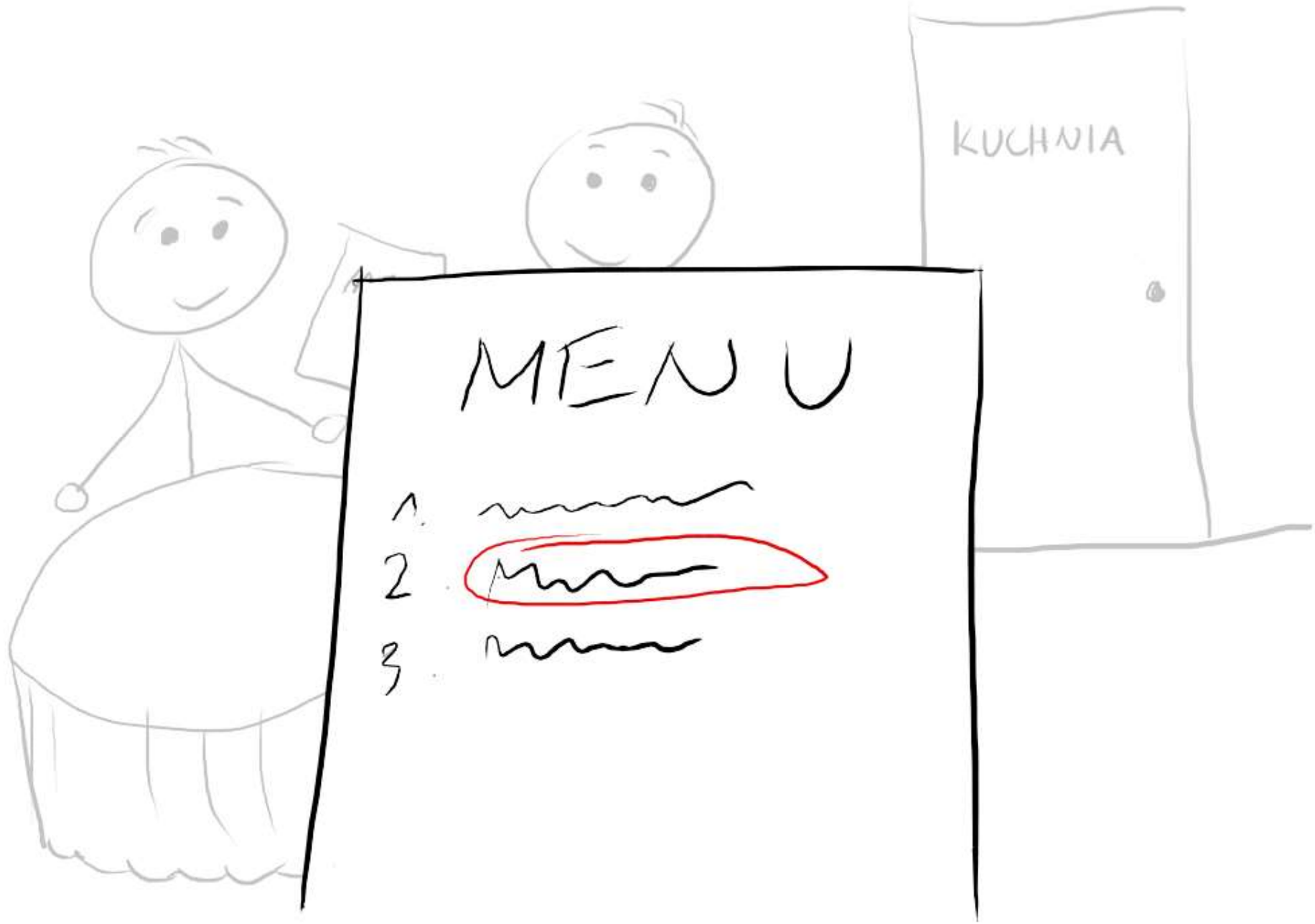
Application Programming Interface

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.


MENU
KUCHNIA


KLIENT
API
SERWER
MENU
KUCHNIA


KUCHNIA
MENU
1.
2
3


KUCHNIA


KUCHNIA

API STYLES

- RPC / SOAP
- gRPC
- GraphQL
- REST

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

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

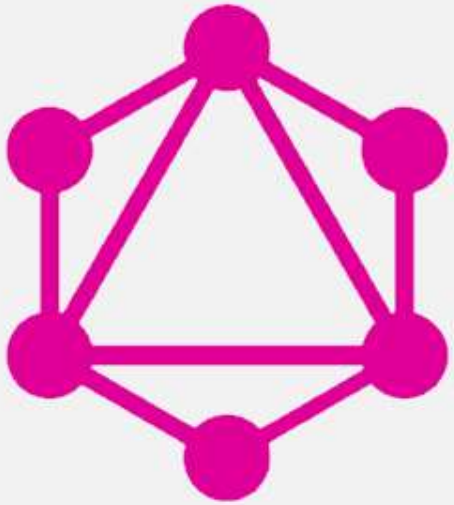
GraphQL

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.

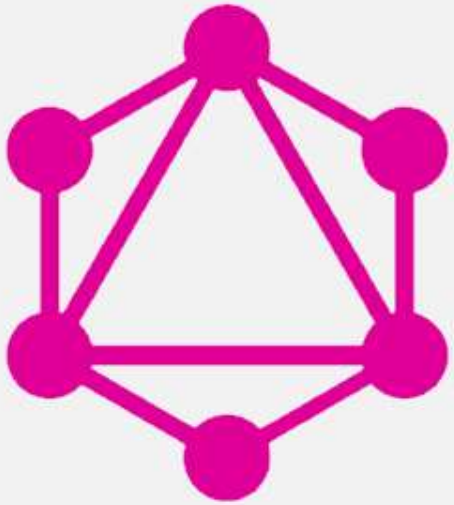
https://graphql.org



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

```
type Query {
  me: User
}

type User {
  id: ID
  name: String
}
```

Query

```
{
  me {
    name
  }
}
```

Result

```
{
  "me": {
    "name": "Luke Skywalker"
  }
}
```

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 - 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 under which it is available

REST example

GET /users

GET /users/:id

GET /users/:id/addresses/: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 - 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

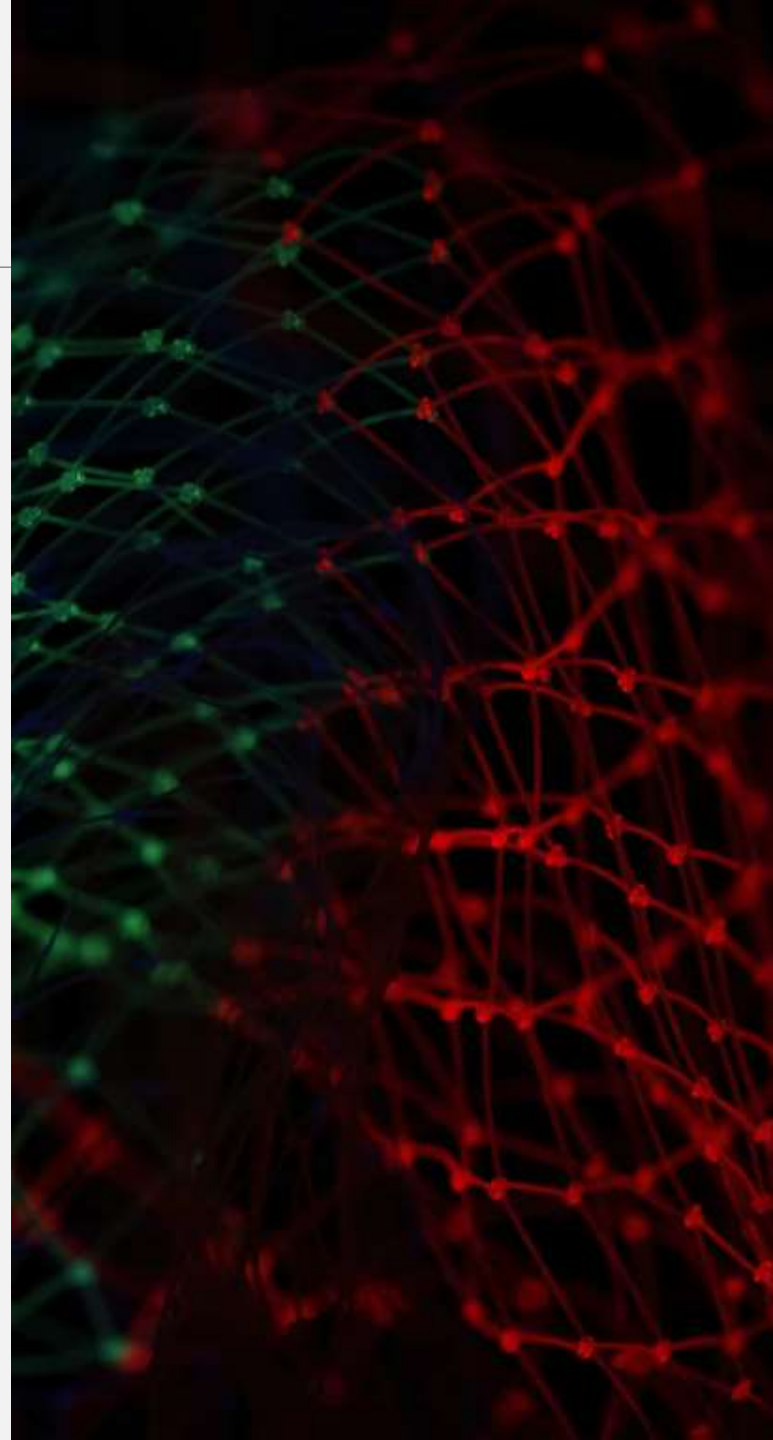
example.org/api/notes/titles – titles of all notes

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

example.org/api/users/1/notes/5 – note with Id=5 created by user with Id = 1

REST - data types

- JSON
- YAML
- XML
- ...



REST - versioning

Usually we cannot change address of resource (backwards compatibility)

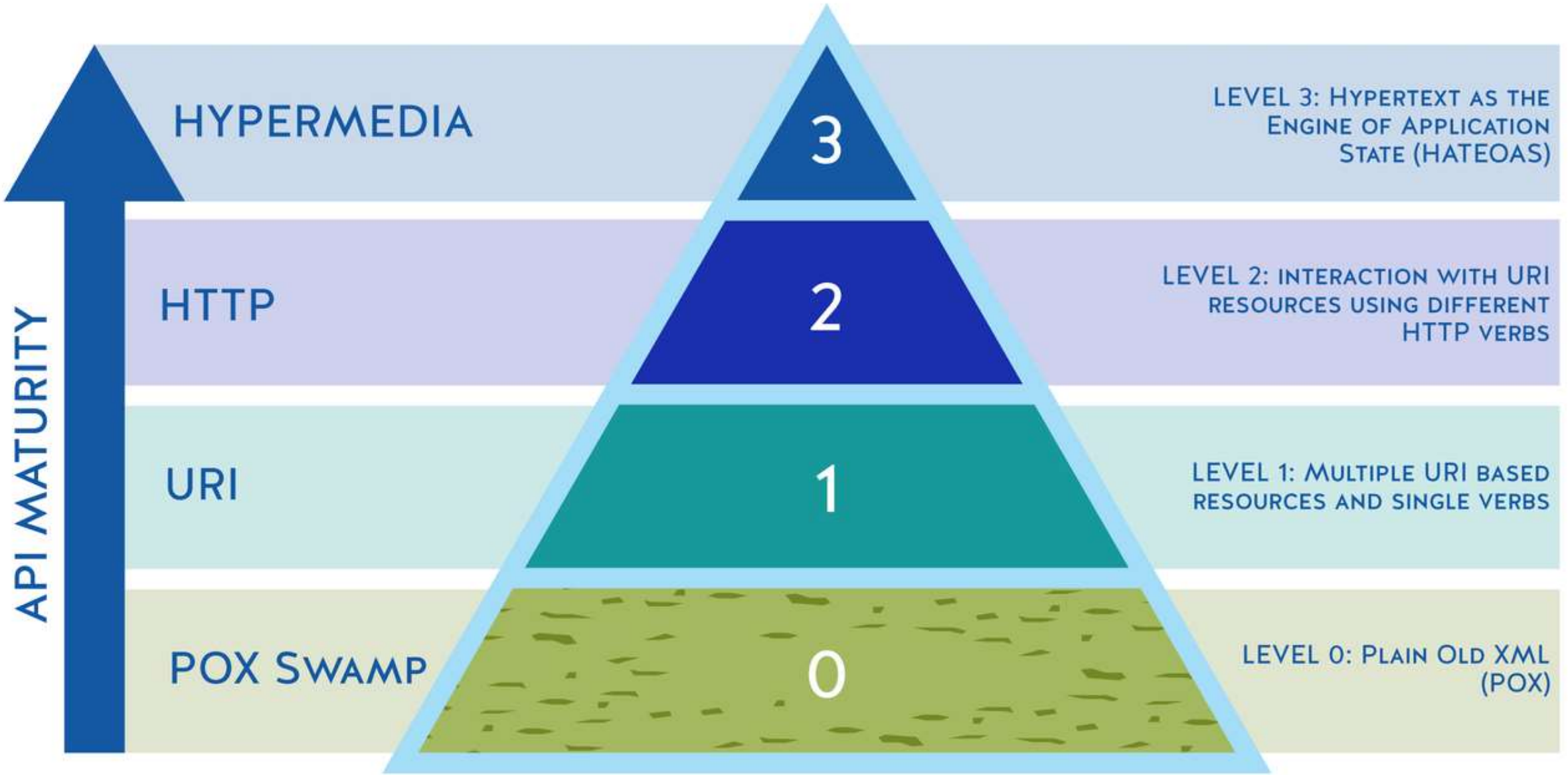
so we must consider other ways:

- 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

RICHARDSON MATURITY MODEL



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 hardcode 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 end users must be introduced in new version

REST - library

- https://en.wikipedia.org/wiki/Representational_state_transfer
- https://technostacks.com/blog/rest-api-development
- https://bykowski.pl/rest-api-efektywna-droga-do-zrozumienia/
- https://restfulapi.net
- https://github.com/microsoft/api-guidelines/blob/vNext/azure/Guidelines.md
- https://www.amazon.com/REST-Practice-Hypermedia-Systems-Architecture/dp/0596805829/

gRPC vs GraphQL vs REST

grpc rest graphql

Wszystko | Grafika | Wideo | Wiadomości | Zakupy | Więcej | Narzędzia

Około 1 310 000 wyników (0,32 s)

https://www.infoq.com › podcasts ▾ Tłumaczenie strony

API Showdown: REST vs. GraphQL vs. gRPC – Which Should ...

17 sty 2022 — The discussion covers some of the pros and cons of **GraphQL** and **gRPC**, and why you might use them instead of a **RESTful** API.

https://blog.logrocket.com › graphq... ▾ Tłumaczenie strony

GraphQL vs. gRPC vs. REST: Choosing the right API

5 kwi 2022 — **gRPC** supports full-duplex streaming out of the box, which makes it suitable for features like video and voice calls. With **REST** on the other hand ...

https://speedscale.com › 2021/07/20 ▾ Tłumaczenie strony

Choosing an API technology: gRPC, REST, GraphQL

20 lip 2021 — Introduced by Facebook in 2015, **GraphQL** represents a different line of API tech evolution. Whereas **gRPC** focuses on blazing fast service-to- ...

https://nordicapis.com › Blog ▾ Tłumaczenie strony

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

21 sie 2018 — **REST**: A stateless architecture for data transfer that is dependent on hypermedia. - **gRPC**: A nimble and lightweight system for requesting data.

https://medium.com › devops-dudes ▾ Tłumaczenie strony

GraphQL vs REST vs gRPC - Medium

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://konghq.com › Blog ▾ Tłumaczenie strony

When to Use REST vs. gRPC vs. GraphQL (Part 1) | Kong Inc.

18 mar 2022 — If you don't have a compelling reason to choose otherwise, **REST** is probably the best option for your application. While **GraphQL** or **gRPC** might ...

gRPC, GraphQL, REST in nutshell

gRPC

- exact and wicked fast

GraphQL

- flexible
- convenient for front-end clients

REST

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

mongoose

Mongoose

Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment.

It manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB.

https://mongoosejs.com

https://github.com/Automattic/mongoose

Mongoose - sample

```
const mongoose = require('mongoose');
mongoose.connect('connection_string’, ...);

const TaskSchema = mongoose.Schema({
    name: String,
    completed: {
        type: Boolean,
        default: true
    },
});
```

Mongoose - sample cd.

```
const Task = mongoose.model('tasks_mongoose', TaskSchema);

(async () => {
    const task = new Task({
        name: 'kupić czekolade',
    });
    let result = await task.save();
})();
```

Mongoose - timestamps

Mongoose schemas support a timestamps option. If you set timestamps: true, Mongoose will add two properties of type Date to your schema:

- createdAt: a date representing when this document was created
- updatedAt: a date representing when this document was last updated

```
const TaskSchema = mongoose.Schema({
    name: String,
    completed: Boolean
},
{ timestamps: true }
);
```

Mongoose - middleware/hooks

Middleware (also called pre and post hooks) are functions which are passed control during execution of asynchronous functions. Middleware is specified on the schema level.

```
TaskSchema.pre('save', function() {
    console.log("task is going to be saved");
});
TaskSchema.post('save', function() {
    console.log("task saved");
});
```

Middleware can be composed into pipelines using next() function.

Mongoose - validation

Validation is middleware. Mongoose registers validation as a pre('save') hook on every schema by default.

```
const TaskSchema = new mongoose.Schema({
    task: String,
    value:  {
        type: Number,
        min: 10,
        max: 100,
        required: true
    }
});
```

Mongoose features

- Validators (async and sync)
- Defaults
- Getters
- Setters
- Indexes
- Middleware
- Methods definition
- Statics definition
- Plugins
- pseudo-JOINs

Mongoose vs native driver

READS	Native	Mongoose
Throughput	1200 #/sec	583 #/sec
Avg Request	0.83 ms	1.71 ms
WRITES	**Native**	**Mongoose**
Throughput	1128 #/sec	384 #/sec
Avg Request	0.89 ms	2.60 ms

https://blog.jscrambler.com/mongodb-native-driver-vs-mongoose-performance-benchmarks/

https://www.ijert.org/research/performance-evaluation-of-mongo-dbnative-driver-and-mongoose-IJERTV13IS030035.pdf