Service Oriented Architecture and Web Services

Martin Kuba, ICS MU
makub@ics.muni.cz

PA160 lecture, spring 2023

Overview

- RPC, RMI, SOA, Microservices
- Web Services
- SOAP/WSDL
- REST
- Web APIs
- OpenAPI
- AJAX, Mash ups
- Authentication and Authorization in Web Services
    - SAML, OAuth 2, OpenID Connect, JWT

Glossary

AJAX - Asynchronous JavaScript and XML

API - Application Programming Interface

GUI - Graphical User Interface

HTTP - Hypertext Transfer Protocol

HTML - Hypertext Markup Language

IDL - Interface Description Language

JSON - JavaScript Object Notation

REST - Representational State Transfer

SSL/TLS - Secure Sockets Layer/Transport Layer Security

SAML - Security Assertion Markup Language

URL - Uniform Resource Locator

XML - Extensible Markup Language

Communication in Distributed Systems

- synchronicity point of view
  - **synchronous** – the calling side blocks until an answer is received
  - **asynchronous** – the calling side does not wait, it is notified of an answer
- persistency point of view
  - **transient** (disappearing with time)
  - **persistent** (storing messages until receiver is ready)
- TCP is transient, JMS or IBM MQ are persistent
- all 4 combinations are possible

RPC - Remote Procedure Calls

- distributed systems are communicating by sending messages
- harder to use than local procedure calls
- **remote** procedure calls try to hide the complexity
- request-response communication:
  - call a procedure, pass parameters by value
  - return values
- client **stub** and server **skeleton** generated from IDL
  - used locally in a given programming language
  - they do marshalling/serialization, communication, unmarshaling/deserialization
- examples: DCE/RPC, XML-RPC, SOAP

RMI - Remote Method Invocation

- distributed **object-oriented** systems need to pass parameters by reference
- a distributed object has *state*, *interface*, and *implementation*
- examples: CORBA, Java RMI, Microsoft DCOM
- original **Java RMI** (JRMP - Java Remote Method Protocol) is pure Java, it can pass implementation of classes between the server and the client
- Java RMI works only between the same version of JVM
- later **Java RMI-IIOP** (Internet Inter-ORB [Object Request Broker] Protocol) is based on CORBA
- **CORBA** implementations from different vendors were never truly interoperable

RMI Problems

- RMI works only in systems under a centralized control
- thus RMI does not scale to Internet-size
- synchronous communication does not scale
- ***tight coupling*** - versioning and evolution of both communicating ends are difficult
- distribution cannot be transparent because of possible partial failure

SOA - Service Oriented Architecture

- SOA is an architectural style whose goal is to achieve loose coupling among interacting software agents. A service is a unit of work done by a service provider to achieve desired end results for a service consumer [1]
- in SOA, services provide only *interface*
- the interface is defined by **messages**, not by operations on data types
- data types are not interoperable, e.g. String[] in Java is different from string[] in .NET, the former may contain nulls, the latter must not

Difference between OO and SOA

- from [1] Hao He: What is Service-Oriented Architecture:
  - a CD player offers a CD playing **service**
  - different quality of service on a portable player and on an expensive stereo
  - in **object oriented** programming style, every CD would come with its own player and they are not supposed to be separated
- SOA more corresponds to how interactions are organized in the real world
- *loose coupling* - independent evolution of clients and services operated by different organizations

Microservices

- popular, but no sound definition
- services are fine-grained and the protocols are lightweight
- microservices are composed using Unix-like pipelines
- inter-service calls over a network have a higher cost in terms of network latency and message processing time than in-process calls
- difficult to maintain data consistency among transaction participants

A web service is a software system designed to support interoperable machine-to-machine interaction over a network.

(W3C, Web Services Glossary)

Brief web services history

1989 - World Wide Web invented

1991 - HTTP 0.9 specified

1992 - Internet at Masaryk University :-)

1993 - first GUI web browser Mosaic

1993 - Common Gateway Interface for executing programs

1995 - JavaScript introduced by Netscape browser

1996 - SSL 3.0 (first usable encryption)

1998 - XML 1.0 (the first interoperable text data format)

1998 - SOAP 1.1 by Microsoft (text-based RPC)

2004 - WS-Interoperability Basic Profile (SOAP usable)

Brief web services history (2)

2000 - REST defined by Roy Fielding

2001 - JSON invented (simple interoperable data format)

2004 - GMail, Google Maps, Web 2.0, wikis, mash-ups

2005 - AJAX, Yahoo offers JSON web services, SAML

2006 - OpenID 2.0 (decentralized authentication)

2008 - HTML5 (first public working draft)

2010 - mobile devices with small screens

2012 - OAuth 2.0 (authorization framework)

2013 - responsive web design as an answer to devices with different screen sizes

Brief web services history (3)

2006-2013 - cloud computing (Amazon 2006, Microsoft 2008, Google 2013)

2014 - HTML5 finalised (APIs for in-browser apps)

2014 - OpenID Connect (authentication standard)

2015 - HTTP/2, JSON Web Tokens

2016 - OpenAPI (IDL for JSON web services)

2018 - TLS 1.3 (weak points removed)

2019 - WebAuthN (hardware authenticators)

2021 - Self-sovereign identity

2022 - HTTP/3.0

HTTP Protocol Versions

- **HTTP/0.9** - 1989 - Tim Berners-Lee at CERN
    - GET only, no HTTP headers, no status/error codes, no versioning
- **HTTP/1.0** - 1996 - IETF and W3C
    - methods GET, HEAD, POST
    - headers, status codes
    - TCP connection terminated immediately after each response
- **HTTP/1.1** - 1997
    - methods GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS
    - persistent and pipelined connections, chunked transfers, compression/decompression, content negotiations, virtual hosting
- **HTTP/2.0** - 2015
    - binary encoding
    - single TCP connection with multiplexing of requests
    - mandatory TLS 1.2+
- **HTTP/3.0** - 2022
    - QUIC/UDP instead of TCP
    - mandatory TLS 1.3+

My definition of a web service

web service client communicates with a web server requesting a web resource identified by a URL, using HTTP protocol secured by TLS exchanging messages in JSON or XML formats

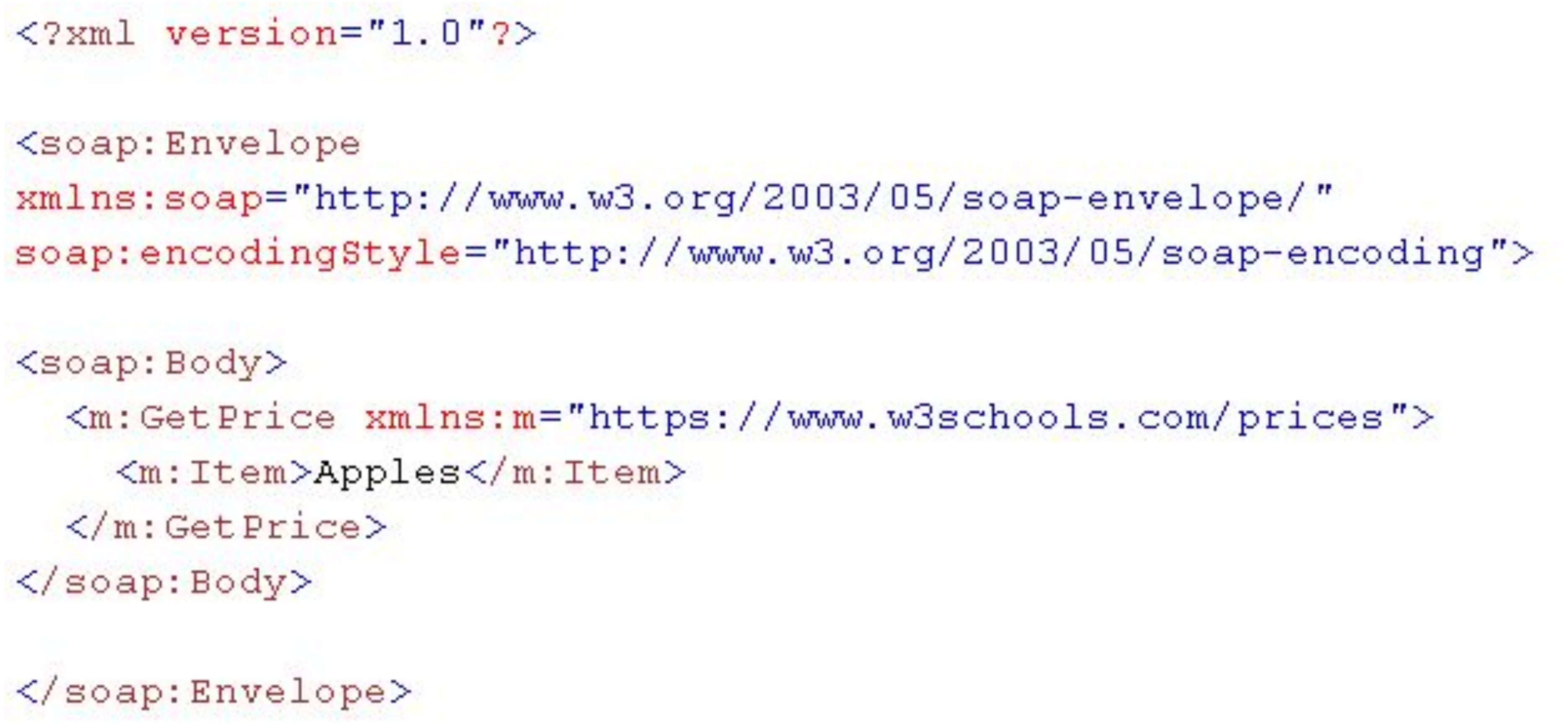
this definition covers

- SOAP/WSDL services
- REST APIs
- dynamic web pages using AJAX

SOAP/WSDL web services

- SOAP was **S**imple **O**bject **A**ccess **P**rotocol
- WSDL is **W**eb **S**ervice **D**escription **L**anguage
- technology for RPC (not RMI!) using exchange of XML messages
- syntax based on XML Schema and Namespaces
- WS-Interoperability Basic Profile needed to ensure interoperability, it requires SOAP 1.1
- many WS-* extensions

SOAP request



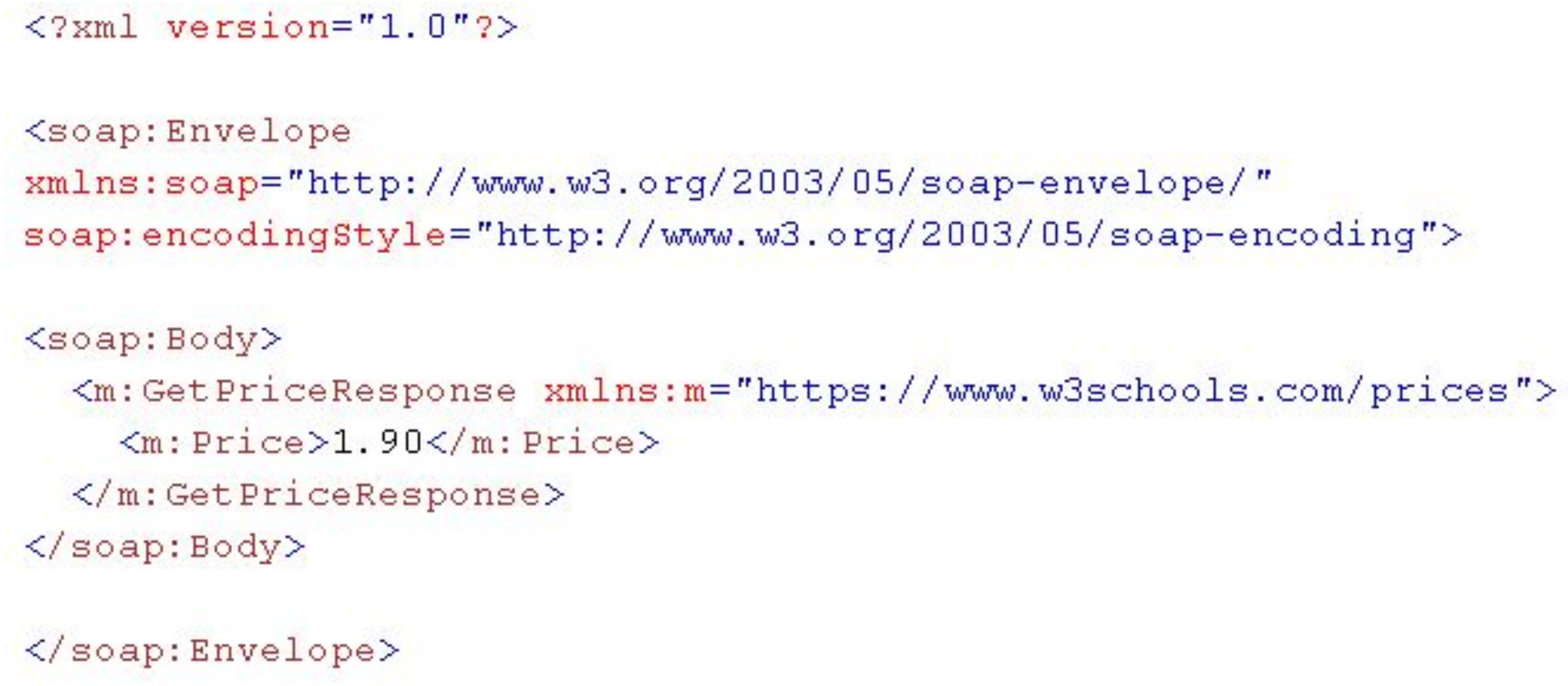
```xml
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body>
  <m:GetPrice xmlns:m="https://www.w3schools.com/prices">
    <m:Item>Apples</m:Item>
  </m:GetPrice>
</soap:Body>

</soap:Envelope>
```

SOAP response



```xml
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body>
  <m:GetPriceResponse xmlns:m="https://www.w3schools.com/prices">
    <m:Price>1.90</m:Price>
  </m:GetPriceResponse>
</soap:Body>

</soap:Envelope>
```

SOAP/WSDL history

- started as XML-based Remote Method Invocation protocol
- changed to Remote Procedure Call protocol (no objects - SOAP is not an abbreviation now)
- introduced its own type system
  - big problems with compatibility followed
- later replaced by XML Schema type system
- main lesson learned - remote interfaces should be defined by ***messages***, not ***operations***

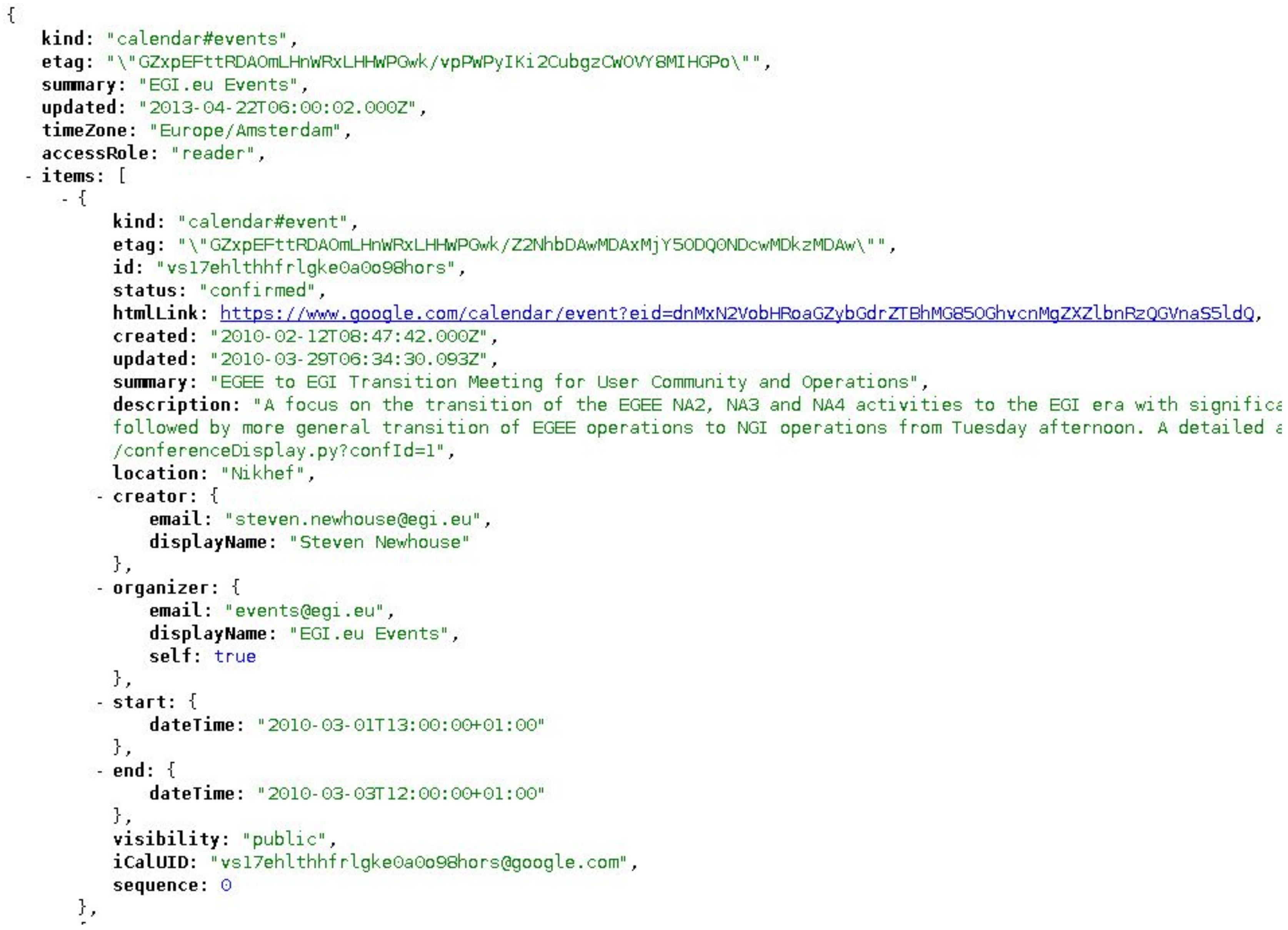
SOAP versus REST

- enterprises prefer complicated stack
  - XML
  - SOAP, WSDL, WS-Interoperability
  - WS-* (WS-Security, WS-Addressing, ...)
  - persistent connections - queues
  - RPC based
  - complex tools and frameworks, need an IT department
- Internet crowd prefers simplicity
  - JSON
  - HTTP requests to URLs, OpenAPI
  - AJAX in browsers
  - transient connections - TCP/IP, HTTP
  - scalable using REST

Web APIs

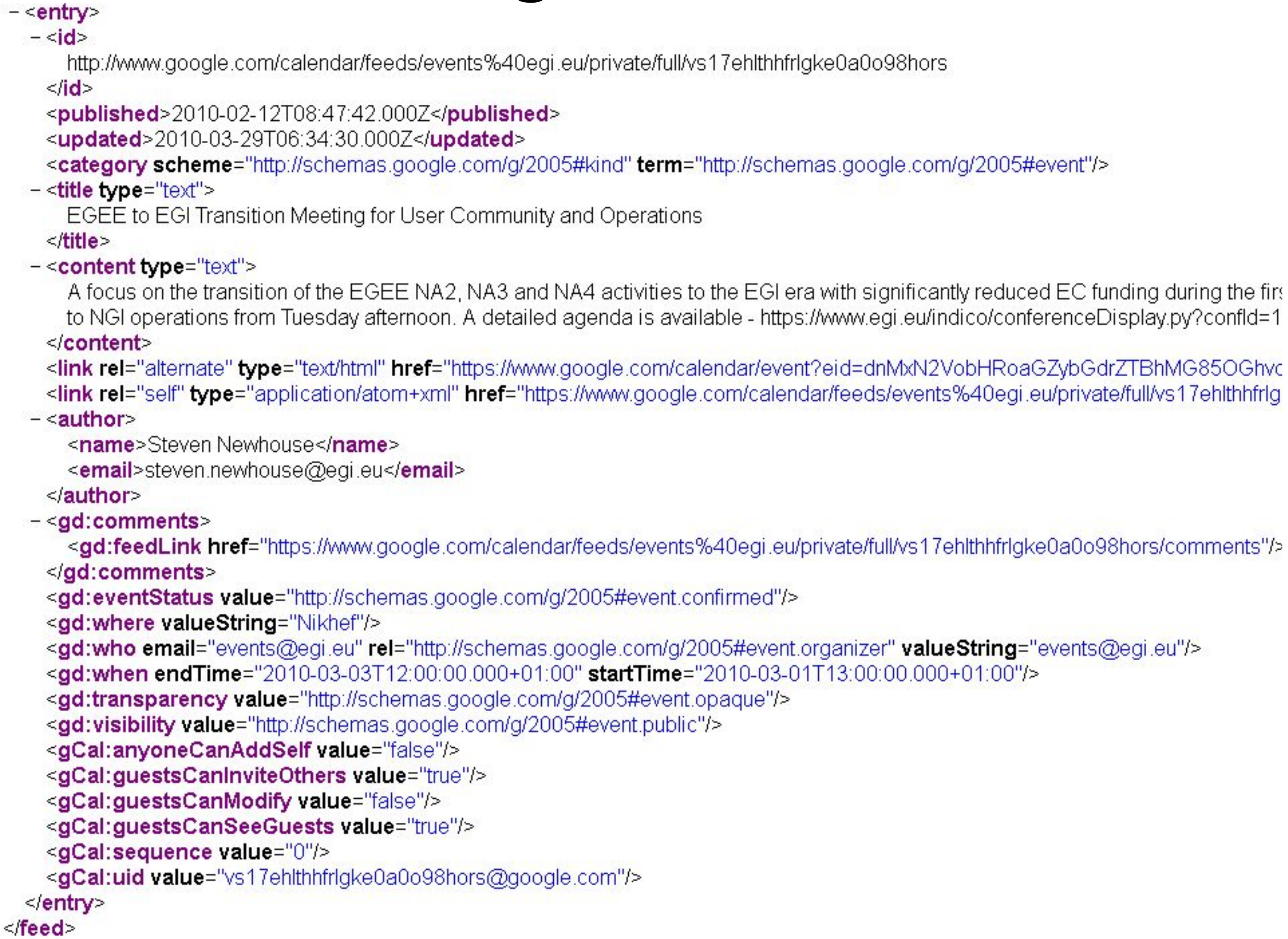
- well-known APIs
  - Google APIs (Calendar, GMail, Maps, ...)
  - Facebook API
  - Twitter API
  - based on HTTP+TLS+JSON+OAuth
- third party clients
  - web, mobile (Android, iOS), desktop, embedded (TV)
- OAuth
  - developer registers an **application** at API provider
  - user authorises the application to use certain operations in the **API**, giving the application an access **token**
  - **application** uses the **token** to use the **API** on behalf of the user

JSON - JavaScript Object Notation



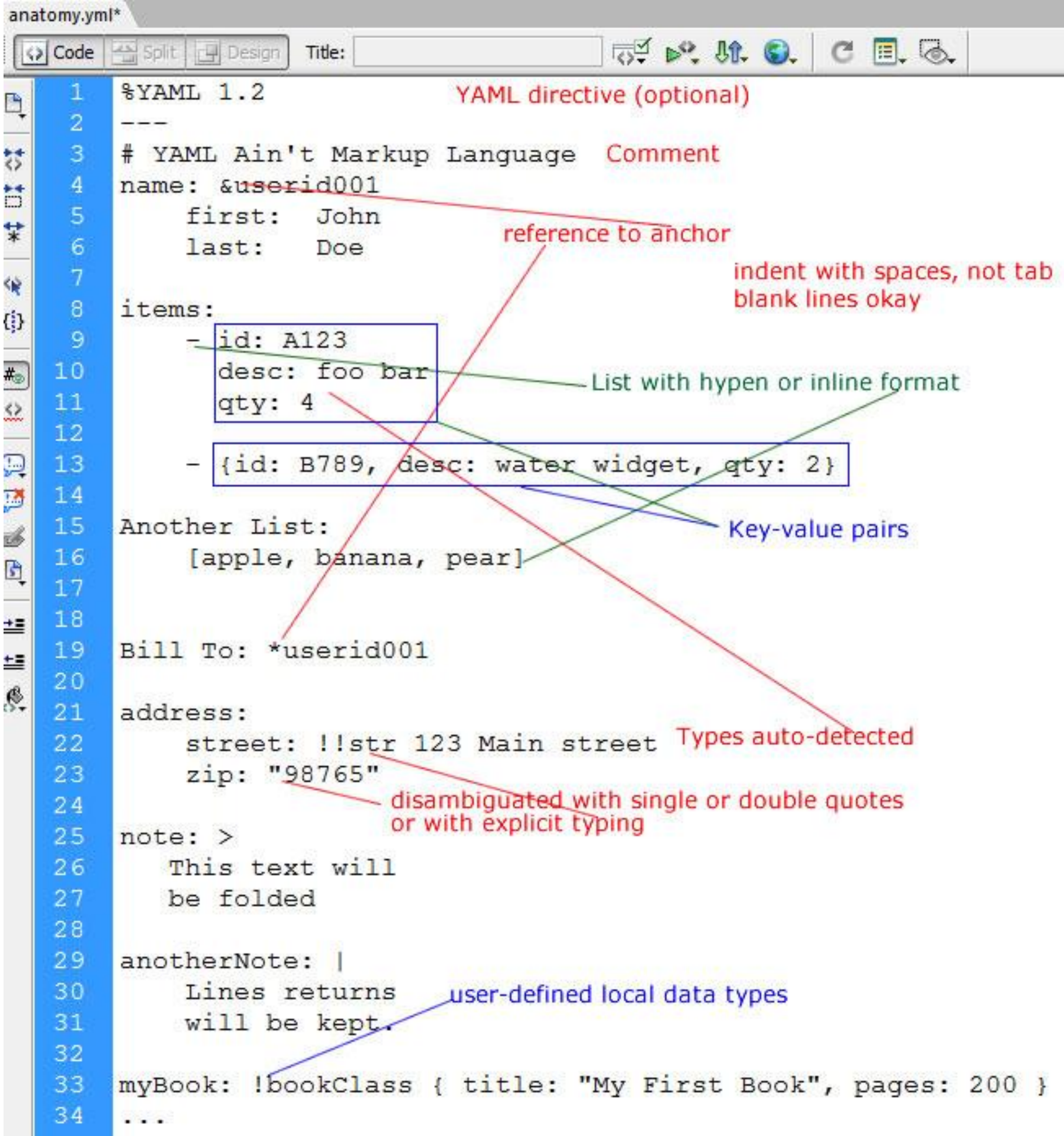
```
{
    kind: "calendar#events",
    etag: "\"GZxpEFttRDAOmLHnWRxLHHWPGwk/vpPWPyIKi2CubgzCWOVY8MIHGPo\"",
    summary: "EGI.eu Events",
    updated: "2013-04-22T06:00:02.000Z",
    timeZone: "Europe/Amsterdam",
    accessRole: "reader",
  - items: [
      - {
            kind: "calendar#event",
            etag: "\"GZxpEFttRDAOmLHnWRxLHHWPGwk/Z2NhbDAwMDAxMjY5ODQ0NDcwMDkzMDAw\"",
            id: "vs17ehlthhfrlgke0a0o98hors",
            status: "confirmed",
            htmlLink: https://www.google.com/calendar/event?eid=dnMxN2VobHRoaGZybGdrZTBhMG85OGhvcnMgZXZlbnRzQGVnaS5ldQ,
            created: "2010-02-12T08:47:42.000Z",
            updated: "2010-03-29T06:34:30.093Z",
            summary: "EGEE to EGI Transition Meeting for User Community and Operations",
            description: "A focus on the transition of the EGEE NA2, NA3 and NA4 activities to the EGI era with significa
            followed by more general transition of EGEE operations to NGI operations from Tuesday afternoon. A detailed a
            /conferenceDisplay.py?confId=1",
            location: "Nikhef",
          - creator: {
                email: "steven.newhouse@egi.eu",
                displayName: "Steven Newhouse"
            },
          - organizer: {
                email: "events@egi.eu",
                displayName: "EGI.eu Events",
                self: true
            },
          - start: {
                dateTime: "2010-03-01T13:00:00+01:00"
            },
          - end: {
                dateTime: "2010-03-03T12:00:00+01:00"
            },
            visibility: "public",
            iCalUID: "vs17ehlthhfrlgke0a0o98hors@google.com",
            sequence: 0
        },
```

The same Google Cal event in XML



```
- <entry>
  - <id>
      http://www.google.com/calendar/feeds/events%40egi.eu/private/full/vs17ehlthhfrlgke0a0o98hors
    </id>
    <published>2010-02-12T08:47:42.000Z</published>
    <updated>2010-03-29T06:34:30.000Z</updated>
    <category scheme="http://schemas.google.com/g/2005#kind" term="http://schemas.google.com/g/2005#event"/>
  - <title type="text">
      EGEE to EGI Transition Meeting for User Community and Operations
    </title>
  - <content type="text">
      A focus on the transition of the EGEE NA2, NA3 and NA4 activities to the EGI era with significantly reduced EC funding during the firs
      to NGI operations from Tuesday afternoon. A detailed agenda is available - https://www.egi.eu/indico/conferenceDisplay.py?confId=1
    </content>
    <link rel="alternate" type="text/html" href="https://www.google.com/calendar/event?eid=dnMxN2VobHRoaGZybGdrZTBhMG85OGhvc
    <link rel="self" type="application/atom+xml" href="https://www.google.com/calendar/feeds/events%40egi.eu/private/full/vs17ehlthhfrlg
  - <author>
      <name>Steven Newhouse</name>
      <email>steven.newhouse@egi.eu</email>
    </author>
  - <gd:comments>
      <gd:feedLink href="https://www.google.com/calendar/feeds/events%40egi.eu/private/full/vs17ehlthhfrlgke0a0o98hors/comments"/>
    </gd:comments>
    <gd:eventStatus value="http://schemas.google.com/g/2005#event.confirmed"/>
    <gd:where valueString="Nikhef"/>
    <gd:who email="events@egi.eu" rel="http://schemas.google.com/g/2005#event.organizer" valueString="events@egi.eu"/>
    <gd:when endTime="2010-03-03T12:00:00.000+01:00" startTime="2010-03-01T13:00:00.000+01:00"/>
    <gd:transparency value="http://schemas.google.com/g/2005#event.opaque"/>
    <gd:visibility value="http://schemas.google.com/g/2005#event.public"/>
    <gCal:anyoneCanAddSelf value="false"/>
    <gCal:guestsCanInviteOthers value="true"/>
    <gCal:guestsCanModify value="false"/>
    <gCal:guestsCanSeeGuests value="true"/>
    <gCal:sequence value="0"/>
    <gCal:uid value="vs17ehlthhfrlgke0a0o98hors@google.com"/>
  </entry>
</feed>
```

YAML 1.2 is superset of JSON


anatomy.yml*
Code
Split
Design
Title:
YAML directive (optional)
Comment
reference to anchor
indent with spaces, not tab
blank lines okay
List with hypen or inline format
Key-value pairs
Types auto-detected
disambiguated with single or double quotes
or with explicit typing
user-defined local data types

```
%YAML 1.2
---
# YAML Ain't Markup Language
name: &userid001
     first:  John
     last:   Doe

items:
     - id: A123
       desc: foo bar
       qty: 4

     - {id: B789, desc: water widget, qty: 2}

Another List:
     [apple, banana, pear]


Bill To: *userid001

address:
     street: !!str 123 Main street
     zip: "98765"

note: >
    This text will
    be folded

anotherNote: |
     Lines returns
     will be kept.

myBook: !bookClass { title: "My First Book", pages: 200 }
...
```

REST

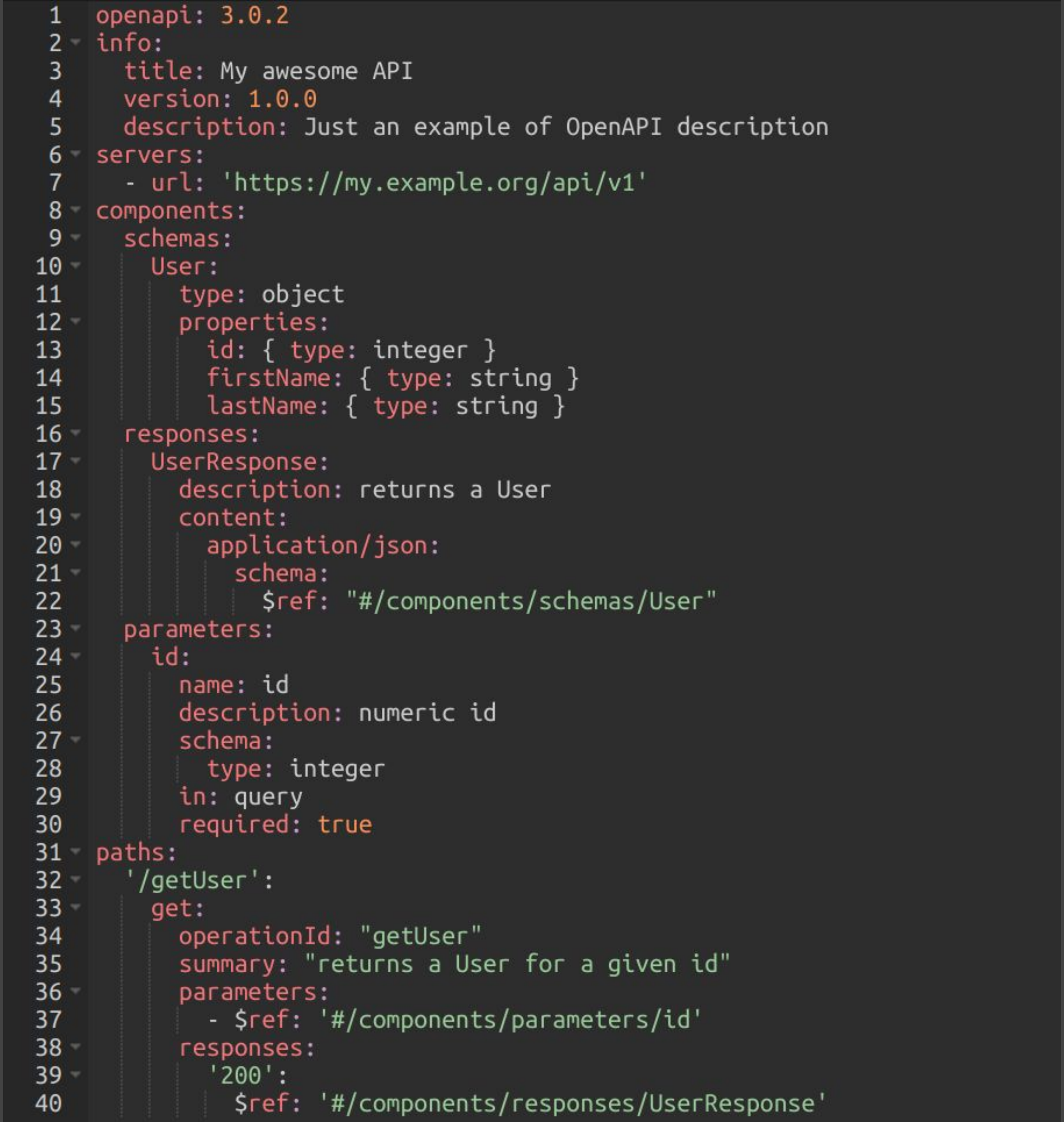
- **Re**presentational **S**tate **T**ransfer
- software **architecture style** for creating scalable web services
- invented by Roy Fielding, author of HTTP 1.1
- resources identified by URIs
- representations of resources as JSON, XML or other formats
- uses HTTP methods GET, PUT, DELETE and POST for manipulating resources
- **verbs** (GET, PUT,...) manipulate **nouns** (resources)
- not every service using HTTP and JSON is RESTful

Web API Descriptions

- API described in human natural language
  - e.g. “*image can be changed by HTTP PUT request to /image/{imageID} with the image in request body*”
- WSDL 2.0 defined in 2007, but never used
- OpenAPI since 2016
  - machine-processable description of HTTP interfaces
  - a form of IDL (Interface Description Language)
  - written in YAML language, which is a more human-readable superset of JSON
  - can describe both RPC-like and RESTful APIs

OpenAPI

- “machine-readable interface files for describing, producing, consuming, and visualizing RESTful web services”
- developed since 2010 as **Swagger**, renamed to **OpenAPI** in 2016
- version 3.0.0 released in 2017
- latest version 3.1 released in February 2021
- API description in file **openapi.yml**
- tool **OpenAPI Generator** can generate client stubs in about 40 programming languages

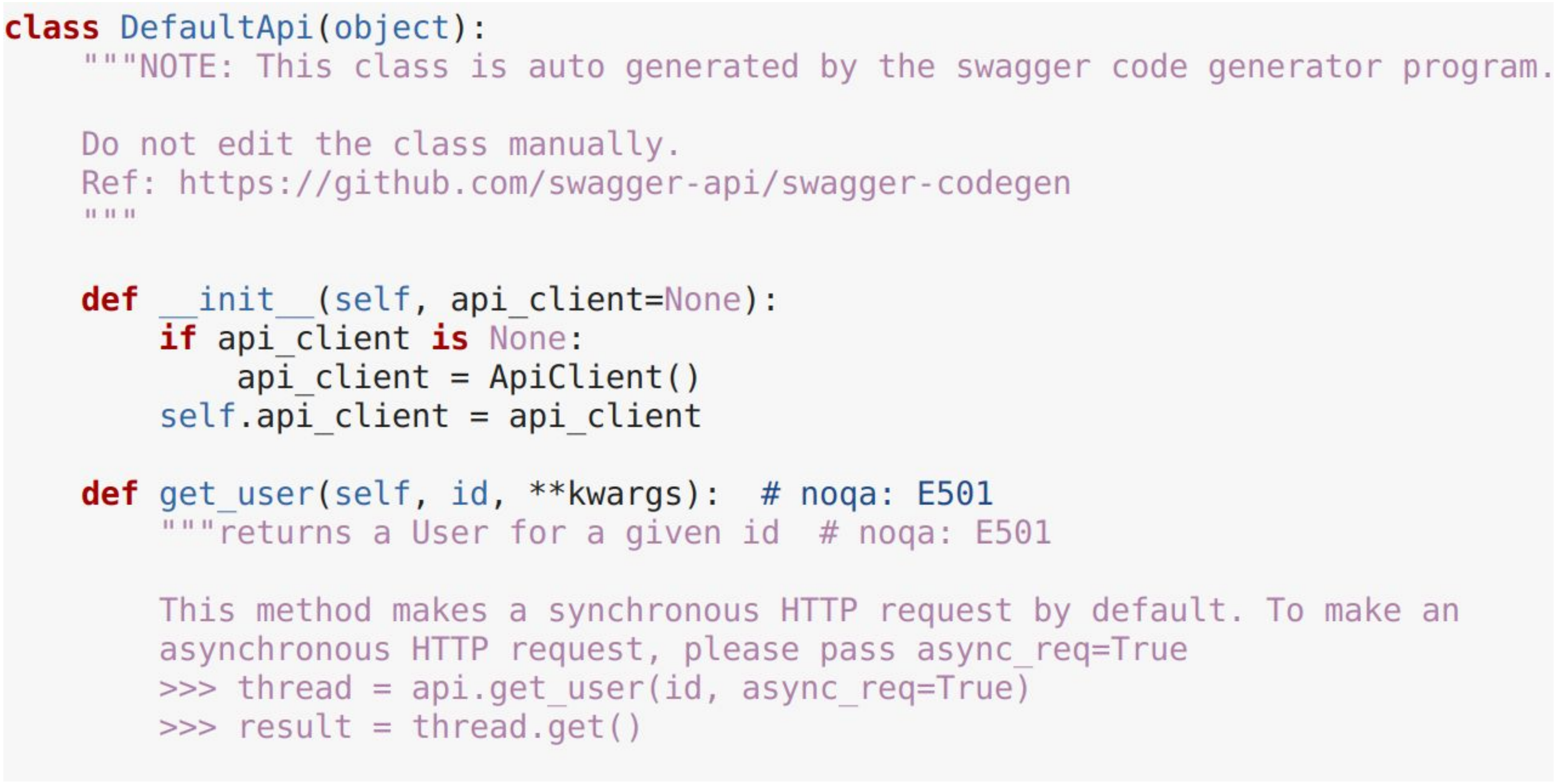


```yaml
openapi: 3.0.2
info:
  title: My awesome API
  version: 1.0.0
  description: Just an example of OpenAPI description
servers:
  - url: 'https://my.example.org/api/v1'
components:
  schemas:
    User:
      type: object
      properties:
        id: { type: integer }
        firstName: { type: string }
        lastName: { type: string }
  responses:
    UserResponse:
      description: returns a User
      content:
        application/json:
          schema:
            $ref: "#/components/schemas/User"
  parameters:
    id:
      name: id
      description: numeric id
      schema:
        type: integer
      in: query
      required: true
paths:
  '/getUser':
    get:
      operationId: "getUser"
      summary: "returns a User for a given id"
      parameters:
        - $ref: '#/components/parameters/id'
      responses:
        '200':
          $ref: '#/components/responses/UserResponse'
```

Java client library generated by OpenAPI Generator

```java
/**
 * returns a User for a given id
 *
 * @param id numeric id (required)
 * @return User
 * @throws ApiException If fail to call the API, e.g. server error
 */
public User getUser(Integer id) throws ApiException {
    ApiResponse<User> resp = getUserWithHttpInfo(id);
    return resp.getData();
}
```

Python client library generated by OpenAPI Generator



```python
class DefaultApi(object):
    """NOTE: This class is auto generated by the swagger code generator program.

    Do not edit the class manually.
    Ref: https://github.com/swagger-api/swagger-codegen
    """

    def __init__(self, api_client=None):
        if api_client is None:
            api_client = ApiClient()
        self.api_client = api_client

    def get_user(self, id, **kwargs):  # noqa: E501
        """returns a User for a given id  # noqa: E501

        This method makes a synchronous HTTP request by default. To make an
        asynchronous HTTP request, please pass async_req=True
        >>> thread = api.get_user(id, async_req=True)
        >>> result = thread.get()
```

AJAX

- **A**synchronous **J**avaScript **A**nd **X**ML
- (Ajax was a Greek mythological hero)
- AJAX does not need XML, uses JSON mostly
- enabled by introduction of **XMLHttpRequest** JavaScript object to web browsers around the year 2006
- asynchronous request to web server
- enables calling REST services from JavaScript

CORS

- JavaScript in browsers has **same-origin policy**
  - limits requests to the same *origin* - triple (scheme, host, port)
  - can be circumvented using CORS
- **CORS** (Cross-origin resource sharing)
  - uses HTTP headers for allowing cross-origin requests
  - client sends header `Origin:` with URL of calling web page
  - server responds with `Access-Control-Allow-Origin:` header with the same URL or `*` for any
  - requests changing data (POST, PUT, …) must do a *preflight request* using OPTIONS method
  - the `Vary:` header should mark CORS headers that cause responses not to be cached by proxies

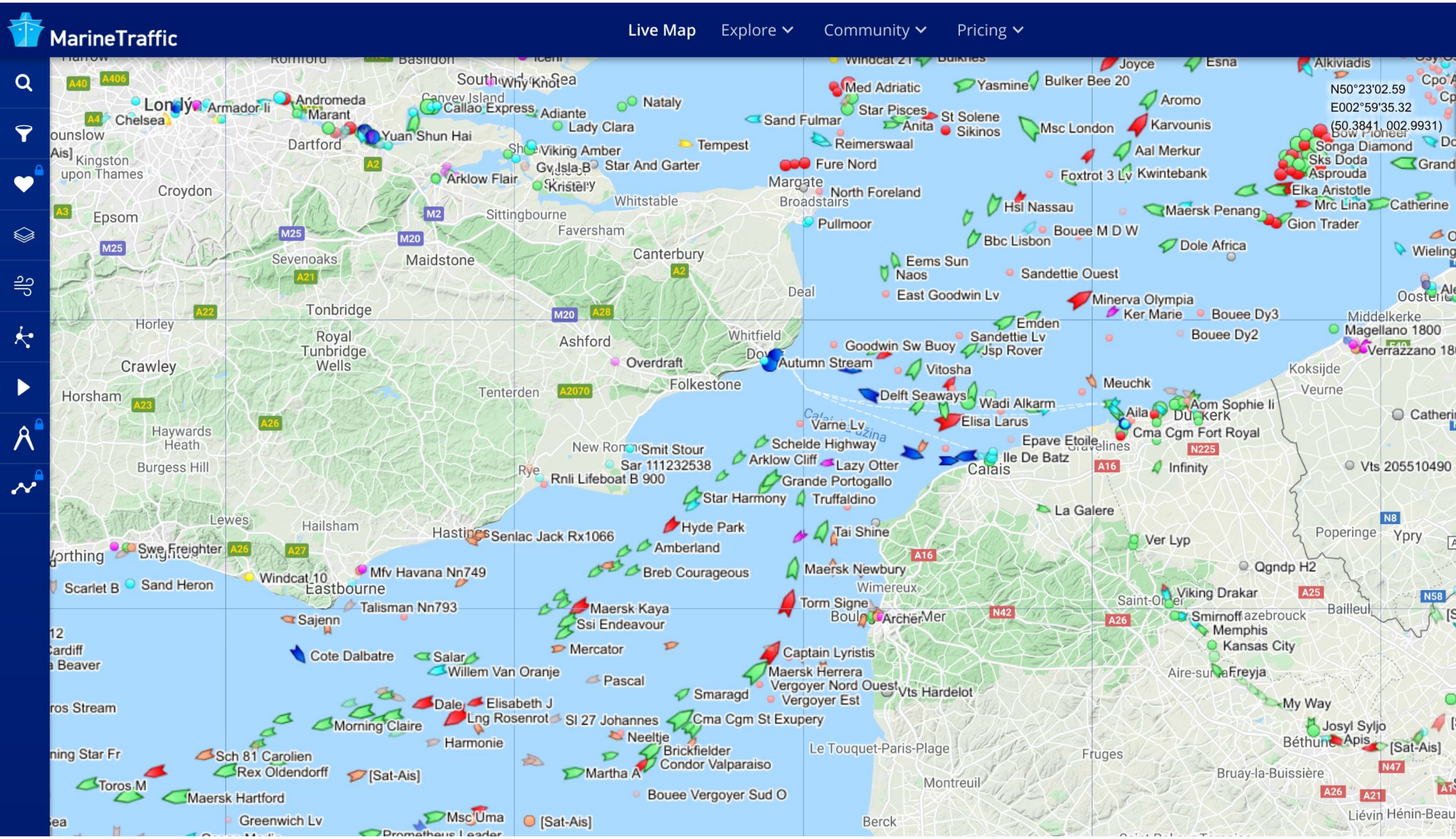
SPA - Single Page Applications

- written in JavaScript
- running in browsers
- transferring data using AJAX calls
- have special security considerations
  - cannot keep secrets (may be reverse-engineered)
  - special types of attacks (XSS, XSRF)

Mash ups

- combine data from various sources
- typically a Google map with some geospatial data
    - ships - http://www.marinetraffic.com/
    - aircrafts - http://www.flightradar24.com/

Mash-up of Google Maps with ships data



Authentication and Authorization in Web Services

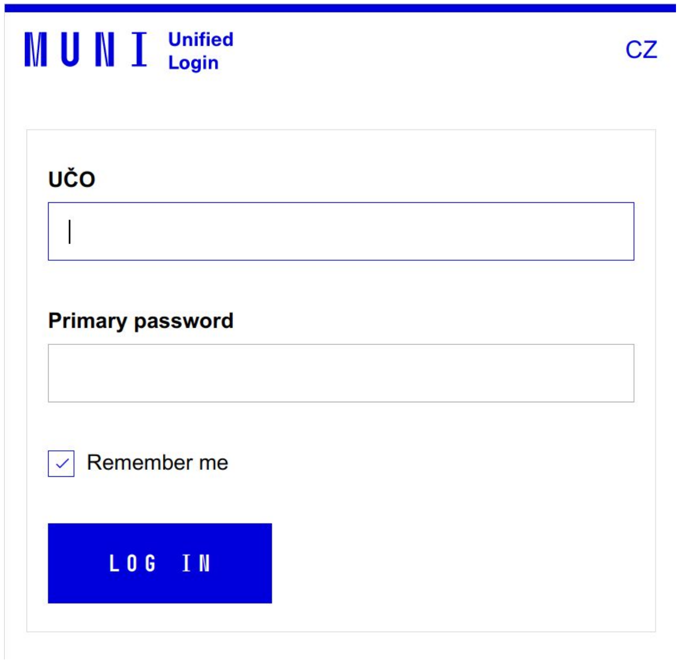
- an important problem in web services is to know who is who (*authentication*) and what to allow them to do (*authorization*)
- the next section talks about
    - federated identity
    - SAML
    - OAuth 2
    - OpenID
    - OpenID Connect
    - JSON Web Tokens

Federated identity

- many authentication mechanisms were developed for the web
  - username+password (hard to remember)
  - X509 digital certificate (complicated to get)
  - digest, Kerberos etc. (not much support in browsers)
- users forget passwords to rarely used accounts
- in federated identity, account from one organisation can be reused at others
- protocols and identity providers:
  - SAML - in academia, Microsoft O365, Google Apps
  - OAuth - Google, Facebook, Twitter, ...
  - OpenID - obsolete
  - OpenID Connect - mix of OpenID and OAuth

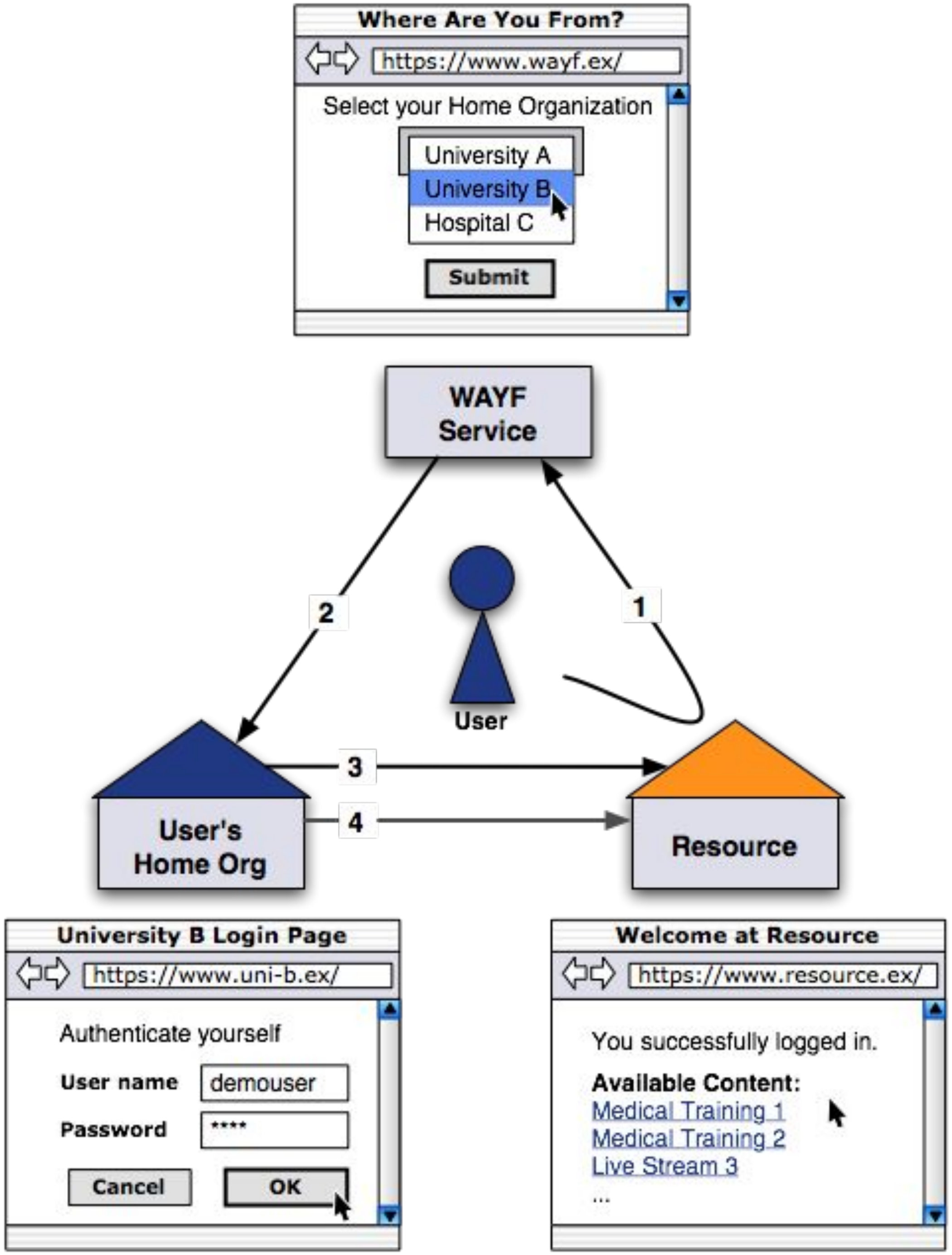
MUNI Unified Login

- **OpenID Connect** protocol for internal MUNI services
- **SAML** protocol for external services in federations eduId.cz and eduGAIN
- see https://it.muni.cz/en/services/jednotne-prihlaseni-na-muni



SAML

- **S**ecurity **A**ssertion **M**arkup **L**anguage
- introduced in 2001
- provides **web browser single sign-on**
- SAML document is XML containing user attributes signed by an identity provider
- trust between identity providers (IdP) and service providers (SP) is established using federations
- a **federation** publishes list of trusted IdPs and SPs complying with federation’s policy
- WAYF - Where Are Your From? service / DS - Discovery Service


Where Are You From?
https://www.wayf.ex/
Select your Home Organization
University A
University B
Hospital C
Submit
WAYF Service
1
2
User
3
4
User's Home Org
Resource
University B Login Page
https://www.uni-b.ex/
Authenticate yourself
User name
demouser
Password
****
Cancel
OK
Welcome at Resource
https://www.resource.ex/
You successfully logged in.
Available Content:
Medical Training 1
Medical Training 2
Live Stream 3
...

OAuth 2.0 Authorization Framework

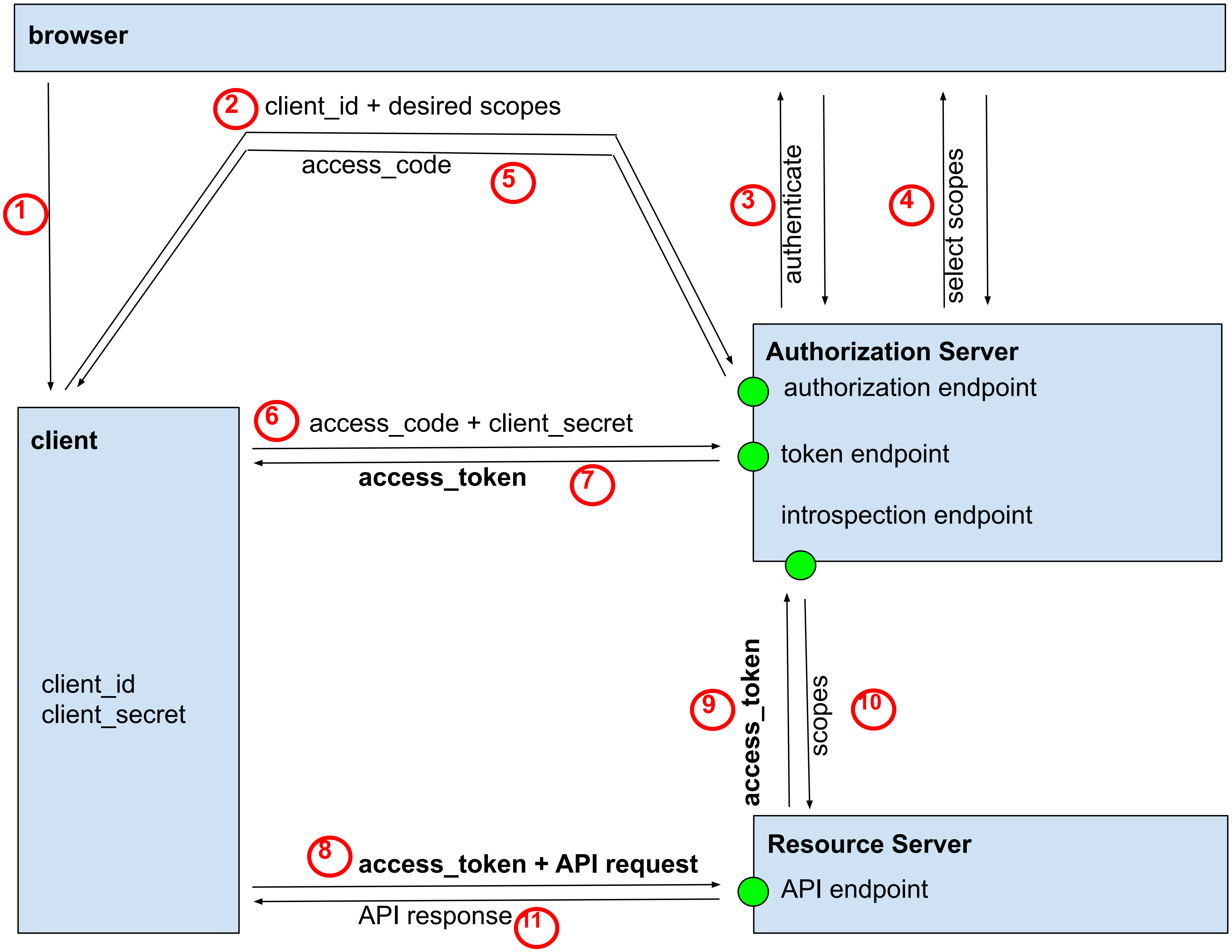
- defined in RFC 6749 in the year 2012
- used by Google, Facebook, Microsoft, Twitter, LinkedIn, GitHub, …
- designed for delegating limited access to third parties, but used for authentication too

OAuth 2 - involved parties

- **resource owner** - the user
- **resource server**
  - maintains user’s data
  - provides API for operations on the data
  - checks **access token** for permissions for sets of operations called **scopes**
- **client** - application that wants to use the API on user’s behalf
- **authorization server**
  - registers all others - the user, the client and the RS
  - authenticates the user, asks which scopes to allow
  - releases an **access token** to the client

OAuth 2 Features

- not limited to web apps, also for mobile, SmartTV, desktop, embedded
- various grant flows depending on abilities to store secrets and user interface
  - if you log into Youtube app in your SmartTV using QR code, that’s OAuth’s “**Device Authorization Grant**”
  - if you log in your mobile app into Google, that’s “**Authorization Code Grant with Proof Key for Code Exchange**”
  - if you log into a server-side web app in your browser, that’s “**Authorization Code Grant**” (on the next slide)


browser
1
2
client_id + desired scopes
access_code
5
3
authenticate
4
select scopes
Authorization Server
authorization endpoint
token endpoint
introspection endpoint
client
client_id
client_secret
6
access_code + client_secret
access_token
7
9
access_token
scopes
10
Resource Server
API endpoint
8
access_token + API request
API response
11

OpenID versions 1 and 2

- obsolete
- introduced the idea of decentralized authentication protocol
- users were identified by URLs
- anybody could run an identity provider
- problem of trust
- only large identity providers like Google were trusted by service providers

OpenID Connect (OIDC)

- promoted as third version of OpenID
- authentication layer built on top of OAuth 2.0
- OAuth 2.0 is for authorization, it does not define API for obtaining user data
- OIDC defines:
    - **UserInfo API** for obtaining user data in JSON
    - **scopes** for the API - openid, profile, email, address, phone
    - **claims** - data about the user (e.g. family_name)
    - well-known URI (RFC 8615) for discovery `/.well-known/openid-configuration`

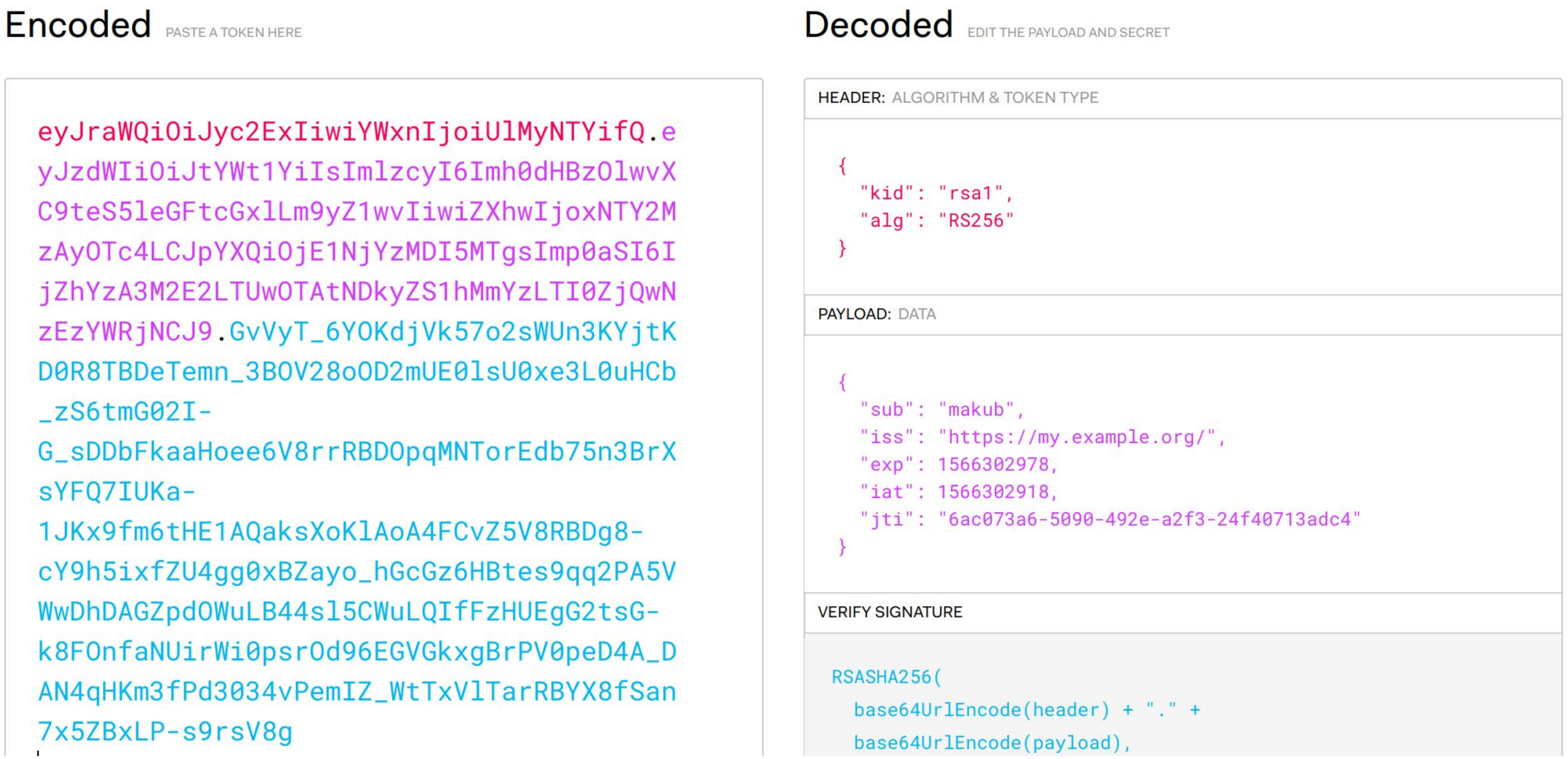
Example of UserInfo response

```json
{
  "sub": "3e65bd2aa4c818bd3579023939b546b69e1@einfra.cesnet.cz",
  "name": "Josef Novák",
  "preferred_username": "pepa",
  "given_name": "Josef",
  "family_name": "Novák",
  "nickname": "Pepan",
  "profile": "https://www.muni.cz/en/people/3988",
  "picture": "https://secure.gravatar.com/avatar/f320c89e39d15da1608c8fc31210b8ca",
  "website": "http://pepovo.wordpress.com/",
  "gender": "male",
  "zoneinfo": "Europe/Prague",
  "locale": "cs-CZ",
  "updated_at": "1508428216",
  "birthdate": "1975-01-01",
  "email": "pepa@gmail.com",
  "email_verified": true,
  "phone_number": "+420 603123456",
  "phone_number_verified": true,
  "address": {
    "street_address": "Severní 1",
    "locality": "Dolní Lhota",
    "postal_code": "111 00",
    "country": "Czech Republic"
  }
}
```

JWT - JSON Web Tokens

- convenient for small digitally signed pieces of structured data
- TLS does not provide signatures of transported data
- JWT is often used for OAuth access tokens
- RFC 7515 - JSON Web Signature
  - **<header>.<payload>.<signature>**
  - all 3 parts are base64-encoded, safe for URLs
  - <header> is JSON metadata identifying signing key
- RFC 7519 - JSON Web Tokens
  - JWS with JSON payload

JSON Web Token example
https://jwt.io/



Encoded PASTE A TOKEN HERE

```
eyJraWQiOiJyc2ExIiwiYWxnIjoiUlMyNTYifQ.e
yJzdWIiOiJtYWt1YiIsImlzcyI6Imh0dHBzOlwvX
C9teS5leGFtcGxlLm9yZ1wvIiwiZXhwIjoxNTY2M
zAyOTc4LCJpYXQiOjE1NjYzMDI5MTgsImp0aSI6I
jZhYzA3M2E2LTUwOTAtNDkyZS1hMmYzLTI0ZjQwN
zEzYWRjNCJ9.GvVyT_6YOKdjVk57o2sWUn3KYjtK
D0R8TBDeTemn_3BOV28oOD2mUE0lsU0xe3L0uHCb
_zS6tmG02I-
G_sDDbFkaaHoee6V8rrRBDOpqMNTorEdb75n3BrX
sYFQ7IUKa-
1JKx9fm6tHE1AQaksXoKlAoA4FCvZ5V8RBDg8-
cY9h5ixfZU4gg0xBZayo_hGcGz6HBtes9qq2PA5V
WwDhDAGZpdOWuLB44sl5CWuLQIfFzHUEgG2tsG-
k8FOnfaNUirWi0psrOd96EGVGkxgBrPV0peD4A_D
AN4qHKm3fPd3034vPemIZ_WtTxVlTarRBYX8fSan
7x5ZBxLP-s9rsV8g
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "kid": "rsa1",
  "alg": "RS256"
}
```

PAYLOAD: DATA

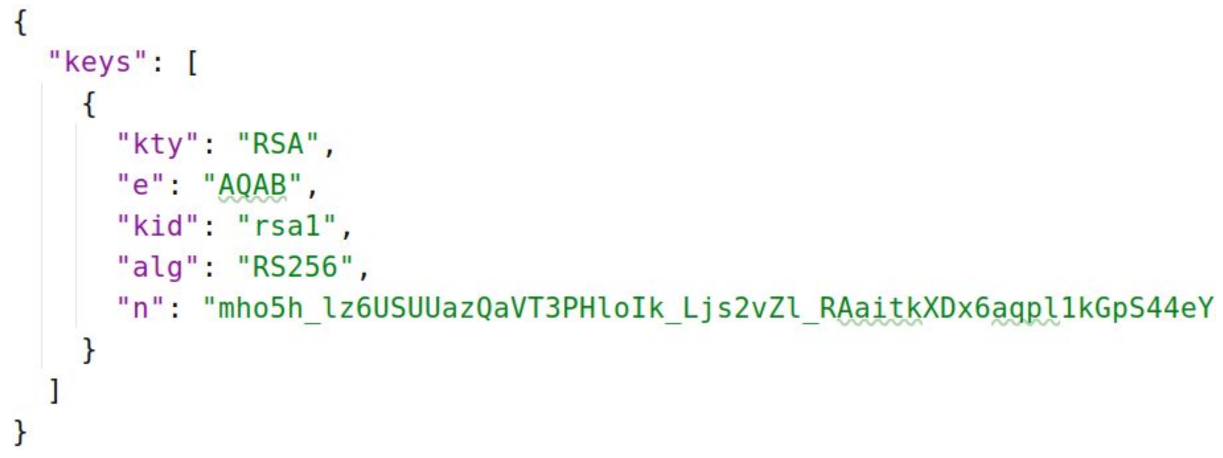
```
{
  "sub": "makub",
  "iss": "https://my.example.org/",
  "exp": 1566302978,
  "iat": 1566302918,
  "jti": "6ac073a6-5090-492e-a2f3-24f40713adc4"
}
```

VERIFY SIGNATURE

```
RSASHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
```

JWKS - JSON Web Key Set

- JSON-formatted web document containing public parts of cryptographic keys
- its URL can be in JWT header in `jku` claim
- its URL can be in OIDC’s metadata at `/.well-known/openid-configuration` in `jwks_uri` claim



```
{
  "keys": [
    {
      "kty": "RSA",
      "e": "AQAB",
      "kid": "rsa1",
      "alg": "RS256",
      "n": "mho5h_lz6USUUazQaVT3PHloIk_Ljs2vZl_RAaitkXDx6aqpl1kGpS44eY
    }
  ]
}
```

That’s it

Thank you for your attention