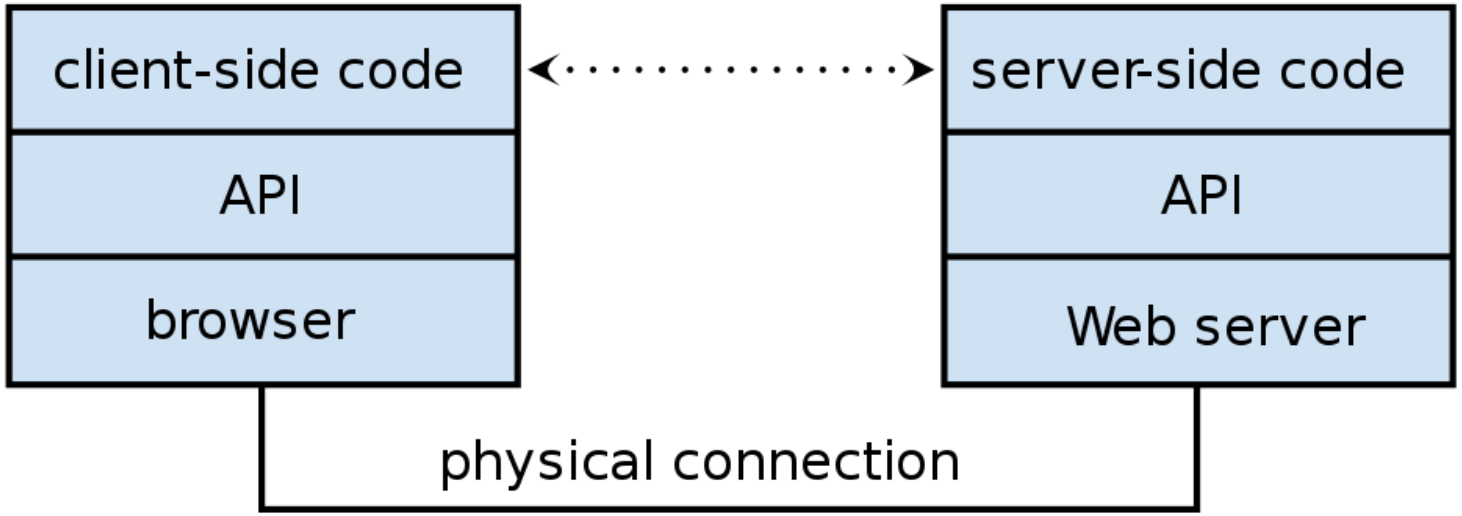


Web API Intro

Frank Walsh

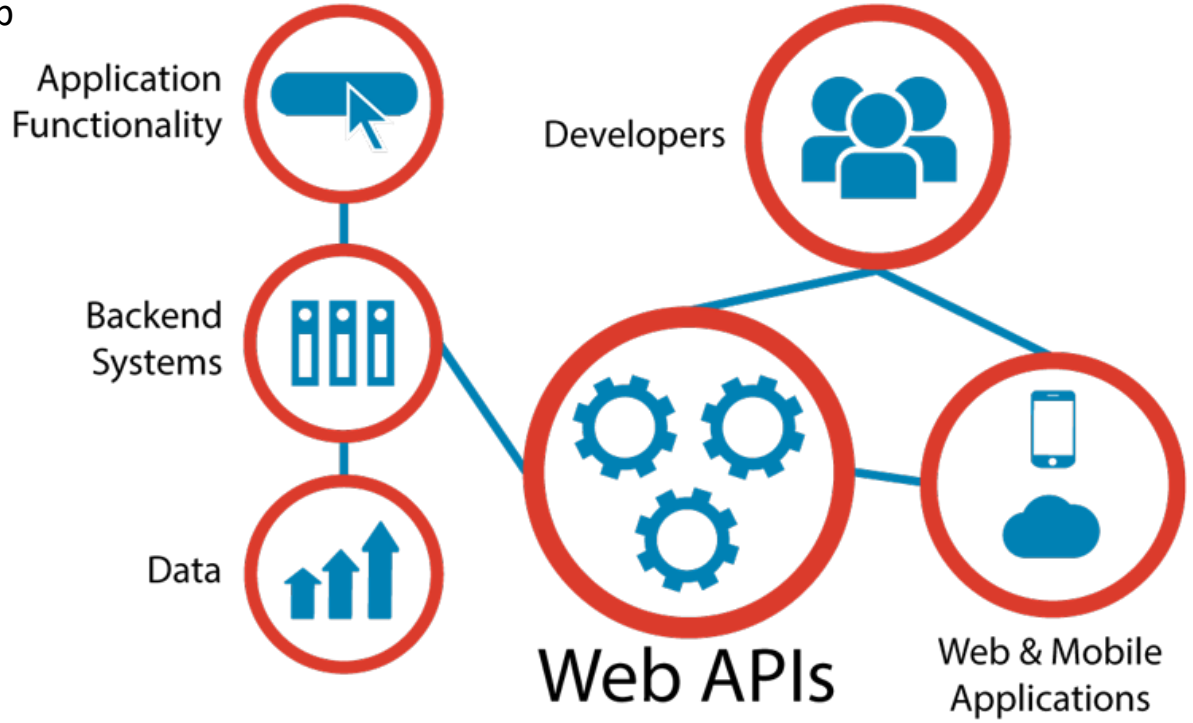
Agenda

- Web API
- REST
- API Value
- API Design

Web APIs

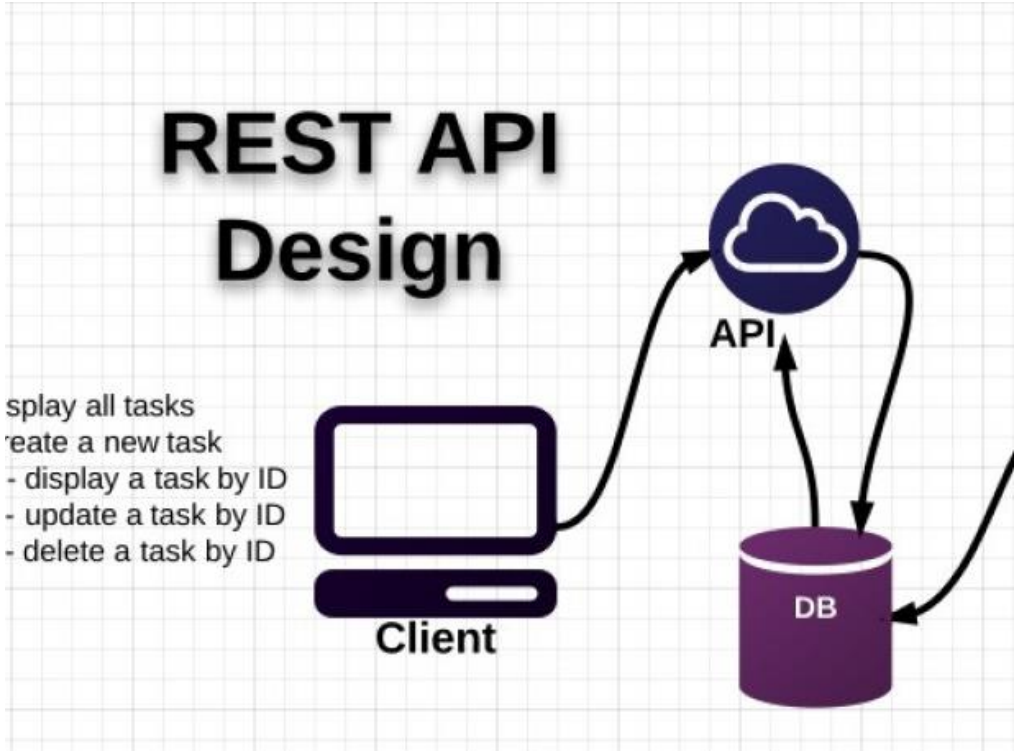
Web APIs

- Programmatic interface exposed via the web
- Uses open standards typically with request-response messaging.
  - E.g messages in JSON or XML
  - HTTP as transport
  - URIs
- Example would be web APIs described in previous lectures.
- Typical use:
  - **allow applications to access data and interact with external software components**
  - Machine to machine communication
  - Distributed systems



Traditional API Design

- API design happens after the release of some a data-rich application
  - Existing application “wrapped” in API
- Created as an afterthought.
  - Tightly bound application needs data/function exposed as API.
  - Shoe-horned in as a separate entity.



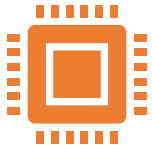
“API First” approach

- Collaboratively design, mockup, implement and document an API **before** the application or other channels that will use it even exist.
- Uses “clean-room” approach.
  - the API is designed with little consideration for the existing IT landscape.
  - the API is designed as though there are no constraints.

Advantages of API First

- Suits multi-device environment of today.
- An API layer can serve multiple channels/devices.
  - Mobile/tablet/IoT device
- Scalable, modular, cohesive and composable
  - If designed properly(e.g. microservice architecture)
  - See later slides
- Concentrate on function first rather than data

API Design Approach



Use principle of developer-first

put target developers' interests ahead of other considerations

Strive for a better developer experience

Commit to RESTful APIs

Take a grammatical approach to the functionality

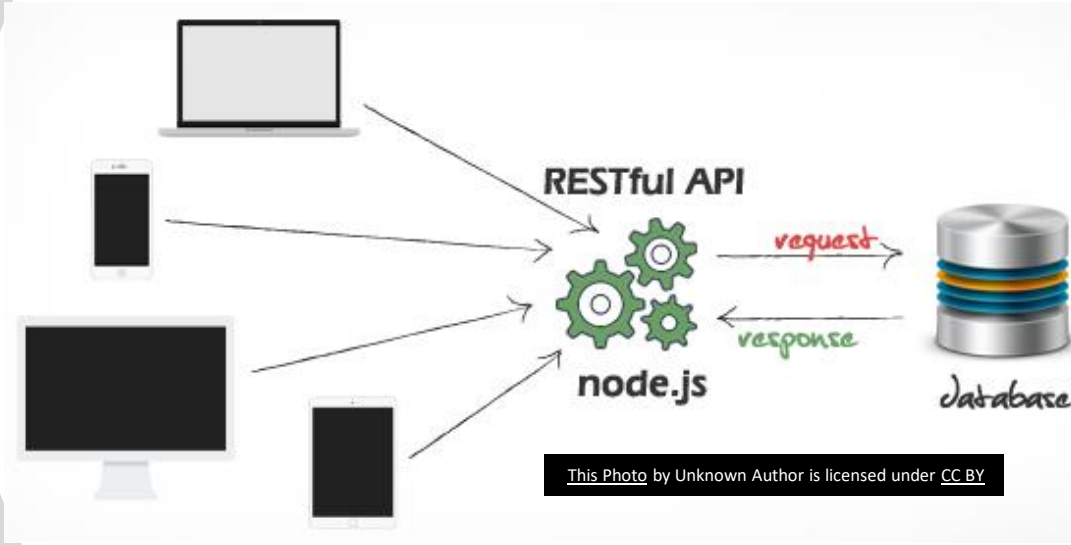
Keep interface simple and intuitive

REST

What’s REST?

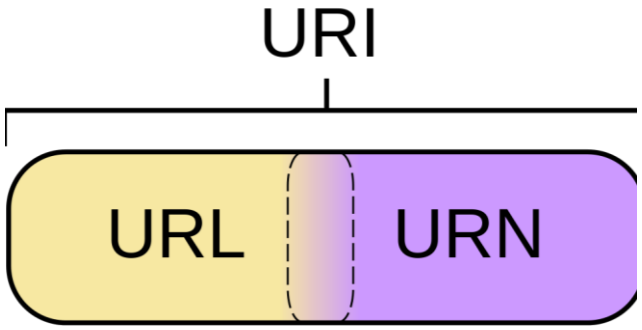
- Short for Representational State Transfer
- Set of Principles for how web should be used
- Coined by Roy Fielding
    - One of the HTTP creators

•A set of principles that define how Web standards(HTTP and URIs) can be used.



Key REST Principles

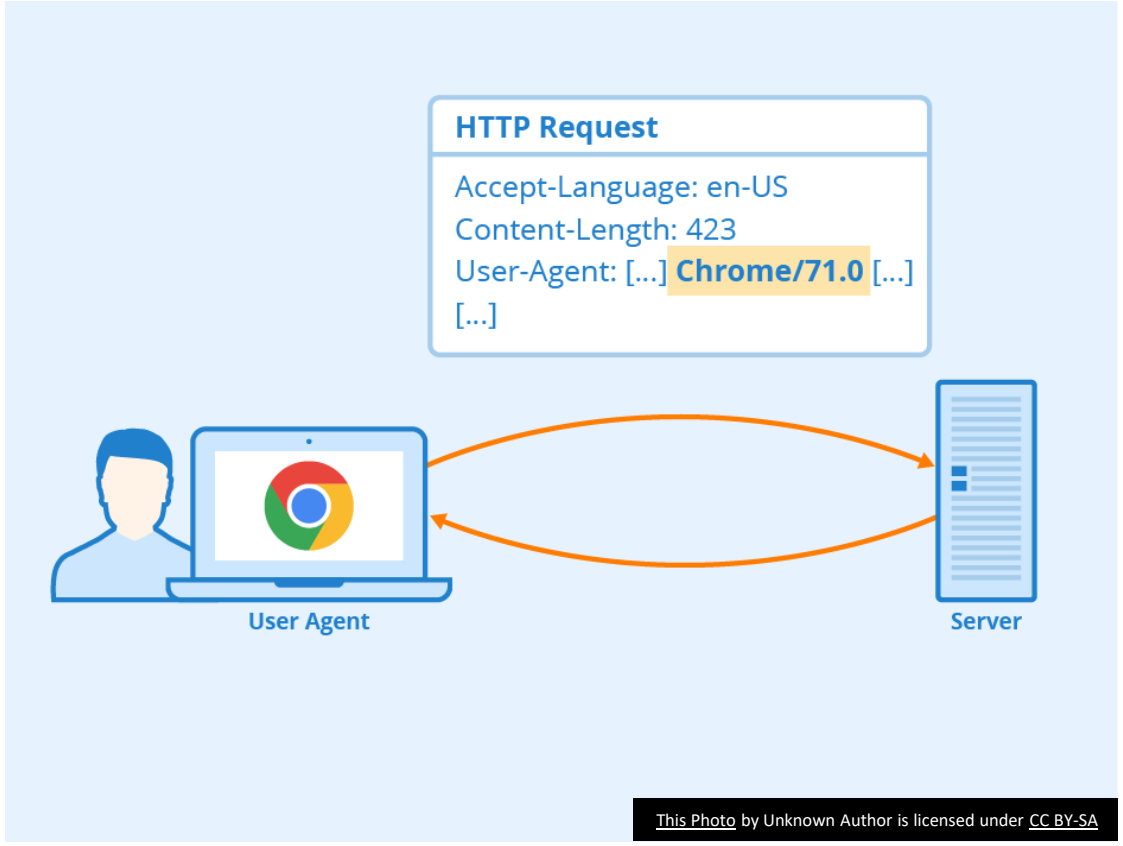
1. Every “thing” has an identity
    - Uniform Resource Identifier
2. Use standard set of methods
    - **HTTP** GET/POST/PUT/DELETE/PATCH
    - Manipulate resources through their representations
3. Resources can have multiple representations
    - JSON/XML/PNG/...
4. Communicate stateless
    - Should **not** depend on server state.

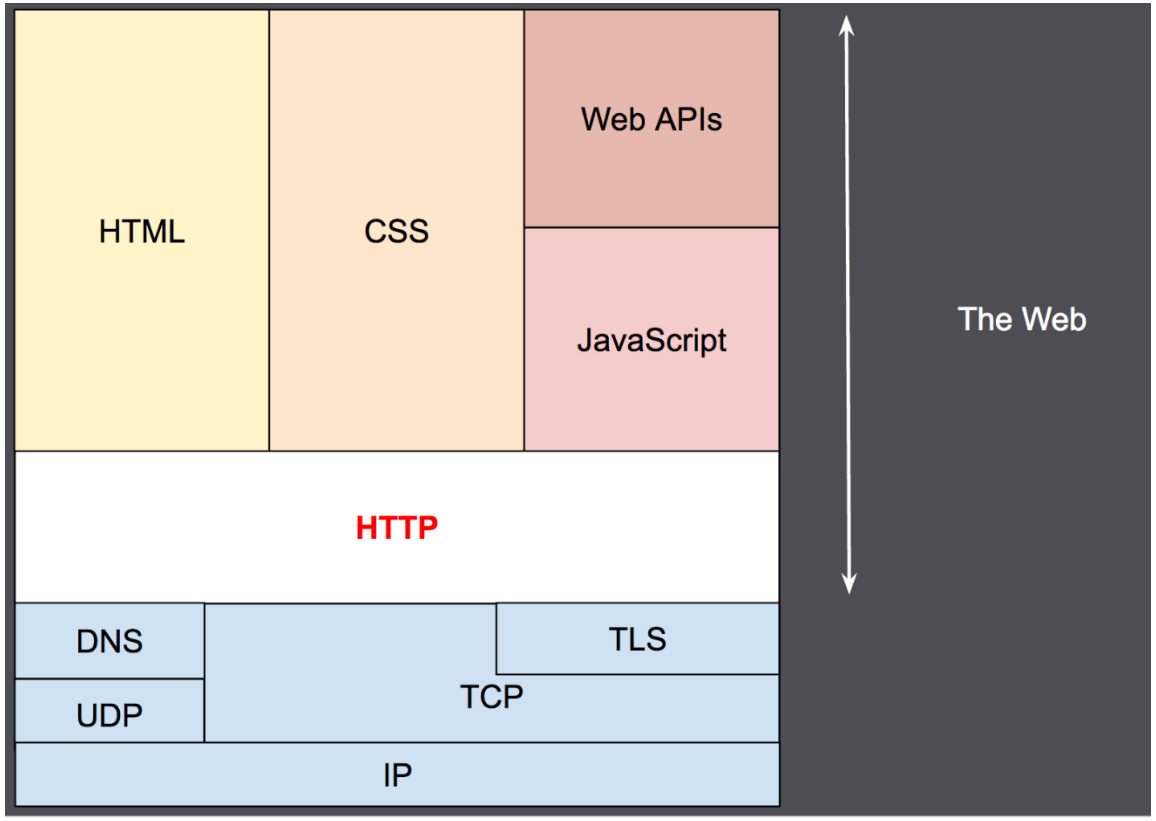


HTTP

Hypertext Transfer Protocol

To design a REST API, you need to know & understand HTTP





HTTP Overview: Components

- **Client:** the **user-agent,** any program that acts for the user – e.g. a browser
- **Server:** *provides* the resource as requested by the client. A
  - Appears as a single machine virtually, however may actually be a collection of servers, sharing the load (load balancing)
  - Can contain complex software interrogating other computers to generate response
- **Proxies:** Computers that relay HTTP messages and perform tasks such as caching, filtering, load balancing, authentication, logging, forwarding

HTTP Protocol (Request)

- HTTP clients (e.g. a browser) translates a **URL** into a request message according to the **specified protocol**; and sends the request message to the server.
- For example, a client could translated the URL http://www.nowhere123.com/api/movies into the following request message:

```
GET /api/movies HTTP/1.1
Host: www.nowhere123.com
Accept: application/json, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

HTTP Protocol (Response)

- When this request message reaches the server, the server can take either one of these actions:
  1. The server interprets the request received, maps the request into a file under the server's document directory, and returns the file requested to the client.
  2. The server interprets the request received, maps the request into a program kept in the server, executes the program, and returns the output of the program to the client.
  3. The request cannot be satisfied, the server returns an error message.

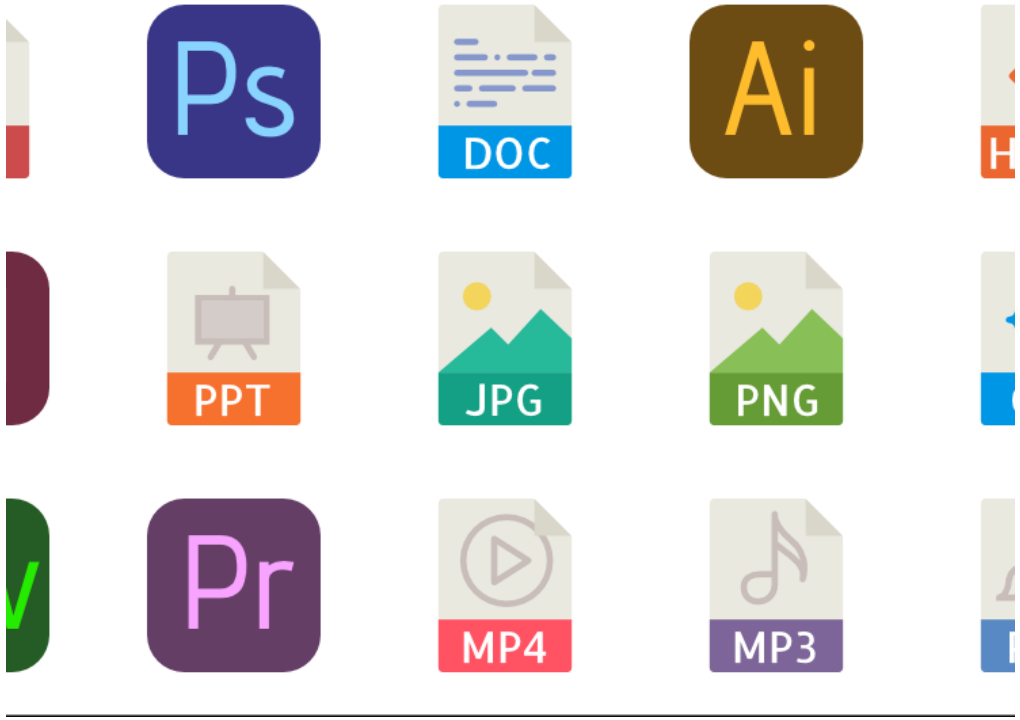
An example of the HTTP response message is below:

HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
Content-Length: 44
Connection: close
Content-Type: application/json

{page:1, total_pages:100, total_results: 1000
“results”: […

HTTP Protocol: **Content-Type Header**

- The Content-Type tells the client what the content type of the returned content.
- Also known as “MIME” ,”media type”, "content type")
  - a video file might be **audio/mpeg**, or an image file **image/png**).
- The Internet Assigned Numbers Authority (IANA) is the official authority for the standardization and publication of these classifications.

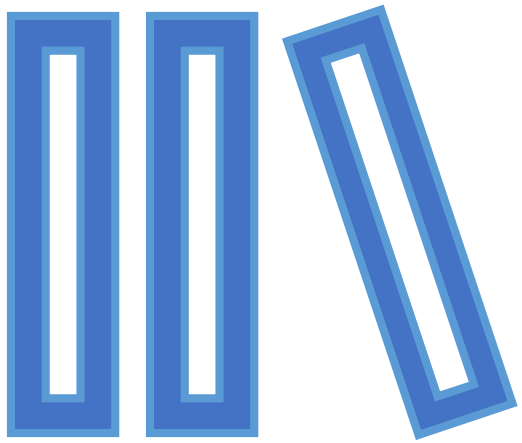


HTTP Protocol: Methods (or Verbs)

- GET
  - Request resources without sending data
- POST
  - Can be used to create new resources with data that you are sending
- PUT/PATCH
  - Modify/ Partially Modify objects with data that you are sending
- DELETE
  - Delete objects without sending data

Safe Method (no action on server/resource, "idempotent")

Usually contains body
(the data sent to server)
Changes stuff!

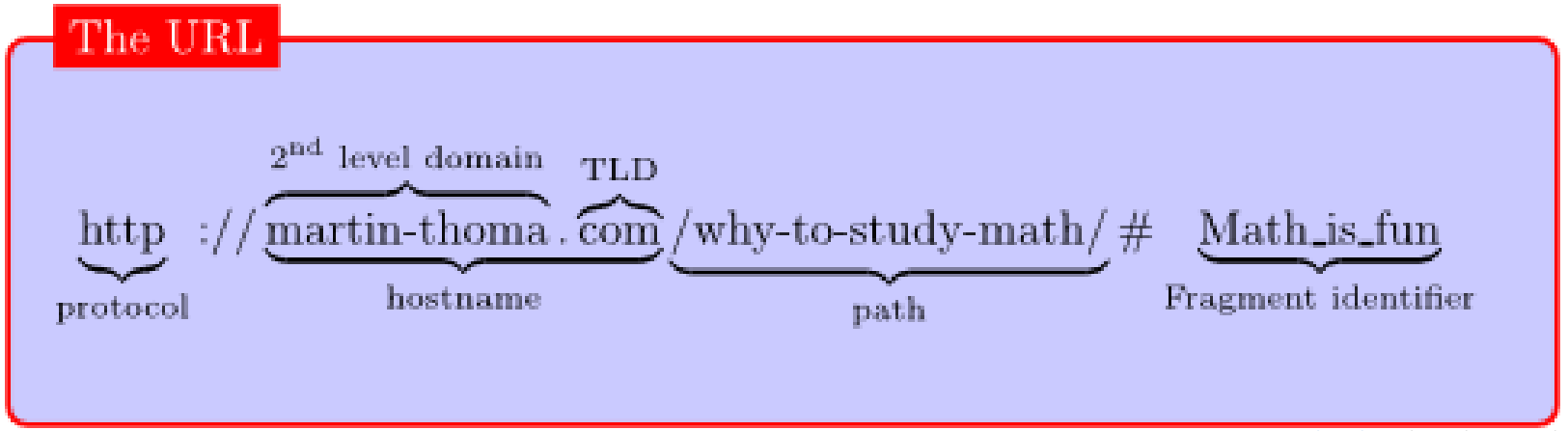


Uniform Resource Indicators

URI

Uniform Resource Locator

- Uniquely identifies a resource over the web. *protocol://hostname:port/path*

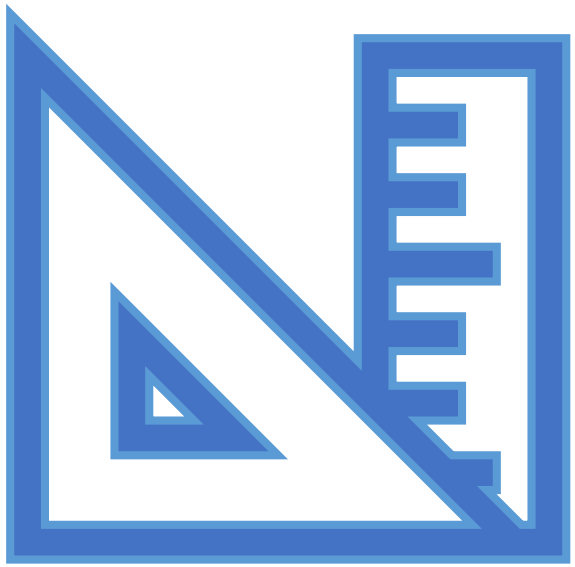


Uniform Resource Locator: Query String

- A query string is a part of a URL that assigns values to specified parameters.
- Often used to filter results returned by API

https://randomuser.me/api?results=10&gender=female

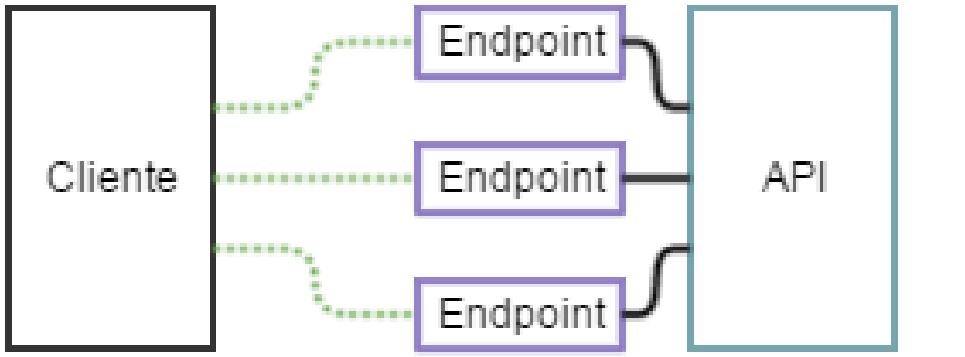
https://api.genderize.io?name=anne



API Design

API Design: Endpoints

- An endpoint is the combination of a HTTP method and an URI
  - GET: /api/friends
- An endpoint can be interpreted as an **action on a resource**.
  - POST: /api/friends means “create a new Friend”



API: Design

- Everything is based around resources
  - the “things” you’re working with are modelled as resources described by URI paths--like /users, /groups, /dogs
  - Notice they are **nouns .**
  - **Verbs in URLs are discouraged**
- The things that you do on these things (or nouns) are characterised by the fixed set of HTTP methods
  - What GET,POST,PUT does is something that the designer/developer gets to put into the model.
- The metadata (the adjectives) is usually encoded in HTTP headers, although sometimes in the payload.
- The responses are the pre-established HTTP status codes and body. (200, 404, 500 etc.)
- The representations of the resource are found inside the body of the request and response

Resource/Path	GET	POST	PUT	DELETE
/friends	List friends	Create New Friend	Bulk Update Friends	Not Applicable
/friends/{id}	Details of Friend {id}	Not Applicable	Update details of Friend {id}	Delete friend {id}

API: Good Practice1

- Always specify **content-type**
- Wrap your responses:
  - Use a standard model for responses to enable easier processing by clients
- Example: TMDB Movie API for GET: /movies?page=1&api_key=c183b23922…
  - Uses a model that defines the page, total pages, total results and results.

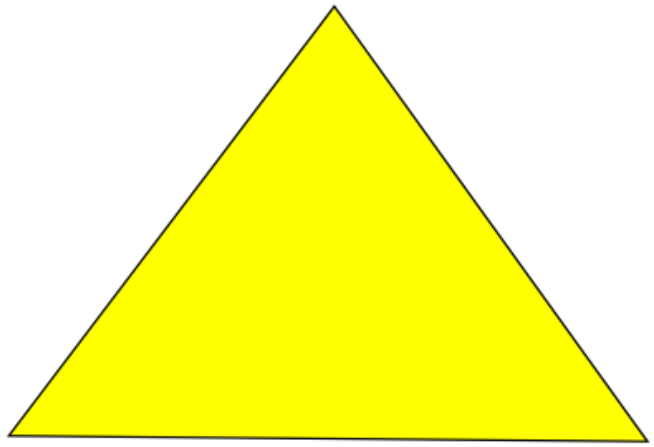

```
{
    "page": 1,
    "total_pages": 500,
    "total_results": 10000,
    "results": [
        {
            "adult": false,
            "backdrop_path": "/hJuDvwzS0SPlsE6MNFOpznQltDZ.jpg",
            "genre_ids
```

API: Good Practice2

- **HTTP Status Codes** are important
- HTTP clients may rely on correct use the standard HTTP status codes ranges.

```
GET: 200 OK
POST: 201 Created
PUT: 200 OK
PATCH: 200 OK
DELETE: 204 No Cor
```

API: Design



Nouns
(Unconstrained)
eg http://wikipedia.org/

Verbs
(Constrained)
eg GET

Content Types
(Constrained)
eg HTML

EXRTA BIT: OpenAPI & Swagger

OpenAPI

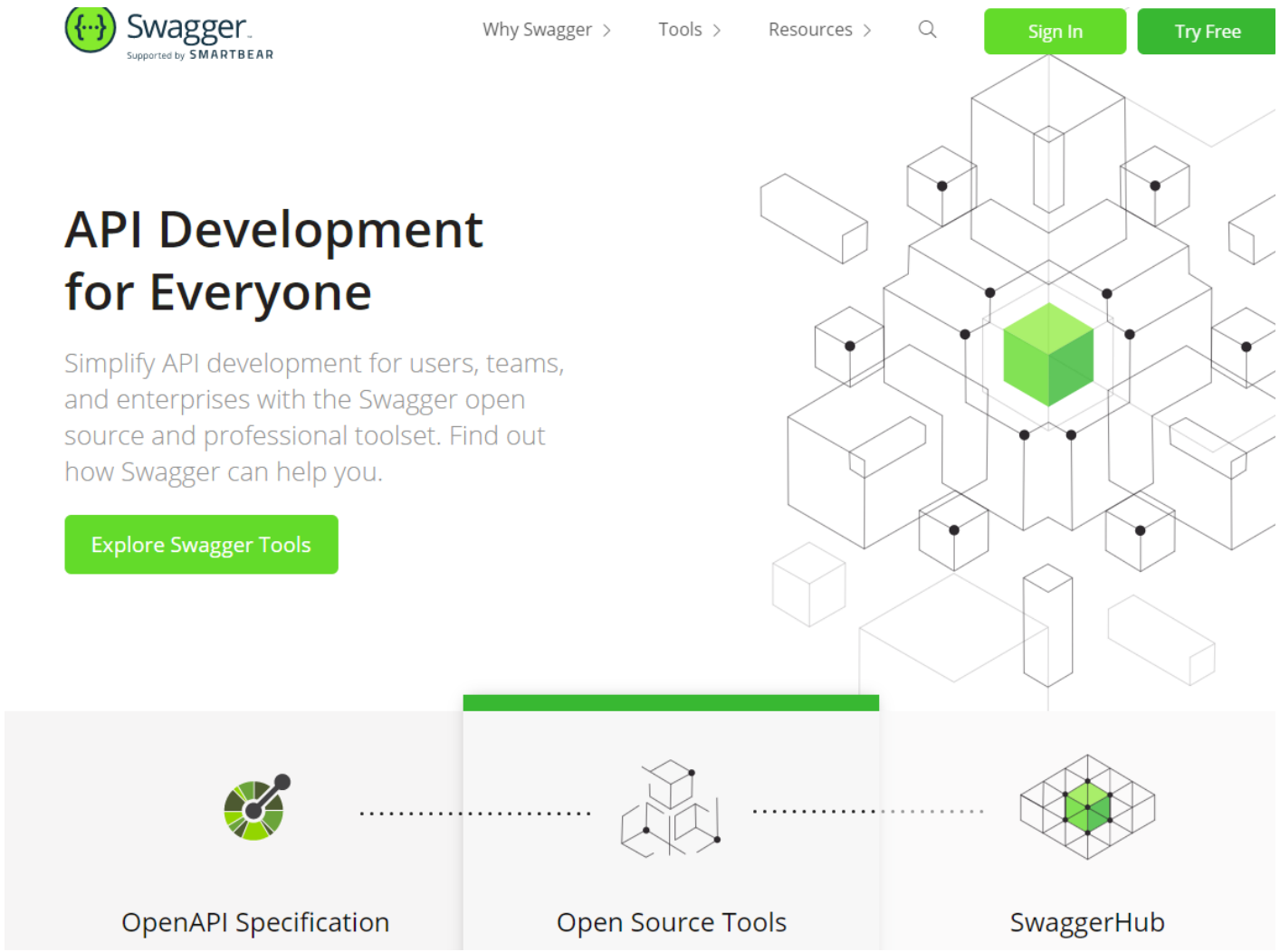
- Specification for machine-readable interface files for describing, producing, consuming, and visualising Restful Web Services
- The OpenAPI Initiative is an open-source collaboration project of the Linux Foundation
- Origins in Swagger… (https://swagger.io/specification/)
- The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs
- YAML can be used to describe an OpanAPI.

Open API: YAML

- Human friendly, cross language, data serialization language.
  - YAML Ain't Markup Language
- Documents begin with --- and end with …
- **Indentation of lines denotes the structure within the document.**
- Comments begin with #
- Members of lists begin with –
- Key value pairs use the following syntax
  - <key>: <value>
- Quick tutorial here
  - https://keleshev.com/yaml-quick-introduction

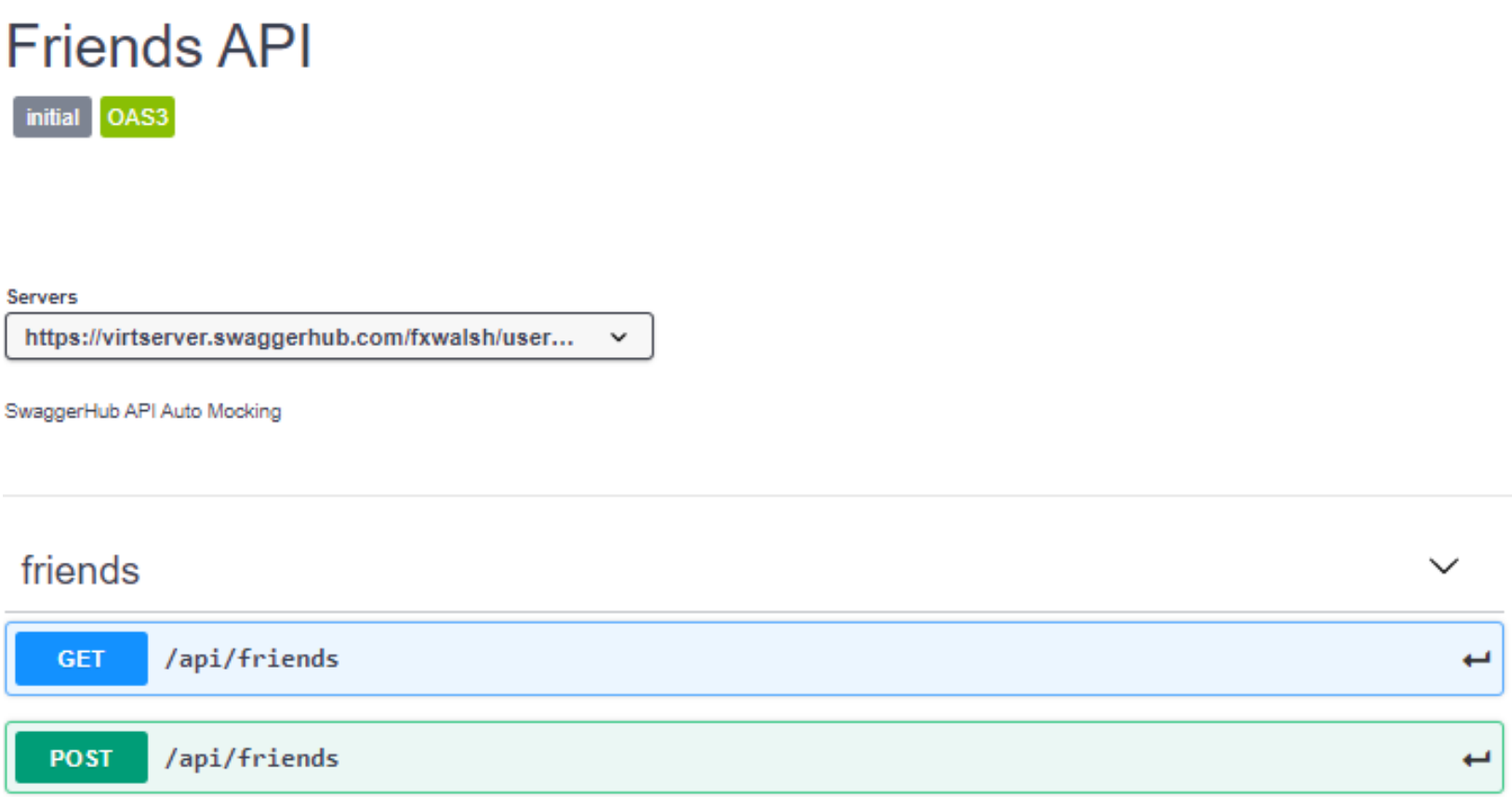
```yaml
---
key: value
map:
    key1: "foo:bar"
    key2: value2
list:
  - element1
  - element2
# This is a comment
listOfMaps:
  - key1: value1a
    key2: value1b
  - key1: value2a
    key2: value2b
---
```

```
some
other
listing
```


Swagger
Supported by SMARTBEAR
Why Swagger
Tools
Resources
Sign In
Try Free
API Development for Everyone
Simplify API development for users, teams, and enterprises with the Swagger open source and professional toolset. Find out how Swagger can help you.
Explore Swagger Tools
OpenAPI Specification
Open Source Tools
SwaggerHub

Swagger

Demo


Friends API
initial
OAS3
Servers
https://virtserver.swaggerhub.com/fxwalsh/user...
SwaggerHub API Auto Mocking
friends
GET
/api/friends
POST
/api/friends