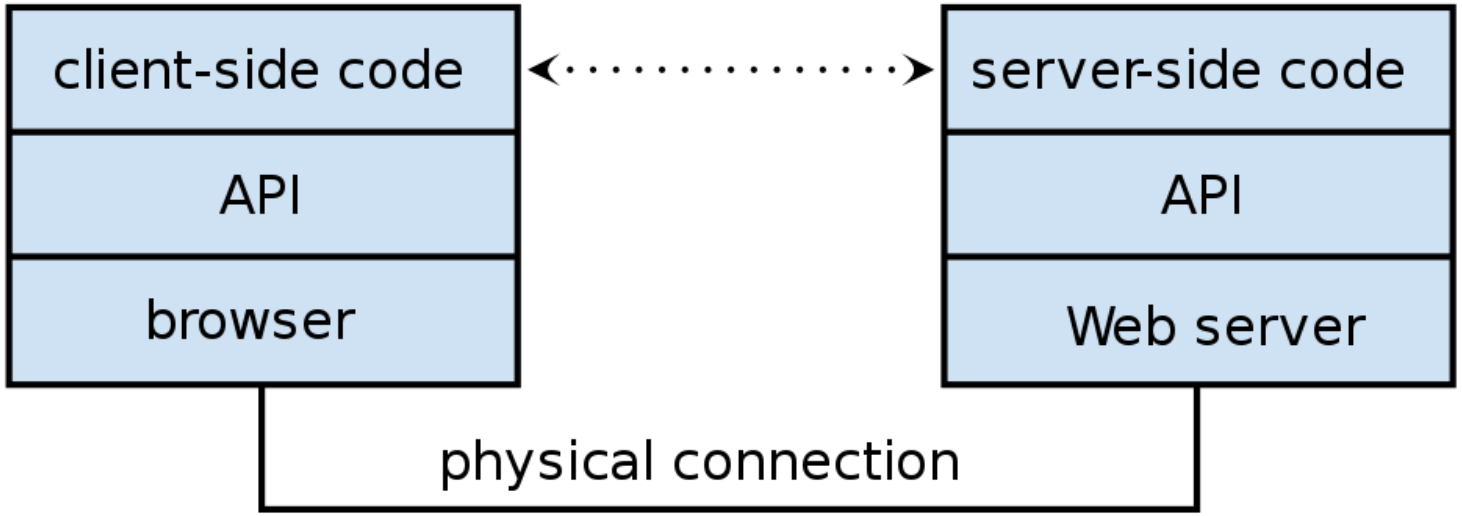


Web API Design

Frank Walsh

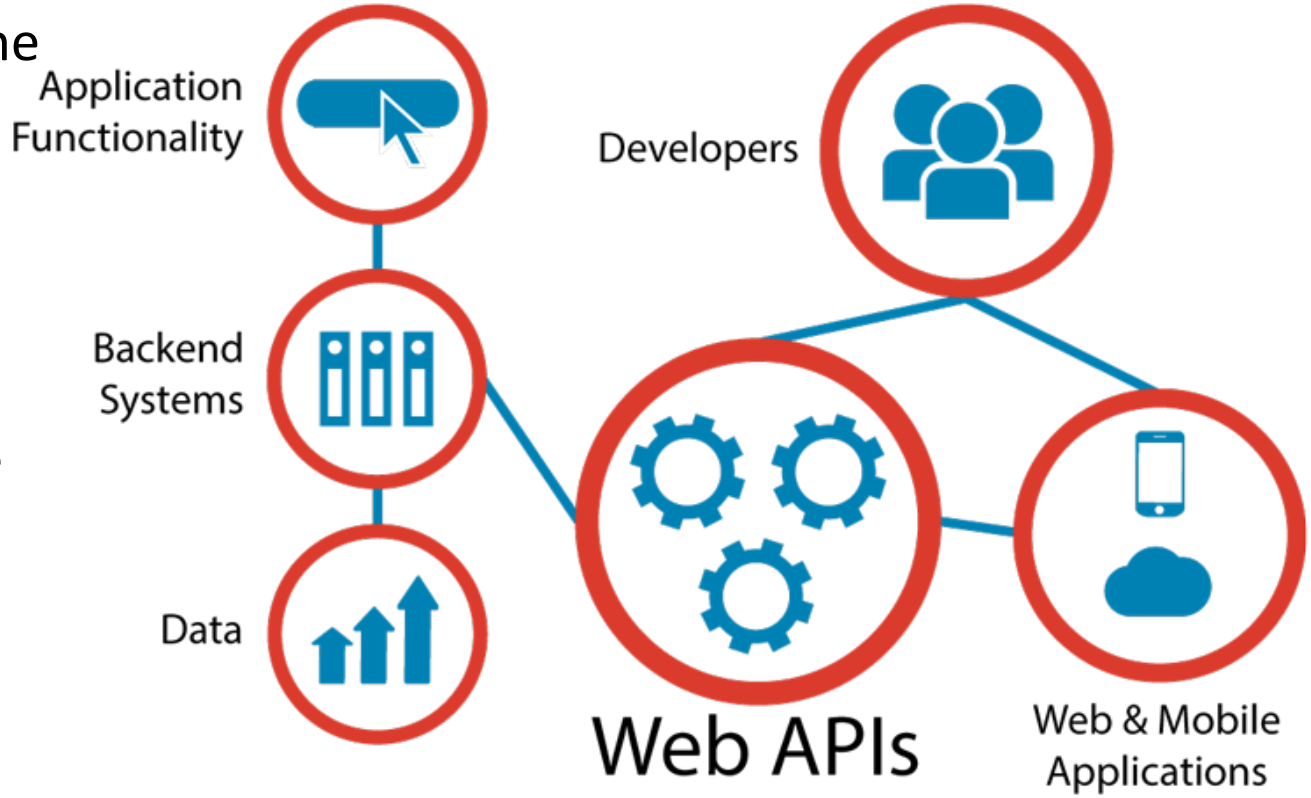
Agenda

- Web API
- REST
- API Value
- API Design

Web APIs

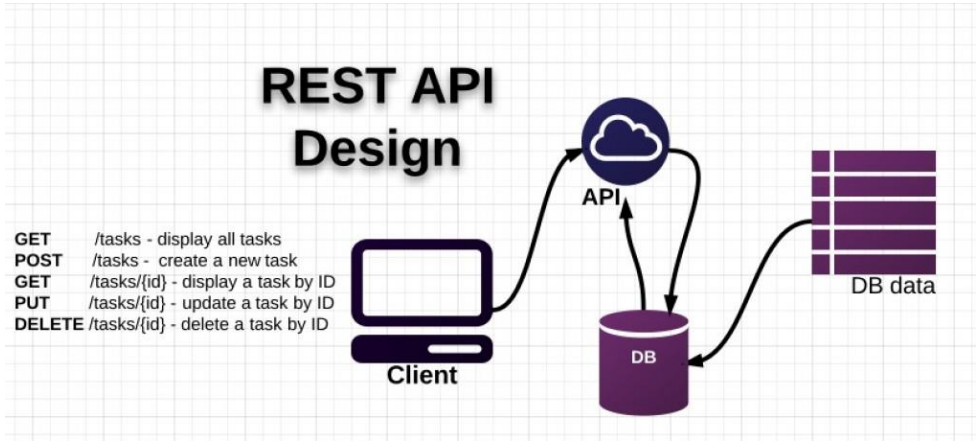
Web APIs

- 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 Restful web service described in previous lectures.
- Typical use:
  - Expose application functionality via the web
  - 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 composeable
  - If designed properly(e.g. microservice architecture)
  - See later slides
- Concentrate on function first rather tha data

APIs in the Internet of Things

- Many new IoT devices being released.
- Devices are limited on their own
  - It's the innovative use of those devices with accompanying APIs that generate value
- "Build a better mousetrap, and the world will beat a path to your door" - Ralph Waldo Emerson
  - Rentokil believe they have using APIs(https://www.computerworlduk.com/it-business/rentokil-on-iot-rat-traps-cash-for-apps-incentives-apis-3612866/)
  - Rentokil increased operational efficiency through the automatic notifications of a caught animal and its size

API Design

- Use principle of developer-first
  - put target developers' interests ahead of other considerations
  - Strive for a better developer experience
- Commit to RESTful APIs
- Use a Interface Description Language like:
  - RESTful API Markup Language (RAML)
  - Swagger (YAML/JSON)
- Take a grammatical approach to the functionality
- Keep interface simple and intuitive

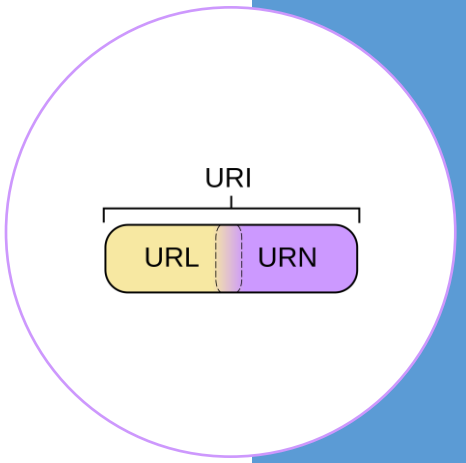
REST

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

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

Key REST Principles

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



API Design

- In Rest, 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 BAD**
- 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

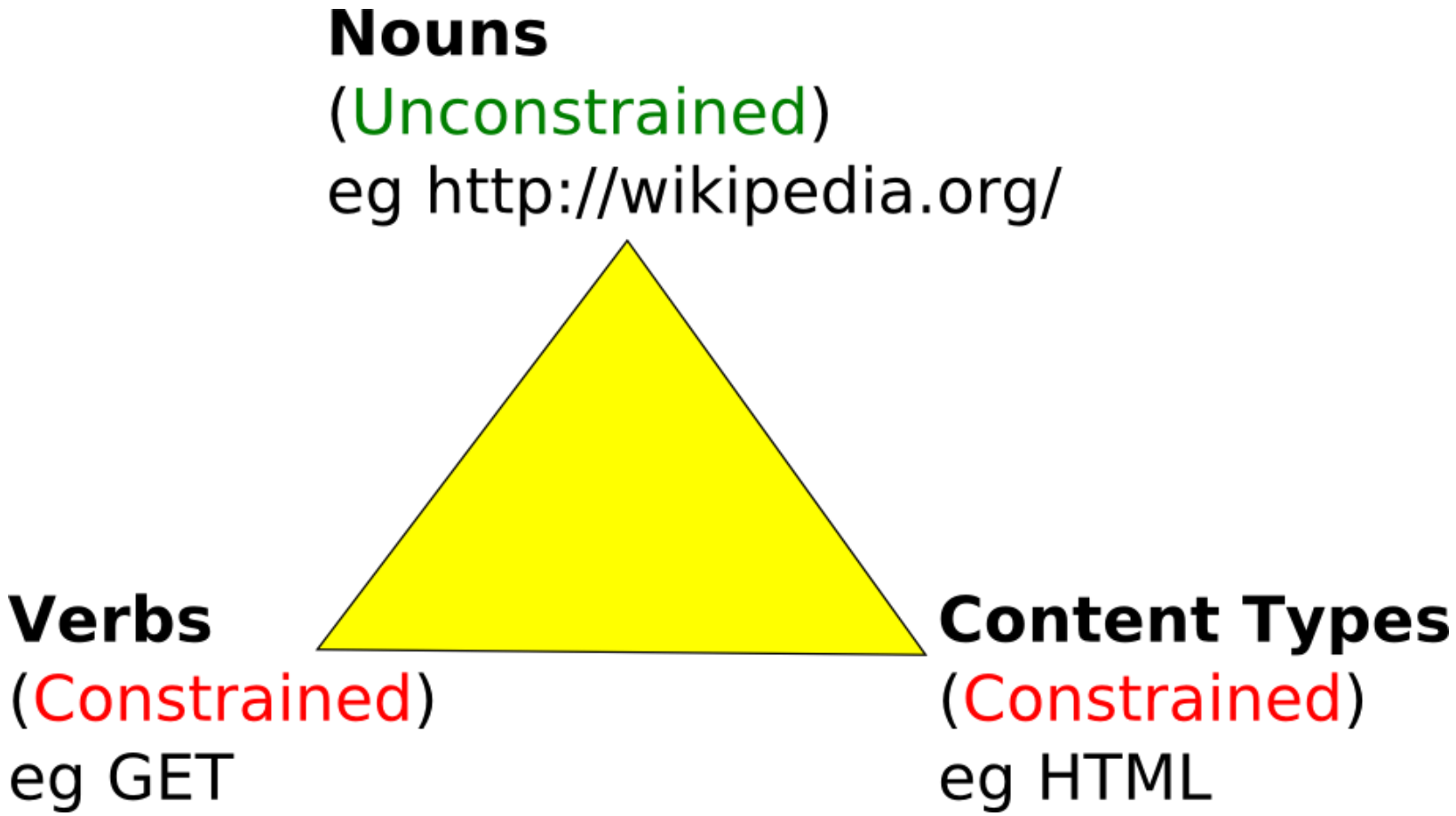
Resource/Path	GET	POST	PUT	DELETE
/dogs	List dogs	Create New Dog	Bulk Update dogs	Not Applicable
/dogs/{id}	Details of Dog {id}	Not Applicable	Update details of dog {id}	Delete dog {id}

API Design - Containment

- URIs embed ids of “child” resources
- Post creates child resources
- Put/Delete for updating /removing resources

Resource	GET	POST	PUT	DELETE
/api/posts	get all posts	add a post	N/A	N/A
/api/posts/:postId	get a post	N/A	update a post	N/A
/api/posts/:postId/upvotes	N/A	upvote a post	N/A	N/A
/api/posts/:postId/comments	Get comments for post	Add a comment	N/A	N/A
/api/posts/:postId/comments/:commentId/upvotes	get upvotes	upvotes a comment	N/A	N/A

API Design



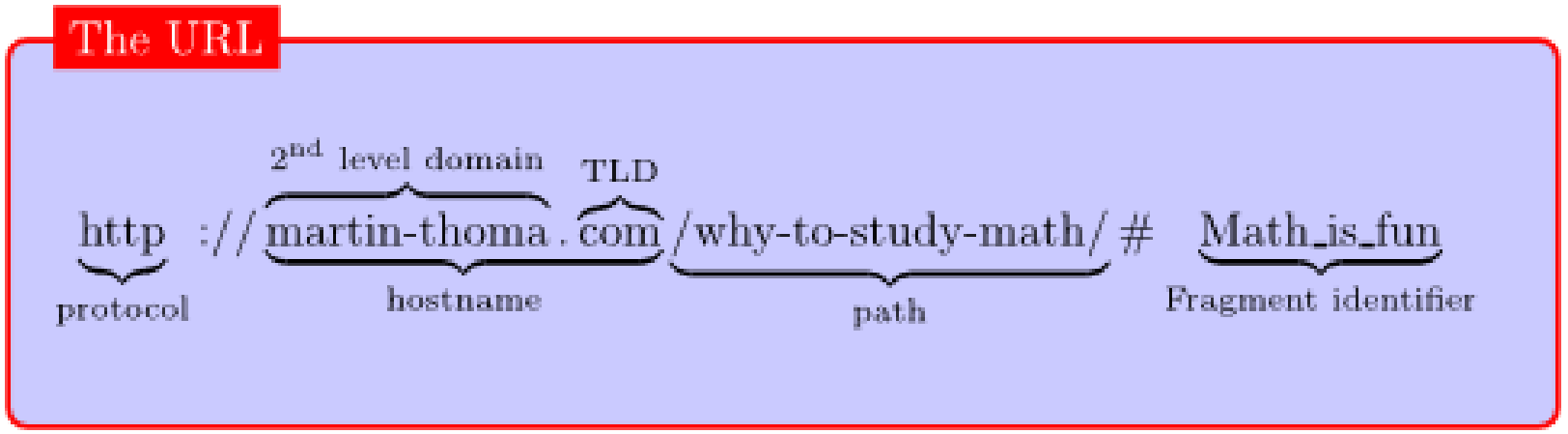
This Photo by Unknown Author is licensed under CC BY

URLs

- Uniquely identifies a resource over the web. *protocol://hostname:port/path*
- Query string used to include data in a URL. For example

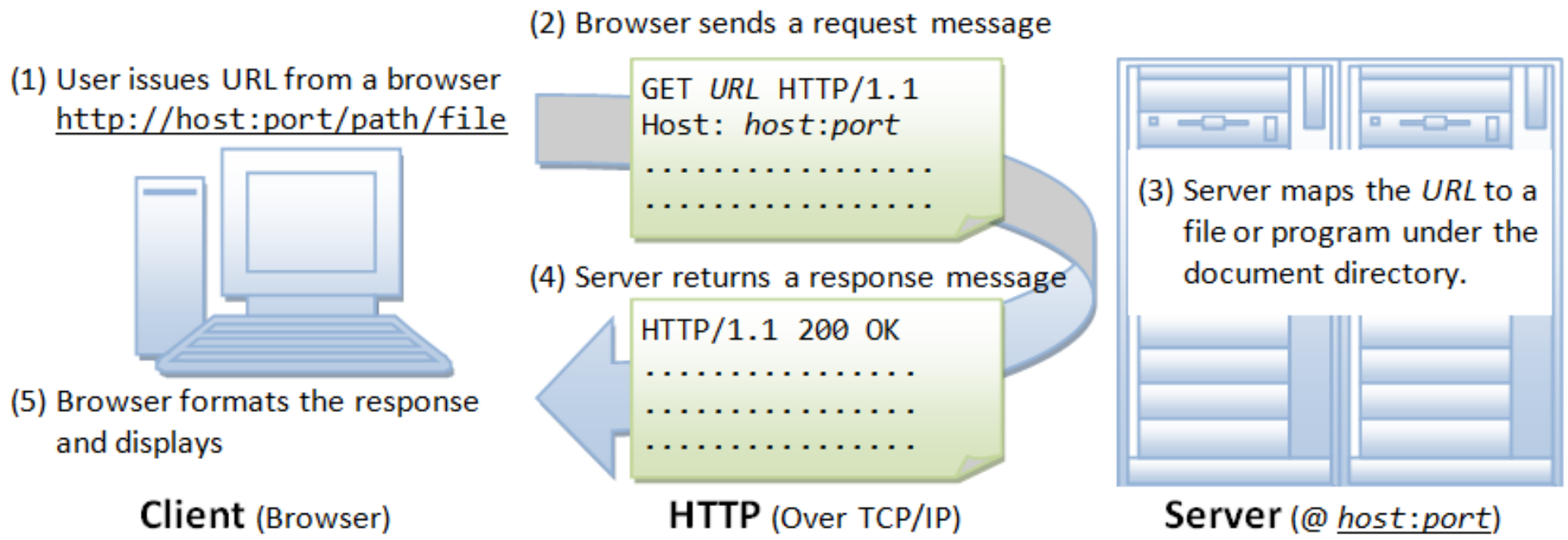
https://www.myhome.com/heating?status=on

This Photo by Unknown Author is licensed under CC BY-SA



HTTP request from App.

Browser:



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, the browser translated the URL http://www.nowhere123.com/doc/index.html into the following request message:

```
GET /docs/index.html HTTP/1.1
Host: www.nowhere123.com
Accept: image/gif, image/jpeg, */*
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: text/html

<html><body><h1>It works!</h1></body></html>

HTTP Methods

- 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!

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.

This Photo by Unknown Author is licensed under CC BY

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
```

This Photo by Unknown Author is licensed under CC BY-SA

Swagger

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