SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Lec4 – Web APIs

50.012 Networks

Jit Biswas

Cohort 1: TT7&8 (1.409-10)

Cohort 2: TT24&25 (2.503-4)

Introduction

- Todays lecture: The application layer
  - Data formatting and parsing
  - Why we need web service APIs
  - Idempotence, safe methods
  - More details on HTTP verbs
  - Example interactive RESTful API
  - Real-world REST APIs
  - HTTP proxies

Motivation

Data formatting and parsing

- Data you send/receive over a socket is just a bitstring
- You can't send objects or similar directly over a socket
- Either you know how to interpret the data, or you have to derive it somehow
- The first option is more efficient for static setups
  - You know to cast the first 32 bit into an int, bit 33 as a Boolean. . .
  - Watch out for platform dependent ordering (1=0001 vs. 1=0100)
- The second option is more flexible (e.g. using JSON, XML, . . .)
  - You can re-create objects like hash maps on the fly
- Sending everything in ASCII is wasteful
  - E.g. decimal numbers: You use 8 bits to send <4 bits of info

SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

Ways to simplify your server

- Have you programmed a server application?
  - What was it doing?
  - What does the general structure look like?
  - What were the challenges?

Ways to simplify your server

- Have you programmed a server application?
  - What was it doing?
  - What does the general structure look like?
  - What were the challenges?

- Tracking each connection in a state machine can be painful
- Requires more code and memory during runtime
- HTTP is stateless for that reason
  - ➢ Representational state transfer (REST) is the general name used for such an architecture
- Client stores state, will re-submit on next connection

Representational state transfer (REST)

- Developed in context of HTTP and WWW, now used for many scalable web services (e.g., Google services, Yahoo, . . .)
- REST is a set of principles on how Web standards (e.g., HTTP and URLs) are supposed to be used
- APIs using REST are supposed to be more scalable
- REST focuses on accessing named resources through a single consistent interface
- General principles
  - Stateless communication
  - Give every resource an ID
  - API exposes standard HTTP methods
  - Support caching
  - Resources can have multiple representations

Design Challenge – API

- You wrote a novel server application
  - Users can submit own content
  - Users can see and vote on other’s content
  - Users can delete their own uploads
  - Users can change their own uploads

How can you maximize the use of your service?

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Design Challenge – API

- You wrote a novel server application
  - Users can submit own content
  - Users can see and vote on other's content
  - Users can delete their own uploads
  - Users can change their own uploads

How can you maximize the use of your service?

Nice website

- Nice app
- Nice desktop application
- Plugins for third parties
- How to connect all that together?

Design Challenge – API II

What you need is:

- An API that allows applications to exchange data
  - Data can be ASCII or binary
  - Create, Read, Update, and Delete objects
  - Easy to interact with (for third parties)
  - Supported by Internet Core and middleboxes
- How would you build that?
- Can the API be self-explanatory?

Web APIs

Commonly Used HTTP Verbs

▪ GET	retrieves a representation of a resource without side-effects (nothing changes on the server).
▪ HEAD	retrieves just the resource meta-information (headers) i.e. same as GET but without the response body – also without side-effects.
▪ OPTIONS	returns the actions supported for specified the resource – also without side-effects.
▪ POST	creates a resource.
▪ PUT	(completely) replaces an existing resource.
▪ PATCH	partial modification of a resource.
▪ DELETE	deletes a resource.

HTTP verbs (more detailed explanation)

- **GET**: requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect
- **HEAD:** The HEAD method asks for a response identical to that of a GET request, but without the response body, (thus avoiding having to transport the entire content).
- **POST:** The POST method requests that the server accept the entity enclosed in the request as a new subordinate of the web resource identified by the URI. The data POSTed might be, for example,
  - an annotation for existing resources;
  - a message for a bulletin board, newsgroup,
  - a mailing list,
  - a comment thread;
  - a block of data that is the result of submitting a web form to a data-handling process;
  - an item to add to a database.
- **PUT**: requests that the enclosed entity be stored under the supplied URI. If already existing resource, it is modified; if not, it is created.
- **DELETE:** The DELETE method deletes the specified resource.

HTTP verbs – further details

- PUT for CREATE possible
- PUT should provide all fields, not a subset
- Idempotence of GET and PUT
- POST not necessarily idempotent
- CRUD – GET, HEAD and DELETE map well
- PUT and POST don't
- POST – recommended return of the object in its entirety (given that it is not too large)
- Sometimes data plan limits make this infeasible
- Similarly, esp. for Android developers – many platform dependent optimizations must be made

Response Status Codes and Reason Phrase

- The first digit of the Status-Code defines the class of response:
  - 1xx: Informational - Request received, continuing process
  - 2xx: Success - The action was successfully received, understood, and accepted
  - 3xx: Redirection - Further action must be taken in order to complete the request
  - 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
  - 5xx: Server Error - The server failed to fulfill an apparently valid request

Two basic type of resources – collections and instances

- Collections
  - /applications
- Instances
  - /applications/a1b2c3

URI, URN, URLs

- Terminology:
  - URI = Uniform Resource Identifier

    A generic URI is of the form:

    scheme:[//[user[:password]@]host[:port]][/path][?query][#fragment] *
  - URL = Uniform Resource Locator
  - URN = Uniform Resource Name
- URNs and URLs are both URIs
- What is the difference?
  - URI: Uniquely identifies a resource
  - URL: identifies + provides location of resource
  - URN: identifies, persistent in time (e.g., after deletion)

Web Server Gateway Initiative (WSGI)

- The python application in Lab 2 is a server-side web application
- The HTTP server is a separate component
  - Directly accepts incoming connections
  - Determines which resource is requested on which host
  - Forwards the request to the right web application (or answers)
- WSGI is a standardized interface between server and app
  - Applications using WSGI can be run on different webservers
- In our lab 2, the web server component is actually Werkzeug
  - The application can also be loaded by Apache (with `mod_wsgi`), or nginx

Idempotence and safe methods

- HTTP uses concepts of idempotence and safe methods
  - Safe methods enable caching and load distribution
  - Idempotence allows to handle lost confirmations by re-sending
- Safe methods will not modify (non-trivial) resources on server
  - Example: GET and HEAD do not change the resource on server
- Idempotent methods may modify resources on the server,
  - But can be executed multiple times without changing outcome
  - Example: duplicate DELETE operations have no additional effect
- POST is not idempotent, multiple POSTs have multiple effects
  - Example: multiple rooms are created for POST /rooms

SOAP

- Simple Object Access Protocol (SOAP)
  - Specific protocol for XML-based data exchange
  - Web service definition language (WSDL) is used to specify available service to client
  - Specific protocol seems to add overhead in many cases
  - Popular in enterprise machine-to-machine communication
- Some IDEs support automatic "learning" of API through WSDL
  - Client-side functions to call APIs are automatically generated
  - *Auto-completion for API calls*

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Example SOAP login call

```
POST https://na1.salesforce.com/services/Soap/c/10.0 HTTP/1.1
Content-Type: text/xml; charset=utf-8
SOAPAction: ""
Content-Length: 510
Expect: 100-continue
Host: na1.salesforce.com

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/xmlns:urn="urn
:enterprise.soap.sforce.com">
   <soapenv:Body>
      <urn:login>
         <urn:username><b>user@domain.com</b></urn:username>
         <urn:password><b>secret</b></urn:password>
      </urn:login>
   </soapenv:Body>
</soapenv:Envelope>
```

Taken from salesforce.com

REST

- Representational state transfer (REST)
  - "Architectural style"
  - Data is often exchanged as XML or JSON
  - No WSDL, any interaction is done on nouns /resources, using Create, Read, Update, Delete (CRUD) operations
  - Reduction to nouns and 4 verbs simplifies API
  - Popular for web service APIs (e.g. stackexchange, facebook, imgur)
- Two types of resources in REST
  - Collections: /rooms
    - Container, referencing other things
  - Instances: /rooms/1.502.14
    - Single instance (representing my office)
- Resources are referenced in the HTTP header
  - GET /rooms/1.502.14 HTTP/1.1

Why REST ?

- Scalability
- Generality
- Independence
- Latency
- Security
- Encapsulation

https://www.youtube.com/watch?v=5WXYw4J4QOU

Why JSON?

- Ubiquity
- Simplicity
- Readability
- Scalability
- Flexibility

https://www.youtube.com/watch?v=5WXYw4J4QOU

Example REST login

```
POST /accounts/login HTTP/1.1
Host: api.nse.sg
Accept: application/json, text/plain, */*
Content-Type: application/json;charset=utf-8
Content-Length: 58

{’password’: ’secret’, ’username’: ’user@domain.com’}
```

Adapted from NSE project APP

PUT vs POST

- Normally, POST is used to create new resources (and get ID), PUT is used to update
- POST can be used to create an element in a collection, without explicit name
  - Server will reply with 201 message with URL of created element
- PUT can be used to update an existing resource
  - Reply will be 200
- Uncommon use (correct, but confusing):
  - PUT can be used to create a resource at user chosen location
    - Reply would be 201 for newly created resource
  - POST can be used to update some properties of a resource, if accessed directly
    - Reply will be 200, with full representation returned

What you do not want in your API

- Your API should be focused on resources (nouns) rather than verbs
- Do not build something like this:
  - myserver.com/doStuff
  - myserver.com/doOtherStuff
  - myserver.com/foobar
  - This will not scale, you end up with unstable API

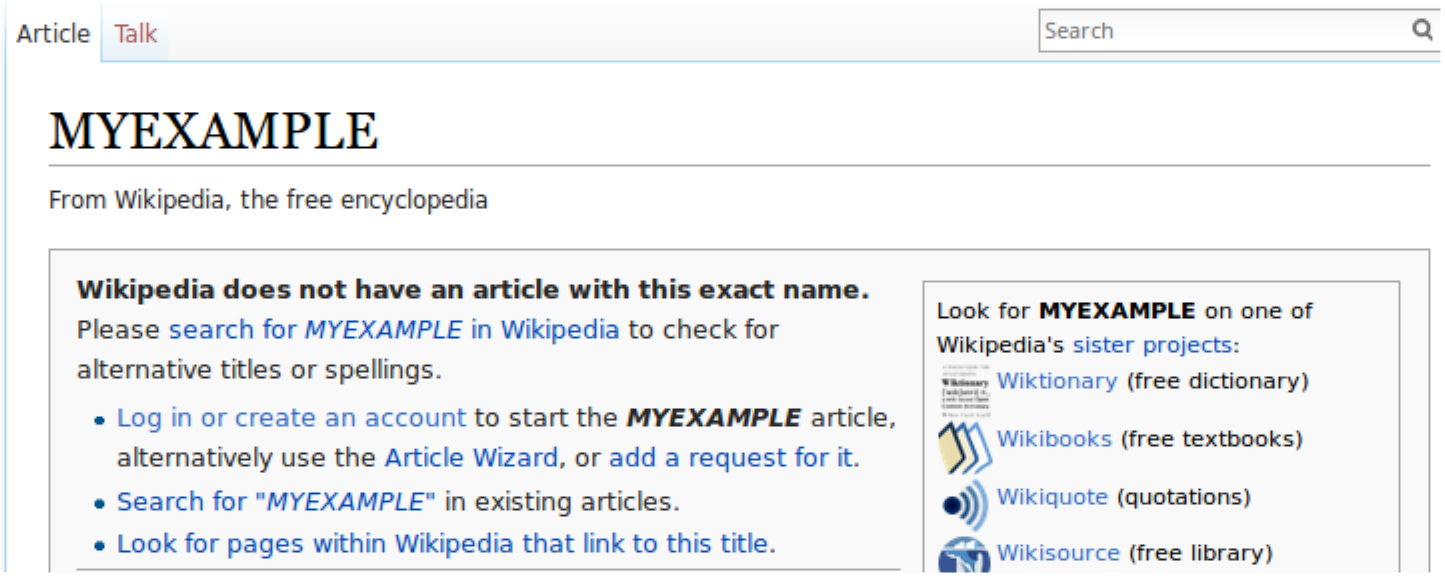
REST Resource naming

- Instead, you will access resources via the HTTP verbs
    - GET, PUT, POST, HEAD, DELETE
- Resources are Collections or Instances
- Examples:
    - example.com/customers/33245/orders
    - example.com/products/66432
    - example.com/customers/33245/orders/8769/lineitems/1
- There could be multiple URLs to access the same data
- GET parameters can also be used, best used for optional arguments
    - `http://bibs.scy-phy.net/bibs?author=tippenhauer&reverse=True`
- More here:

`http://www.restapitutorial.com/lessons/restfulresourcenaming.html`

Dynamic content creation

- In simple static settings each resource corresponds to single file
  - Example: simple static .htm files on your web server
- In dynamic settings, the content at the resource URL might not exist yet
  - The server will interpret the URL as parameters
  - Dynamically create content for the provided parameters
  - Example: `http://en.wikipedia.org/wiki/MYEXAMPLE`



MYEXAMPLE

From Wikipedia, the free encyclopedia

Wikipedia does not have an article with this exact name. Please search for *MYEXAMPLE* in Wikipedia to check for alternative titles or spellings.

- Log in or create an account to start the ***MYEXAMPLE*** article, alternatively use the Article Wizard, or add a request for it.
- Search for "*MYEXAMPLE*" in existing articles.
- Look for pages within Wikipedia that link to this title.

Look for **MYEXAMPLE** on one of Wikipedia's sister projects:

Wiktionary (free dictionary)

Wikibooks (free textbooks)

Wikiquote (quotations)

Wikisource (free library)

Interactive REST demo

- Let's experiment a bit with a demo REST API:

  `http://jsonplaceholder.typicode.com/`

- Try adding posts using POST/PUT (difference?)
- Try reading info about a specific post using GET
- To interact, recommended: `curl -X GET http://jsonplaceholder.typicode.com/posts/1` or similar
- To send JSON: `curl -H "Content-Type: application/json" -X PATCH -d '{"title":"Mytitle"}' http://jsonplaceholder.typicode.com/todos/199`

REST API examples

- Github REST API `(https://developer.github.com/v3/)`
- Stackexchange API:

  `https://api.stackexchange.com/docs`
- curl is a useful command line tool to GET HTTP
  - try the -v switch for more information on exchanged messages
- Recommended: requests as Python library to test these

ACTIVITY 4: Experimenting with python ‘requests’

Try out the following on using python on your laptop along with the requests library. (you may need to use python3). Submit your answers on eDimension:

```
>>> r = requests.get('https://api.github.com/user',\
    auth=('user', 'pass'))
>>> r.status_code
YOUR ANSWER
>>> r.headers['content-type']
YOUR ANSWER
>>> r.encoding
YOUR ANSWER
>>> r.text
YOUR ANSWER
>>> r.json()
YOUR ANSWER
```

REST authentication

- Three basic options (without HTTPS client auth):
  - No authentication
  - Identification (claiming an ID)
  - Authentication (verifying an ID)
- Identification: e.g., Google Maps API key for billing
  - Possession of ID value is enough
  - You can distribute this in APP
- Authentication:
  - Basic HTTP authentication in header field (username:password)
  - More advanced authentication tokens (e.g. OAuth)
  - Basic HTTP authentication is often good enough if used over HTTPS

Resource Representations

- URLs represent unique resources
- Same resource might have different representations
  - Example: .png or .svg version of same image
- HTTP header or request specifies accepted format:
  - Example: "Accept: application/json"
  - HTTP reply will contain: "Content-Type: application/json"
  - These types are MIME meta data (originally for mail)
- But now we have file ending, and content type, why both?
  - Filename extensions are explicit legacy way
  - Requesting foo/bar with image/png is alternative
- foo.com/nils.png vs. foo.com/nils (with ’accept: image/png ’)
  - Latter is preferred, both yield same result

Further RESTful tutorials on the web

- Designing a Beautiful REST+JSON API

`https://www.youtube.com/watch?v=5WXYw4J4QOU`

- Google Android REST tutorial:

`https://www.youtube.com/watch?v=xHXn3Kg2IQE`

HTTP Proxies

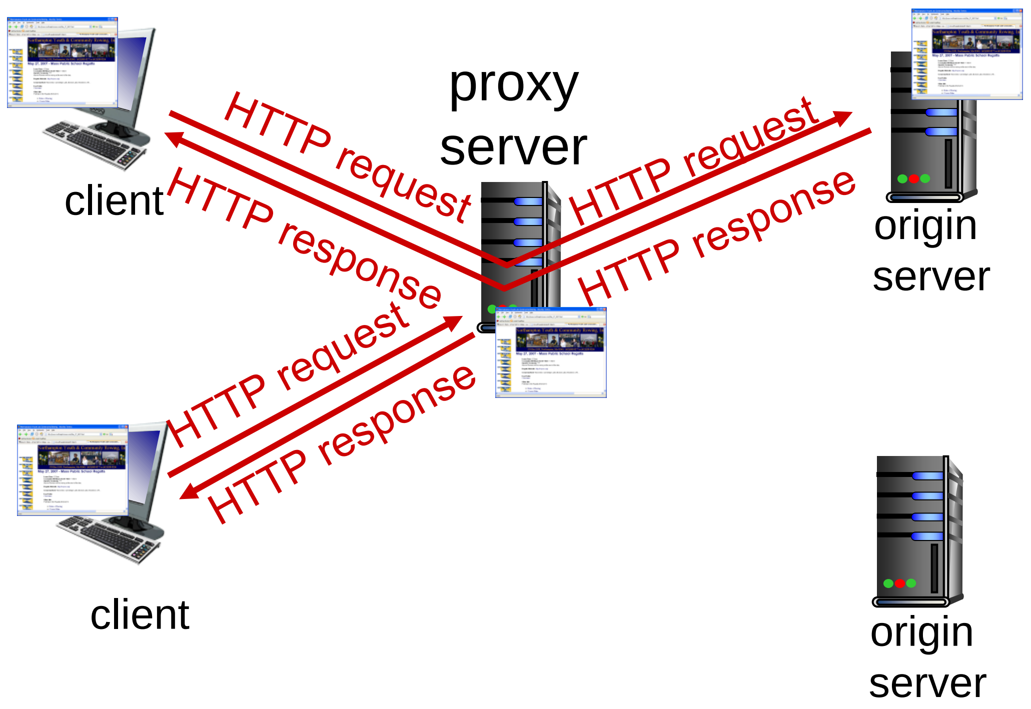
Caching and proxies

- Proxy IRL: "a person authorized to act on behalf of another"
- An intermediate server that is performing requests for us
    - For example, to allow us to access the Internet with private IP
- A caching proxy keeps copies of resources for the client
    - E.g. results of HTTP GET queries
    - Results of non-idempotent operations are not cached, i.e. POST
- These cached results are served to subsequent queries
    - These clients do not have to be the same as original clients
    - As long as GET was requesting the same resource

Web caches (proxy server)

goal: satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



More about Web caching

why Web caching?

- reduce response time for client request
- reduce traffic on an institution's access link

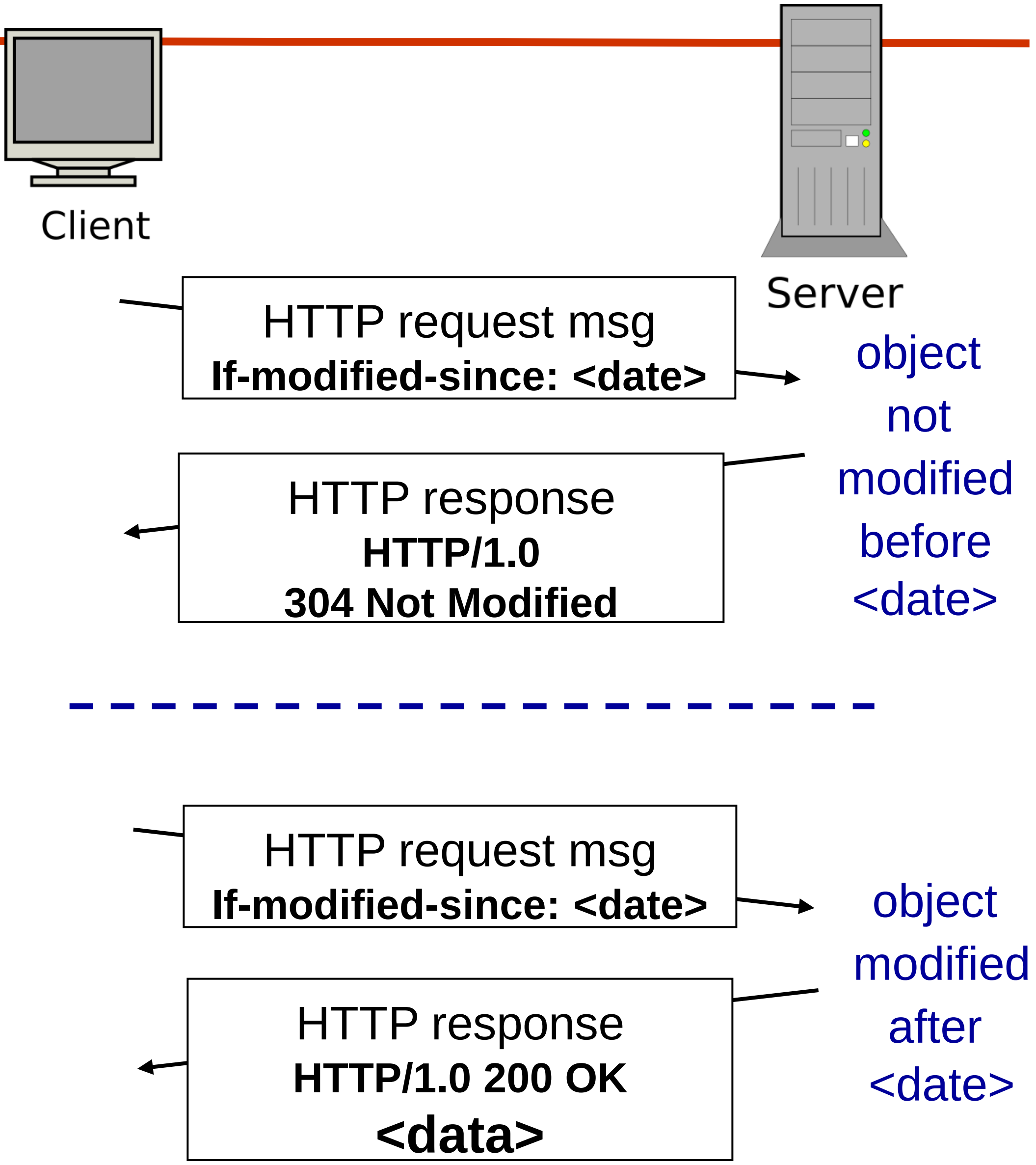
- typically cache is installed by ISP (university, company, residential ISP)
- cache acts as both client and server
  - server for original requesting client
  - client to origin server

Conditional GET

- *Goal: don't send object if cache has up-to-date cached version*
  - no object transmission delay
  - lower link utilization
- *cache: specify date of cached copy in HTTP request*

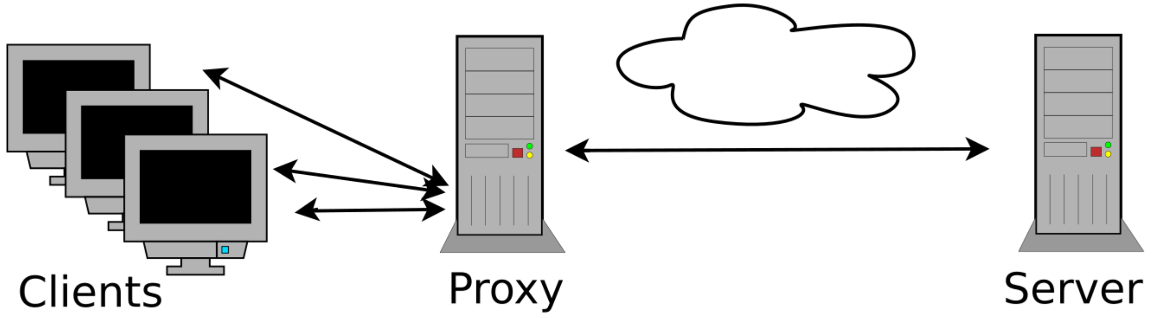
  `If-modified-since: <date>`
- *server: response contains no object if cached copy is up-to-date:*

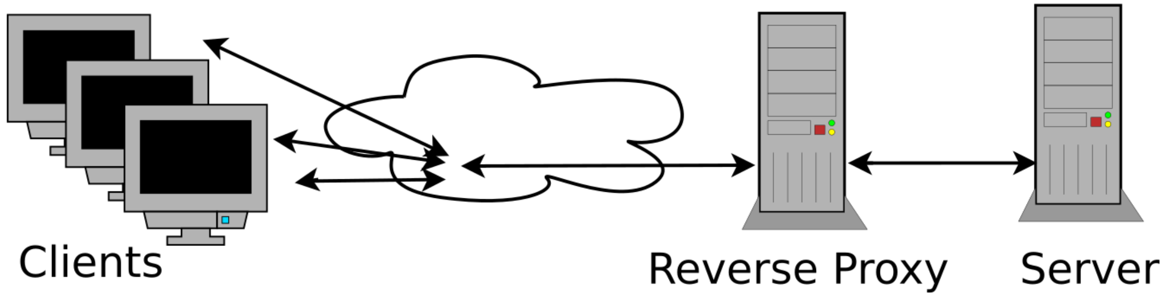
  `HTTP/1.0 304 Not Modified`



Proxy Architectures

- Forward Proxies
  - Content filters: by only forwarding selected queries and results
  - Content logging and eavesdropping: by logging requested resources
  - Content accelerators: by reducing delay and load on outgoing connections
- Reverse Proxies can also cache queries in front of servers





Open API (public API)

An **open API** (often referred to as a public API) is a publicly available **application programming interface** that provides developers with programmatic access to a proprietary software application or web service.

Source: Wikipedia

- A software intermediary that makes it possible for application programs to interact with each other and share data. Often implemented in the REST paradigm, thus protecting the application.
- Open APIs are published on the internet and shared freely. A startup software company, for example, might publish a series of APIs to encourage third-party developers in vertical industries to be innovative and come up with new ways to use the startup's software product.

https://searchmicroservices.techtarget.com/definition/open-API?track=NL-1823&ad=922465&src=922465&asrc=EM_NLN_98755286&utm_medium=EM&utm_source=NLN&utm_campaign=20180810_Word%20of%20the%20Day:%20open%20API

Benefits and Drawbacks of Open APIs

Benefits

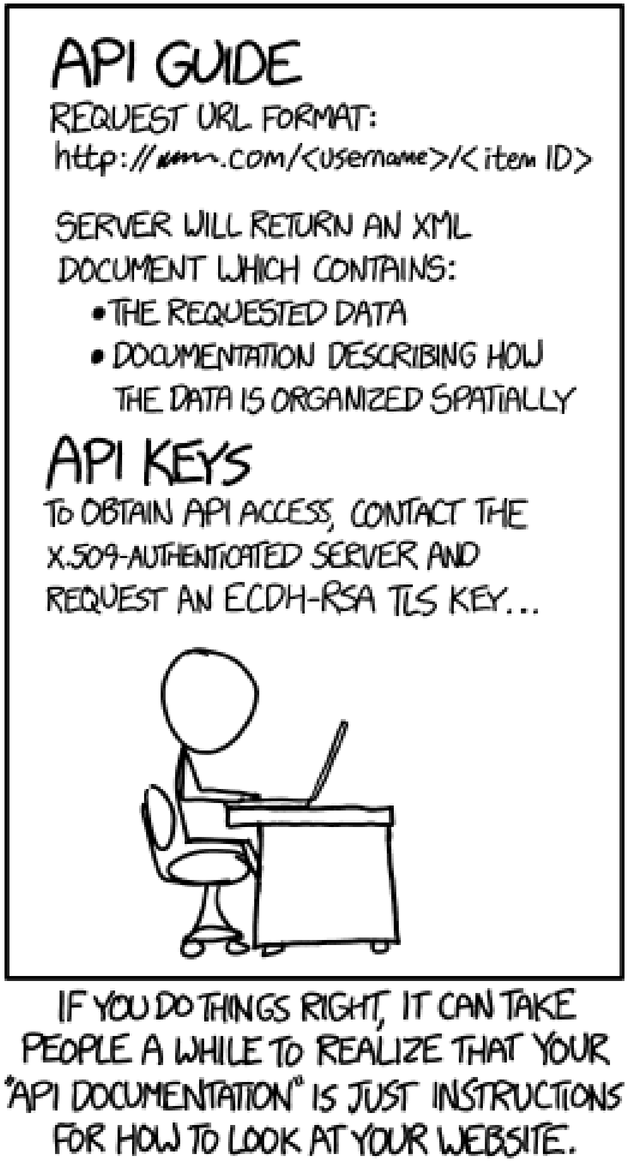
- Third-party developer can make money by licensing a new program, a mashup with advanced functionalities or an innovative use case.
- Meanwhile, the open API's publisher gets to expand its user base without having to spend any money, and keeps source code proprietary.

Drawbacks

- The reputation of the company can improve or suffer damage depending upon how the open API is received by the developers using it. It's crucial to avoid problems with open APIs. The should be treated just like any application an organization might release to the public.

Conclusion

- Why we need web APIs
- SOAP
- REST
- More details on RESTful APIs
- HTTP proxies
- Open APIs


API GUIDE
REQUEST URL FORMAT:
http://[illegible].com/<username>/<item ID>
SERVER WILL RETURN AN XML DOCUMENT WHICH CONTAINS:
• THE REQUESTED DATA
• DOCUMENTATION DESCRIBING HOW THE DATA IS ORGANIZED SPATIALLY
API KEYS
TO OBTAIN API ACCESS, CONTACT THE X.509-AUTHENTICATED SERVER AND REQUEST AN ECDH-RSA TLS KEY...
IF YOU DO THINGS RIGHT, IT CAN TAKE PEOPLE A WHILE TO REALIZE THAT YOUR "API DOCUMENTATION" IS JUST INSTRUCTIONS FOR HOW TO LOOK AT YOUR WEBSITE.