

TK2100: Informasjonssikkerhet

Lesson 08: HTTP and Web

Dr. Andrea Arcuri
Westerdals Oslo ACT
University of Luxembourg

Goals

- Understand how Browsers communicate with Web Servers using HTTP/S
- Understand how Web Applications are architected
- Base knowledge of how to do AJAX calls in JavaScript

[Gmail](#)

[Images](#)



[Sign in](#)



Google Search

I'm Feeling Lucky

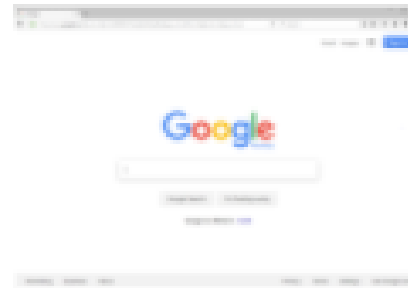
Google.no offered in: [norsk](#)

World Wide Web (WWW)

- Invented by Tim Berners-Lee in 1989, at CERN, Switzerland
 - I was in high school when it was first available...
- WWW is a set of documents/resources distributed among different machines
- Resources/documents are identified with a Uniform Resource Locator (URL)
- Resources are accessed/downloaded over the *internet*, typically HTTP over TCP
- A web page is a resource written in HTML format
- *Browsers* are tools used to download/visualize HTML pages, and enable the following of *links*



www.google.com



Send a HTTP request, and get back a HTML page which will be visualized in the browser

(Briefly) HTTP

- Hypertext Transfer Protocol (HTTP)
- When browser connects to server, will be a TCP connection on a given port (usually 80 and 443)
- HTTP represents how the messages are structured
 - eg, retrieve a certain file
 - eg, send data, like username/password for login
- **HTTPS**: the S stands for Security. Protocol in which HTTP is encrypted

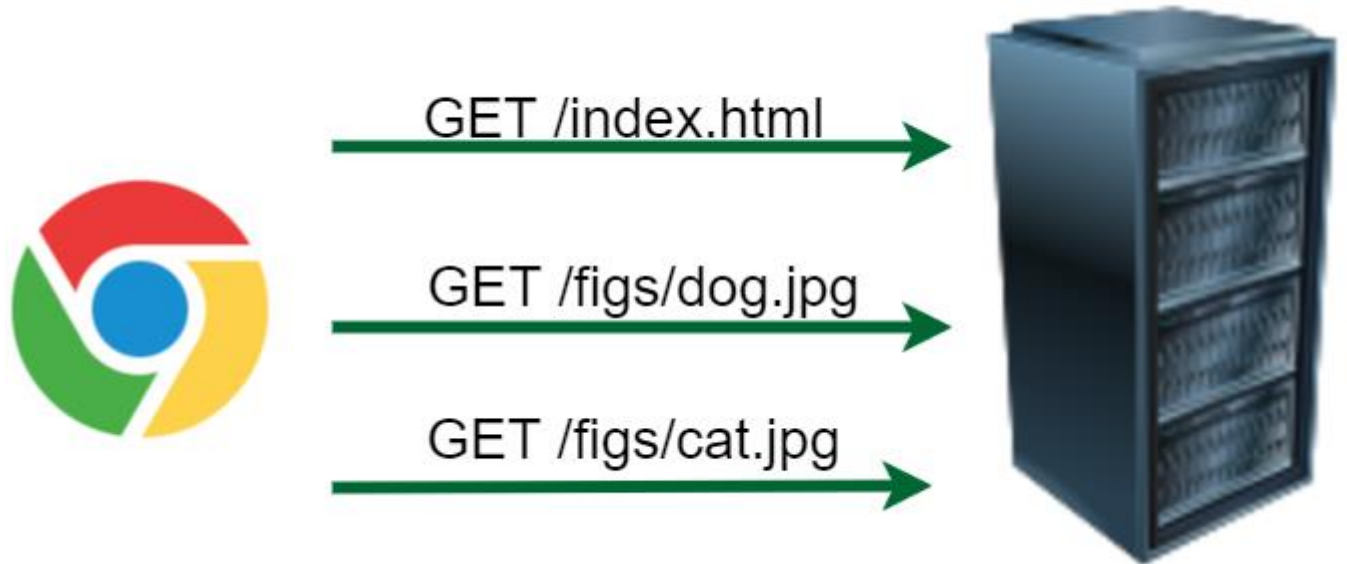
A Cat and a Dog



```
<HTML>
  <HEAD>
    <TITLE>Example</TITLE>
  </HEAD>
  <BODY BGCOLOR="gray">
    <h2> A Cat and a Dog </h2>
    <DIV>
      <IMG SRC="figs/cat.jpg" style="width:256px">
      <IMG SRC="figs/dog.jpg" style="width:256px">
    </DIV>
  </BODY>
</HTML>
```


Links to other files

- To display the page in the previous slide, after a GET of the *index.html* page, the browser will do 2 further HTTP GET requests
- Those further 2 requests could be done in parallel on two different TCP connections



Many browsers...

- Chrome (from Google)
- Firefox (from Mozilla)
- Edge (from Microsoft)
 - Mainly used to download Chrome/Firefox
- Safari (from Apple)
 - Mainly used to download Chrome/Firefox, but OK on iPhone
- Opera
- ... and others: PaleMoon, SeaMonkey, etc.

Alright Edge, let's see who you really are!



Internet Explorer???



Conversation Between Browsers

What are we?



Browsers!



...



Browsers! Browsers!



What do we want?



Faster!



...



Faster!



Faster!



When do we want it?



Now!



...



Now!



Now!



Browsers!



VERY IMPORTANT: Chrome -> More Tools -> Developer tools

The screenshot shows the Google Chrome browser with the Developer Tools panel open. The browser window displays the Google logo and the text "Google.no offered in: norsk". The Developer Tools panel is set to the "Network" tab, showing a list of requests. The selected request is "www.google.com", which is a GET request with a status code of 302. The "Response Headers" section is expanded, showing various headers including "alt-svc", "cache-control", "content-length", "content-type", "date", "location", and "status".

Google Chrome Developer Tools interface showing the Network tab. The selected request is `www.google.com` with status 302. The response headers are visible:

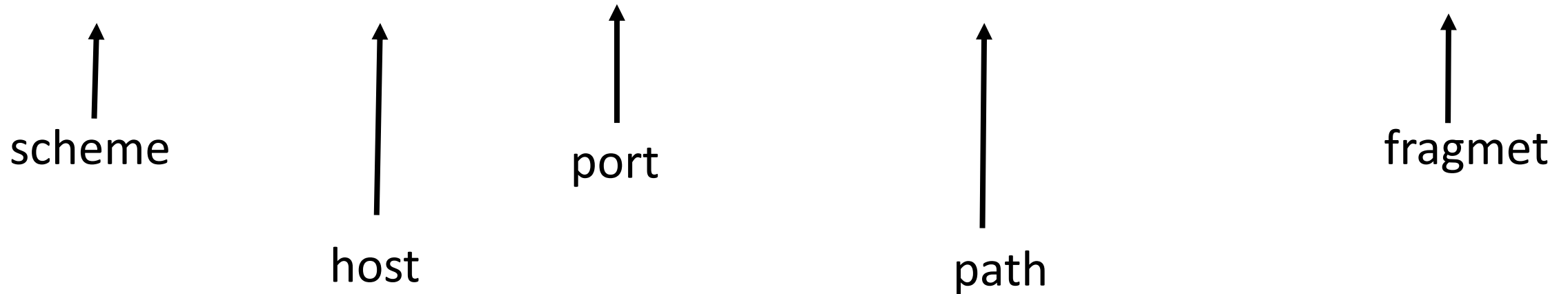
- General**
 - Request URL: `https://www.google.com/`
 - Request Method: `GET`
 - Status Code: `302`
 - Remote Address: `81.175.29.152:443`
- Response Headers**
 - `alt-svc`: `quic=":443"; ma=2592000; v="35,34"`
 - `cache-control`: `private`
 - `content-length`: `259`
 - `content-type`: `text/html; charset=UTF-8`
 - `date`: `Tue, 17 Jan 2017 11:14:36 GMT`
 - `location`: `https://www.google.no/?gfe_rd=cr&ei=nPx9WPGfCOXk8Aed6JGYAw`
 - `status`: `302`
- Request Headers**
 - Provisional headers are shown
 - `Upgrade-Insecure-Requests`: `1`
 - `User-Agent`: `Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36`

Defaults

- When nothing specified, browsers do default to known ports for the given protocol
 - Default protocol is HTTP, and default resource is the root “/”
- So typing *www.google.com* is equivalent to ***http://www.google.com:80/***
- Typing *https://www.google.com* is equivalent to ***https://www.google.com:443/***
- Note: the page you request might not be the one you will get, as you could get a HTTP *3xx redirection*

URL (Uniform Resource Locator)

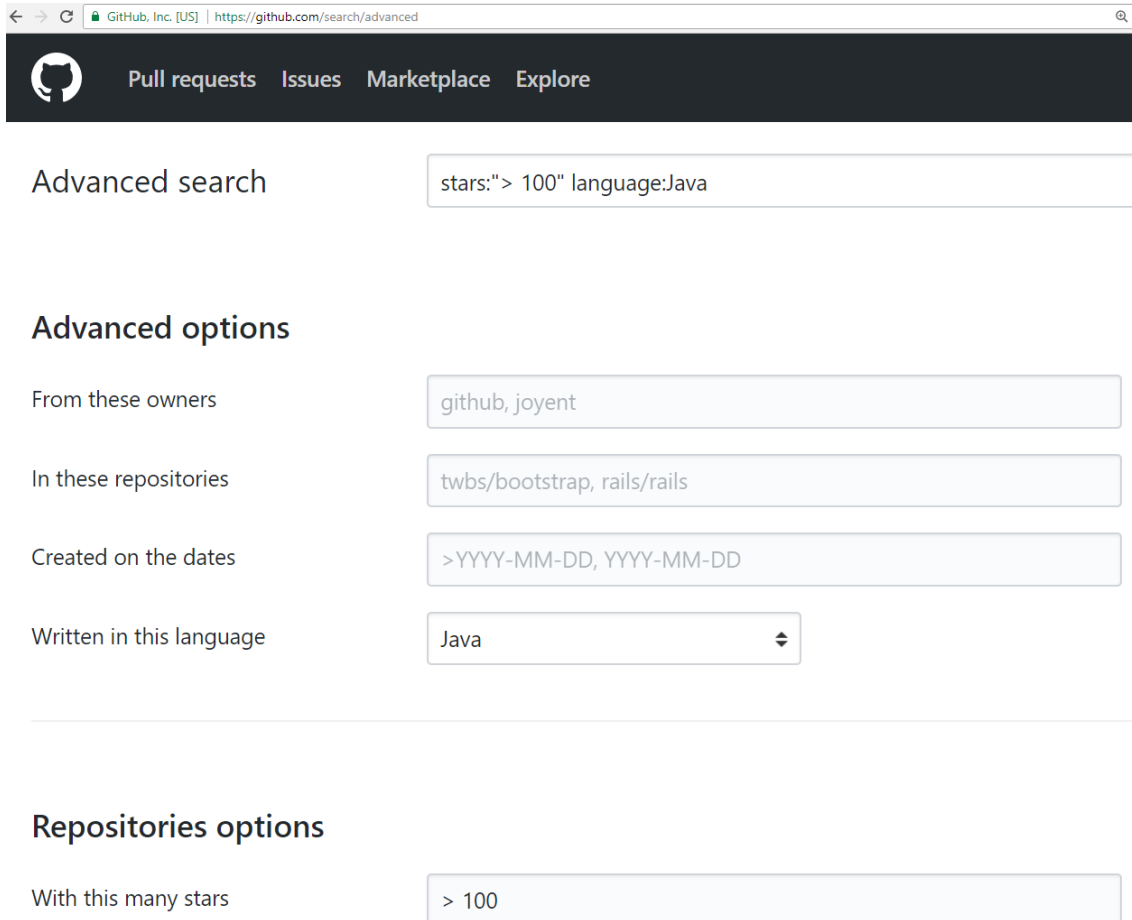
- Reference to a web resource and how to retrieve it
- **scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]**
- https://en.wikipedia.org:443/wiki/Uniform_Resource_Locator#Syntax



Cont.

- **Scheme:** how to access the resource
 - http, https, file, ftp, etc.
- **Host:** the name of the server, or directly its numeric IP address
- **Port:** the listening port you will connect to on the remote server
- **Path:** identifies the resource, usually in a hierarchical format
 - Eg, /a/b/c
- **Query:** starting with “?”, list of <key>=<value> properties, separated by “&”
 - eg `https://github.com/search?q=java&type=Repositories&ref=searchresults`
- **Fragment:** identifier of further resource, usually inside the main you requested
 - Eg, a section inside an HTML page

- <https://github.com/search?utf8=%E2%9C%93&q=stars%3A%22%3E+100%22+language%3AJava&type=Repositories&ref=advsearch&l=Java&l=>
- The asked page/resource is **/search**, where it is retrieved in different ways based on the list of query “?” parameters



Advanced search

stars:"> 100" language:Java

Advanced options

From these owners

github, joyent

In these repositories

twbs/bootstrap, rails/rails

Created on the dates

> YYYY-MM-DD, YYYY-MM-DD

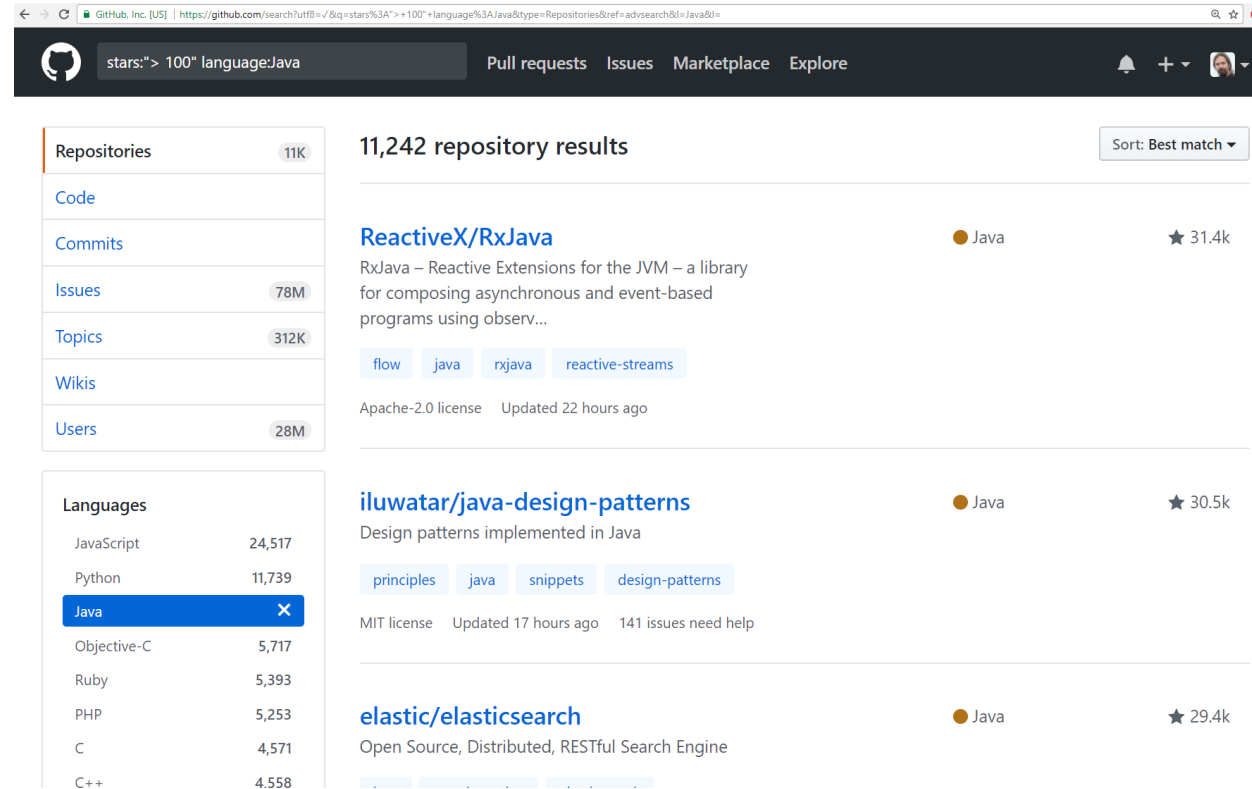
Written in this language

Java

Repositories options

With this many stars

> 100



stars:"> 100" language:Java

11,242 repository results

Sort: Best match

Repositories	11K
Code	
Commits	
Issues	78M
Topics	312K
Wikis	
Users	28M

Languages	
JavaScript	24,517
Python	11,739
Java	X
Objective-C	5,717
Ruby	5,393
PHP	5,253
C	4,571
C++	4,558

ReactiveX/RxJava

Java

★ 31.4k

RxJava – Reactive Extensions for the JVM – a library for composing asynchronous and event-based programs using observ...

flow java rxjava reactive-streams

Apache-2.0 license Updated 22 hours ago

iluwatar/java-design-patterns

Java

★ 30.5k

Design patterns implemented in Java

principles java snippets design-patterns

MIT license Updated 17 hours ago 141 issues need help

elastic/elasticsearch

Java

★ 29.4k

Open Source, Distributed, RESTful Search Engine

URI (Uniform Resource Identifier)

- String of characters used to identify a resource
- A URL is a URI:
 - Exactly same format
 - In URL, the resource is typically located on a network
 - Given a URL, you should be able to access the resource, which is not necessarily true for URI
- The distinction between URL and URI is conceptually very thin
 - Most people use the two terms interchangeably

HTTP

HTTP History

- Started at CERN in 1989
- 1995: version 0.9
- 1996: version 1.1
- 1999: “updates” to 1.1
- 2014: more “updates” to 1.1
- 2015: version 2.0

Http Versioning: What a Mess!!!

- HTTP is one the **worst** examples of versioning done **wrong**
- Changing specs and semantics over 18 years, but still keeping the same version number **1.1!!!**
- Why? To support the largest number of browsers, even very old ones
- Not many people realized there was an update in 2014... you might still find quite a few libraries/tools that wrongly use the 1999 version

RFC (Request for Comments)

Technically, a RFC is not a “standard” yet, but it is de-facto in practice

- RFC 7230, HTTP/1.1: Message Syntax and Routing
- RFC 7231, HTTP/1.1: Semantics and Content
- RFC 7232, HTTP/1.1: Conditional Requests
- RFC 7233, HTTP/1.1: Range Requests
- RFC 7234, HTTP/1.1: Caching
- RFC 7235, HTTP/1.1: Authentication
- RFC 7540, HTTP/2
- Etc.
- For the ones of you that will work as frontend/backend developers, recommended to study these documents in details

HTTP 1.1 vs 2

- v2 is quite recent (2015), and still not so common
- Unless otherwise stated, we'll just deal with v1.1
- From user's perspective, v2 is like v1.1
 - Same methods/verbs, just better optimization / performance improvement
 - More like adding functionalities, not replacing it
- Main visible difference: v1.1 is “text” based, whereas v2 has its own byte format (less space, but more difficult to read/parse for humans)

HTTP Messages: 3 Main Parts

- First line specifying the action you want to do, eg GET a specific resource
- Set of *headers* to provide extra meta-info
 - eg in which format you want the response: JSON? Plain Text? XML?
 - In which language? Norwegian? English?
- (Optional) Body: can be anything.
 - Request: usually to provide user data, eg, login/password in a submitted form
 - Response: the actual resource that is retrieved, eg a HTML page

First line

- <METHOD> <RESOURCE> <PROTOCOL> \r\n
- Ex.: GET / HTTP/1.1
 - <method> **GET**
 - <resource> /
 - <protocol> **HTTP/1.1**
- A resource can be anything
 - html, jpeg, json, xml, pdf, etc.
- A resource is identified by its *path*
 - Recall URI, and such path is same as file-system on Mac/Linux, where “/” is the root

Different kinds of Methods

- **GET**: to retrieve a resource
- **POST**: to send data (in the HTTP body), and/or create a resource
- **PUT**: to replace an existing resource with a new one
- **PATCH**: to do a partial update on an existing resource
- **DELETE**: to delete a resource
- **HEAD**: like a GET, but only return headers, not the resource data
- **OPTIONS**: to check what methods are available on a resource
- **TRACE**: for debugging
- **CONNECT**: tunneling connection through proxy

Method Semantics

- Each of the methods has a clear semantics
 - Eg GET does retrieve a resource, whereas DELETE should delete it
- But *how* the application server does handle them is completely up to it
 - Eg, an application server could delete a resource when a GET is executed

Verbs should not be in paths

- Given a resource “/x.html”
- **Wrong:** GET on “www.foo.org/x.html/delete” to delete “x.html”
 - Here the resource would be “/x.html/delete”
- Also wrong to use query, eg
“www.foo.org/x.html?*method=delete*”
- Paths should represent/identify resources, and NOT actions on those

Idempotent Methods

RFC 7231: “A request method is considered *idempotent* if the intended effect on the server of multiple identical requests with that method is the same as the effect for a single such request...

... if a client sends a ... request and the underlying connection is closed before any response is received, then the client can establish a new connection and retry the idempotent request.”

Which methods are idempotent?


GET 

POST 

DELETE 

PUT 

PATCH 

HEAD 

Headers

- Extra meta-information, besides Method/Resource
- Pairs <key>:<value>
- For example:
 - In which format am I expecting the resource? HTML? JSON?
 - In which language do I want it?
 - Who am I? (important for user authentication)
 - Should the TCP connection be kept alive, or should it be closed after this HTTP request?
 - Etc.

▼ Hypertext Transfer Protocol

> GET / HTTP/1.1\r\n

Host: google.com\r\n

Connection: keep-alive\r\n

Upgrade-Insecure-Requests: 1\r\n

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)

X-Chrome-UMA-Enabled: 1\r\n

X-Client-Data: CKiIyQEIHbJJAQiltskBCMS2yQEIsIrKAQj6nMoBCKmdygE=\r\n

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n

Accept-Encoding: gzip, deflate, sdch\r\n

Accept-Language: en-US,en;q=0.8\r\n

- Request for *www.google.com*
- Recall, you can use WireShark or Chrome Developer Tools

“Host” Header

- For example “Host: google.com\r\n”
- REQUIRED header (it has to be there)
- A server running on an IP x.y.z.w might serve for different host names, eg foo.com, foo.org, foo.no, bar.com, etc., all pointing to the same IP x.y.z.w
- Host names are translated to IP addresses (recall DNS)
- A TCP connection (used for HTTP) knows about IP, not host names
- Without the “Host” header, server would not know for which host the request is for
 - How to differentiate a request for “www.foo.org:80/index.html” from “www.bar.no:80/index.html” running on same server?
- Application-level routing mechanism

“Connection” Header

- To send an HTTP message to *www.foo.org:80*, I need a TCP connection toward *foo.org* on port 80
- What to do after a message is sent and got response?
- Should the server keep the connection open for further requests from the same client?
- Opening a TCP connection is expensive, so should be re-used as much as possible
- But keeping alive a non-needed TCP connection is expensive as well
- Solution: “*Connection: keep-alive*” and “*Connection: close*”

“upgrade-insecure-request” header

- When doing a non-secure request over HTTP, header used to tell the server that, if they support HTTPS, then switch to it
- HTTPS is encrypted
- Note: this is independent from authentication/authorization of a user
- Eg many websites are moving to HTTPS even for displaying resources that do not need a registered user (eg, searches on Google)
- *Why???*
- Encryption is not so expensive / time consuming any more
- Avoid proxies / internet providers to modify the returned responses, eg inject their own **ads** in the retrieved HTML pages

“User-Agent” Header

- Tells information about the client, for example browser type/version, OS, etc.
- The server might reply differently based on the client capabilities
 - This was a huge problem with Internet Explorer
- You will see that most browsers identify themselves as “Mozilla/5.0”
 - I.e., they claim to be compatible with Mozilla version 5.0
 - Mozilla Browser is the old name for FireFox Browser
- *Why???* Some servers, *in the old days*, gave degraded functionalities/pages if agent was not compatible with “Mozilla/5.0”
- Nowadays? Might use “User-Agent” to check if client is a mobile, and give different pages from the ones served to a PC

“Accept” Header

- Specify in which format we expect the response
 - HTML? Plain text? JSON? XML? JPEG? Etc.
- Might specify more than one, where parameter “q in [0,1]” specify preference
- Comma “,” separated list, where each element can have properties with “;”, for example:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,/*;q=0.8*
 - text/html, application/xhtml+xml and image/webp are the main preferences (default q=1)
 - the “+xml” means that the specified format (xhtml) is a syntactically valid xml
 - application/xml;q=0.9 is second preference (0.9 < 1)
 - */* means “any”, and last preference (0.8)

“Accept-Encoding” Header

- To save bandwidth, a resource could be compressed (eg zipped) when sent by the server
- But need to tell the server which kind of algorithm the client (browser) can support, eg be able to decode/unzip
- Accept-Encoding: **gzip, deflate, sdch**\r\n

“Accept-Language” Header

- A web application might support its pages in different languages
- When sending a HTTP request, might use header to specify a preference for the language
- Accept-Language: en-US,en;q=0.8\r\n
 - Main preference “en-US” (default q=1)
 - Second preference “en” (0.8)
- If server does not support a language, might just return default one
- Recall: those headers just express a “preference” of the client
- Note: server might just ignore that header, and give you language based on your IP geolocation (very annoying when travelling...)

“Cookie” Header

- Likely the most important header
- HTTP is stateless, so cannot know if 2 requests come from same user
- User can send an identifying token in the cookie value, given by server in the first response
 - Eg, “cookie: sfjdfg4397217846tyjewfs”
- There is a lot privacy concern regarding cookies, eg can track user behaviors even if s/he does not register and agree
- Depending on the country, there are laws regulating how cookies can be used (eg, have to ask permission to the user)
- We will go into its details later in the course

Custom Headers

- There is a pre-defined, standardized set of HTTP headers
 - Many more than what shown in these slides...
- But you can use any custom header if you want... but clients/servers need to know how to interpret them
- “Usually” (ie convention) custom headers would start with a “x-”, to clearly distinguish them from the standard ones
 - Eg, “x-Chrome-UMA-enabled”

HTTP Body

- After last header, there must be an empty line
- Any data after that, if any, would be part of the payload, ie HTTP body
- Request: needed for **POST**, **PUT** and **PATCH**
- Response: needed for **GET** (also the other methods “might” have body, but **HEAD**)

<http://www.rd.com/wp-content/uploads/sites/2/2016/04/01-cat-wants-to-tell-you-laptop.jpg>



GET with no body

```
▼ Hypertext Transfer Protocol
  > GET /wp-content/uploads/sites/2/2016/04/01-cat-wants-to-tell-you-laptop.jpg HTTP/1.1\r\n
    Host: www.rd.com\r\n
    Connection: keep-alive\r\n
    Cache-Control: max-age=0\r\n
    Upgrade-Insecure-Requests: 1\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
    Accept-Encoding: gzip, deflate, sdch\r\n
    Accept-Language: en-US,en;q=0.8\r\n
```

Have a look at the “Accept” header... there is no “jpg” there, why?

```
▼ Hypertext Transfer Protocol
  > HTTP/1.1 200 OK\r\n
    Date: Tue, 07 Mar 2017 10:26:42 GMT\r\n
    Content-Type: image/jpeg\r\n
  > Content-Length: 210054\r\n
    Connection: keep-alive\r\n
    Cache-Control: public, max-age=14400\r\n
    ETag: "57153573-33486"\r\n
    Last-Modified: Mon, 18 Apr 2016 19:28:51 GMT\r\n
    CF-Cache-Status: HIT\r\n
    Vary: Accept-Encoding\r\n
    Expires: Tue, 07 Mar 2017 14:26:42 GMT\r\n
    Accept-Ranges: bytes\r\n
    Server: cloudflare-nginx\r\n
    CF-RAY: 33bcd8a7557742c1-OSL\r\n
  \r\n
  [HTTP response 1/2]
  [Time since request: 0.020112000 seconds]
  [Request in frame: 852]
  [Next request in frame: 1024]
  [Next response in frame: 1025]
  File Data: 210054 bytes

▼ JPEG File Interchange Format
  Marker: Start of Image (0xffd8)
  > Marker segment: Reserved for application segments - 1 (0xFFE1)
  > Marker segment: Reserved for application segments - 12 (0xFFEC)
  > Marker segment: Reserved for application segments - 1 (0xFFE1)
  > Marker segment: Reserved for application segments - 13 (0xFFED)
  > Marker segment: Reserved for application segments - 14 (0xFFEE)
  > Marker segment: Define quantization table(s) (0xFFDB)
  > Start of Frame header: Start of Frame (non-differential, Huffman coding) -
  > Marker segment: Define Huffman table(s) (0xFFC4)

00000190 0a ff d8 ff e1 00 18 45 78 69 66 00 00 49 49 2a .J.....E xif..II*
000001a0 00 08 00 00 00 00 00 00 00 00 00 00 ff ec 00 .....
000001b0 11 44 75 63 6b 79 00 01 00 04 00 00 00 1e 00 00 .Ducky..
000001c0 ff e1 04 4c 68 74 74 70 3a 2f 2f 6e 73 2e 61 64 ...Lhttp://ns.ad
000001d0 6f 62 65 2e 63 6f 6d 2f 78 61 70 2f 31 2e 30 2f obe.com/ xap/1.0/
000001e0 00 3c 3f 78 70 61 63 6b 65 74 20 62 65 67 69 6e .<?xpack et begin
000001f0 3d 22 ef bb bf 22 20 69 64 3d 22 57 35 4d 30 4d ="..." i d="W5M0M
00000200 70 43 65 68 69 48 7a 72 65 53 7a 4e 54 63 7a 6b pCehiHzeSzNTczk
00000210 63 39 64 22 3f 3e 20 3c 78 3a 78 6d 70 6d 65 74 c9d"?> < x:xmpmet
00000220 61 20 78 6d 6c 6e 73 3a 78 3d 22 61 64 6f 62 65 a xmlns: x="adobe
00000230 3a 6e 73 3a 6d 65 74 61 2f 22 20 78 3a 78 6d 70 :ns:meta /" x:xmp
00000240 74 6b 3d 22 41 64 6f 62 65 20 58 4d 50 20 43 6f tk="Adob e XMP Co
00000250 72 65 20 35 2e 33 2d 63 30 31 31 20 36 36 2e 31 re 5.3-c 011 66.1
```

- In this case, payload is in JPEG format
- “Content-type” header:
 - need to specify the format, eg JPEG. Note this is necessary because what requested by user (“Accept”) might be a list, and also server might return something different
- “Content-length” header:
 - Essential, otherwise HTTP parser cannot know when payload is finished
- Cache handling: headers like “Cache-Control”, “ETag”, “Last-Modified”, etc.
 - If visiting page for second time, no need to re-download image if hasn’t changed

HTTP Response

- Same kind of headers and body as HTTP request
- Only first line does differ
- <PROTOCOL> <STATUS> <DESCRIPTION>
 - Eg, “HTTP/1.1 200 OK”
 - Note: only 1 space “ ” between the tags, I added extras just for readability
- When making a request, a lot of things could happen on server, and the “status” is used to say what happened

HTTP Status Codes

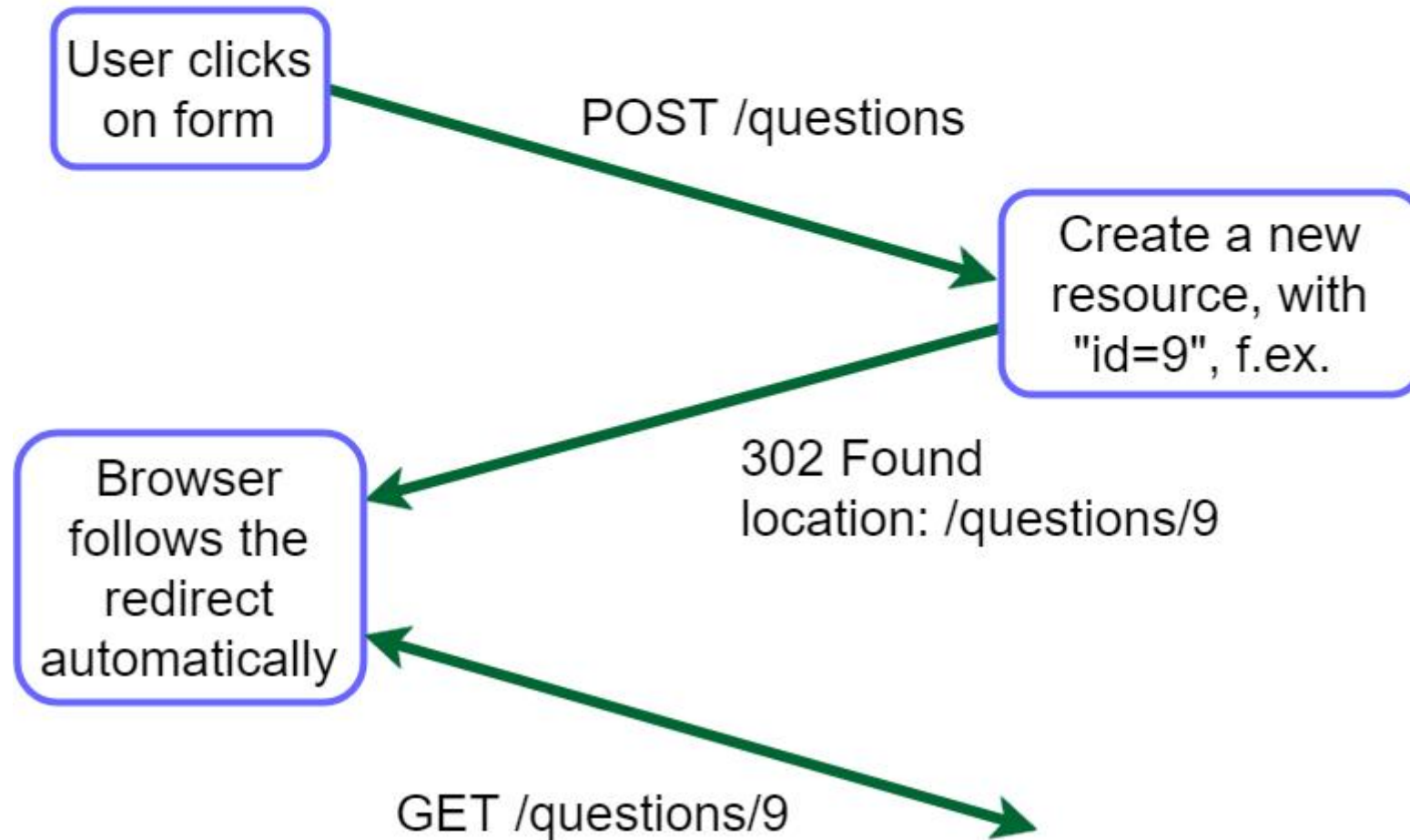
- 3 digit number, divided into “families”
- 1xx: informational, interim response
- 2xx: success
- 3xx: redirection
- 4xx: user error
- 5xx: server error

2xx Success

- **200:** OK
- **201:** resource created
- **202:** accepted, but not completed (eg, background operation)
- **204:** no content (eg, as result of PUT or DELETE)

3xx Redirection

- Note: the semantics of these codes is a “**mess**”, being changing with different “updates” of HTTP 1.1... and being left in an inconsistent state
- **301** permanent redirection
 - If X redirects to Y, then client will never ask for X again, and go straight for Y
- **302** temporary redirection
 - “May” change verb, eg from POST to GET
- **307** temporary redirection, but same verb
 - Eg, a POST stays a POST
- “*Location*” header: URI of where we should redirect



4xx User Error

- **400**: bad request (generic error code)
- **401**: unauthorized (user not authenticated)
- **403**: forbidden (authenticated but lacking authorization, or not accessible regardless of auth)
 - Note: RFC 7231/7235 are rather ambiguous/confusing when it comes to define authentication/authorization, and differences between 401 and 403
- **404**: not found (likely the most famous HTTP status code)
- **405**: method not allowed (eg doing DELETE on a read-only resource)
- **415**: unsupported media type (eg sending XML to JSON-only server)

Even if error (eg 404), response can have a body, eg an HTML page to display

The image shows a web browser displaying a 404 error page from GitHub. The page features a large '404' in the top left, a cartoon cat character on the right, and a speech bubble in the center that reads: 'This is not the web page you are looking for.' The browser's address bar shows the URL: `https://github.com/somestringthatshouldnotrepresentanyresourceongithub`.

On the right side of the browser window, the 'Network' tab is open, showing a list of resources. The first resource, `somestringthatshouldnot...`, is selected. The 'Response' tab for this resource is active, displaying the following HTML content:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta http-equiv="Content-type" content="text/html; charset=utf-8">
5     <meta http-equiv="Content-Security-Policy" content="default-src 'self';>
6     <meta content="origin" name="referrer">
7     <title>Page not found &middot; GitHub</title>
8     <style type="text/css" media="screen">
9       body {
10         background-color: #f1f1f1;
11         margin: 0;
12       }
13     </style>
14   </head>
15   <body>
```

5xx Server Error

- **500: Internal Server Error**

- Often, a bug, eg an exception in the business logic (like a NPE), that propagates to the application server will be handled with a 500 response
 - Note: if whole application server does crash, then you get no response...
- Required external services have problems, eg database connection failed

- **503: Service Unavailable**

- Eg, overloaded of requests, or scheduled downtime for maintenance

HTTPS (HTTP Secure)

- Encrypted version of HTTP, using Transport Layer Security (TLS)
- Usually on port 443 instead of 80
- URIs are the same as in HTTP (just the “scheme” does change)
- Note, the whole HTTP messages are encrypted, **but still using TCP**
 - this means it is still possible to find out the IP address and port of remote server, although cannot decipher the actual sent messages
 - this issue can be avoided by going through proxy networks like TOR , or any VPN provider (but this latter would know what you visit)
- More on this later in the course

Web Applications

HTTP Server Application

- A program that *opens* a TCP port (eg, 80 or 443) and *listens* on incoming requests
- For each request, will send a HTTP response with the given requested resource
- The “resource” might be an *existing* file on disk (e.g., an HTML page or JPEG image), or *created* on the fly (eg based on content in database)
- In the “old days”, a “web application” was just a set of static files accessible over HTTP

Old Days Server



`/where/installed/`

`/index.html`

`/figs/cat.jpeg`

`/figs/dog.jpeg`

Application server running on port 80, providing files from the folder “/where/installed”

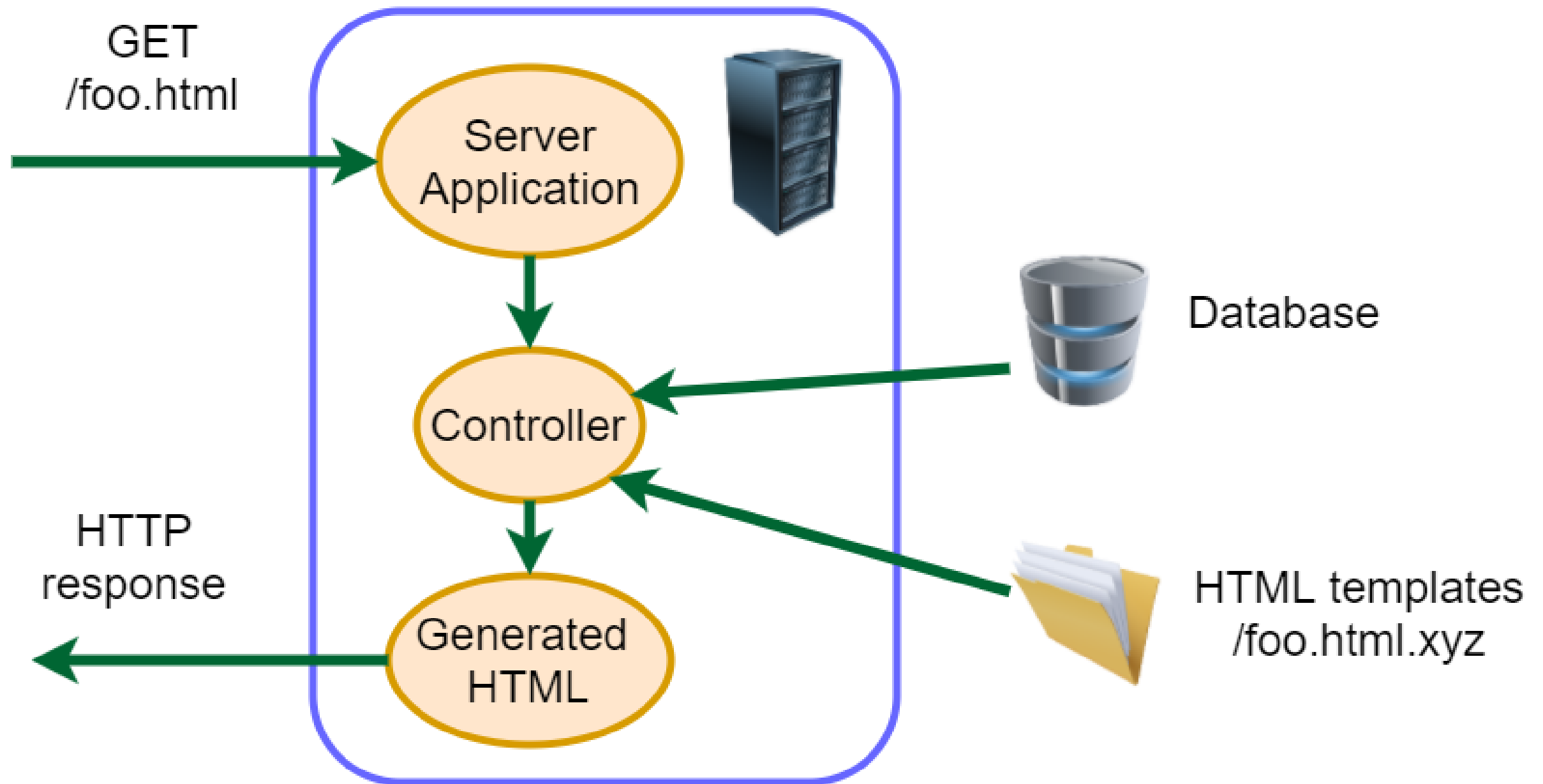
If browser asks for “www.foo.org/index.html”, the server will check if a file called “index.html” is under the folder “/where/installed”, and return it as Body of the HTTP response

Cont.

- When asked for “www.foo.org/index.html”, the browser will:
 - Resolve the IP address of www.foo.org with DNS
 - Establish a TCP connection toward <IP address>:**80**
 - Do a HTTP request with command: “GET /index.html HTTP/1.1”
 - Here the “index.html” is the requested resource
 - If asking for “www.foo.org/figs/cat.jpeg”, the request command will be “GET /figs/cat.jpeg HTTP/1.1”
 - What after the <IP address>:<port> is the so called “path” that identifies the resource on the server
 - Note: the client browser has no clue of where the files are actually stored on the server, ie the “/where/installed” folder

Dynamic Pages

- Static, pre-defined HTML pages are not enough for modern applications
- You might want to base the HTML pages on data from database or dynamic content
 - Web forum
 - Shopping cart
 - Live chats
 - Etc.
- HTML pages will need to be created on the fly at each HTTP request
- Browser will still just see a HTML file: no clue if automatically generated



Server-Side HTML Rendering

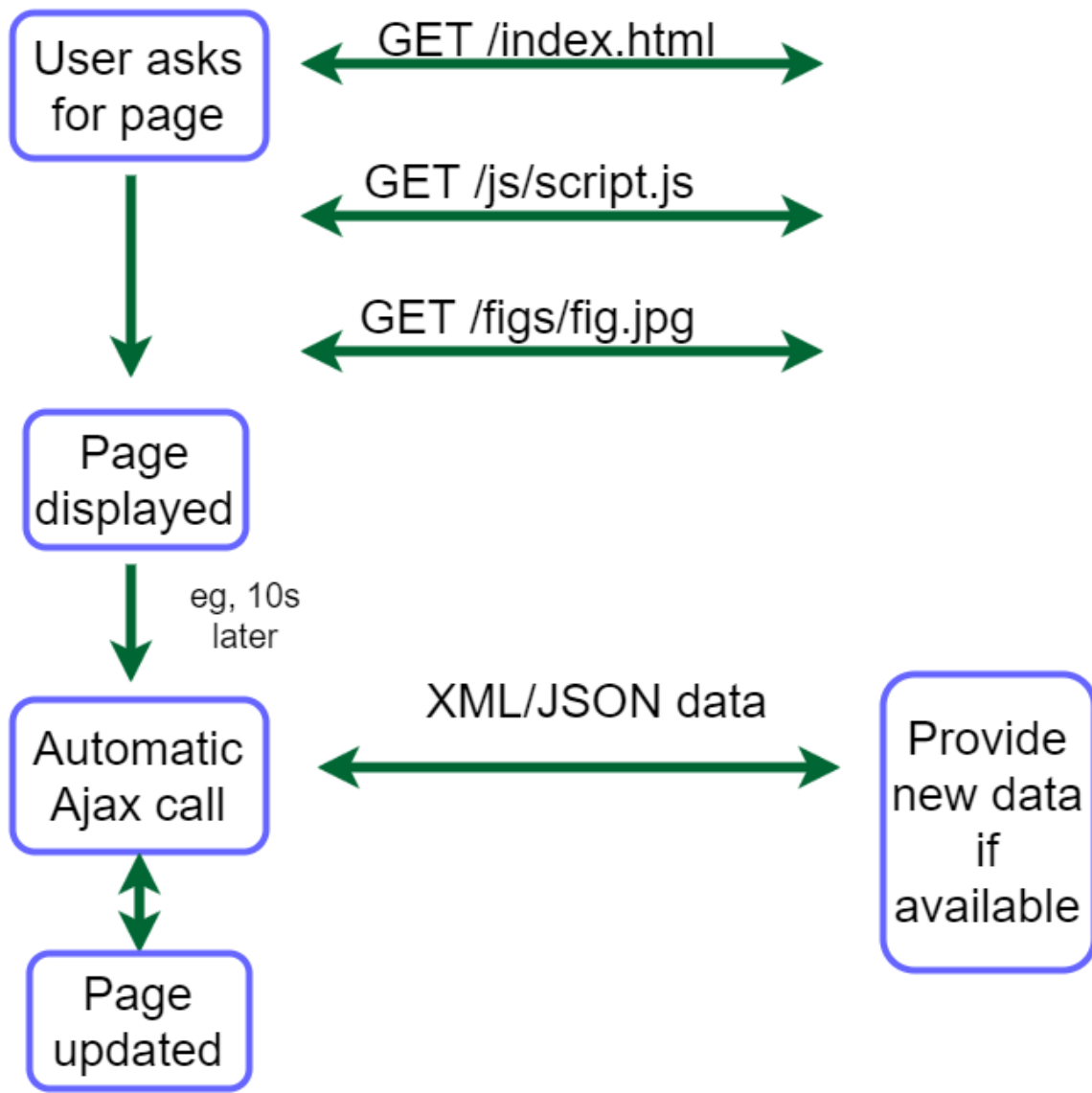
- The whole HTML page is created on the server in one go
- HTML templates
 - Files that mix together HTML data and instructions/code on how to create the dynamic parts
- Many different template technologies, even within the same language
 - eg, PHP, JSP/JSF for Java, Ruby on Rails, etc.

HTML Triggering Actions

- ****
 - Do a HTTP **GET** on the given resource specified by the “target” URI
 - *Note: <a> does have an attribute “type”, but many browsers just ignore it... yep, welcome to the beautiful world of internet standards!!!*
- **<form><input /></form>**
 - Will do a **GET** or a **POST** (based on *method* attribute), with data from the input fields
- How can I do something like “DELETE /questions/5” ???
- In HTML, you simply CANNOT
- You need to use JavaScript...

AJAX (Asynchronous JavaScript and XML)

- Executing JS on the client browser opens the door to many possibilities
- JS can retrieve new data from the server in the background, and update the current webpage without the need for the user to reload it completely
 - Chats
 - News (eg, refreshed every hour)
 - Etc.
- JS can run in background, and do AJAX calls every X seconds to server to retrieve data in XML/JSON format, and update DOM accordingly



WebSocket

- Limitation of AJAX is that client has to make the request for data
- Asking at time intervals (eg every 10s) is inefficient:
 - Many requests, often even if new data is not available
 - If new data is available, cannot get it immediately, and have to wait till next scheduled AJAX call
- WebSocket is a protocol working on TCP to allow full-duplex communication between client and server
- Server can “push” new data even if client did not ask for it
- A relatively recent protocol... first version in Chrome in 2009

Front-End Development

- Front-end development is becoming more complex
 - Can be 10s or 100s of thousands of lines of JS
- Making good GUIs requires special skills, eg in interaction design
- In large organizations, not strange to have separated teams for front-end and back-end development
- In such cases, using “template” technologies might not be the best option
 - Front-end developers might be HTML/CSS/JS specialists that might not know about the specific language(s) chosen for the backend
 - When your pages depend on running the backend, it is more difficult to prototype the GUI

Possible Solution

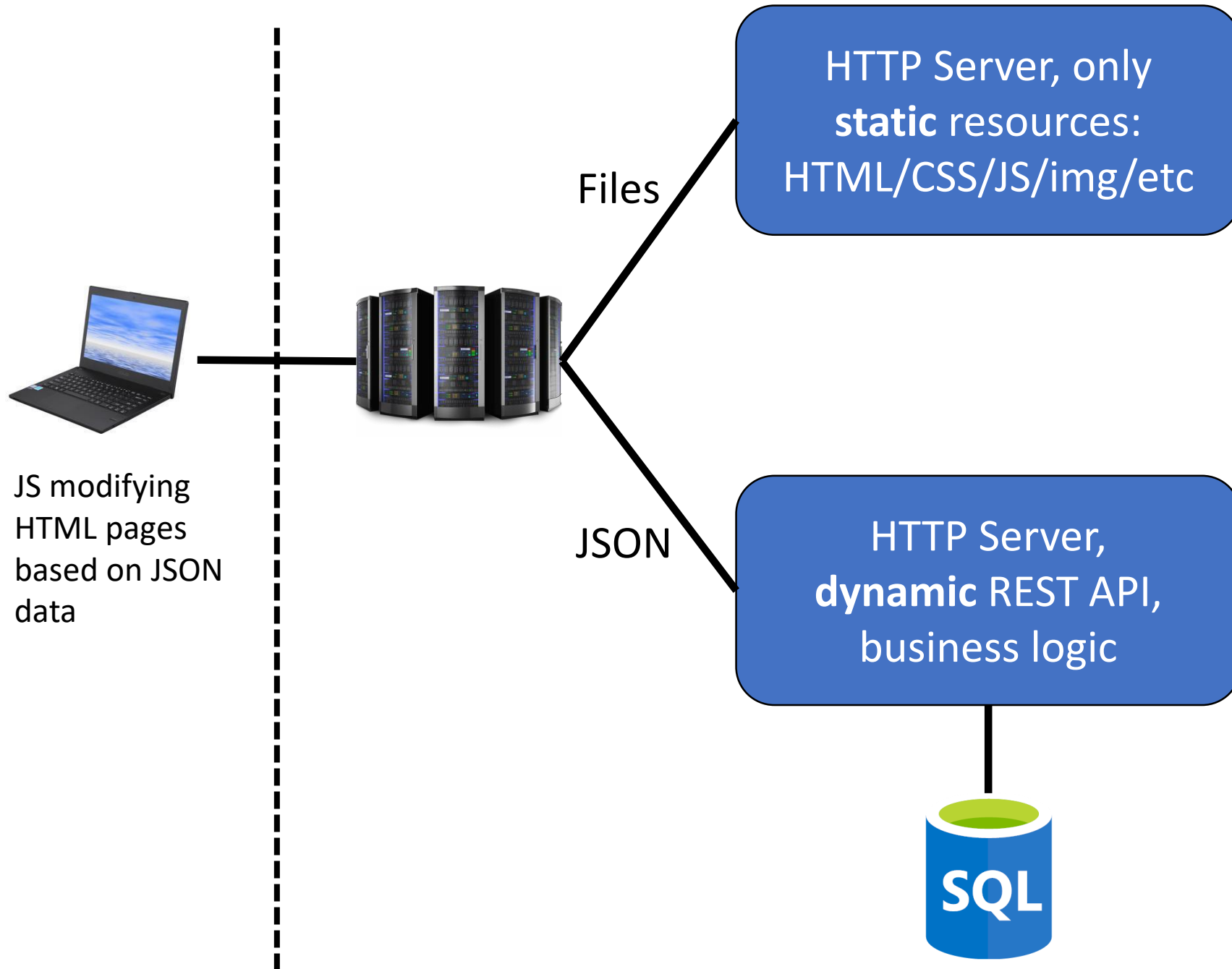
- Have a complete separation between frontend and backend
- Frontend: only “static” files like HTML/CSS/JavaScript
 - No template language
- Backend: provide just data in JSON format, and the client JS will update the DOM from the HTML static files
 - I.e., client-side rendering
 - Data can be read via AJAX when HTML page is loaded in browser
- How to provide JSON data? **RESTful** Web Services

REST (Representational State Transfer)

- Not a protocol, but rather an architectural style
- Used to define how web resources should be structured and accessed
- Based on the HTTP protocol
- Server will answer to different URL requests, where the path represents the data to return in an hierarchical format, eg
 - /menus , return all menus
 - /menus/today , return the menu of today
 - /menus/today/dishes, return all the dishes in today's menu
- Data can be returned in different formats, but in most cases it is in JSON (JavaScript Object Notation), as client browser JS can directly use it without the need to unmarshal it

JavaScript + REST

- More and more companies are moving to this approach to develop web applications instead of server-side templates
- Different RESTful services can be written in different languages, serving the same page
 - Java, Scala, C#, etc.
- RESTful can also serve other GUIs, eg mobile apps
- Frontend can be tested/prototyped without a running backend server (can just stub out the JSON responses with static files)
- *No silver bullet*: client-side rendering puts more strain on the client, which can become an issue on mobile web browsers



- Server will have *static* resources and *dynamic* ones
- Provided by same server, or different ones behind a *gateway*

Web Security

Overview

- In this class, we will not go into the details of web security
- Need to understand how web works before you can “hack” it
- Brief overview of topics will see in the next following classes

Cross-Site Request Forgery (CSRF)

- We will see how authentication is done in web applications
- Once user is authenticated on site A, a CSRF attack is done when visiting a malicious site B
 - malicious B has code to do operations (ie HTTP calls) on A
 - eg transfer money to attacker if A is bank application, and when user visits B while logged in on A
 - can trick user to visit B via social engineering

Cross-Site Scripting (XSS)

- **NEVER TRUST USER INPUTS!!!**
- Problem when HTML is based on user inputs
 - eg, display message in a chat/forum
 - eg, show back user's "query" when doing a search query on a site
- If data is not *sanitized*, user can inject messages containing HTML tags and JS code that *will be executed* when page is updated
 - e.g., replace whole web page with a Jolly Roger (or any other image)...
- *Sanitization*: replace/delete characters that can alter the context in which the input is used
 - eg "<" and ">" in HTML/XML documents

SQL Injection

- As for XSS, **NEVER TRUST USER INPUTS!!!**
- If SQL queries on database are based on user inputs, if not sanitized, can trick backend to execute arbitrary SQL queries
- Can leak info from database!!!
 - major issues for passwords and credit card numbers
 - we will see how to protect passwords even in the case of *successful* SQL Injection attacks
- In some (embarrassing) cases, can login without passwords...

JSON and AJAX

JavaScript Object Notation (JSON)

- Lightweight data-interchange format
- Currently JSON most used format in web communications
 - in the past XML was more common, especially with SOAP web services
- Can be directly used by JavaScript (eg when running in browser) as an object

Two Main Structures

- Objects {}, as collection of unordered pairs name-value
- Eg: **{“username”: “foo”, “age”:18, “hobbies”:{}} }**
 - 3 fields: *username*, *age* and *hobbies*
 - Name of fields are in “” quotes
 - Types can be primitives (eg **18**), strings (eg **“foo”**), objects (eg empty {}), etc.
 - Operator “:” for assignment
- Ordered arrays [], where each element separated by “,”
- Eg: **[12, {“foo”:4}, 15]**
 - note: type of elements in array can be different, eg numbers and objects



Introducing JSON

العربية Български 中文 Český Dansk Nederlands English
Esperanto Français Deutsch Ελληνικά עברית Magyar Indonesia
Italiano 日本 한국어 فارسی Polski Português Română Русский
Српско-хрватски Slovenščina Español Svenska Türkçe Tiếng Việt

ECMA-404 The JSON Data Interchange Standard.

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It

object
`{}`
`{ members }`

Elements Console Sources Network Performance

top Filter De

```
> var user = {"username": "foo", "age":18, "hobbies":{}}
```

```
< undefined
```

```
> user.username
```

```
< "foo"
```

```
> user.age
```

```
< 18
```

```
> user.hobbies
```

```
< ► {}
```

```
> var data = [12, {"foo":4}, 15]
```

```
< undefined
```

```
> data[0]
```

```
< 12
```

```
> data[1]
```

```
< ► {foo: 4}
```

```
> data[1].foo
```

```
< 4
```

```
> data[2]
```

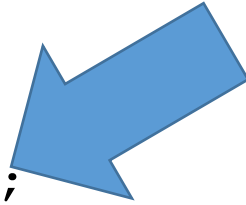
```
< 15
```

AJAX (Asynchronous JavaScript and XML)

```
someNameForAFunction = function() {  
    var xmlhttp = new XMLHttpRequest();  
  
    xmlhttp.onreadystatechange = function() {  
        if (xmlhttp.readyState === XMLHttpRequest.DONE) {  
            if (xmlhttp.status === 200) {  
                // do something if call was successful  
            } else {  
                alert('something went wrong');  
            }  
        }  
    };  
  
    xmlhttp.open("GET", "http://localhost/someURL", true);  
    xmlhttp.send();  
};
```


The name **XMLHttpRequest** is only for historical reasons, from a time in which XML was the common exchange format instead of JSON

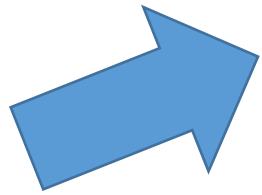
```
someNameForAFunction = function() {  
    var xmlhttp = new XMLHttpRequest();
```



```
    xmlhttp.onreadystatechange = function() {  
        if (xmlhttp.readyState === XMLHttpRequest.DONE) {  
            if (xmlhttp.status === 200) {  
                // do something if call was successful  
            } else {  
                alert('something went wrong');  
            }  
        }  
    };  
};
```

```
xmlhttp.open("GET", "http://localhost/someURL", true);  
xmlhttp.send();  
};
```

- When a HTTP call is done with AJAX, it might take some time (few ms)
- Here, we register a “*callback*”, which is a function which is executed every time there is a state change in the call
- In particular, we want to wait for when the request is fully DONE
- Once done, we code different behavior based on the HTTP status
- Recall: 2xx (OK), 4xx (user error), 5xx (server error)



```
xmlhttp.onreadystatechange = function() {  
    if (xmlhttp.readyState === XMLHttpRequest.DONE) {  
        if (xmlhttp.status === 200) {  
            // do something if call was successful  
        } else {  
            alert('something went wrong');  
        }  
    }  
};
```

The HTTP method to execute

Whether the call should be asynchronous (you want it *true* most of the time)

The URL we send the message to



```
xmlhttp.open("GET", "http://localhost/someURL", true);
```

```
xmlhttp.send();
```

The actual sending of the HTTP request


```
var payload = ...  
  
xmlhttp.open("POST", "http://localhost/someURL", true);  
  
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");  
  
xmlhttp.send(payload);
```



If sending a payload as part of a POST/PUT request, then must also specify the format in which such payload should be sent, eg, JSON, XML, JPEG, PNG, Plain Text and URL Encoded.

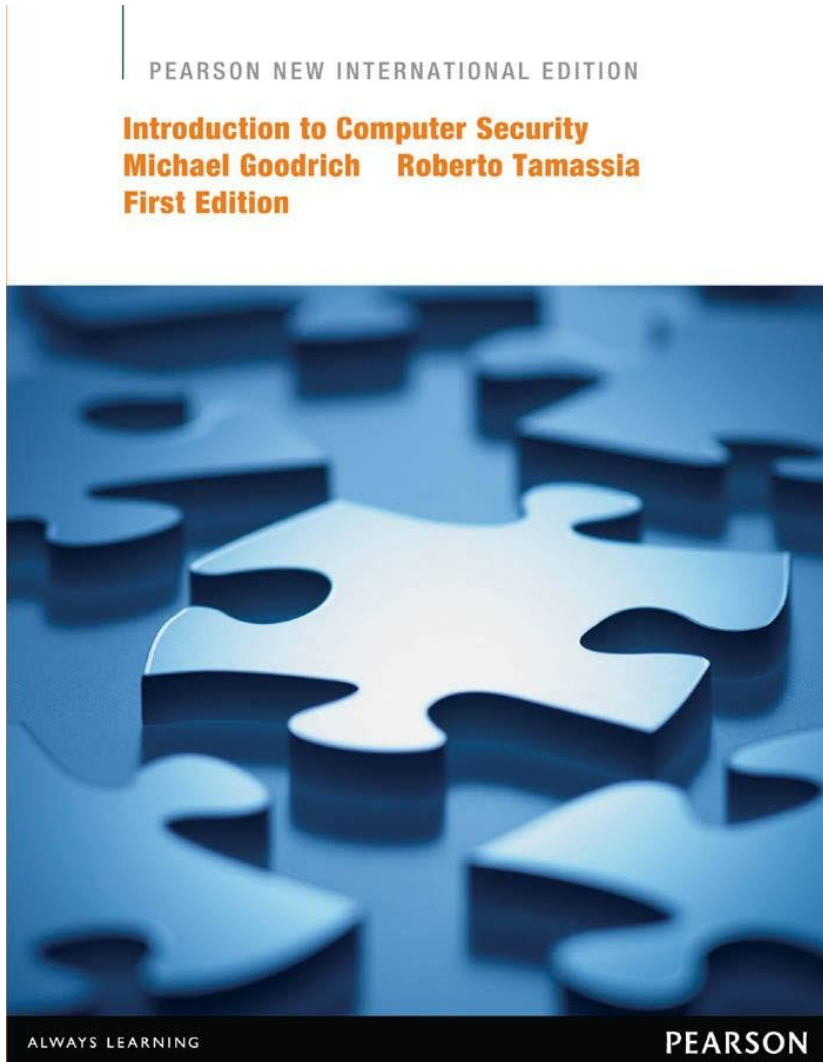
[illegible]

Can pause,
and execute
1 statement
at a time



Inspect
values of
variables

For Next Week



- Book pages: 328-333, 339-341
- Note: when I tell you to **study** some specific pages in the book, it would be good if you also *read* the other pages in the same chapter at least once
- Exercises for Lesson 8 on GitHub repository