

Enterprise Programming 1

Lesson 07: JSF and Docker

Dr. Andrea Arcuri

Web Applications

[Gmail](#)

[Images](#)



[Sign in](#)



Google Search

I'm Feeling Lucky

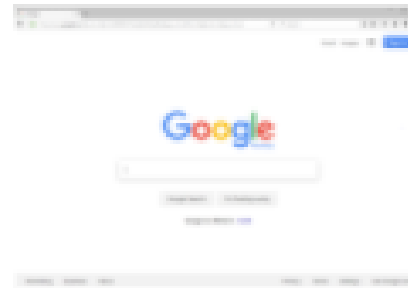
Google.no offered in: [norsk](#)

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

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!

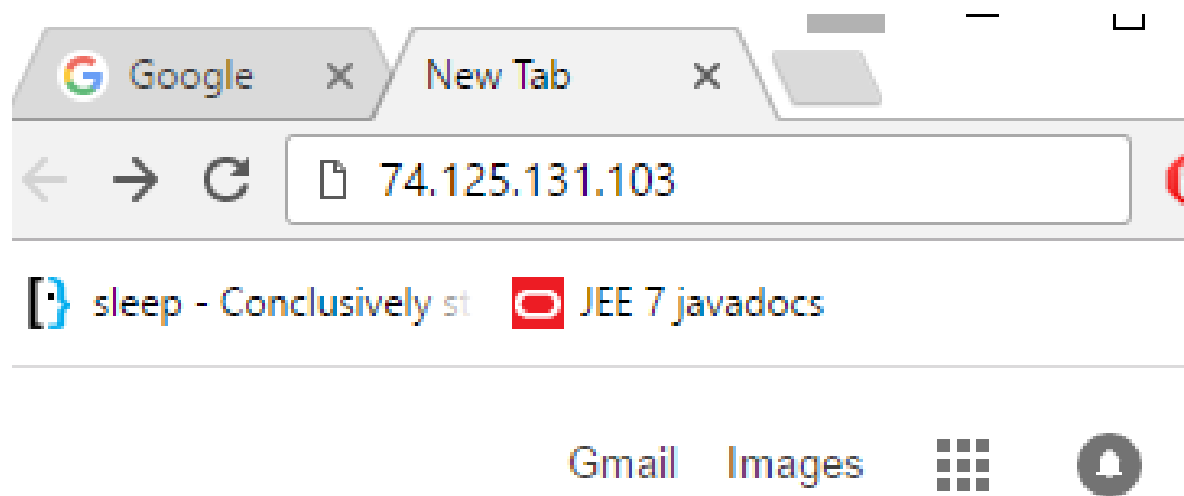


Browsers!



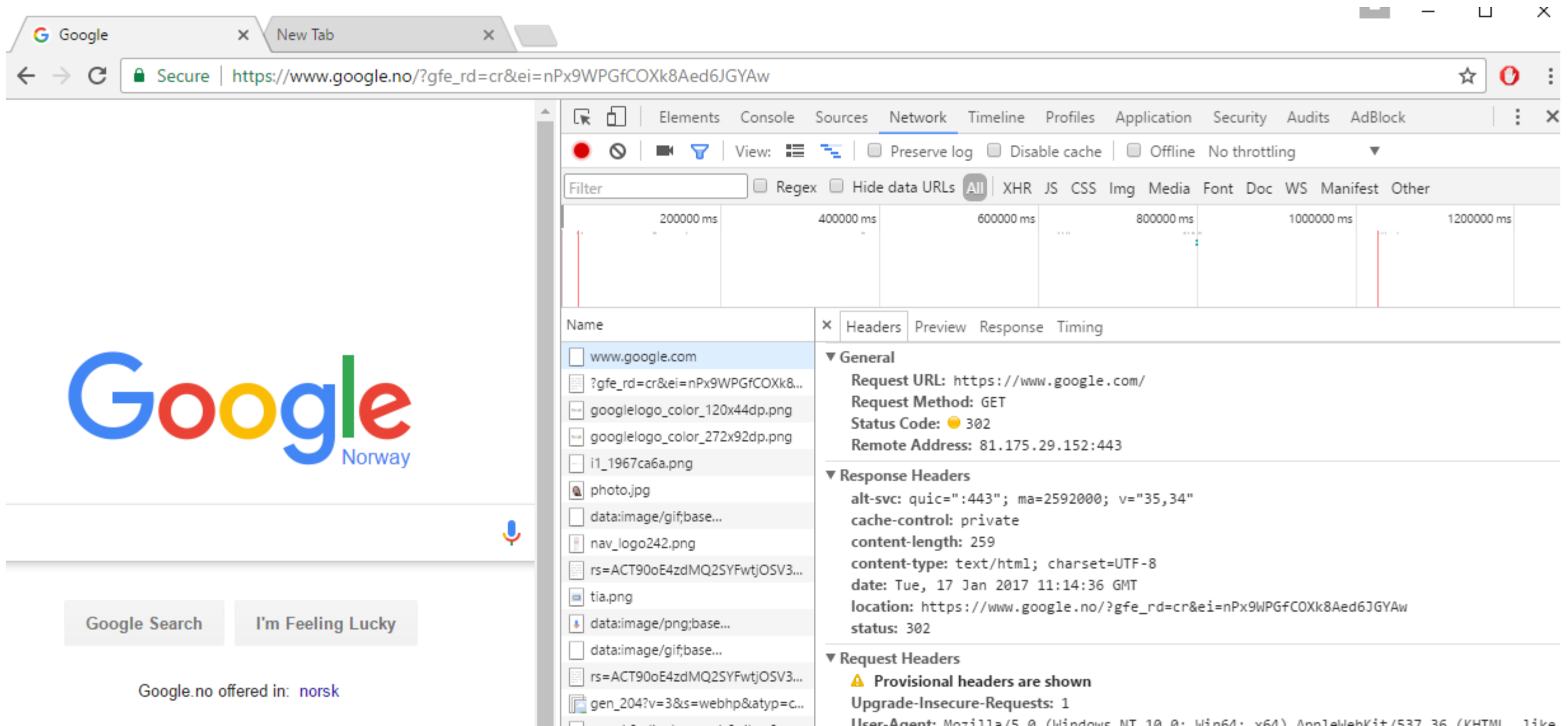
Domain Name System (DNS)

- When you type “www.google.com” in your browser, how does it now which machine to “connect” to?
- Internet connections are based on IP addresses
 - A 32 bit number, usually represented with 0-255 octets
 - 74.125.131.103 for www.google.com
 - IPv4 is most currently used. IPv6 is sloooowly replacing it
- To make the translation from “www.google.com” to 74.125.131.103, your machine needs to contact a DNS server
- Discovery handled by the operating system (OS): either hardcoded known host (DNS roots) or broadcast on local network to get reply from your ISP (Internet Service Provider)



- If you know the IP address, you can type it directly
- Same “name” can be mapped to different IP addresses, ie different servers
- The mapping can change

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



The screenshot shows a Google Chrome browser window with the Developer Tools panel open. The browser is displaying the Google homepage for Norway. The Developer Tools panel is set to the Network tab, showing a list of requests. The first request, 'www.google.com', is selected, and its details are shown in the right-hand pane. The details pane shows the 'General' tab, which includes the Request URL, Request Method, Status Code, and Remote Address. The 'Response Headers' section is also visible, showing various headers like 'alt-svc', 'cache-control', 'content-length', 'content-type', 'date', 'location', and 'status'.

Google Chrome browser window showing the Google homepage (Google.no) and the Developer Tools panel open.

The address bar shows the URL: https://www.google.no/?gfe_rd=cr&ei=nPx9WPGfCOXk8Aed6JGYAw.

The Developer Tools panel is open, displaying the Network tab. The selected request is `www.google.com`.

The request details are shown in the right pane:

- 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`

← → ↻ Gandi [FR] | https://www.gandi.net/domain

gandi.net no bullshit™

Service status Shopping cart Webmail kr English

Domain names Hosting SSL Corporate Why Gandi? Discussion ? Help Log in

Home Register New TLDs Transfer Renew Restore Whois Reseller


.CN for just US\$ 7.60 (6,33€) per year
From January 1 through February 28, get a .CN for 25% off

Afilias The first half of 2017 is half-off on Afilias domains
A half-year of half-price domains from Afilias

50% Fifty percent off .osaka
Get a .osaka domain for half price this winter.

Domain name registration

[Transfer](#) [Renew](#) [Bulk registration](#) [See all domain prices](#)





- ☒ Generic (472)
- ☐ Americas (25)
- ☒ Europe (50)
- ☐ Asia/Oceania (31)
- ☐ Africa (9)
- ☐ All (587)

Promos!

































- ☐ The first half of 2017 is half-off o...
- ☐ Fifty percent off .osaka
- ☐ Like .me in 2017
- ☐ Half-price .eu this year
- ☐ Radix Registry promo for the ne...
- ☐ Big fat .promo, 50% off

Discover the new gTLDs .app, .blog, and .web...
There are already more than **One Million** pre-registrations!
[Pre-register yours for free!](#)

Every domain name includes:

-  Full domain management
-  5 mailboxes and 1000 forwarding addresses

Popular TLDs at Gandi

| | |
|---|---------------------------------|
|  .at | 166,50 NOK |
|  .be | 111,00 NOK |
|  .biz | 135,05 NOK |
|  .ca | 129,50 NOK |
|  .cam | 332,44 NOK |
|  .ch | 106,38 NOK |
|  .cloud | 240,13 NOK |
|  .club | 95,55 NOK |
|  .cn | 58,55 NOK 78,07 NOK |
|  .co | 264,55 NOK |
|  .co.uk | 74,00 NOK |
|  .com | 115,99 NOK |
|  .cz | 148,00 NOK |
|  .de | 111,00 NOK |
|  .es | 111,00 NOK |
|  .eu | 55,50 NOK 144,00 NOK |
|  .fi | 138,75 NOK |
|  .fr | 111,00 NOK |
|  .gg | 527,25 NOK |
|  .info | 64,75 NOK 129,50 NOK |
|  .io | 268,25 NOK |
|  .it | 111,00 NOK |
|  .lu | 208,13 NOK |
|  .me | 74,00 NOK 148,00 NOK |
|  .mobi | 60,13 NOK 120,25 NOK |
|  .net | 138,75 NOK |
|  .ninja | 139,49 NOK |
|  .nl | 111,00 NOK |
|  .no | 138,75 NOK |
|  .no.com | 244,20 NOK |
|  .nz | 148,00 NOK |
|  .online | 37,00 NOK 398,24 NOK |

Registering a domain mapped to an IP address of your choice is not particularly expensive: eg 139 NOK per year for a “.no” address

For example, I used it to register “*arcuriandrea.org*”

Ports

- An IP address is not enough to establish a TCP connection to a remote server
- Need to also know the “port”, which is a number in 0-65535 ($2^{16}-1$)
- A *server application* running on a remote *server machine* will need to specify on which “port” it is listening to
- On same machine, you can have several different applications binding to different ports
- The range 0-1023 is for *reserved* ports, for very specific, well known types of applications

Ports to know

- 0: dynamically allocated (see next slide)
- 22: for SSH connections
 - Very common when you need to connect to a remote server using a terminal
- 80: for a HTTP connection
 - When browsing the web without encryption
- 443: for HTTPS, ie secure/encrypted HTTP over TLS/SSL
 - More and more common nowadays, even when no user authentication
- 8080: unreserved port.
 - “Typically” used by tools when running a HTTP server locally on your machine

Ephemeral/Dynamic ports

- “Typically” in range 49152–65535
 - But will vary based on the OS
- For short-lived communications
 - When you establish a connection to a remote server, a port will be open locally to read the responses
- During development, when using port 0
 - Get a dynamic port which is free, not used
 - Eg, if you want to start a server locally for debugging/testing, you might want to avoid conflicts with other applications using the same port
 - Essential when you run tests in parallel, and need several server instances

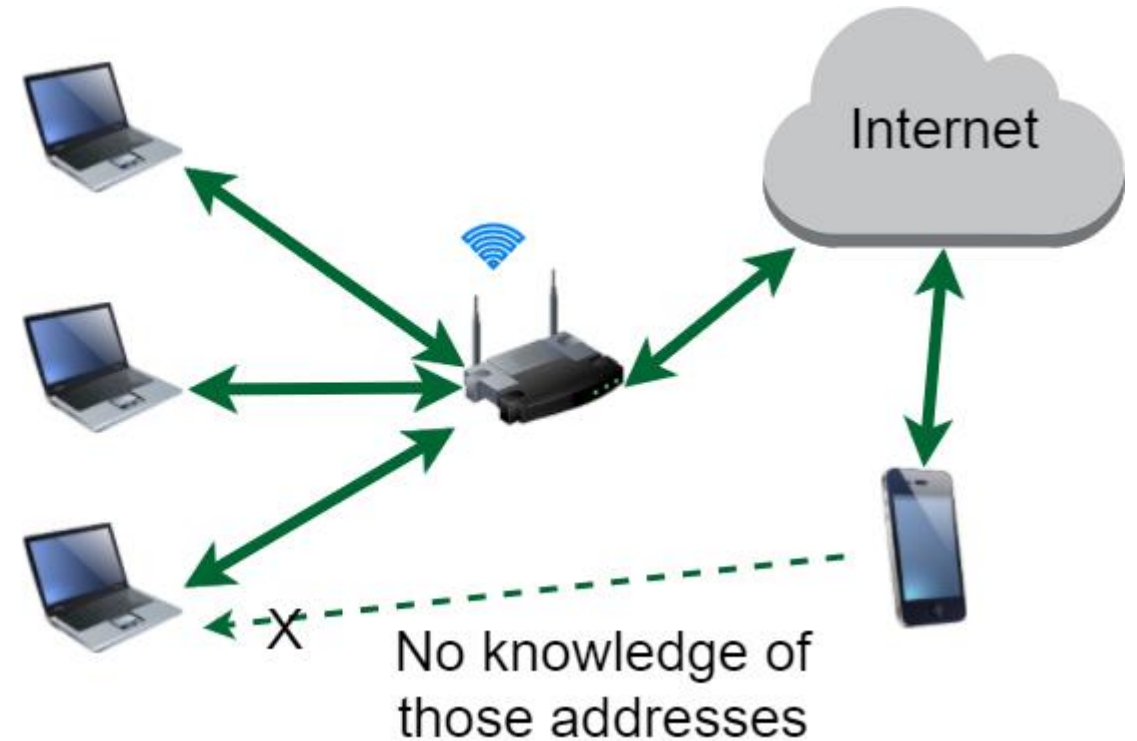
Defaults

- 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*

Local Networks

- Router has IP accessible from internet
- Machines connected to it have local IP not visible from outside
- Cannot use mobile to connect to such machines, unless special settings on router, or WiFi directly to same router
- Router might block machine-to-machine communications on same network for security reasons

10.x.x.x
172.[16-31].x.x
192.168.x.x



HTTP message

- When you ask your browser to display “*http://www.google.com:80*”, the browser will do a TCP connection to the remote server
- The browser will send a command on this TCP connection, which is a stream of bytes
 - Think about it like a `byte[]` array
- HTTP (Hypertext Transfer Protocol) is the *protocol* used to define the structure of the sent/received `byte[]` arrays

HTTP Protocol (Brief)

- Will go into low level details next course, Enterprise 2
- 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
- Most used version is 1.1, where data is sent like a readable string
 - HTTP 2.0 send data in binary format, but uses same commands/headers

*Ethernet 2

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http

| No. | Time | Source | Destination | Protocol | Length | Info |
|------|------------|----------------|----------------|----------|--------|--|
| 3466 | 251.884769 | 10.96.16.12 | 216.58.209.142 | TCP | 54 | 50154→80 [ACK] Seq=1 Ack=1 Win=65536 Len=0 |
| 3470 | 251.920257 | 10.96.16.12 | 216.58.209.142 | HTTP | 966 | GET / HTTP/1.1 |
| 3471 | 251.951558 | 216.58.209.142 | 10.96.16.12 | TCP | 60 | 80→50154 [ACK] Seq=1 Ack=913 Win=44800 Len=0 |

> Frame 3470: 966 bytes on wire (7728 bits), 966 bytes captured (7728 bits) on interface 0

> Ethernet II, Src: AsustekC_01:a3:02 (38:d5:47:01:a3:02), Dst: CiscoInc_9f:f0:d8 (00:00:0c:9f:f0:d8)

> Internet Protocol Version 4, Src: 10.96.16.12, Dst: 216.58.209.142

> Transmission Control Protocol, Src Port: 50154, Dst Port: 80, Seq: 1, Ack: 1, Len: 912

▼ 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) Chrome/55.0.2883.87 Safari/537.36\r\n

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

X-Client-Data: CKilyQEIHbJAQiltskBCMS2yQEIsIrKAQj6nMoBCKmdyGE=\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

▼ [truncated]Cookie: SID=7QNNk0soLHjmlPbDmKKMbje6E8VAgxtNb9rIG5wAVp2nKCNx2BXm0DfE4Q3nZBAFGSBisA.; HSID=ASgoRi3c02qtFU-5S; AI

Cookie pair: SID=7QNNk0soLHjmlPbDmKKMbje6E8VAgxtNb9rIG5wAVp2nKCNx2BXm0DfE4Q3nZBAFGSBisA.

Cookie pair: HSID=ASgoRi3c02qtFU-5S

Cookie pair: APISID=0t9_C_lg56MQobyj/ACMf0YuteULlLIIsbb

Cookie pair: CONSENT=YES+NO.no+20150906-13-0

Cookie pair [truncated]: NID=95=WlvqT2lJvLp8oM_LQ8FXPugSgQcp3RPUBqt_hBoU2YZFICb1Mdd-WhrMYkw52PVu5a0tbi0IN0EvE8bPI3QID_S_

\r\n

[Full request URI: <http://google.com/>]

[HTTP request 1/1]

[Response in frame: 3472]

Use
Wireshark
to see
exactly
what is
sent over
your
network
card

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

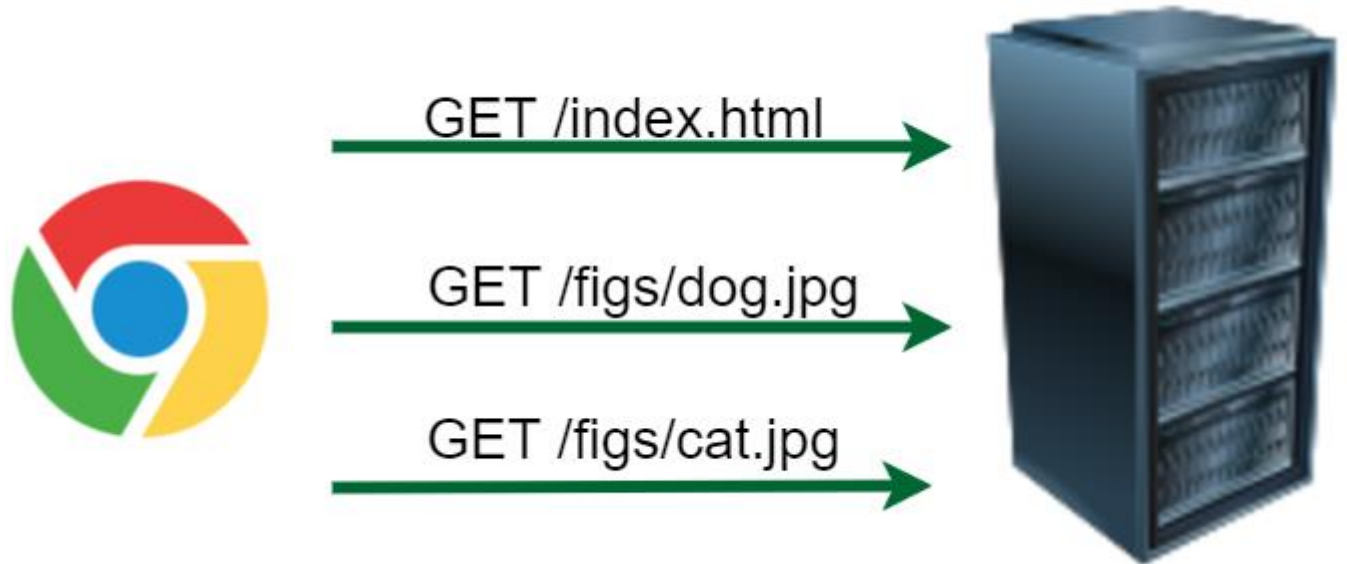
A Cat and a Dog



```
<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 two further HTTP GET requests
- Those further 2 requests could be done in parallel on two different TCP connections

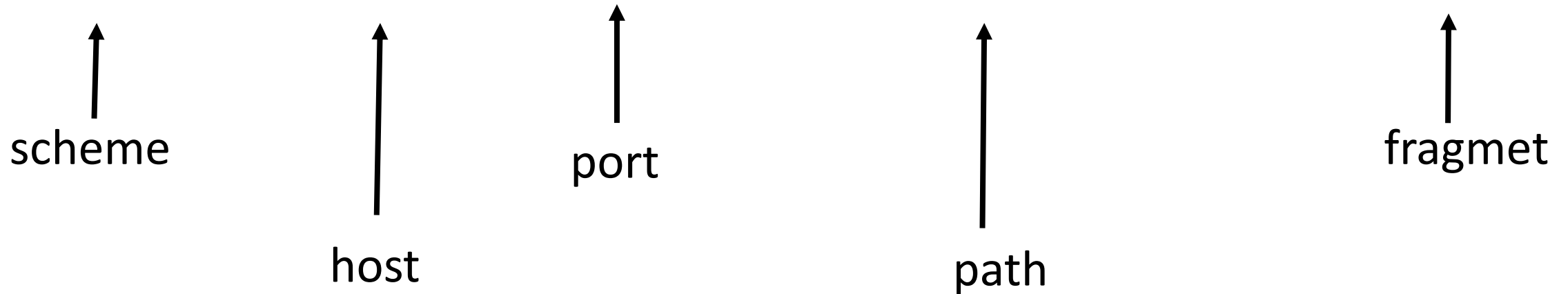


Cont.

- When asked for “*www.foo.org/index.html*”, the browser will:
 - Resolve the IP address of *www.foo.org*
 - 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

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.

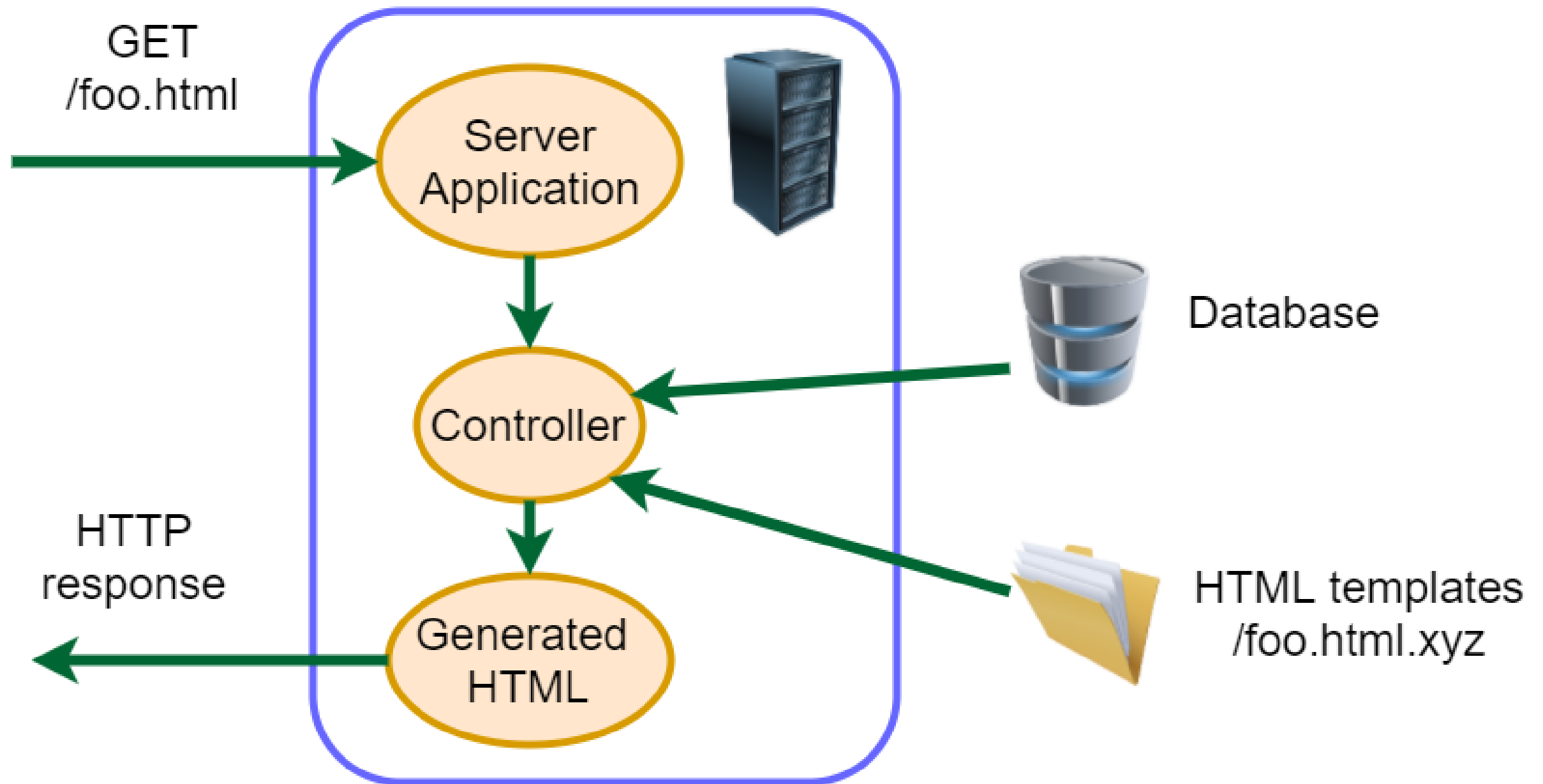
- *Scheme*: how to access the resource
 - http, https, file, ftp, etc.
- *Host*: the name of the server, or directly its numeric IP address
- *Port*: the listening port you will connect to on the remote server
- *Path*: identifies the resource, usually in a hierarchical format
 - Eg, /a/b/c
- *Query*: 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

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

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 chart
 - 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

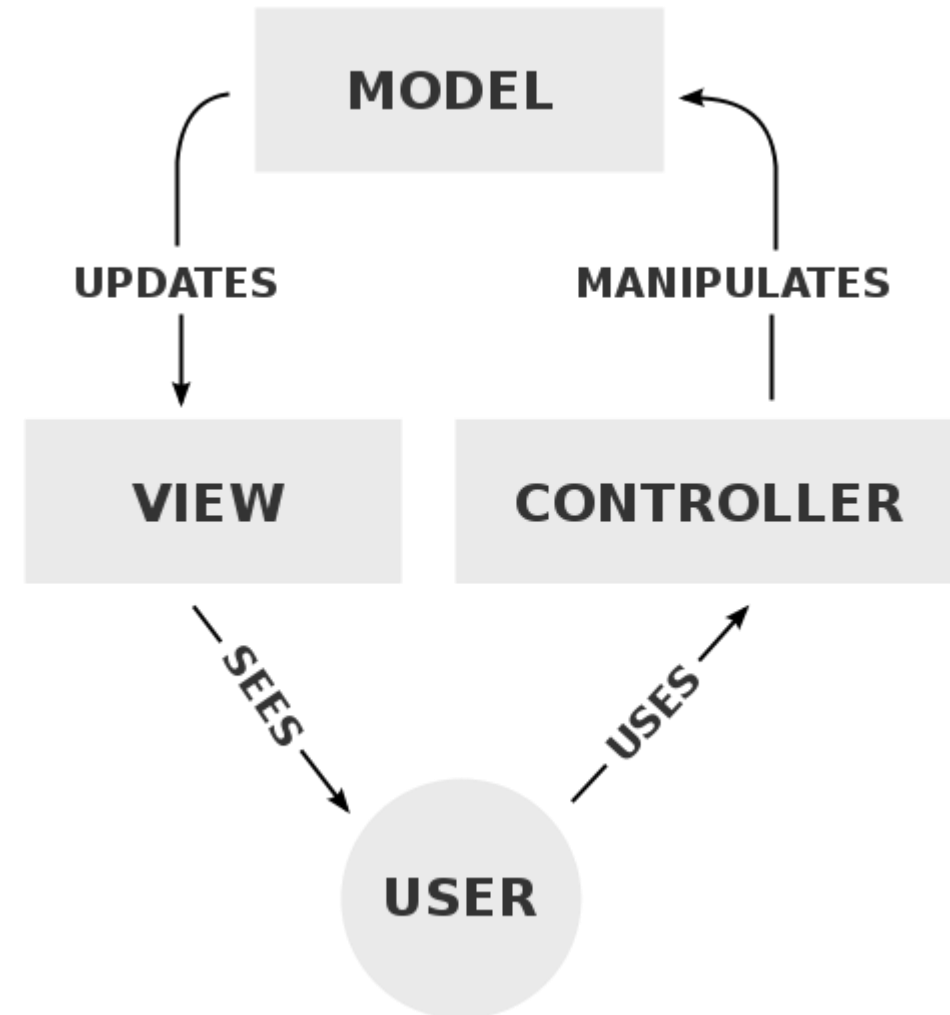


Server-Side HTML Rendering

- 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, JSF (JavaServer Faces) for Java (.xhtml)

MVC (Model–view–controller)

- A design pattern for developing GUI/Web applications
- Clear separation of concerns: easier to develop and maintain
- **Model**: internal state of the application
- **Controller**: running code with the business logic. Receive inputs from User, and modify the state, ie the Model
- **View**: what the user will receive, eg displayed HTML pages



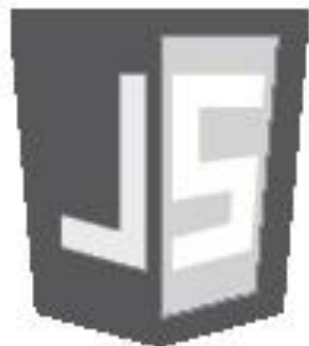
JavaScript

- JavaScript (JS) has nothing to do with Java
- *Programming* language executed in the browser
- JS code referenced by webpages like any other resource (eg images and CSS files), or can be embedded directly in HTML
- JS can manipulate the DOM (Document Object Model) to alter the webpages structure/content based on user's interactions (eg mouse clicks)

JAVA *is to*
JAVASCRIPT

as

HAM *is to*
HAMSTER

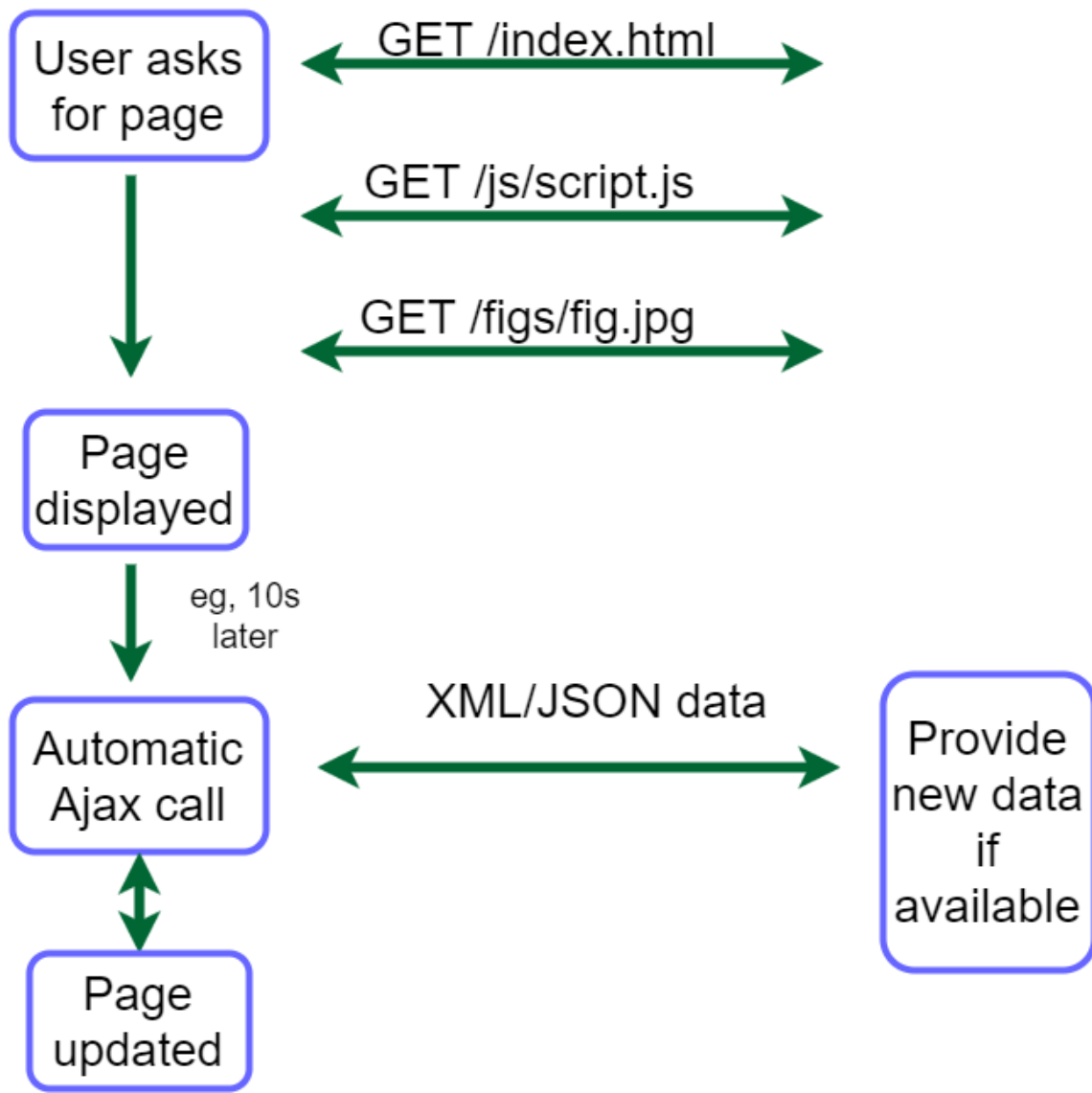


JavaScript is King on Browser

- If web page needs to execute code on browser, you use JS
- But historically there were other options in the (not so long ago) past:
 - Java with Java Applets (practically dead)
 - Flash (still found in some old web pages)
 - Silverlight
 - Etc.
- Those were not natively supported by browsers, and you had to install plugins to run them

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

Frontend 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 (eg JSF) 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 front-end 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
- 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

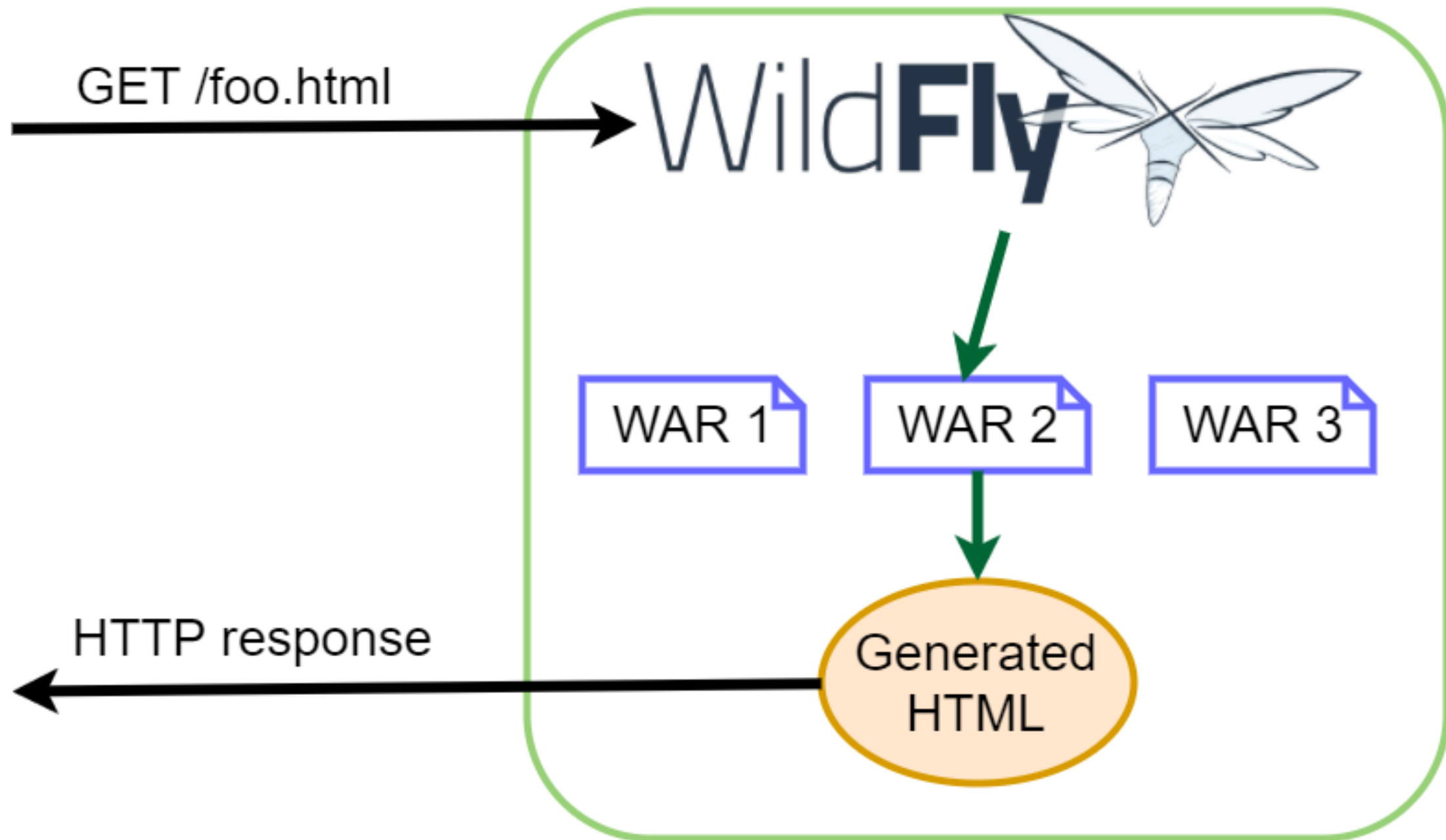
Enterprise 1 vs Other Courses

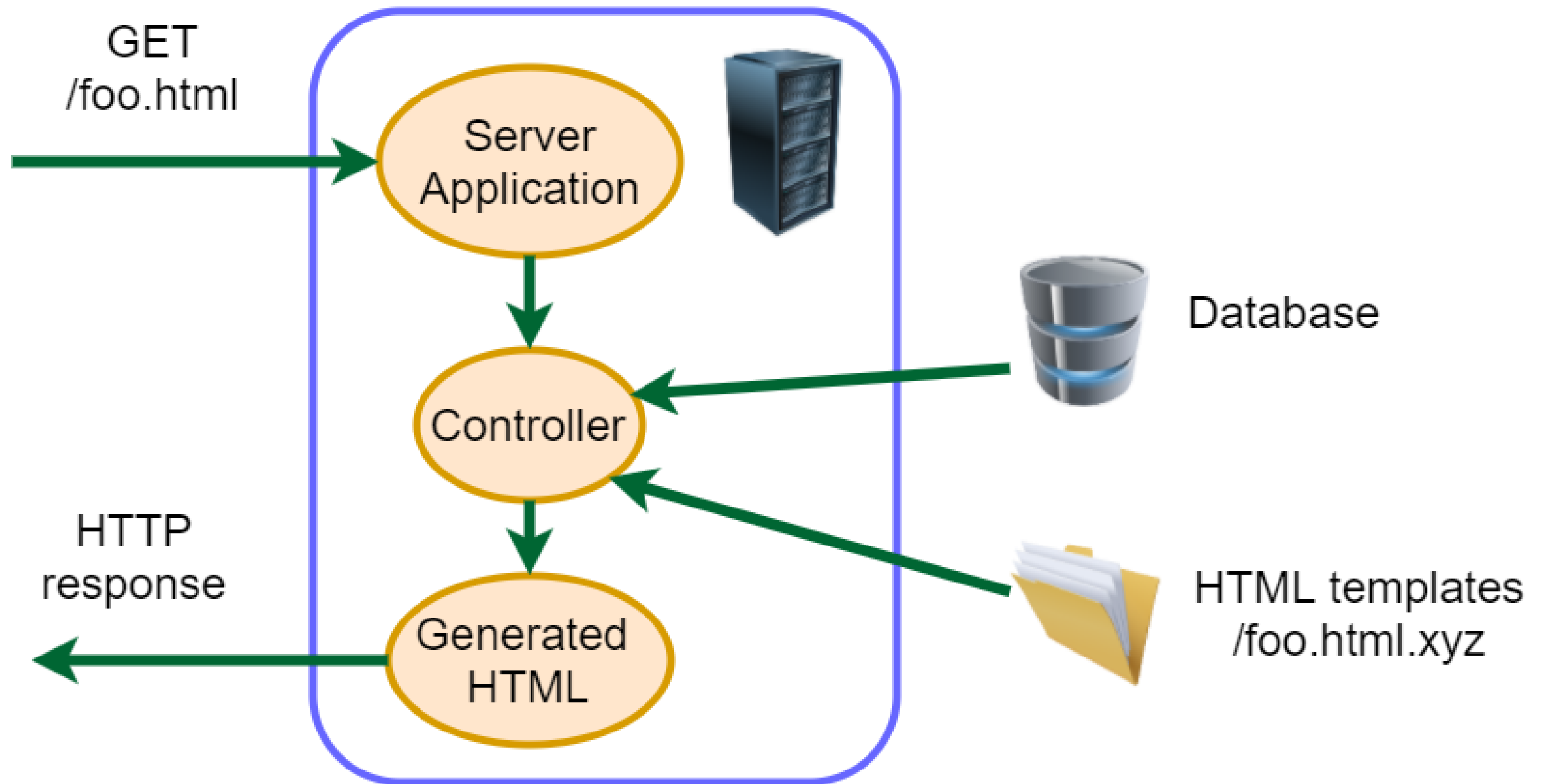
- In E1, we are going to only see server-side HTML rendering with a template technology like JSF (which is tight to Java-backends)
- In WDAD, we will see REST and client-side rendering with AJAX and WebSockets, with frameworks like React and Angular (which are not tied to any specific backend technology)
- In E2, we will have JVM backends with JS frontends
- Note 1: in current job market, learning a framework like *React is better than JSF...* but still important to get some experience with such kind of frameworks
- Note 2: we only going to scratch the surface of JSF, using the minimum necessary to make some non-trivial pages...

JSF

JEE Containers

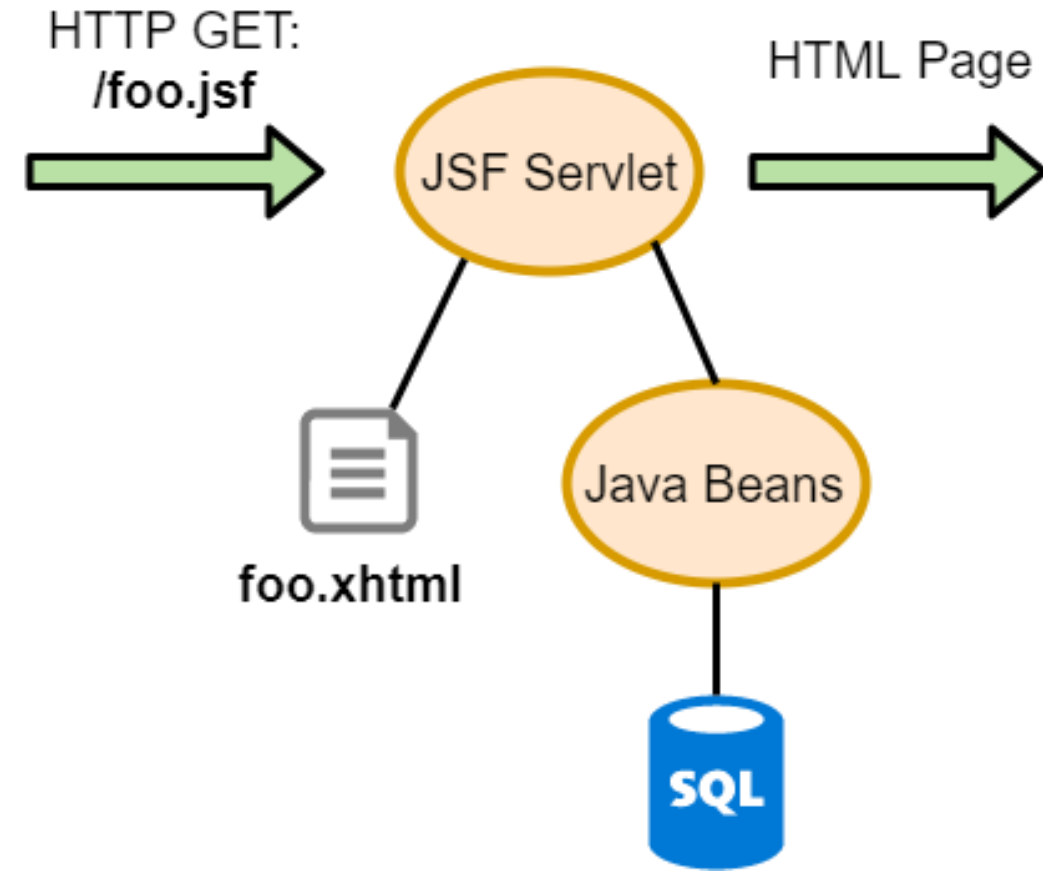
- Besides handling specs like JPA and EJB, a Container also provides a web server handling HTTP requests
- A *servlet* is the part that handles the HTTP requests, and creates dynamic HTML pages based on templates
- Different templating technologies, using servlets underneath
 - JSF main one in JEE, replacing older JSP
- As container can have different WARs installed, first step when receiving HTTP request is to check which WAR should handle it (based on URL path)





JSF: JavaServer Faces

- Template technology
- Templates written in *XHTML*: Extensible Hypertext Markup Language
- Valid XML files, resembling HTML
- Mixing together HTML-tags (eg <p>, <a>) with special tags that are going to be handled and transformed by JSF Servlet



JSF Template .XHTML example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html
    xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html"
    xmlns:f="http://xmlns.jcp.org/jsf/core"
>
<body>
<h2>Example of Dynamic Page</h2>
<p>
    Back to <a href="../index.html">Home</a>
</p>
<p>
    Current date: <h:outputText value="#{session.lastAccessedTime}">
        <f:convertDateTime pattern="MM/dd/yyyy" type="date"/>
    </h:outputText>
</p>
</body>
</html>
```

Example of Dynamic Page

Back to [Home](#)

Current date: 01/25/2018

Elements Console Sources Network

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  ▶ #shadow-root (open)
    <head></head>
    ▼ <body>
      <h2>Example of Dynamic Page</h2>
      ▼ <p>
        "
        Back to "
        <a href=" ../index.html">Home</a>
      </p>
      ... <p>
        Current date: 01/25/2018
      </p> == $0
    </body>
  </html>
```

Tags

- Based on tags, JSF decides how to translate *<tags>* into HTML
- JSF has a predefined sets of tags that it can handle
- In some cases, direct mapping to HTML
- Other cases, will execute code to generate HTML
- **<h:outputText value="#{session.lastAccessedTime}">**
 - Create text based on Java code executed inside “#{ }”
- **<f:convertDateTime pattern="MM/dd/yyyy" type="date"/>**
 - Specify how a Java Date object should be printed

Namespaces (ns)

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
```

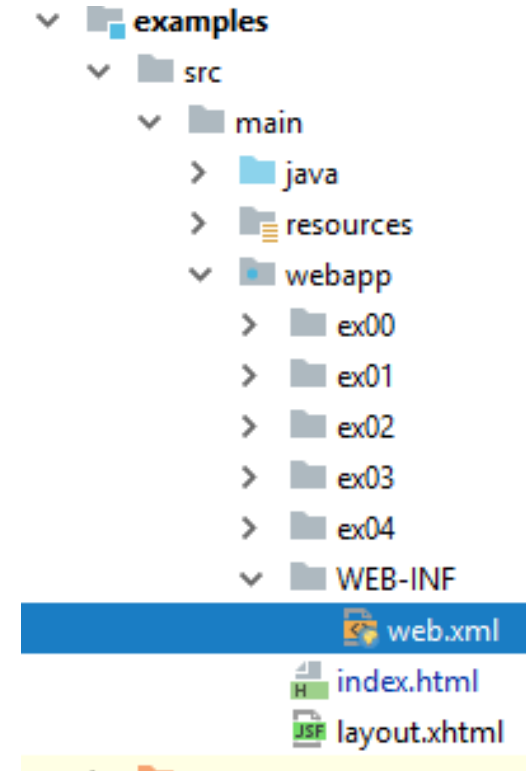
- Namespaces are like “*import packages*” in Java, used to avoid name conflicts
- Used namespaces need to be declared at beginning, with a custom variable name (usually just a letter, like *h* and *f*, to avoid too much typing)
- `<a>` belongs to default namespace
 - “http://www.w3.org/1999/xhtml” in this example
- `<h:outputText>` belongs to namespace called *h*
 - “http://xmlns.jcp.org/jsf/html” in this example

JSF Tags

- JSF provides several tags with different namespaces
- JSF can also be extended with more tags, with libraries like *PrimeFaces* and *OmniFaces*
 - But we are not going to see them...
- Available tags/namespaces (see online for full list of available tags):
 - <http://xmlns.jcp.org/jsf/html>
 - <http://xmlns.jcp.org/jsf/core>
 - <http://xmlns.jcp.org/jsf/facelets>
 - <http://xmlns.jcp.org/jsf/composite>
 - <http://xmlns.jcp.org/jsf/passthrough>
 - <http://xmlns.jcp.org/jsp/jstl/core>
 - <http://xmlns.jcp.org/jsp/jstl/functions>

Maven Folder Structure

- Template *.XHTML* and other assets (HTML/CSS/JS/images/etc.) are not source files to compile
- Need to be put under “*src/main/webapp*”
- Anything under such folder will be available with HTTP request to server, but “*WEB-INF*”
- “*WEB-INF/web.xml*” is where configurations are set, like JSF handling



Docker

Deploy OS Images

- When developing applications, not limit to just package your code
 - Java, NodeJS, PHP, etc.
- Create a whole image of an OS, including all needed software
 - Eg the version of JRE/.Net/Ruby/etc. that you need
 - Eg having a JEE Container like WildFly
- Particularly useful when developing web applications to install on a server
- Do not install the OS image on the server, but rather run it in a virtual machine
- Also, instead of installing a database, could just load a OS image with it
- How to automate all this?

Docker to the Rescue



- Automate the deployment of application inside software containers
- Create whole OS images, based on predefined ones
- Eg, a Linux distribution with the latest version of the frameworks you need
 - NodeJS, PHP, JDK, etc.
- Large *online* catalog of existing base images at Docker Hub
- Your application, and any needed third-party library, will be part of the OS image
- Use Docker (and tools built on / using it) to deploy your OS images and start them locally or on remote servers

How to Use Docker?

- First you need to install it
 - <https://store.docker.com/>
 - Note: if you are using Windows, Home Edition might not be enough. You would need a better version, like the Educational one, which you should be able to freely get from school
- To run existing images, you just need to type commands from a shell terminal (eg, GitBash)
- When you are writing your own projects, you need to create configuration files
 - *Dockerfile*: specify how to build an OS image
 - *docker-compose.yml*: for handling multi-images
- Then, use *docker* and *docker-compose* commands from the command line

Docker Examples

- <https://docs.docker.com/get-started/>
- <https://hub.docker.com/r/docker/whalesay/>
- ***docker run docker/whalesay cowsay boo***
 - This will install the image “docker/whalesay”, and run it with input “cowsay boo”
 - First time you run it, the “docker/whalesay” image will be downloaded

< boo >

///

The diagram illustrates the process $e^+e^- \rightarrow e^+e^- + 3\gamma$. It features a central vertex labeled '0' where a virtual photon splits into three photons and an electron-positron pair. The incoming electron and positron lines are represented by wavy lines, and the outgoing electron and positron lines are also wavy. The three photons are represented by wavy lines, and the electron-positron pair is represented by a loop of fermion lines.

```
Andreas-MBP-2:~ foo$
```

Custom Images

- Extend existing images to run the applications you develop
- Just need to create a text file called “Dockerfile”
- **FROM:** specify the base OS image
- **RUN:** execute commands in the virtual OS to set it up, like installing programs or create files/directories
- **CMD:** the actual command for your application
- **ENV:** define an environment variable
- **ADD:** take a file X from your hard-disk, and copy it over to the Docker image at the given path Y
 - When Docker image runs, it can access X at path Y, even when you deploy the image on a remote server
- **WORKDIR:** specify the working directory for the executed commands
 - Think about it like doing “cd” to that folder, so all commands/files are relative to that folder, and you do not need to specify full path
- **#** are comments

```
# Specify which OS image to run.  
# In our case, we are using an OS image with WildFly started  
# as a daemon/service  
FROM jboss/wildfly:10.1.0.Final  
  
# Copy the generated WAR from "target" folder into the Docker image,  
# in the folder where Wildfly is expecting to find installed WAR files  
ADD target/examples.war /opt/jboss/wildfly/standalone/deployments/  
  
# No need of CMD here, as Wildfly is automatically started as a service
```

Docker Commands

- ***docker build -t <name> .***
 - Create an image with name <name>, from the Dockerfile in the current “.” folder
- ***docker run <name>***
 - Run the give image
- **docker ps**
 - Show running images
- **docker stop <id>**
 - Stop the given running image. Note: an image can be run in several instances, with different ids
- In IntelliJ, you can also install “Docker integration” plugin

Docker in This Course

- Java is quite portable... for monolithic Java servers (ie single applications), using Docker is not critical (compared to other languages/frameworks...)
- But it simplify starting JEE Containers
- We will also use it for databases (eg Postgres) and browsers (eg Chrome) for testing (ie Selenium)
- Docker will be critical in Enterprise 2, when dealing with microservices (where we will need to orchestrate many applications...)

Git Repository Modules

- *NOTE: most of the explanations will be directly in the code as comments, and not here in the slides*
- **intro/jee/jsf/base**
- **intro/jee/jsf/examples**
- Exercises for Lesson 07 (see documentation)