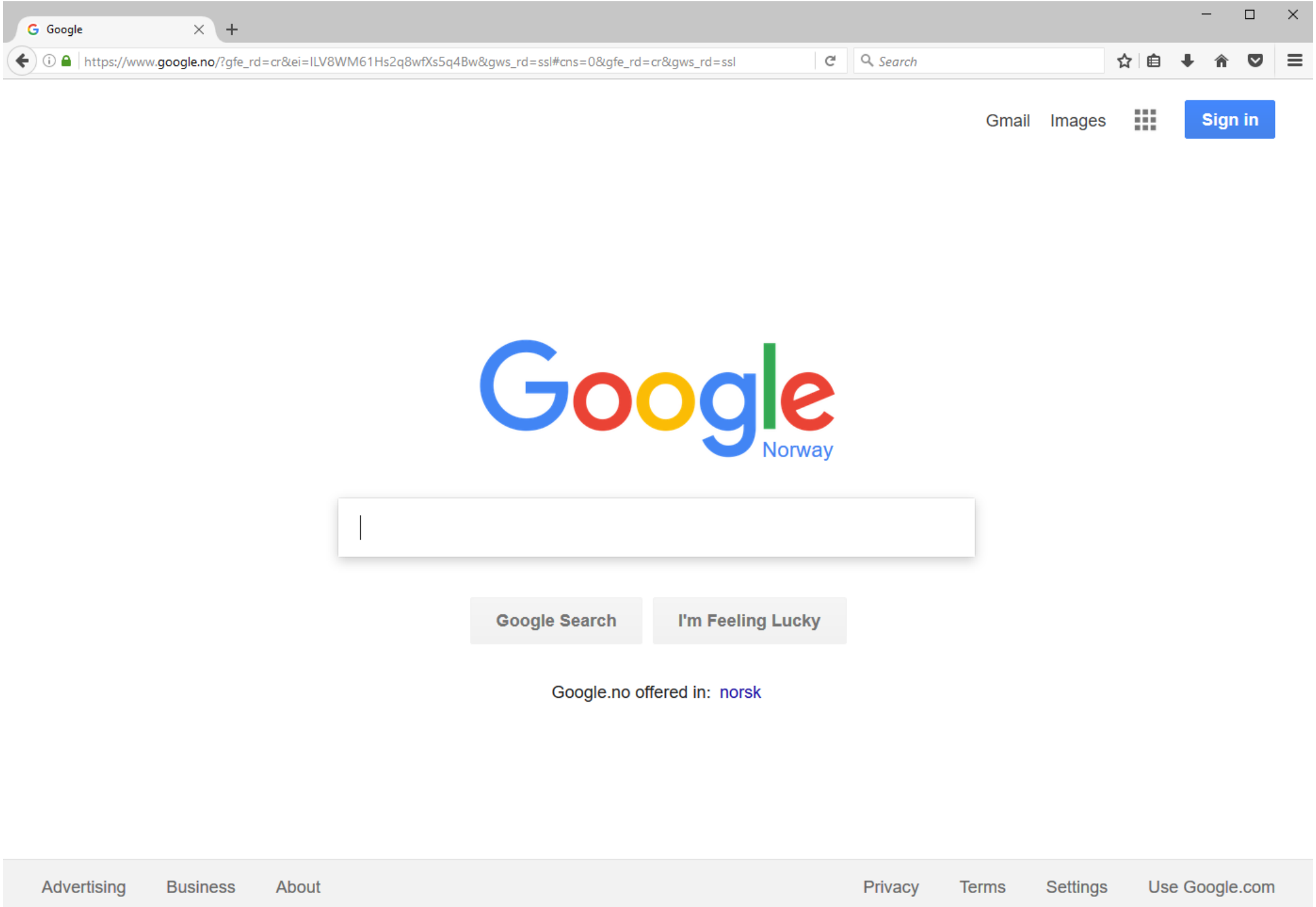
Enterprise Programmering 1

Lesson 07: JSF and Docker

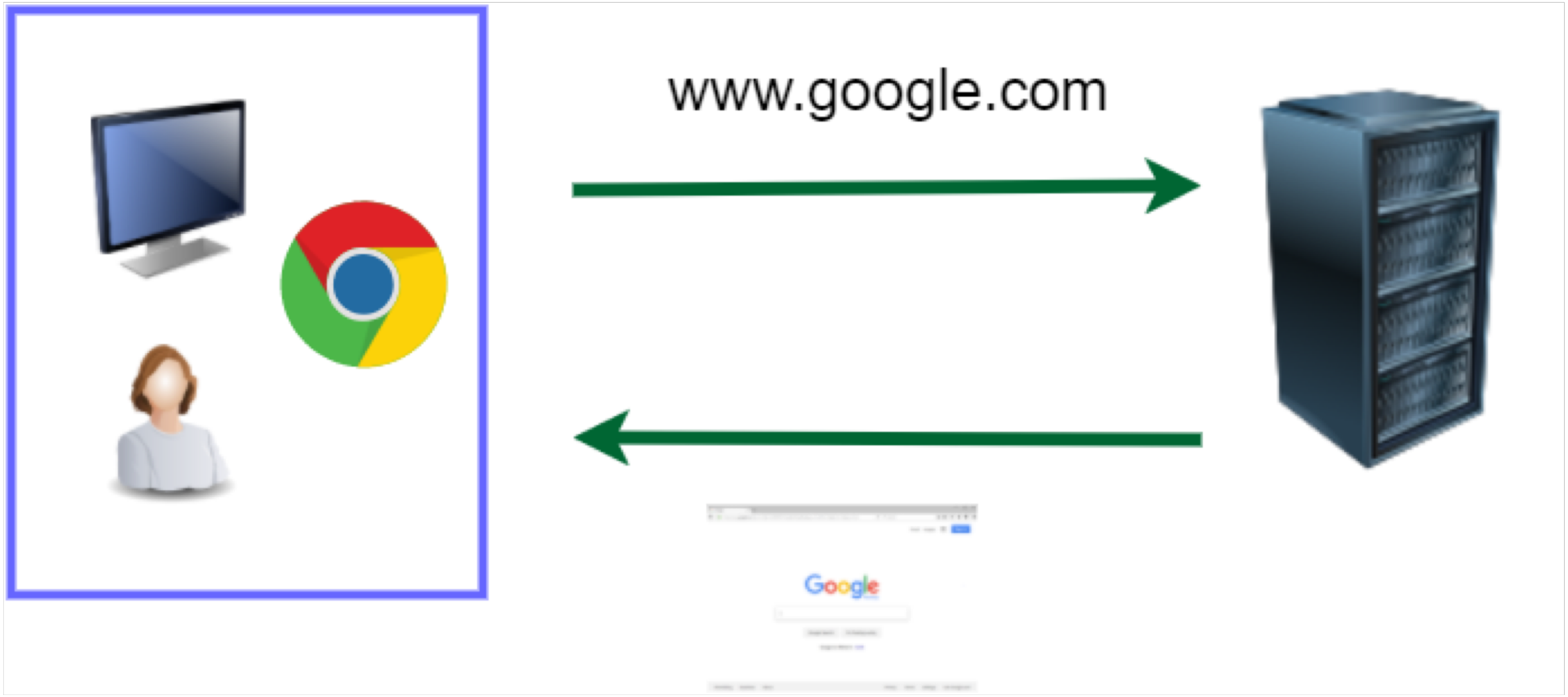
Prof. Andrea Arcuri

Web Applications


Google
https://www.google.no/?gfe_rd=cr&ei=ILV8WM61Hs2q8wfXs5q4Bw&gws_rd=ssl#cns=0&gfe_rd=cr&gws_rd=ssl
Search
Gmail
Images
Sign in
Google
Norway
Google Search
I'm Feeling Lucky
Google.no offered in: norsk
Advertising
Business
About
Privacy
Terms
Settings
Use Google.com

World Wide Web (WWW)

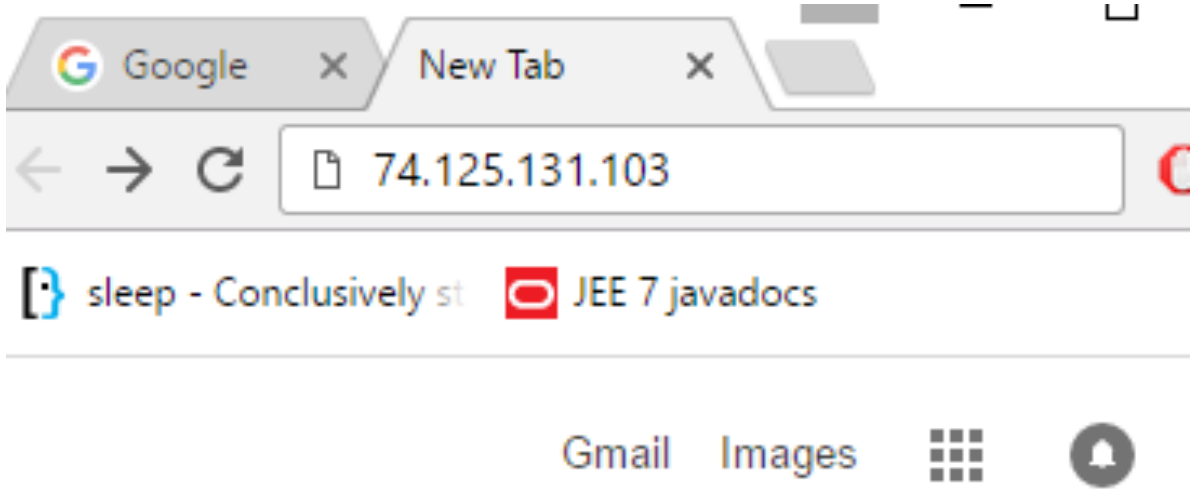
- Invented by Tim Berners-Lee in 1989, at CERN, Switzerland
  - I was in high school when it was first available…
- WWW is a set of documents/resources distributed among different machines
- Resources/documents are identified with a Uniform Resource Locator (URL)
- Resources are accessed/downloaded over 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*



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

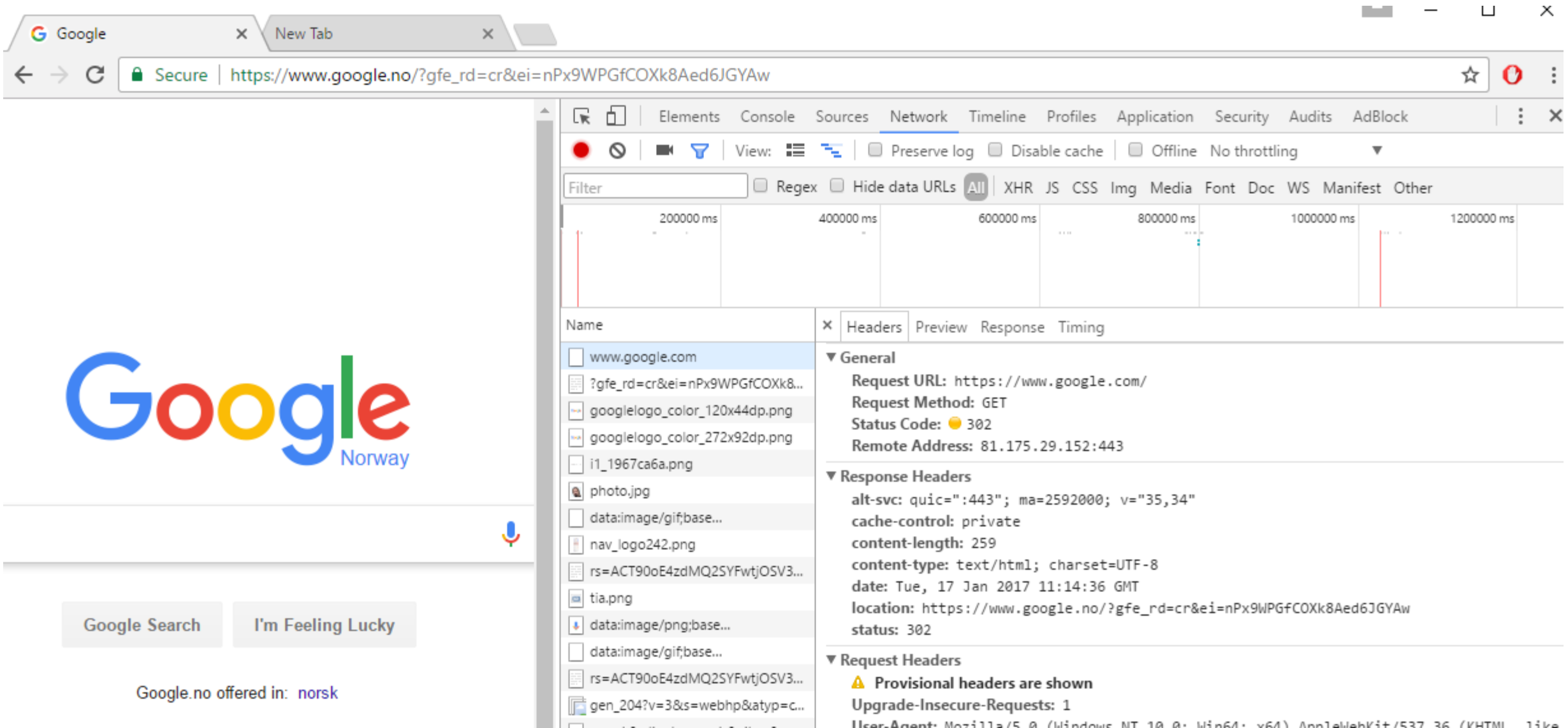
Domain Name System (DNS)

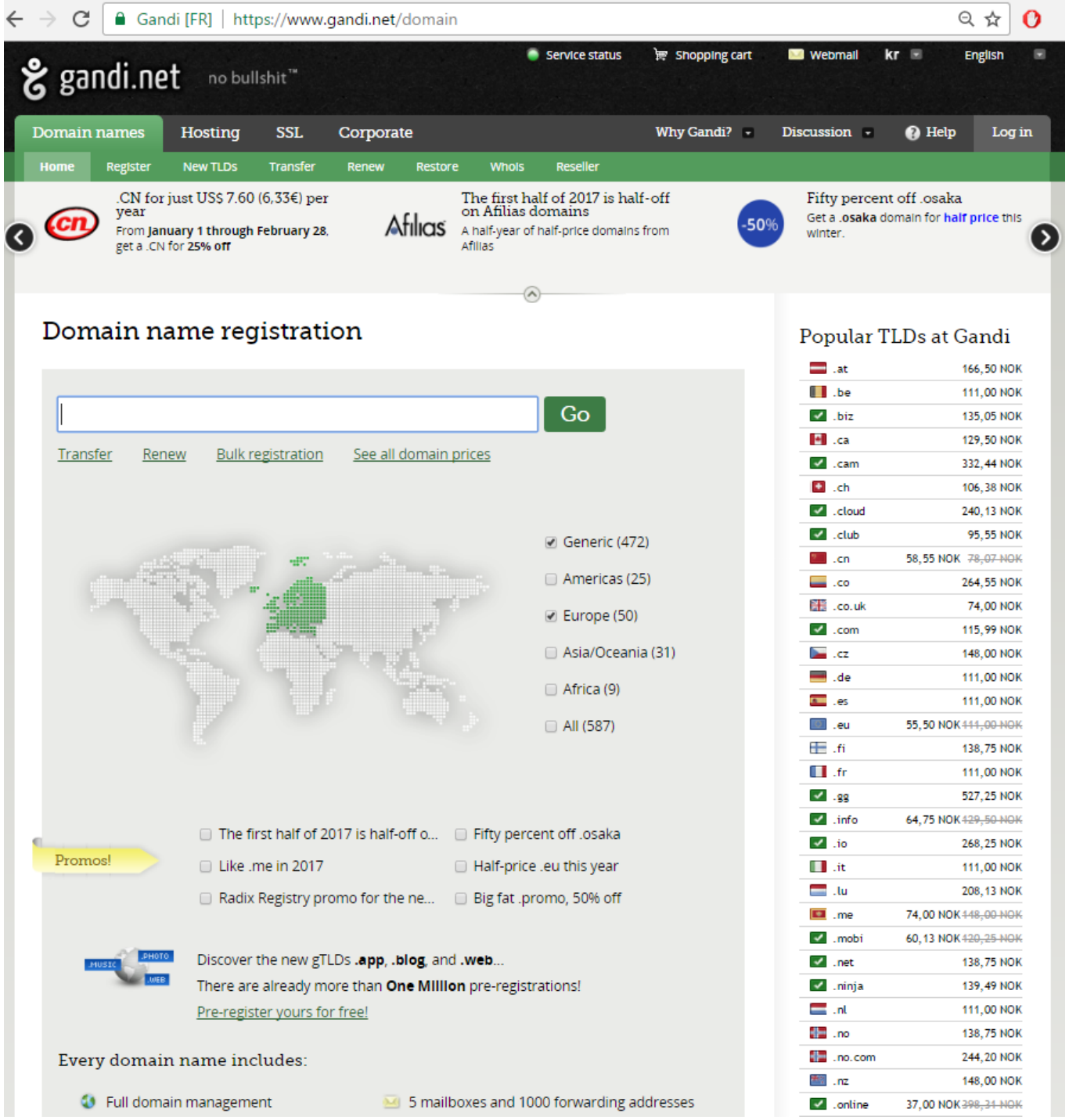
- When you type "*www.google.com*" in your browser, how does it now which machine to "connect" to?
- Internet connections are based on IP addresses
  - A 32 bit number, usually represented with 0-255 octets
    - 74.125.131.103 for www.google.com
  - IPv4 is most currently used. IPv6 is slooooowly replacing 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





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

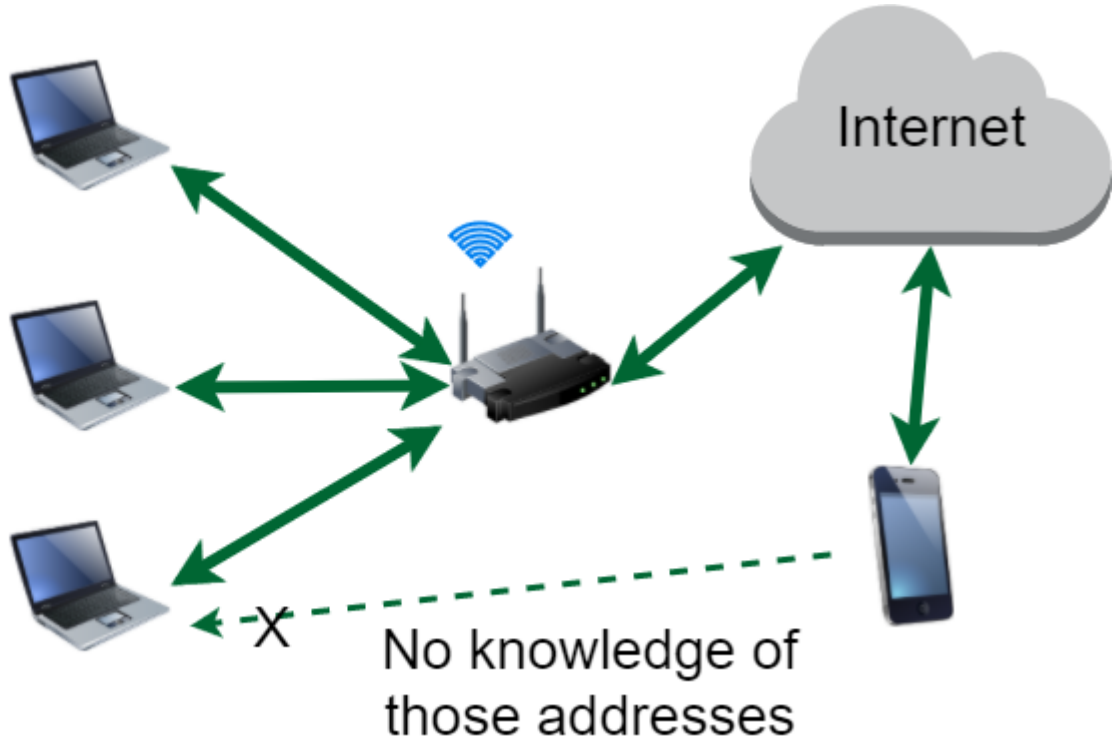
Defaults

- 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 http**s**://www.google.com is equivalent to http**s**://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

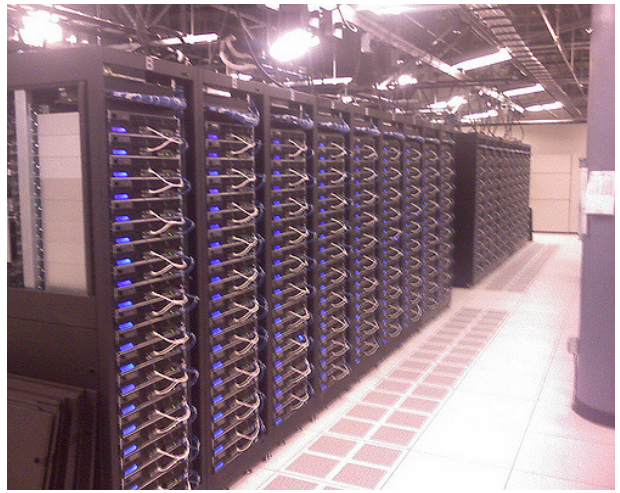
HTTP Protocol (Brief)

- 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

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

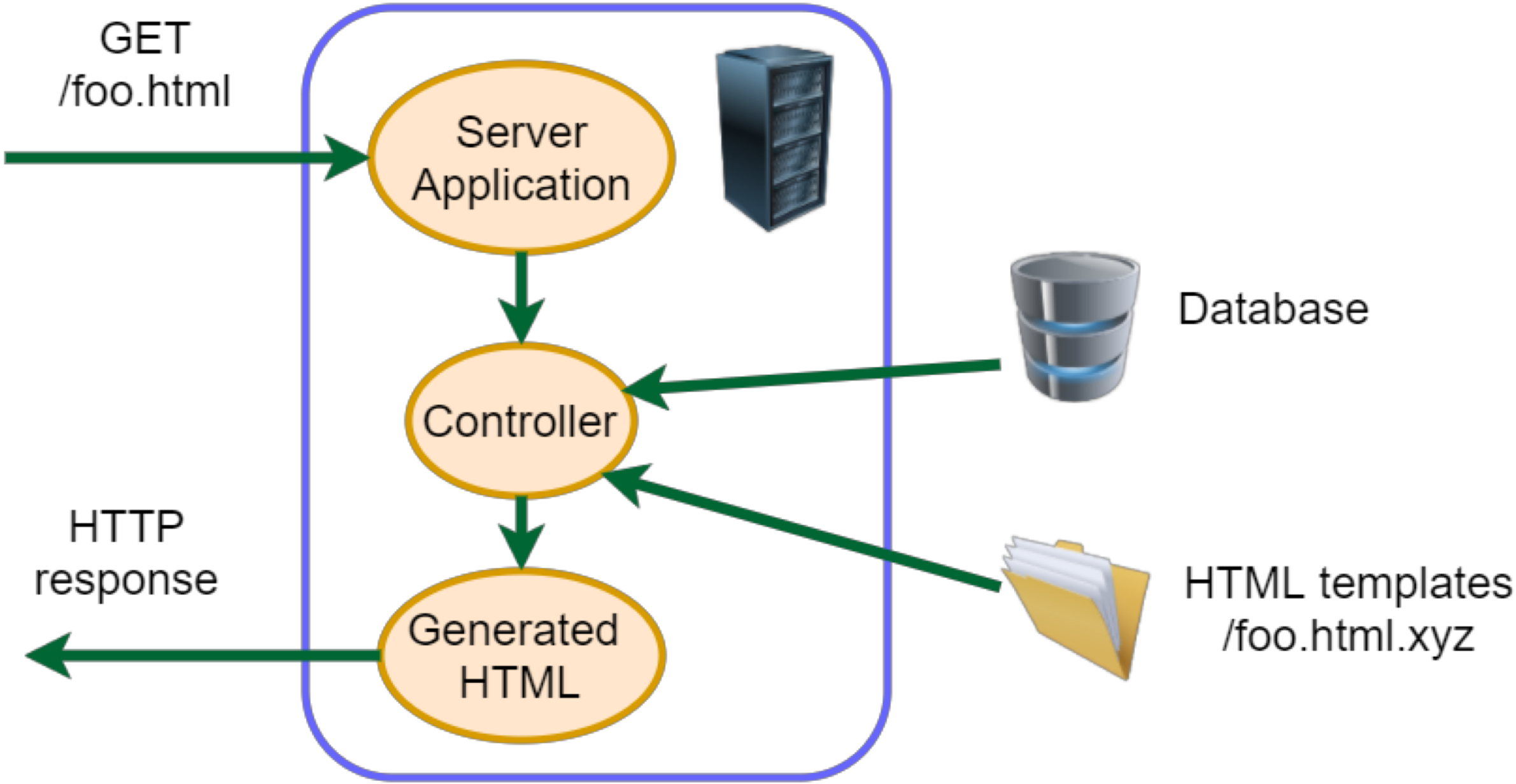
- To display the page in the previous slide, after a GET of the *index.html* page, the browser will do two further HTTP GET requests
- Those further 2 requests could be done in parallel on two different TCP connections

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

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

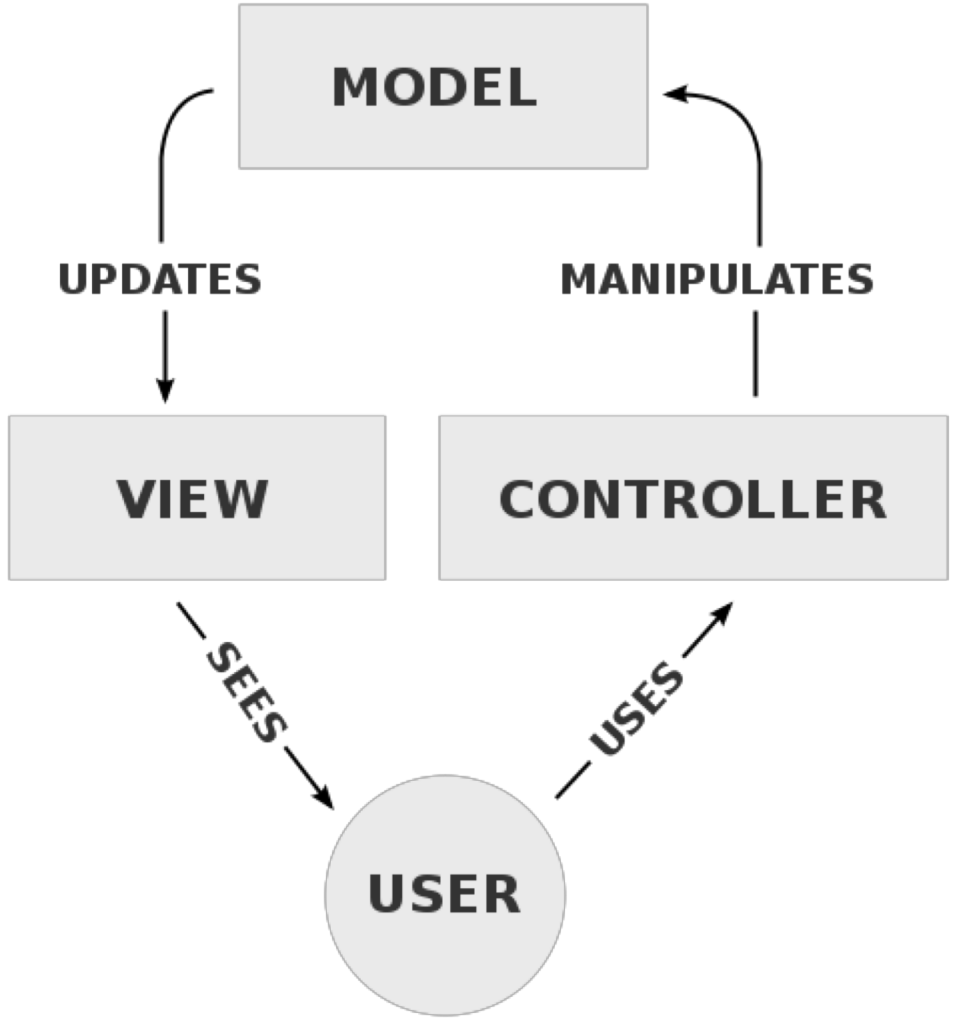

GET
/foo.html
Server
Application
Controller
Generated
HTML
HTTP
response
Database
HTML templates
/foo.html.xyz

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



Client-Side Rendering with 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)

JS: AJAX and WebSockets

- Executing JS on the client browser for DOM manipulation opens the door to many possibilities with *AJAX* and *WebSockets*
- *AJAX* can be used to retrieve only the needed data (eg JSON), instead of whole HTML page
- *WebSockets* enables "*pushes*" from the server
  - essential for chats and other live-applications like real-time games

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
  - Ie, 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 see REST and client-side rendering with AJAX and WebSockets, with libraries like React (which are not tied to any specific backend technology), in so-called *Single-Page-Applications*
- In E2, we will have JVM backends with JS frontends
- Note 1: in current job market, learning a library like *React is better than JSF*…
- Note 2: we only going to scratch the surface of JSF, using the minimum necessary to make some non-trivial pages…

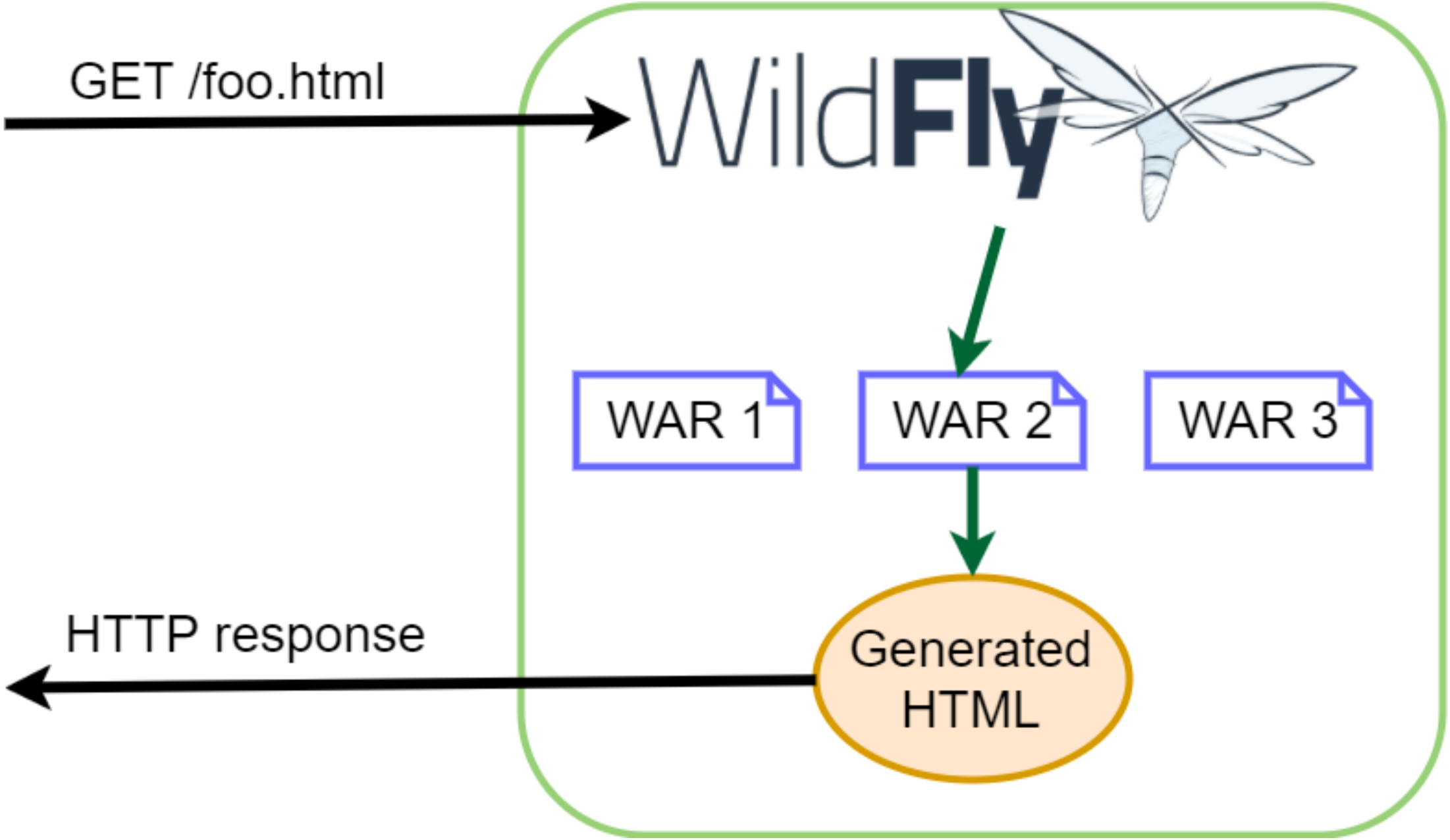
So... Why Learning JSF???

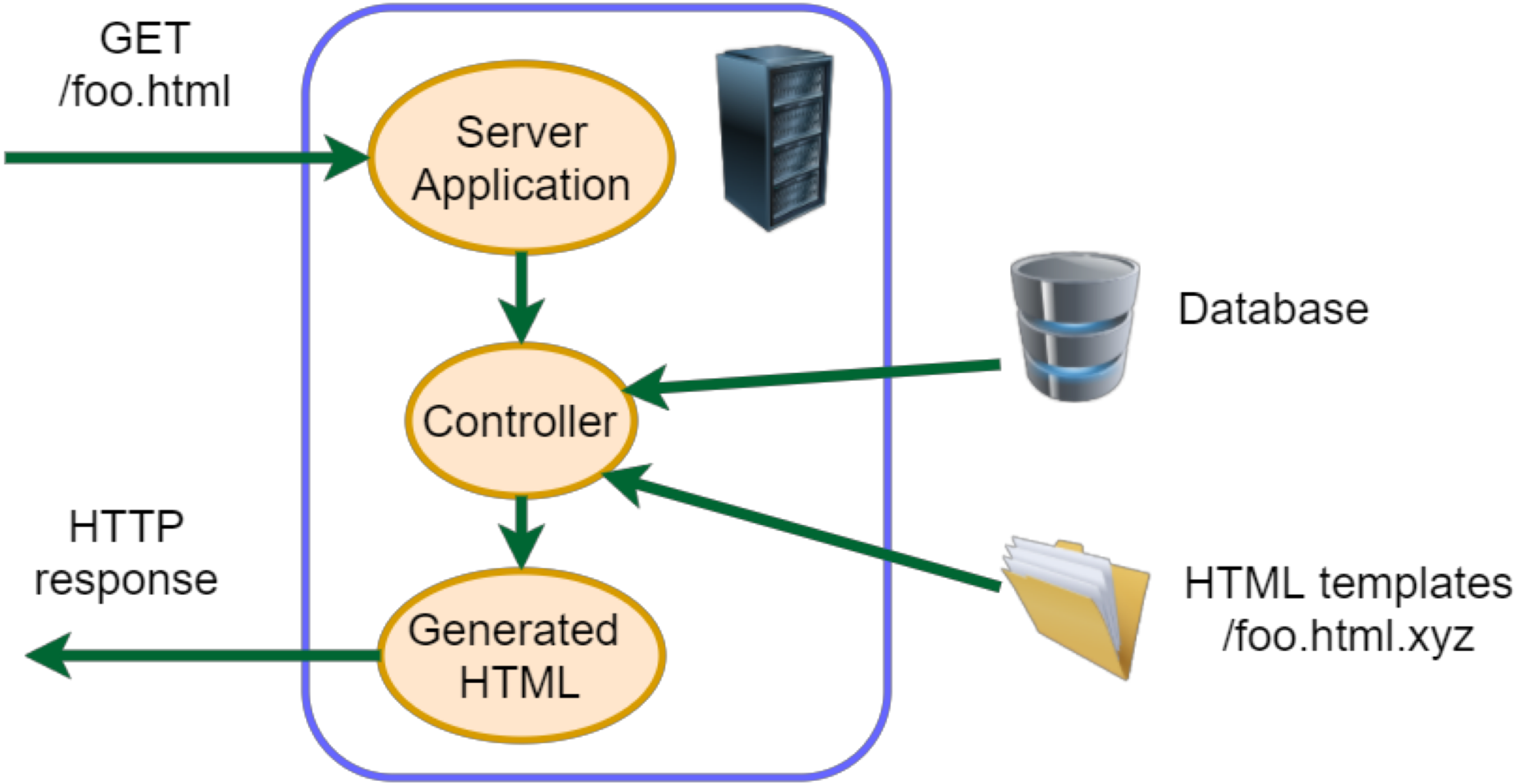
- Important to learn the foundations of web development
  - need to learn how full web applications can be developed with no JS
  - just studying the latest trends does not give you the depth and insight to understand the new technologies of tomorrow
- Need some experience with *server-side HTML rendering*
- Could have used any other template technology… chose JSF just because can run on both JEE and Spring

JSF

JEE Containers

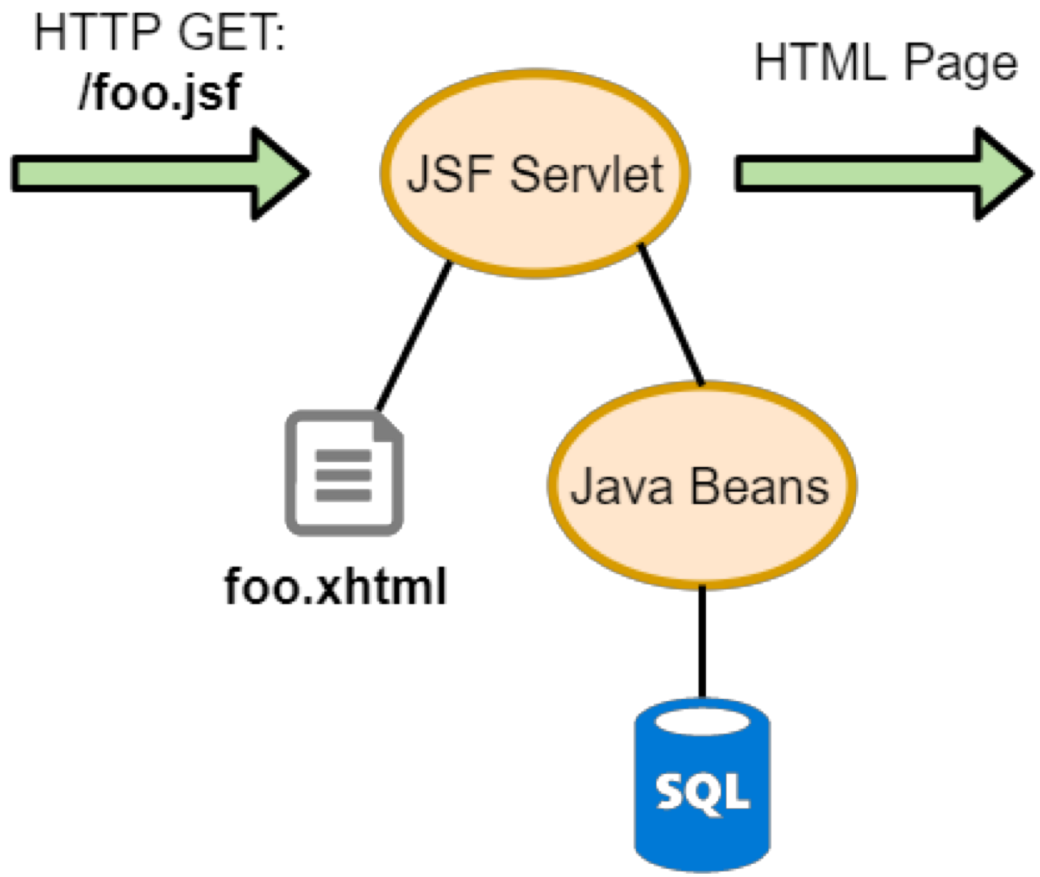
- Besides handling specs like JPA and EJB, a Container also provides a web server handling HTTP requests
- A *servlet* is the part that handles the HTTP requests, and creates dynamic HTML pages based on templates
- Different template technologies, using servlets underneath
  - JSF main one in JEE, replacing 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)


GET /foo.html
WildFly
WAR 1
WAR 2
WAR 3
Generated HTML
HTTP response


GET
/foo.html
Server
Application
Controller
Generated
HTML
HTTP
response
Database
HTML templates
/foo.html.xyz

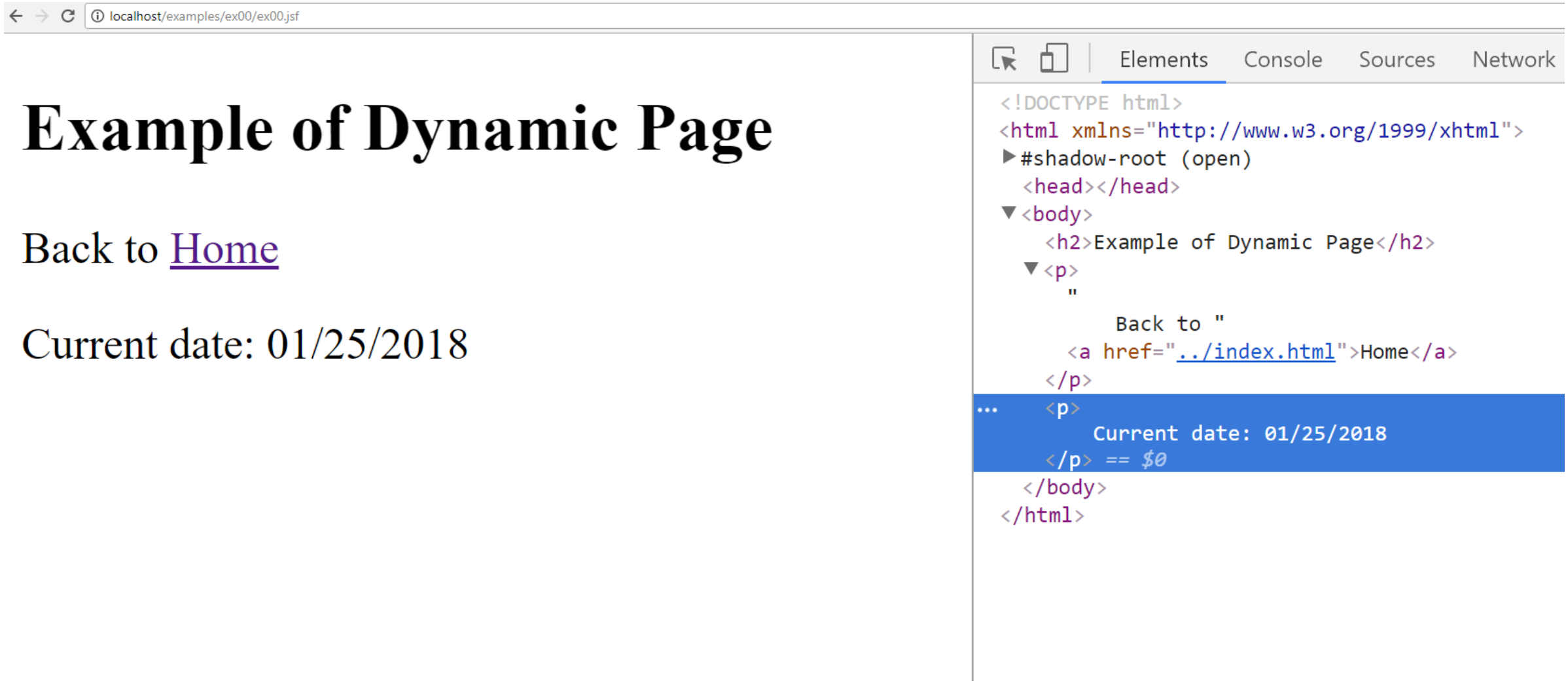
JSF: JavaServer Faces

- 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>
```


localhost/examples/ex00/ex00.jsf
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>

"
Back to "
Home

Current date: 01/25/2018
 == $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">
```

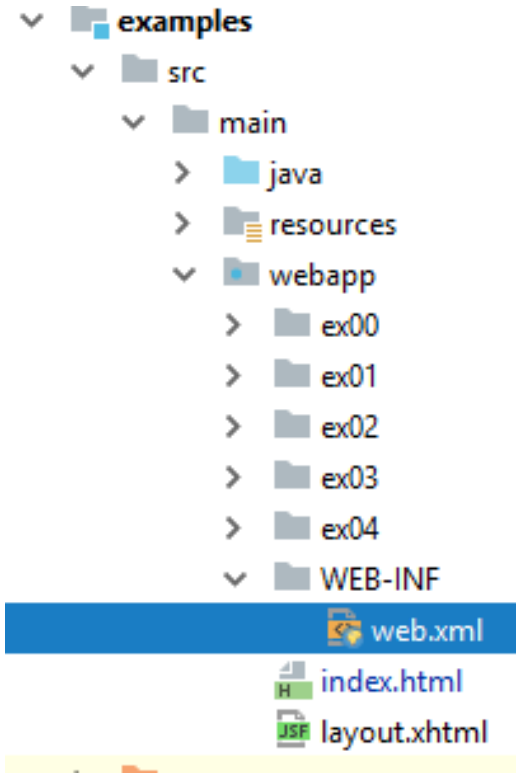
- Namespaces are like "*import packages*" in Java, used to avoid name conflicts
- Used namespaces need to be declared at beginning, with a custom variable name (usually just a letter, like *h* and *f*, to avoid too much typing)
- *<a>* belongs to default namespace
  - "http://www.w3.org/1999/xhtml" in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



JSF Managed Beans

- From XHTML, we can call Java code, inside **#{}** blocks in the tag attributes
  - "*value*" can be read when generating HTML server-side
  - "*action*" can be executed once server receives HTTP POSTs
- JSF uses special beans as a bridge between the HTML GUI and the business logic (eg EJBs)
- Different kinds of JSF-Beans, but we will just need 2:
  1. *@RequestScoped*, created for each HTTP request
  2. *@SessionScoped*, created for each user session (based on cookies)

Cont.

- To be able to be referenced from XHTML, the beans need to be marked with *@Named*
  - can then use name of the class with first letter in lower case to reference an instance of the bean, eg a class *FooBar* can be referenced with a variable called *fooBar*
- JSF-Beans can have injected EJBs
- JSF-Beans should not really have business logic (which should be in the EJBs)
- JSF-Beans should mainly deal with handling of form data and page navigation

Page Navigation

- Can simply use *<a>* tags for simple navigation between pages
- Complexity arises when *<form>* POSTs execute JSF-Bean code on the backend
  - this is will be defined inside the "*action*" attribute
  - eg, *<h:commandButton value="X" action="#{fooBar.someMethod}" />*
- If the JSF-Bean method returns void, then the POST returns the rendered HTML of the current page
- If otherwise returns a String, that can be a URL path toward another page to *redirect* to
  - eg, once submitted data in a form, want to go to page showing this new data
  - eg, after login (userId/password sent by a form), automatically go to home-page

Forward and Redirect

- In JSF, there are 2 main ways for navigation
- **Forward**: return the rendered HTML of the linked-page as part of the body payload of the POST request
  - this is the default behavior
  - issue: the URL on address bar in browser does NOT change
- **Redirect**: return a 302 redirection, where path returned by "*action*" will be in the *Location* HTTP header
  - issue: browser will need to make 2 calls (first the POST, and then a GET), but the address bar will be updated correctly
  - needs to be activated with URL attribute: "**?faces-redirect=true**"
  - this is the one you should use most of the time

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

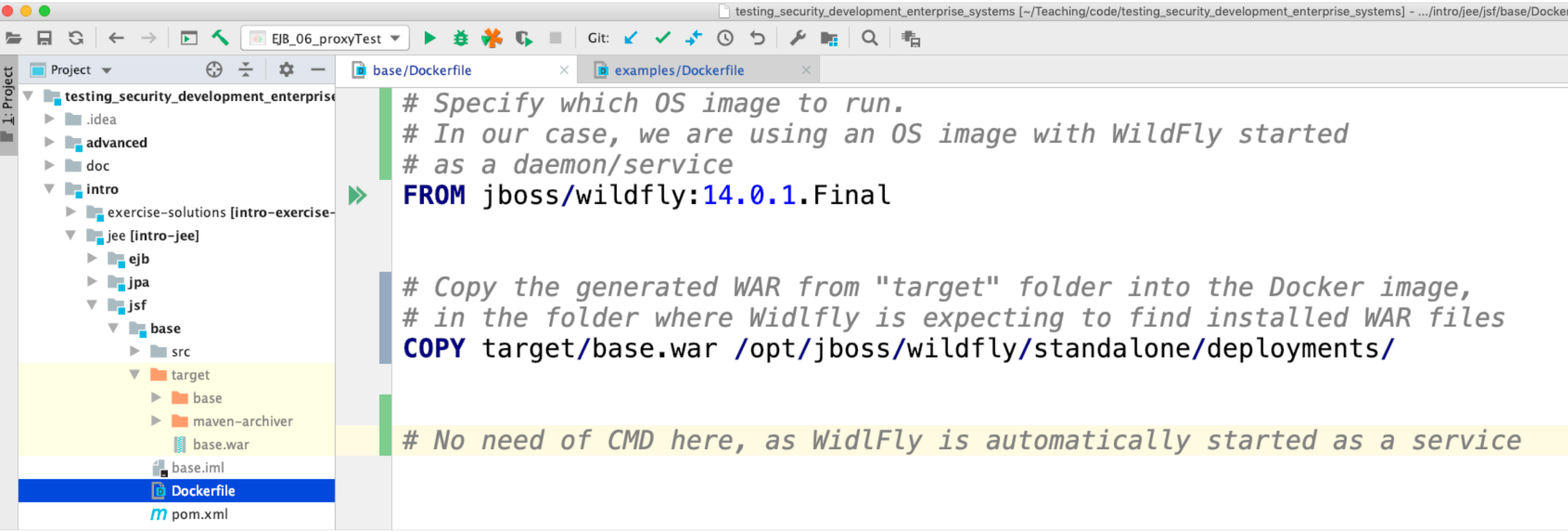
foo — -bash — 136×29

```
Andreas-MBP-2:~ foo$ docker run docker/whalesay cowsay boo
Unable to find image 'docker/whalesay:latest' locally
latest: Pulling from docker/whalesay

e190868d63f8: Pull complete
909cd34c6fd7: Pull complete
0b9bfabab7c1: Pull complete
a3ed95caeb02: Pull complete
00bf65475aba: Pull complete
c57b6bcc83e3: Pull complete
8978f6879e2f: Pull complete
8eed3712d2cf: Pull complete
Digest: sha256:178598e51a26abbc958b8a2e48825c90bc22e641de3d31e18aaf55f3258ba93b
Status: Downloaded newer image for docker/whalesay:latest
 _____
< boo >
 -----
    \
     \
      \
                    ##        .
              ## ## ##       ==
           ## ## ## ##      ===
       /""""""""""""""""___/ ===
  ~~~ {~~ ~~~~ ~~~ ~~~~ ~~ ~ /  ===- ~~~
       \______ o          __/
        \    \        __/
          \____\______/
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
- **COPY**: 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
  - Note, there is also an **ADD** option. Do NOT use it (ie, not recommended, as having side-effects besides adding files)
- **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


testing_security_development_enterprise_systems [~/Teaching/code/testing_security_development_enterprise_systems] - .../intro/jee/jsf/base/Docke
EJB_06_proxyTest
Git:
Project
base/Dockerfile
examples/Dockerfile
testing_security_development_enterprise
.idea
advanced
doc
intro
exercise-solutions [intro-exercise-
jee [intro-jee]
ejb
jpa
jsf
base
src
target
base
maven-archiver
base.war
base.iml
Dockerfile
pom.xml
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:14.0.1.Final
Copy the generated WAR from "target" folder into the Docker image,
in the folder where Widlfly is expecting to find installed WAR files
COPY target/base.war /opt/jboss/wildfly/standalone/deployments/
No need of CMD here, as WidlFly 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 given 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

Networking

- When you run a server on your local host, it will open a TCP port, typically 80 or 8080
- A server running inside Docker will open the same kind of ports, but those will not be visible from the *host* OS
- You need to explicitly make a mapping from *host* to *guest* ports
- Ex.: **docker run –p 80:8080 foo**
  - When we do a connection on localhost on port 80, it will be redirected to 8080 inside Docker

CTRL-C

- When running Docker (eg "*docker run*") in a terminal, you can use CTRL-C to stop it
- On Windows/GitBash it *might* happen that the image still run in background
- Use "*docker ps*" to check if indeed the case
- Use "*docker stop <id>*" to stop an image manually
- If you have Docker images running in the background, that can be a problem if they use TCP ports

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)