

# Lecture 2: presentation tier & MVC

---

Olivier Liechti  
AMT

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

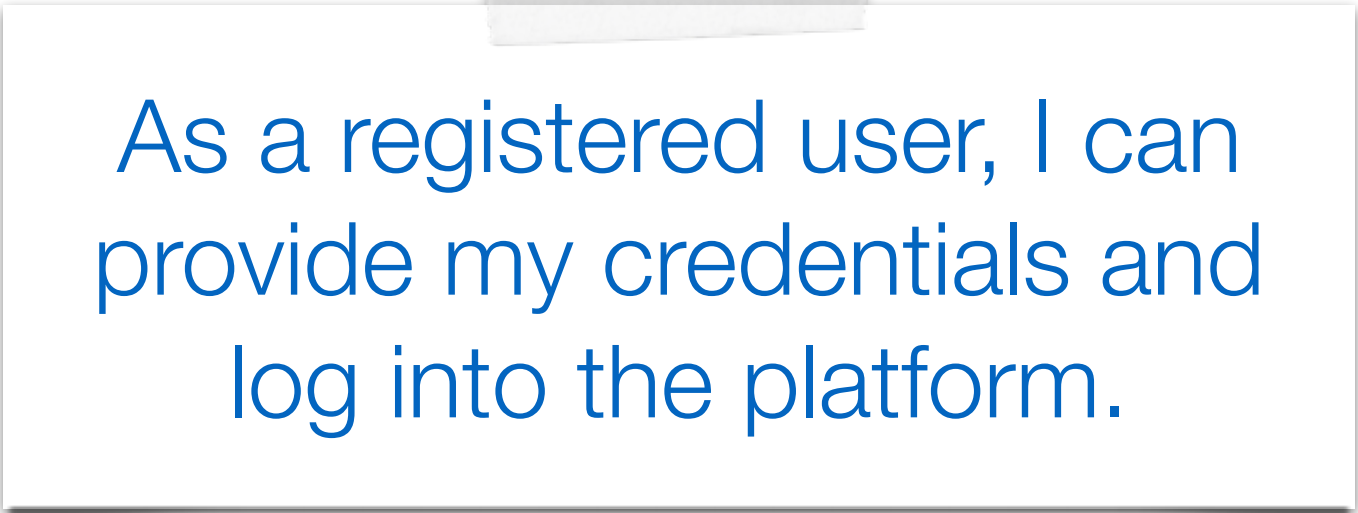
# Project - 2 features

---



As a guest, I can register  
and create an account.

*but the data does not have to be stored in  
a database. It can be stored “in memory”*



As a registered user, I can  
provide my credentials and  
log into the platform.

# Feature 1 tasks

Design and implement a page with a “protected” page

Design and implement a page with a login form (and a register link)

Design and implement a page with a registration form

Implement a security filter to protect pages from unauthenticated users

Implement a “users manager” service with an in-memory data store

Use the HTTP session to store current user identity

Validate that a guest can register and access protected pages.

**Update the docker-compose environment**

Validate that a registered user can login and access protected pages

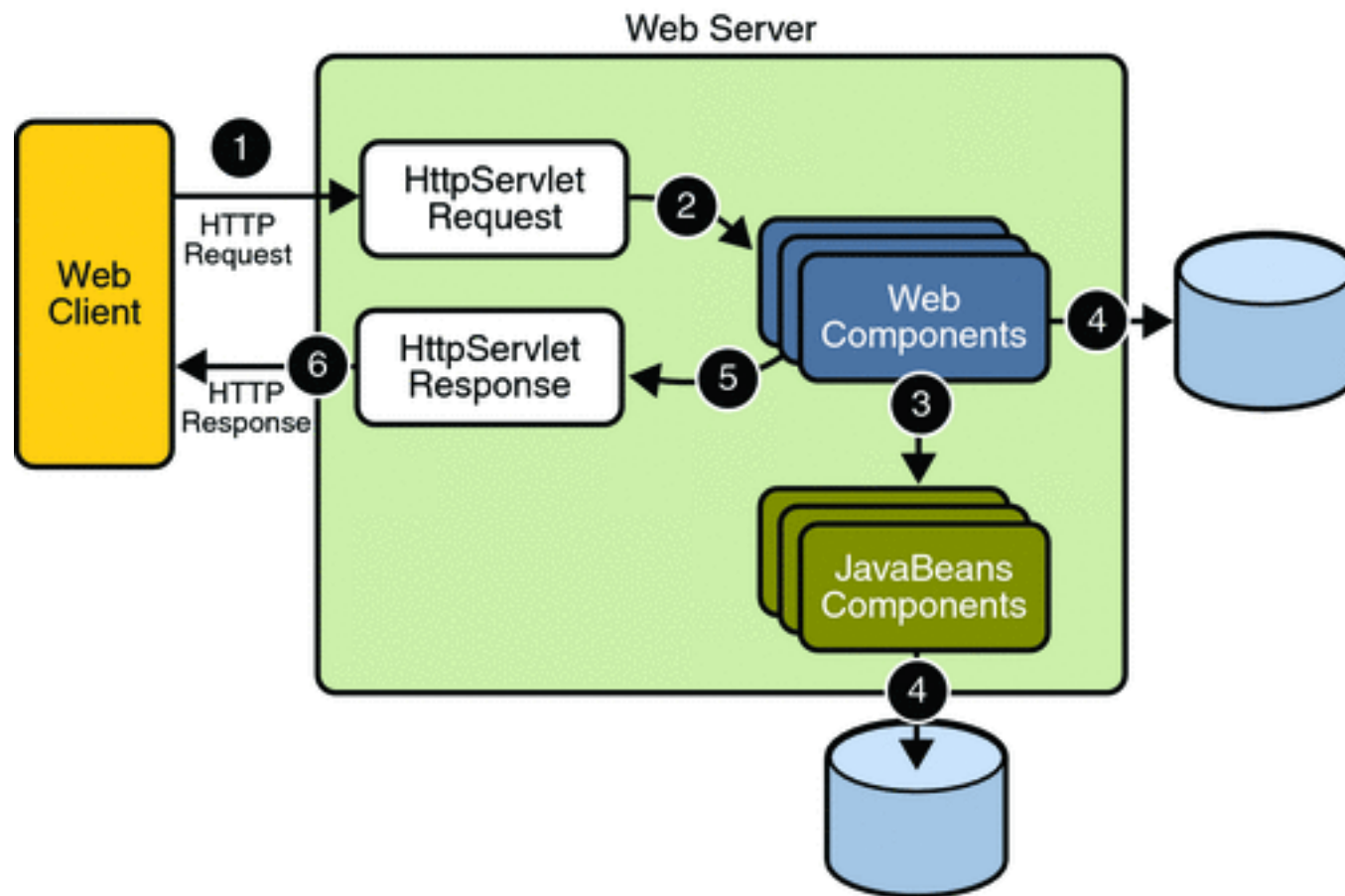
**Validate the acceptance criteria.**

Validate that an authenticated user can log out and not access protected pages anymore.

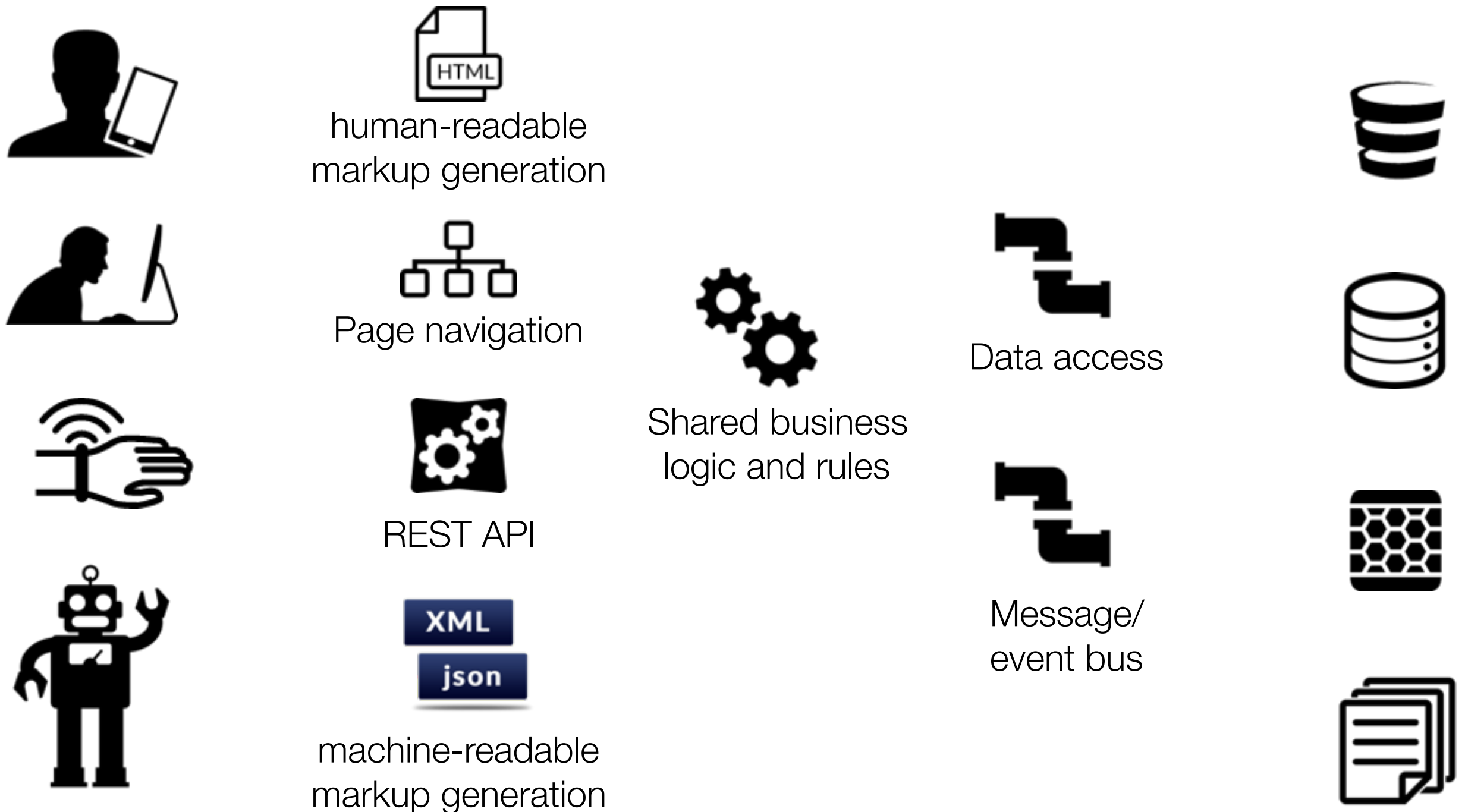
# Today's agenda

---

<b>13:15 - 13:35</b>	<b>20'</b>	<b>Intro: presentation tier &amp; MVC</b>
13:35 - 14:00	25'	In practice: setup local environment
14:00 - 15:30	90'	Step-by-step implementation of MVC
15:30 - 15:40	10'	Wrap-up



# Presentation Tier & MVC



User

Client

Presentation

Business

Integration

Resources

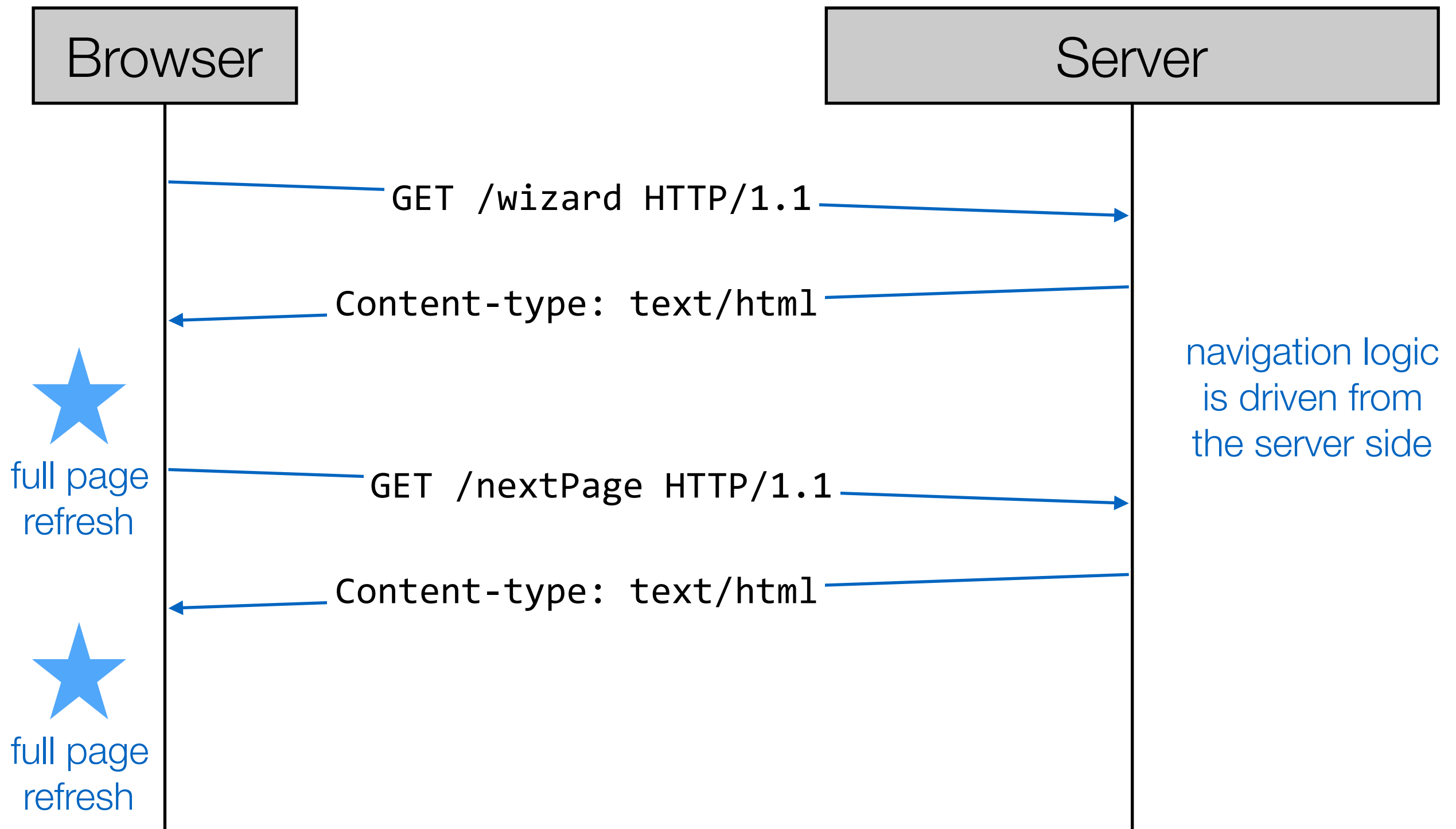


What is the relation between the **client** and the **presentation** tiers?

- The **client tier** is located on the **user device** (laptop, mobile phone, etc.).
- The **presentation tier** is located on the **server**. It generates content (HTML, JSON, PNG, etc.) that is used in the client tier (it also consumes content sent by the client tier).
- Components in the client and presentation tier use a **communication protocol**. Today, it is most often HTTP, but other protocols could be used.
- In many applications, the client tier consists of a **web browser**, a **JavaScript engine** and **scripts executed locally**.
- The client tier can also consist of a **native application** (it is often the case for mobile applications), making requests to the presentation tier. You could implement a native application in Java, with Swing.
- Some client applications **don't have a GUI** (command line tools, robots, etc.).



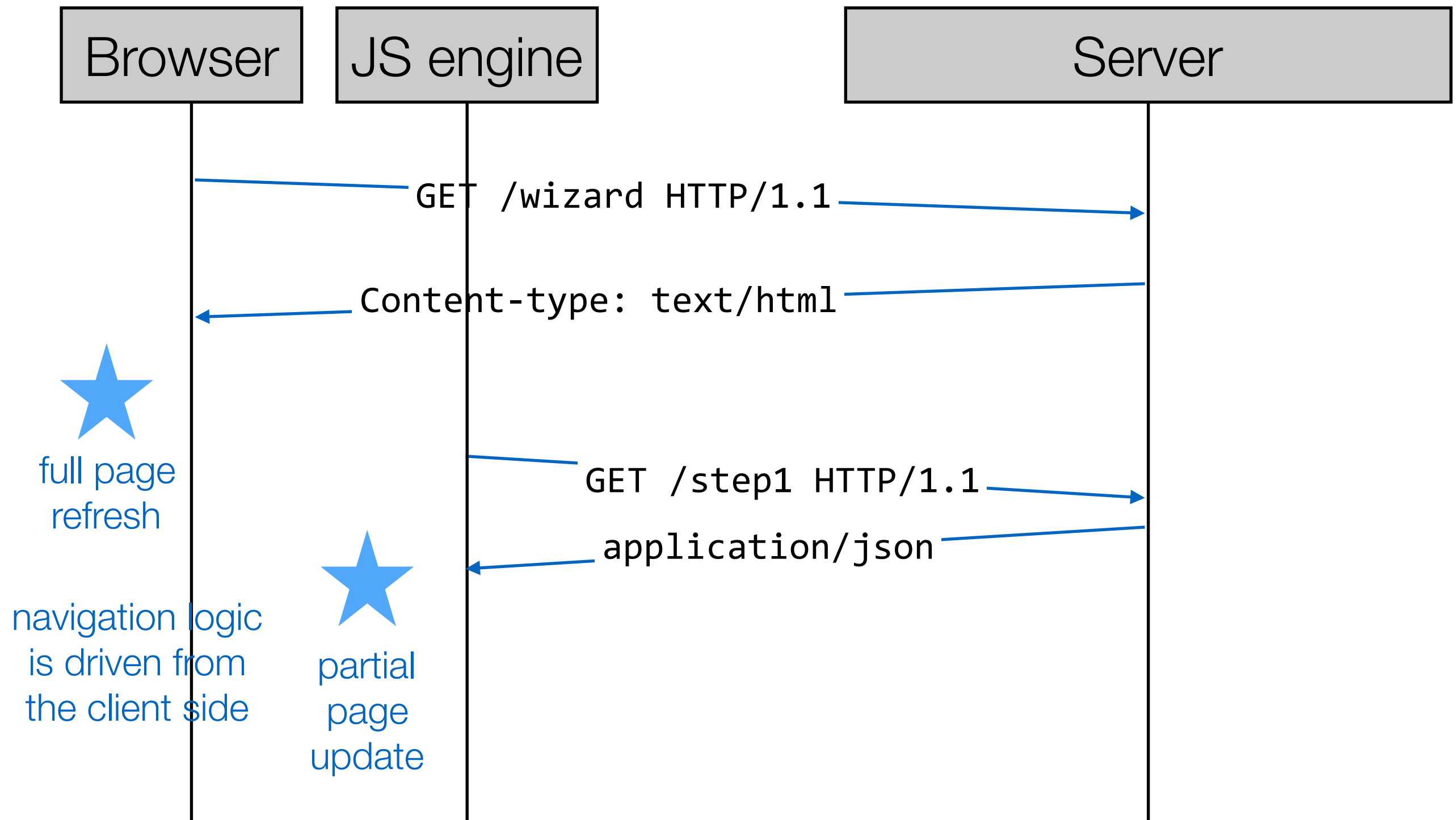
How do clients and server interact in the “traditional” MVC model?







How do clients and server interact in the  
“Single Page Application” model?





What is the ~~ugliest~~ simplest way to process an HTTP request in Java EE?

- HTTP requests sent by clients are received by the application server.
- The application server looks at the first element in the URL path to figure out **which application** should process the request. A **servlet mapping** is then used to figure out **which servlet** should process the request.
- The servlet has access to a **request** and a **response objects**. It can access **HTTP data** (URL, headers, payload) via these objects. The servlet can generate an HTML and send it back via the response object.

```
@WebServlet("/greeting") _____ GET /theApp/greeting HTTP/1.1
public class GreetingServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html lang=\"en\">");
        out.println("It is a bad idea to write HTML in a servlet. ");
        out.println("I will never, ever, do that.</html>");
        out.close();
    }
}
```

javax.servlet.http

## Class HttpServlet

java.lang.Object

javax.servlet.GenericServlet

javax.servlet.http.HttpServlet

### All Implemented Interfaces:

Serializable, Servlet, ServletConfig

---

```
public abstract class HttpServlet  
extends GenericServlet
```

Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site. A subclass of `HttpServlet` must override at least one method, usually one of these:

- `doGet`, if the servlet supports HTTP GET requests
- `doPost`, for HTTP POST requests
- `doPut`, for HTTP PUT requests
- `doDelete`, for HTTP DELETE requests
- `init` and `destroy`, to manage resources that are held for the life of the servlet
- `getServletInfo`, which the servlet uses to provide information about itself

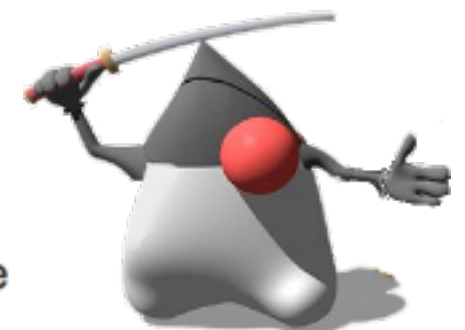
There's almost no reason to override the `service` method. `service` handles standard HTTP requests by dispatching them to the handler methods for each HTTP request type (the `doXXX` methods listed above).

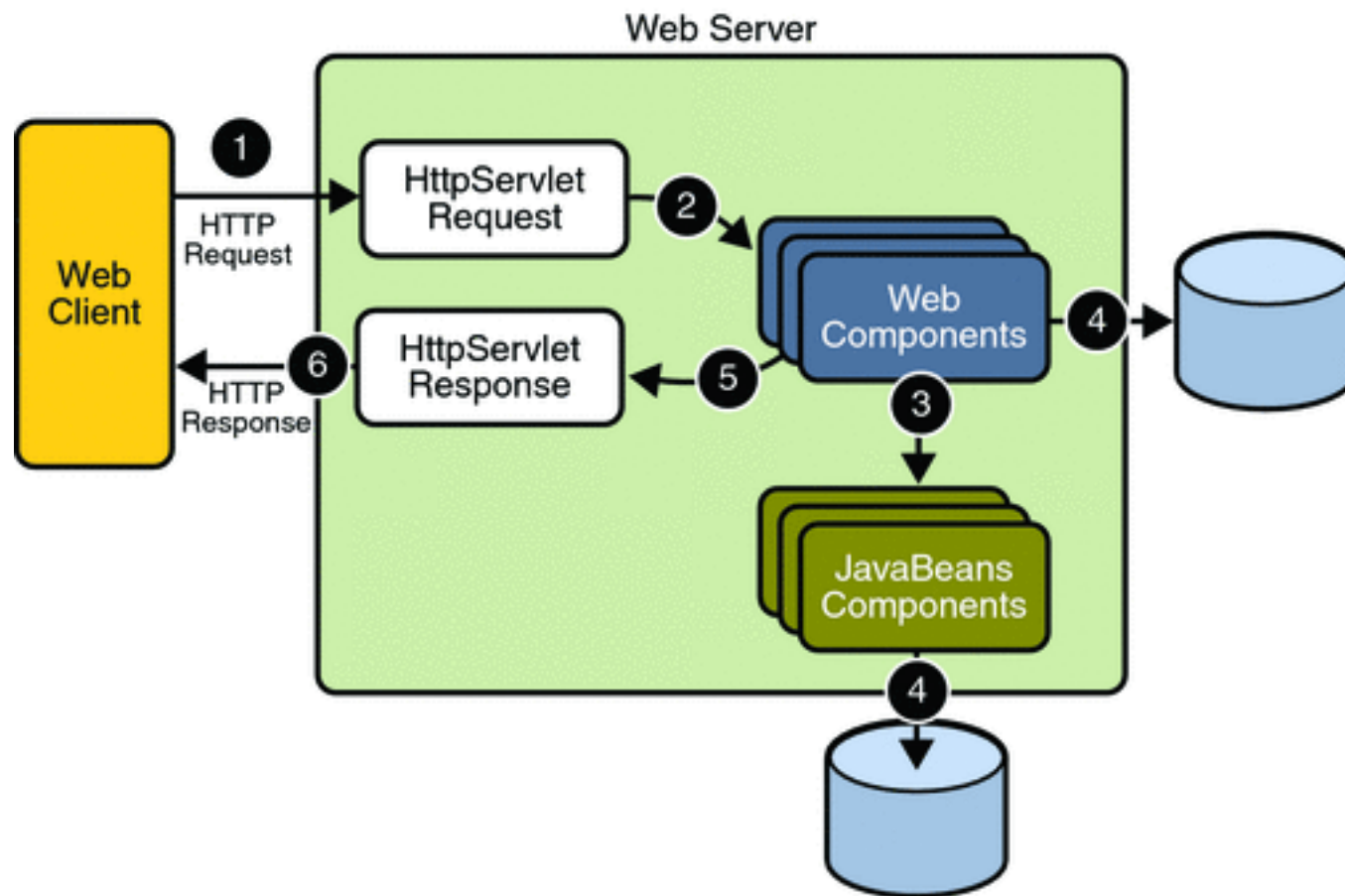
Likewise, there's almost no reason to override the `doOptions` and `doTrace` methods.

Servlets typically run on multithreaded servers, so be aware that a servlet must handle concurrent requests and be careful to synchronize access to shared resources. Shared resources include in-memory data such as instance or class variables and external objects such as files, database connections, and network connections. See the [Java Tutorial on Multithreaded Programming](#) for more information on handling multiple threads in a Java program.

eig-vd

te Ecole d'Ingénierie et de Gestion  
Canton de Vaud





# Let's put it in practice...

# Let's put it in practice

---

- Set up the **local environment** (in addition to the docker-compose setup)
- Create a new web project
- Create our first servlet

# What's wrong with this approach?

---

- There is **no separation of concerns**: the presentation and business logic is coded in the same module.
- The code is **hard to read** and to **maintain**.
- It is **not possible to share and reuse business logic** (the service) across different views. What happens when a business service needs to be accessed via a browser and a native mobile application?
- It is **not possible to distribute the work** between the back-end and the front-end developer.
- How do we implement and enforce **page navigation logic** in this scenario?



# The Model-View-Controller (MVC) pattern

---

- The MVC design pattern has initially been developed in **native GUI toolkits**:
  - The **model** is responsible to capture the state and behavior of a business object.
  - The **view** is responsible for rendering the model and present it to the user.
  - The **controller** is responsible to **react to events** (mouse, keyboard, etc.). In reaction to events, it invokes operations on the model (which changes its state) and asks

# What does MVC mean in Web Apps?

- The **model** is still responsible to capture the state (and behavior) of a business object.
- The **view** is responsible for rendering the model and present it to the user. Hence, the view is generating HTML, JSON, XML, PNG.
- The **controller** is responsible to **react to events**.
  - An event originates with a **user action** (clicking on a link, submitting a form, typing in a URL).
  - It is **encoded in an HTTP request** (with a URL, query string params, headers).



In web apps, **MVC can be implemented both on the server side and on the client side** (javascript frameworks). Here, we are talking about server-side MVC.





How are responsibilities shared in the MVC design pattern?

**model**

*I am (domain) **data***

**view**

*I know how to **present data** in a specific way.*

**controller**

*I know **what data** is needed.*

*I know **how** to generate data.*

*I know **which view** can present the data*



How can the MVC design pattern be implemented with Java EE technologies?

**model**

*I am a **JavaBean***

**view**

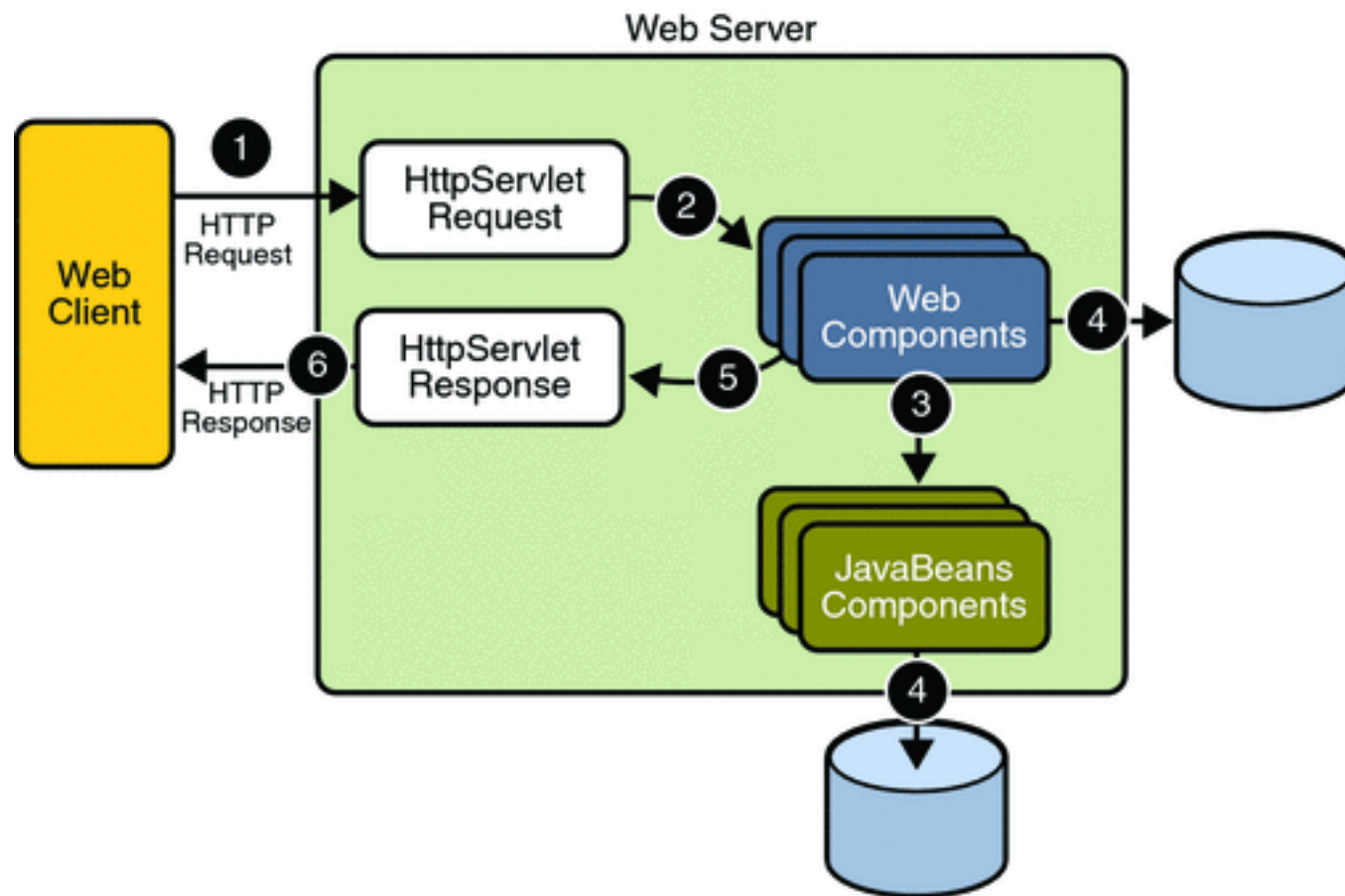
*I am a **JSP** page*

**controller**

*I am a **servlet***

*I can do the work myself or, better,  
**delegate** it to a **service**.*

*I know how to **delegate** work to a JSP*



Let's put it in practice...

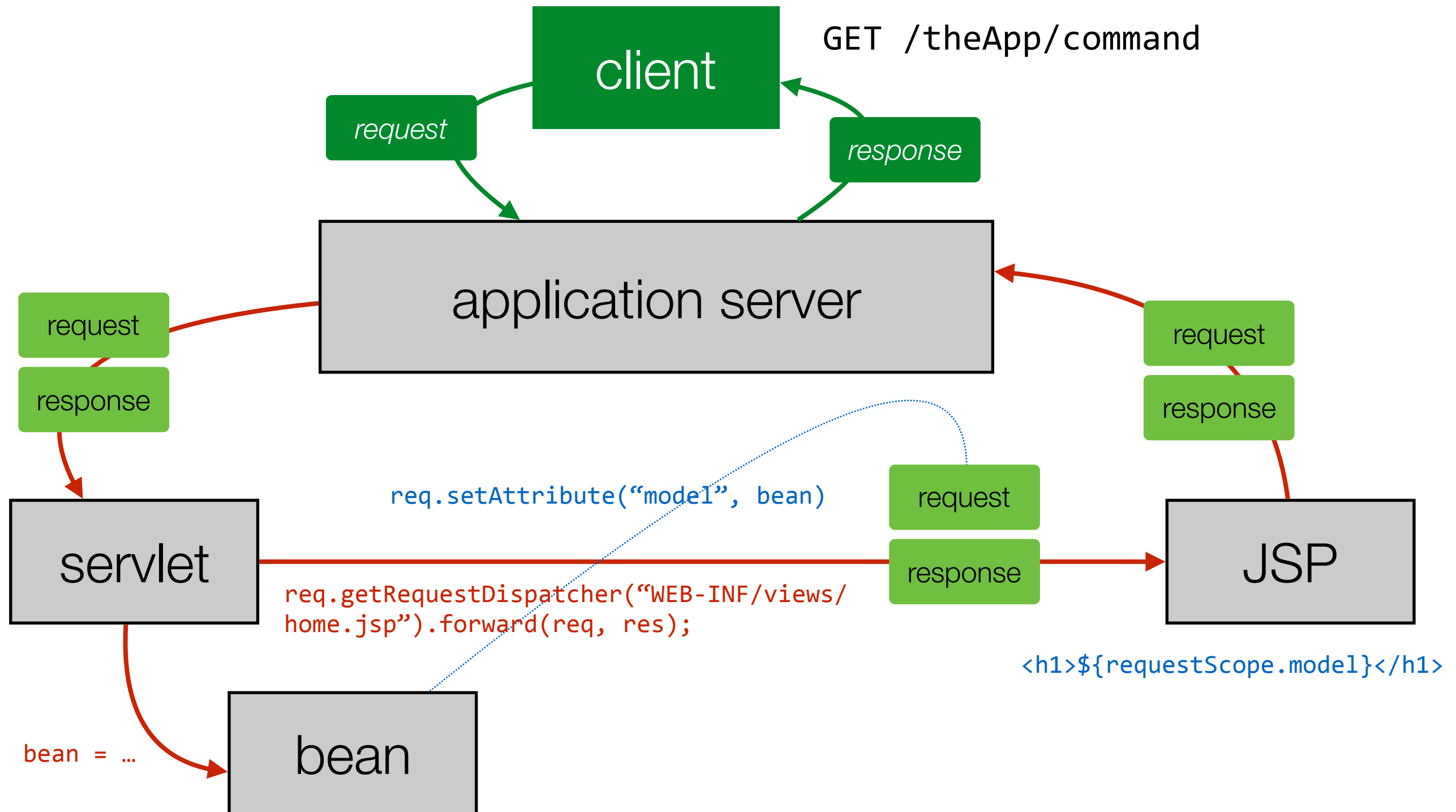
# Let's put it in practice

---

- Create the structure of packages
- Create a model class
- Create a service class
- From the controller (servlet), get a reference to the service and invoke it



How can the MVC design pattern be implemented with Java EE technologies?





What is the proper way to process an HTTP request in Java EE?

```
@WebServlet("/greeting")  
public class GreetingServlet extends HttpServlet {
```

**Servlet <<controller>>**

```
    @Override  
    public void doGet(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException {  
        String message = "A First Java EE Web App";  
        request.setAttribute("message", message);  
        request.getRequestDispatcher("WEB-INF/views/home.jsp").forward(request, response);  
    }  
}
```

```
<div class="page-header">  
    <h1>${requestScope.message}</h1>  
</div>  
  
<div class="page-header">  
    <h1>${message}</h1>  
</div>
```

**JSP <<view>>**

**Object <<model>>**

"A First Java EE Web App"



If you read the Java EE 7 Tutorial, you will read about **JSF**... but where is the **JSP** section???

- **Java Server Pages** (JSP) is a core Java EE technology since the early days. It still is fully supported by application servers (backward compatibility).
- At some stage, **Java Server Faces** (JSF) was added as a complementary presentation tier API. The promise was to offer a component-oriented programming model.
- **JSF has never gained broad adoption** (issues in the first versions of the spec, complexity, alternatives, growing popularity of javascript frameworks combined with REST APIs, etc.).
- Nevertheless, **application server vendors** are still trying to push the standard. They **would like** JSF to be the technology that developers use in the web tier...
- You simply need to do a bit of “documentation archeology” (Java EE 5):
  - <http://docs.oracle.com/javaee/5/tutorial/doc/bnagx.html>
  - <https://jstl.java.net/>



HTTP requests are processed in a **pipeline**. What does it mean?

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud

- HTTP requests sent by clients are received by the application server.
- Remember that several applications may be deployed in the application server. Each application has an associated **context**, which is a URL prefix.
- The application server **inspects the URL**. Based on its prefix, it is responsible to find the first application component that will initiate the processing of the request. Typically, this will be a servlet or a servlet filter.
- In well structured applications, **several components are involved in the processing of each HTTP request**. At the very least, a controller is responsible to invoke the appropriate service (based on URL path and parameters) and a view is responsible to present the data in a particular format. In more complex scenarios, several components (**filters**) may apply some processing in sequence (security checks, logging, compression, etc.).
- The components involved in the processing are organized in a pipeline. The request and the response object are **passed** from one component to the other.

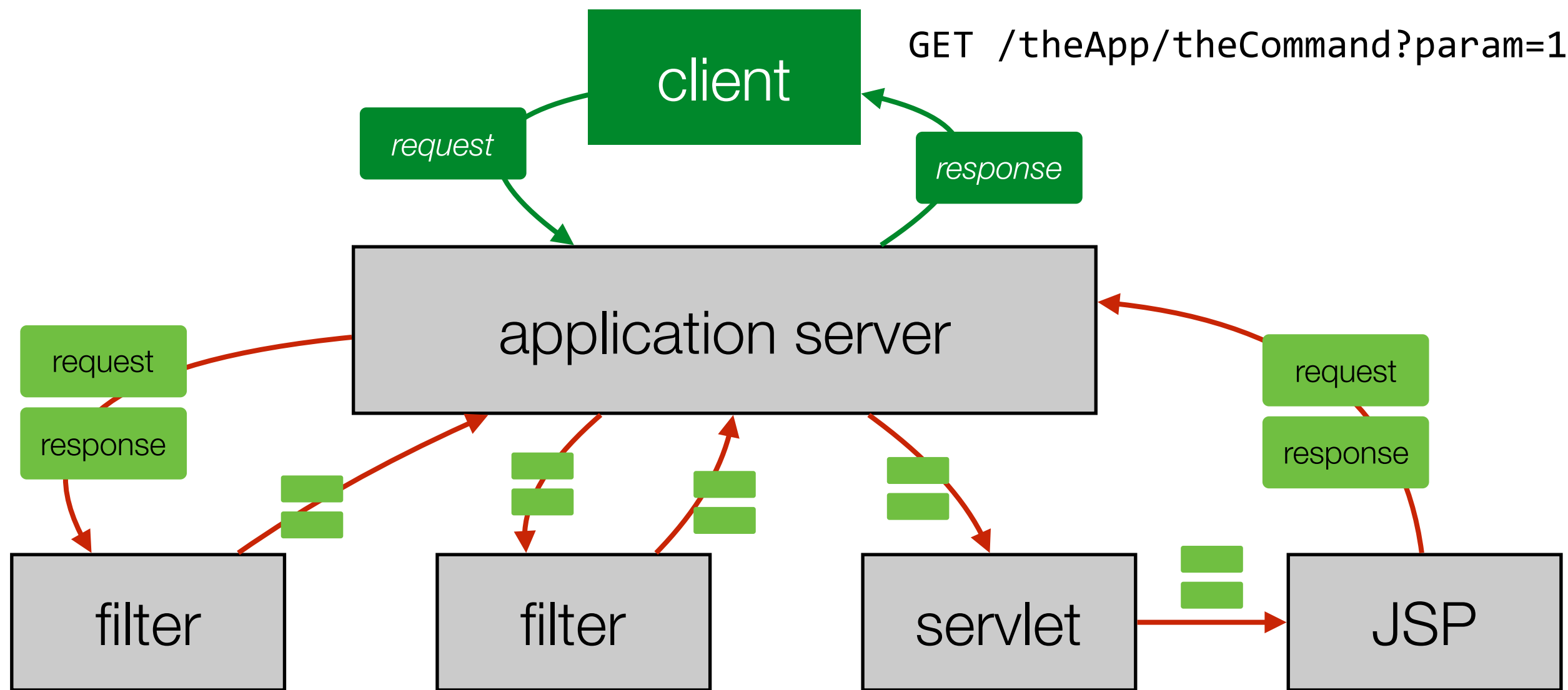




HTTP requests are processed in a **pipeline**. What does it mean?

heig-vd

Haute Ecole d'Ingénierie et de Gestion  
du Canton de Vaud



```
public void doFilter(ServletRequest request,
ServletResponse response, FilterChain chain)
throws IOException, ServletException {
```

```
    chain.doFilter(request, response);
```

```
}
```

```
public void doGet(HttpServletRequest req,
HttpServletResponse res)
throws ServletException, IOException {
```

```
    req.getRequestDispatcher("WEB-INF/views/
home.jsp").forward(req, res);
```

```
}
```

## Interface Filter

```
public interface Filter
```

A filter is an object that performs filtering tasks on either the request to a resource (a servlet or static content), or on the response from a resource, or both.

Filters perform filtering in the `doFilter` method. Every Filter has access to a `FilterConfig` object from which it can obtain its initialization parameters, and a reference to the `ServletContext` which it can use, for example, to load resources needed for filtering tasks.

Filters are configured in the deployment descriptor of a web application.

Examples that have been identified for this design are:

1. Authentication Filters
2. Logging and Auditing Filters
3. Image conversion Filters
4. Data compression Filters
5. Encryption Filters
6. Tokenizing Filters
7. Filters that trigger resource access
8. XSL/T filters
9. Mime-type chain Filter

### Method Summary

#### Methods

Modifier and Type	Method and Description
void	<b>destroy()</b> Called by the web container to indicate to a filter that it is being taken out of service.
void	<b>doFilter(ServletRequest request, ServletResponse response, FilterChain chain)</b> The <code>doFilter</code> method of the Filter is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain.
void	<b>init(FilterConfig filterConfig)</b> Called by the web container to indicate to a filter that it is being placed into service.