M3: HTTP, Servlets, JSP

Table of Content

World Wide Web

History

- Developed by Tim Berners-Lee (1989) at the CERN

  "The World Wide Web (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents." [1]

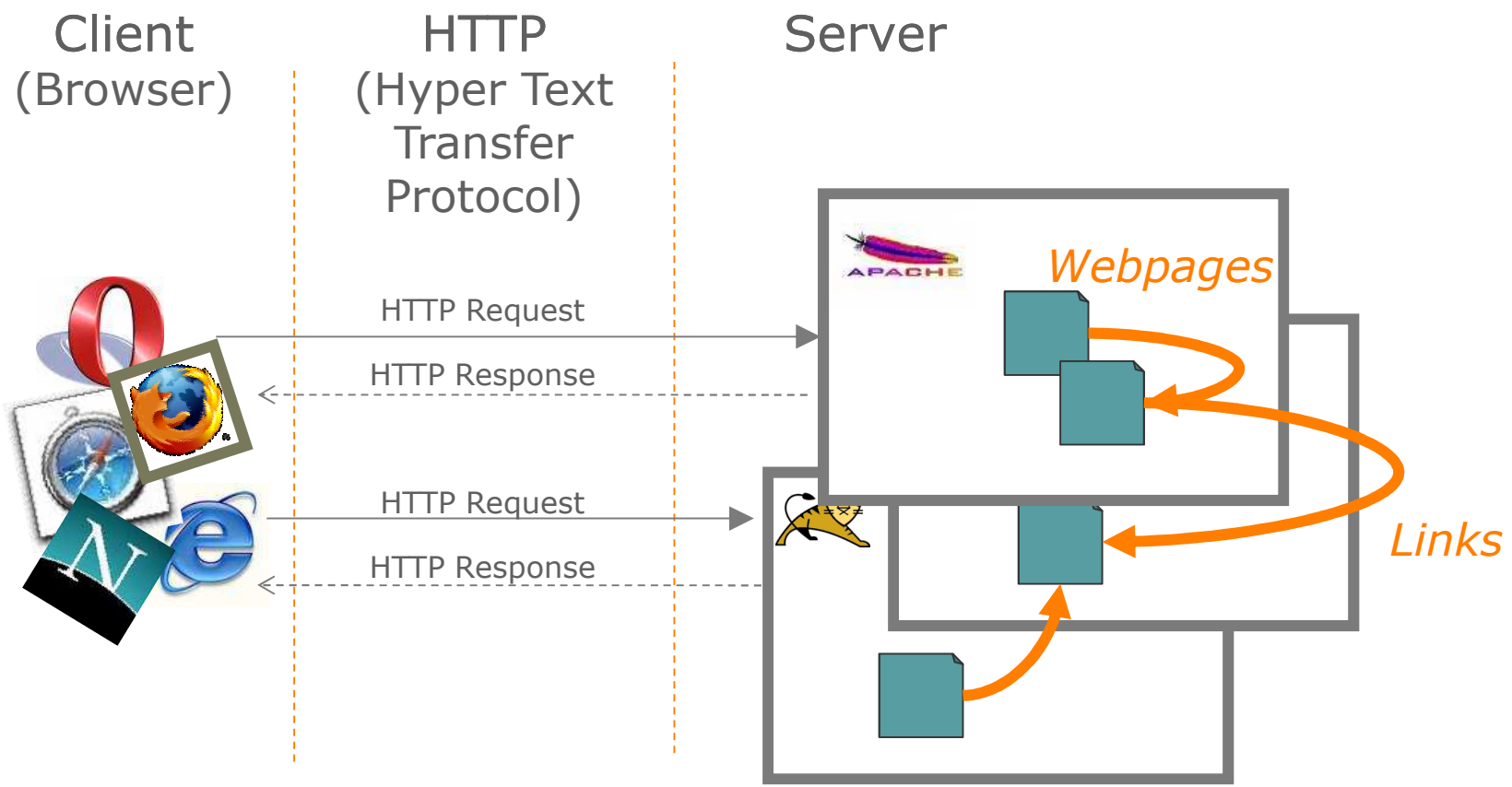
- 3 core standards
  - HTML (Hyper Text Markup Language)
  - URL (Universal Resource Locator)
  - HTTP (Hyper Text Transfer Protocol)

- Success factors
  - Simple publishing language (HTML)
  - Unidirectional links (URL)
  - Free protocol (HTTP)

[1] W3 history, http://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html

HTTP

Architectural Overview



HTTP

Resources and URLs

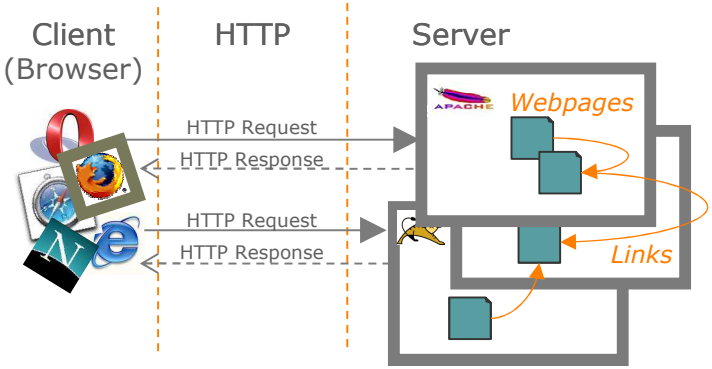
- Resource
  - Abstract concept for the nodes in the hypertext (e.g., HTML files, documents, images)
  - Data types defined by MIME (RFC 2045) (e.g., "text/html", "image/png", "application/xml“)
- Uniform Resource Locator (URL) RFC 1738
  - Subtype of Uniform Resource Identifier (URI)
  - Identification and addressing of a resource
  - HTTP URL scheme:

<scheme>://[<user>[:<password>]@]**<server>**[:<port>]**/**[<path>][?<query>][#<fragment>]

  - Example: http://www.google.com/search?q=ietf+http&hl=de

HTTP

The HTTP Protocol



- IETF Standard RFC 2616
- Builds upon TCP/IP
- Synchronous request-response protocol
  - Client (web browser) sends request
  - Web server replies with appropriate answer
- "Stateless" protocol
  - Each request-response pair is independent
  - No permanent connection between server and browser (allows for a high number of users per server)
- Proxies mediate between browser and server (caching, filtering, etc.)

HTTP

Request Message

- Refers always to a certain **resource** (identified by its URL)
- Has always a certain **type** ("method")
- Can contain **application data** ("body")
  - e.g., the data of a form (POST, PUT)
- Can contain **application metadata**, e.g.:
  - Preferred data type and language (for GET, POST) – Content Negotiation
  - Data type of the body (for POST, PUT)
  - Partial access (e.g., for download in pieces)
  - Versioning information
- Can contain **request metadata**, e.g.:
  - Target host
  - User authentication
  - Cookies

HTTP

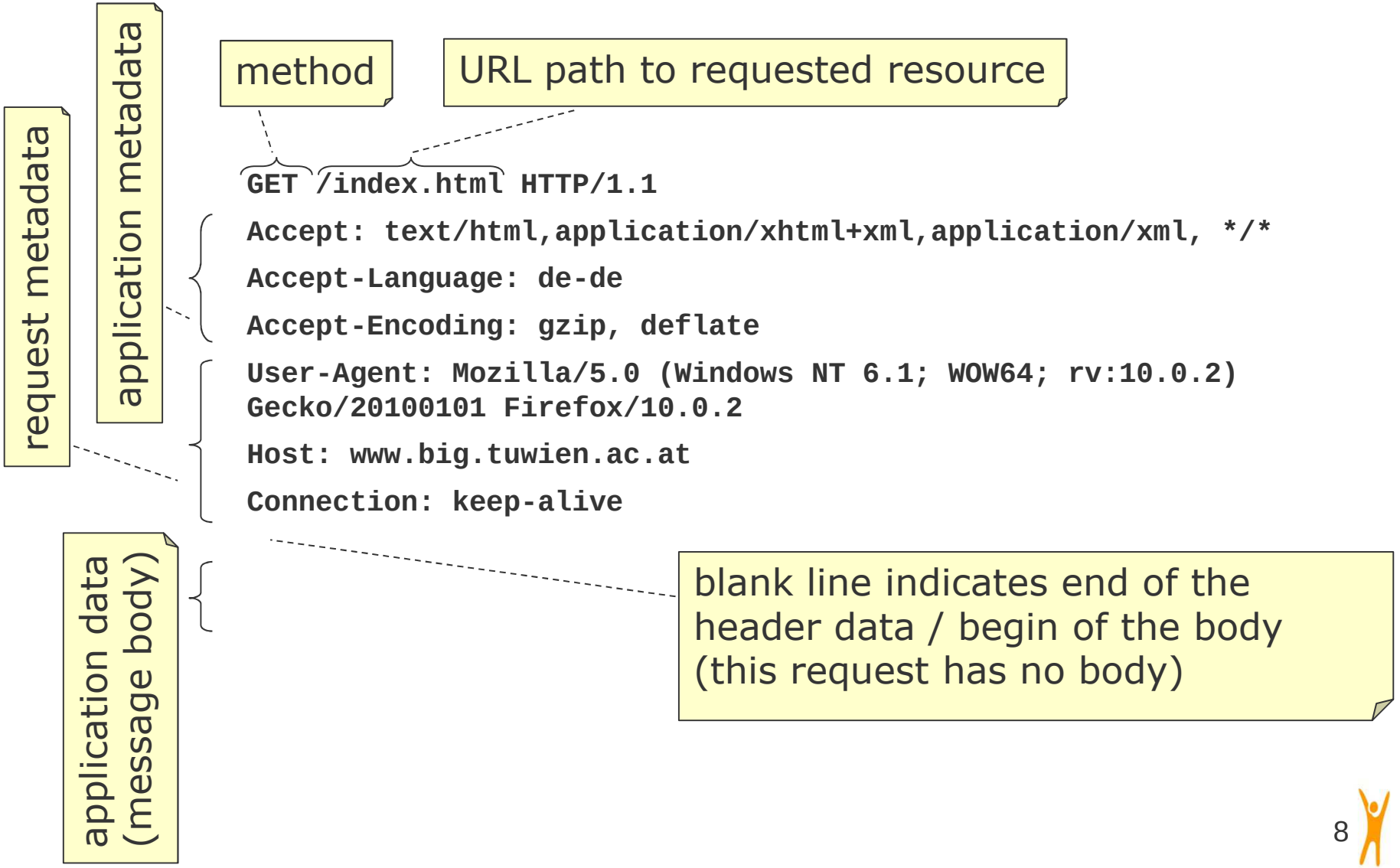
HTTP Request Methods

- Each access to a resource has a certain type ("method")

- **GET**: request a resource
  - Only retrieves data, i.e., is „safe and repeatable“
  - → no changes of the resource are possible
- **POST**: submit data to a resource
  - Data is included in body of the request
  - May result in creation of new resource or update of existing resource
- PUT: update/create a resource
- DELETE: delete a resource
- OPTIONS, TRACE, HEAD, CONNECT, PATCH: access to the metadata of the servers, the Internet connection, the resource, etc.

HTTP

Example of a HTTP Request Message



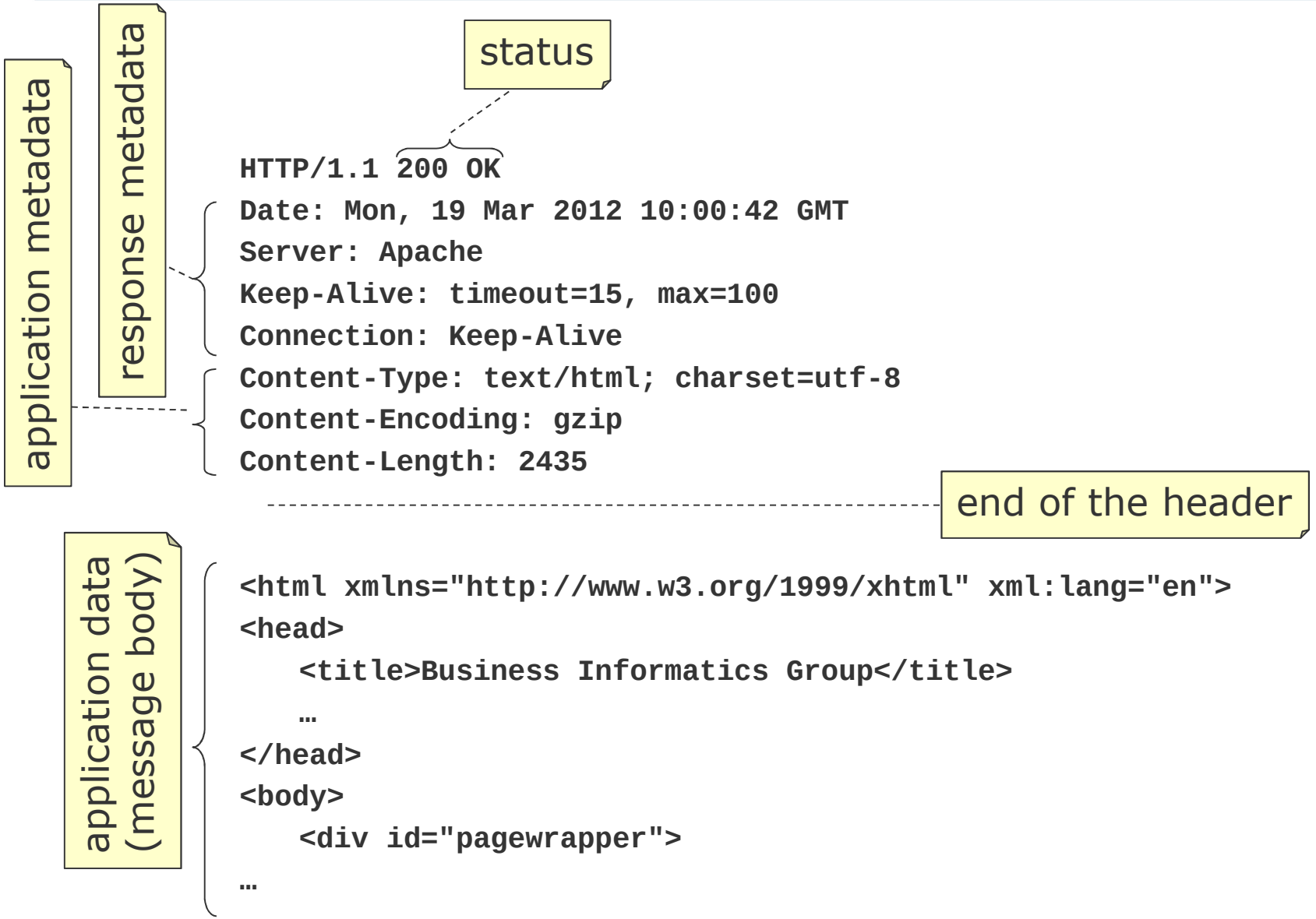
HTTP

Response Message

- Always follows a request message
- Contains a **status code**
  (success, redirection, client error, server error, etc.)
- Can contain **application data** („body“)
- Can contain **application metadata**, e.g.:
  - Data type and encoding of the application data
  - Caching possibilities and expiring date
  - Current URL of a transferred resource (for GET)
- Can contain **response metadata**, e.g.:
  - Server
  - Date
  - State of the TCP-connection

HTTP

Example of a Response Message



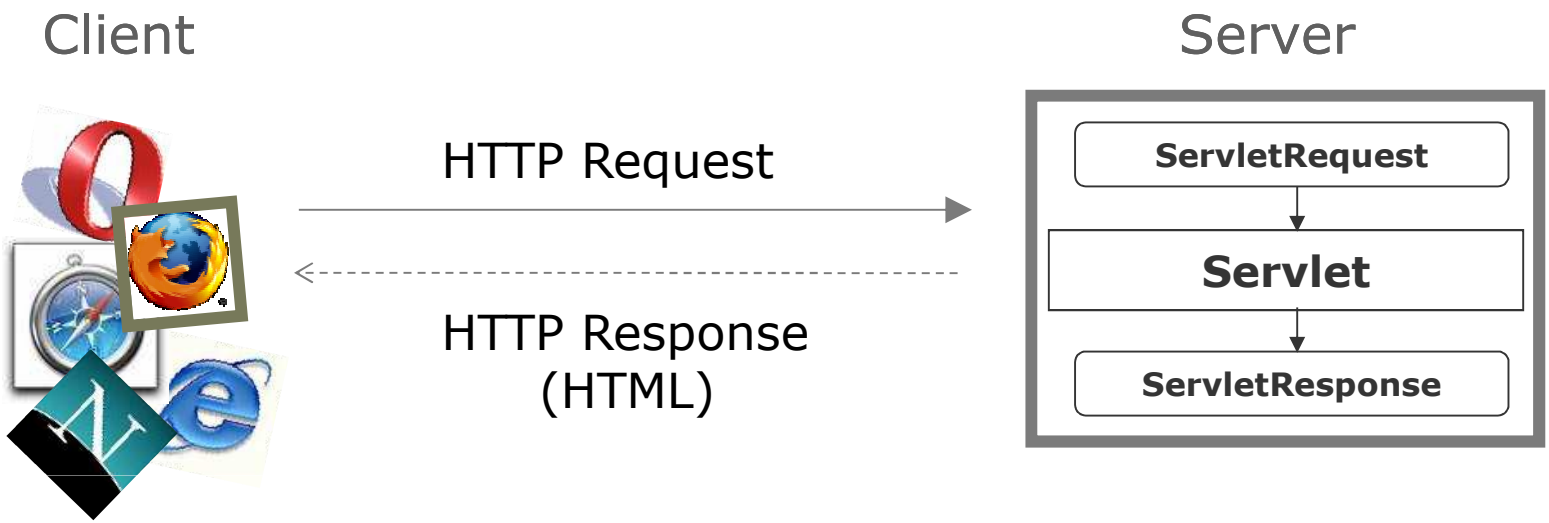
Servlets

Creating Dynamic Content

- Execute programs on the server side for dynamically generating HTML documents
- Common Gateway Interface (CGI)
  - Extension mechanism to web servers
  - Problems with performance (solution: FastCGI)
- 1996 Server-side applets by Netscape
- 1996 Resources as server-side Java modules
- 1997 Servlet technology by JavaSoft
- Many server-side extensions (e.g., ASP by Microsoft)

Servlets

Architecture



Greeting.html

```
<html>
    <body>
        <h1>What's your name?</h1>
        <form action="GreetingServlet" method="GET">
            <input type="text" name="userName" value="" /><br/>
            <input type="submit" value="submit" name="submit" />
        </form>
    </body>
</html>
```

Servlets

Components of a Servlet-based Web Application

- A web application consists of
  - **Static resources** (e.g., HTML files, images)
  - **Dynamic generated resources** (e.g., realized by servlets)
  - **Further program elements** (e.g., Java classes for data structures and helper functions)
  - **Deployment descriptor**, which describes the configuration of the web application (e.g., which servlets exist, init parameters, access rights)
- All files of a web application can be assembled in **one archive: *.war file** (web application archive)

Servlets

Structure of a Web Application

RegistrationMVC.war

- WEB-INF
  - lib
    - AbsoluteLayout.jar
  - classes
    - controller
      - LoginServlet.class
    - model
      - User.class
      - UserPool.class
  - web.xml
- META-INF
- login.html
- loginfail.html
- registration.html
- userdata.jsp
- userdatafail.html
- userpage.jsp

- **Document root directory**
  - Contains HTML-documents, images, JSPs, subdirectories, etc.
  - Accessible to clients
  - Example: http://localhost:8084/RegistrationMVC/login.html
- Special subdirectory **WEB-INF**
  - Content is not directly accessible to clients
  - Contains the **web.xml** file (web application deployment descriptor)
  - *.class files in the subdirectory **classes**
  - jar-archives in the subdirectory **lib** e.g., JDBC-drivers

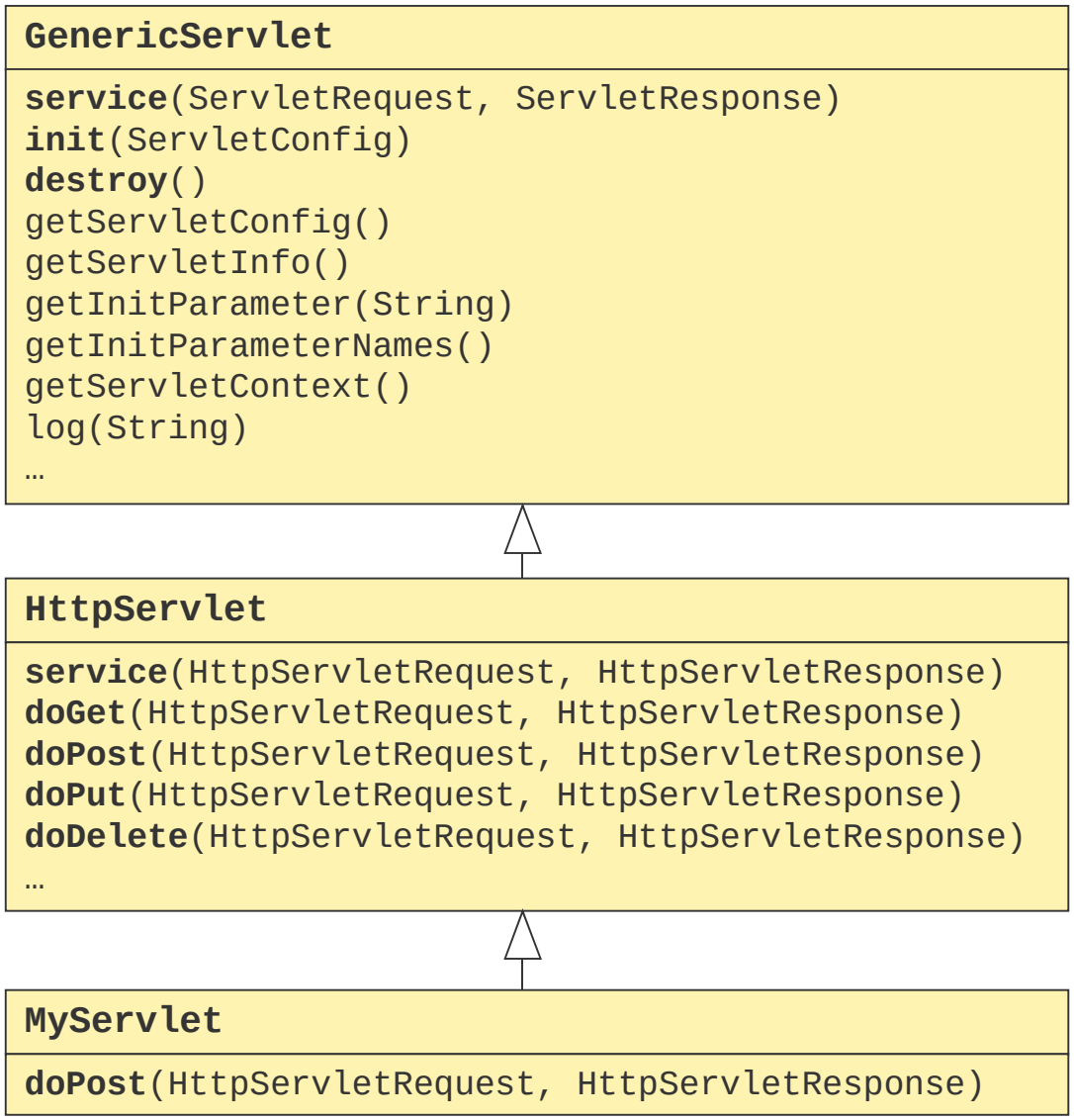
Servlets

Deployment of Servlets

- Servlets are deployed as *.war files
- A web application is deployed and executed within a **Servlet Container**
  - Main function of the container is to load, initialize and execute servlets (i.e., it is responsible for managing the lifecycle of servlets)
  - Often part of a web server or an application server
  - Reference implementation: Apache Tomcat (http://tomcat.apache.org/)
- Most servlet containers have a special directory to copy the *.war files into for auto deployment
  - Directory structure depends on the servlet container
- Apache Tomcat:
  - Default installation in the subdirectory **webapps**
  - The basic URL of the web application corresponds to the directory name
  - Special configuration of a web application can be done in the file conf/server.xml (e.g., basis URL, log file, user database)

Servlets

The Servlet API



javax.servlet.GenericServlet
Abstract class that defines a generic, protocol-independent servlet.

javax.servlet.http.HttpServlet
Abstract class to be subclassed to create an HTTP servlet. Implements the `service()` method. A subclass must override at least one of the `doXXX()` methods.

MySerlvet
Own servlet implementation for handling HTTP post requests.

Servlets

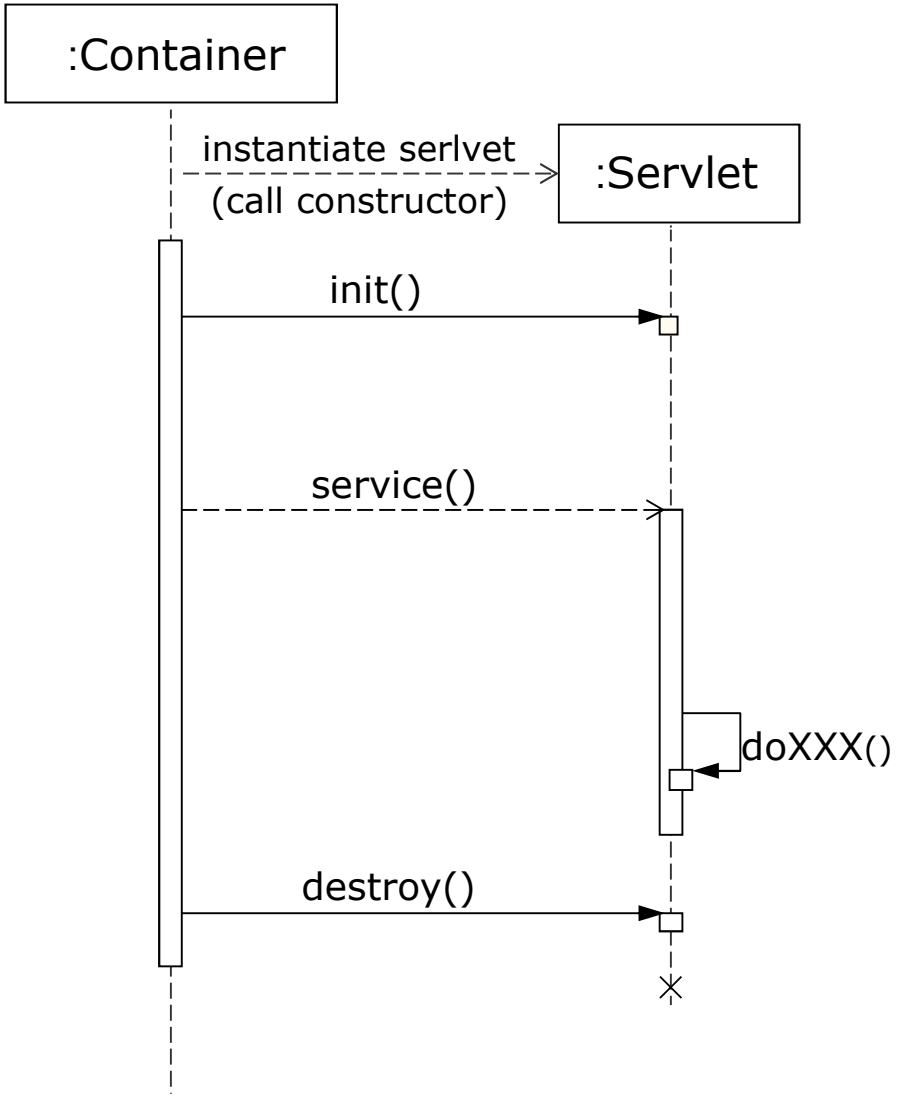
A Servlet‘s Lifecycle

Methods which are overwritten can only be called by the server (*Hollywood principle: "Don´t call us, we will call you"*):

```
void init(ServletConfig config)
void doGet(HttpServletRequest req, HttpServletResponse res)
void doPost(HttpServletRequest req, HttpServletResponse res)
void doPut(HttpServletRequest req, HttpServletResponse res)
void doDelete(HttpServletRequest req, HttpServletResponse res)
void doHead(HttpServletRequest req, HttpServletResponse res)
void doOptions(HttpServletRequest req, HttpServletResponse res)
void doTrace(HttpServletRequest req, HttpServletResponse res)
void destroy()
```


Servlets

A Servlet‘s Lifecycle



`init()` is called after loading and instantiating the servlet class

`service()` is called when a client sends a request and it dispatches the HTTP request to the handler methods for each HTTP request type (the **`doXXX()`** methods)

`doXXX()` is called by the **`service()`** method depending on the HTTP request type to handle the request

`destroy()` is called if a servlet is being taken out of service

Servlets

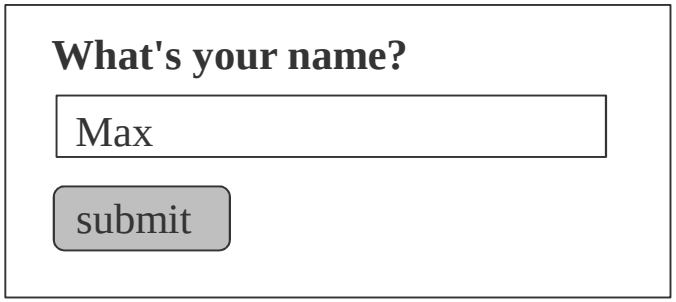
Example

Servlet Example:

Greeting.html

```
<html>
    <body>
        <h1>What's your name?</h1>
        <form action="GreetingServlet" method="GET">
            <input type="text" name="userName" value="" /><br/>
            <input type="submit" value="submit" name="submit" />
        </form>
    </body>
</html>
```

http://localhost:8084/GreetingServlet/



TUWEL course resources: GreetingServlet

Servlets

Example

GreetingServlet.java

```
public class GreetingServlet extends HttpServlet {

  protected void doGet(HttpServletRequest request,
                       HttpServletResponse response)
            throws ServletException, IOException {
     response.setContentType("text/html;charset=UTF-8");
     PrintWriter out = response.getWriter();
     out.println("<html>");
     out.println("<body>");
     out.println("<h1>Hello " + request.getParameter("userName") + "</h1>");
     out.println("</body>");
     out.println("</html>");
    }
}
```

http://localhost:8084/GreetingServlet/

TUWEL course resources: GreetingServlet

Servlets

Example

web.xml (Deployment Descriptor)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" …>
  <description>This servlet greets its users</description>
  <display-name>Greeting</display-name>
  <servlet>
    <servlet-name>Greeting</servlet-name>
    <servlet-class>GreetingServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Greeting</servlet-name>
    <url-pattern>/GreetingServlet</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>Greeting.html</welcome-file>
  </welcome-file-list>
</web-app>
```

Defines an internal name for the servlet which is used for the servlet mapping

Specifies the full qualified name of the servlet class

Defines the URL path for accessing the servlet with the name "Greeting"

Specifies the start page of the web application

TUWEL course resources: GreetingServlet

Servlets

Request Processing

- **`HTTPServletRequest`** object can be used to access header and body data of the HTTP request
- Access to HTML **form data**:
  - `String param = request.getParameter(name);`
  - `String[] params = request.getParameterValues(name);`
  - `Enumeration paramnames = request.getParameterNames();`
- Session Tracking:
  - `String sid = request.getRequestedSessionId();`
  - `HttpSession s = request.getSession();`
- Further header data:
  - `String host = request.getRemoteHost();`
  - `String user = request.getRemoteUser();`
  - …

Servlets

Request Processing

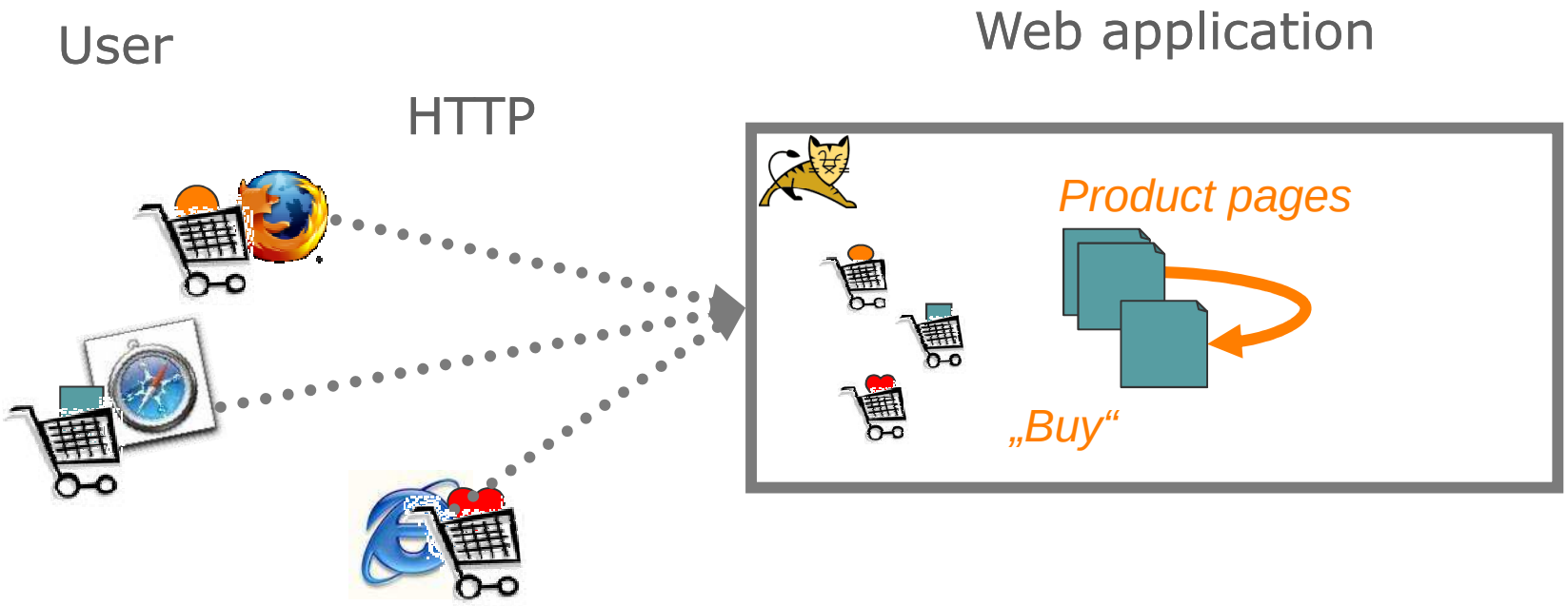
- A **`HttpServletResponse`** object is created to respond to the request
- Write header data
  - `response.setContentType("text/html;charset=ISO-8859-1");`
  - Further methods:
    `response.setContentLength(len),`
    `response.addCookie(cookie), …`
- Write body data
  - Text data: `PrintWriter out = response.getWriter();`
  - Binary data: `ServletOutputStream s = response.getOutputStream();`
  - Generate HTML by println instructions

```
PrintWriter out = response.getWriter();
out.println("<html><body>");
out.println("Hello " + request.getParameter("userName"));
out.println("</body></html>");
```


Servlets – Sessions

Session Tracking

- Web applications have to handle multiple users at the same time
- Often it is necessary to have user specific state information
- Example:



Servlets – Sessions

Session Tracking

- Problem: HTTP is stateless
  - There are no "sessions", i.e., each request is handled individually
  - Navigation within context-dependent documents (like forms) or the split up of search results are difficult
  - Functionality like a shopping cart demand for storing state information
- Different variants to realize sessions:
  - HTTP authentication of the user
  - Hidden fields in forms
  - Encoding of state information in the URL
  - Cookies
  - **Java Servlet Session Tracking API**

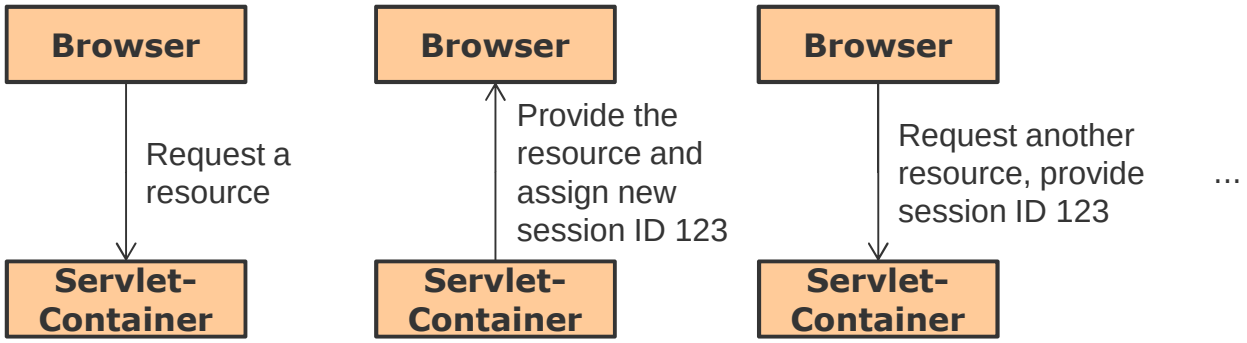
Servlets – Sessions

The Session Tracking API (1/2)

- Each user of a server has exactly one session object
- This can be requested by

```
HttpSession session = request.getSession();
```

  - If the request contains a session ID, the corresponding session object is returned to the client
  - Otherwise a new session object is created, and a new session ID is sent back to the client
  - The client can save this session ID and send the ID back to the server in further requests
- Used mechanism (usage of cookies or URL encoding) for exchanging the session ID depends on server and client



Servlets – Sessions

The Session Tracking API (2/2)

- The session objects can contain session-specific data like the user name or temporary data
- Access methods:
  - `session.setAttribute(attrName, attrValue);`
  - `Object o = session.getAttribute(attrName);`
  - `Enumeration names = session.getAttributeNames();`
- A session has only a limited period of validity
  - Set in the deployment descriptor (default: 30 minutes)
  - Can be made explicitly invalid, e.g., at the logout, by `session.invalidate();`
  - On a request with an invalid session ID, a new one is created

TUWEL course resources: RegistrationServlet

Servlets – Security

Security Risks

- Web applications are usually publicly accessible
- Server-side risks by "security holes" in web applications like
  - Lacking access control, which allows for public access to confidential data
  - Insecure data transfer, which allows for eavesdropping
  - Processing of not validated input data, which allows for the introduction of script code in form input or manipulated URL
  - Errors in the applications may result in outputting security relevant configuration information
  - …
- Important issues: Authentication, Authorization, Confidentiality, Data integrity

Servlets – Security

Security Techniques

- Encryption of the data transfer by SSL/TLS (HTTPS)
- Authentication
  - By username and password via HTTP-authentication or HTML-forms
  - Via SSL server certificates and client certificates
  - Demands for the administration of the user data and the certificates
- Access control
  - By configuration of the network (LAN, Firewall)
  - By the web application itself (**programmatic security**)
    e.g., `if(request.getRemoteUser() != "root")`
  - By configuration of the web server (**declarative security**)
    Administration of access rights necessary, flexible by the usage of roles,
    e.g., `request.isUserInRole("admin")`

Servlets – Security

Example: Declarative Security

```
<web-app>
  …
 <security-constraint>
    <web-resource-collection>
      <web-resource-name>Admin-Pages</web-resource-name>
      <url-pattern>/admin/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
       <role-name>administrator</role-name>
    </auth-constraint>
    <user-data-constraint>
       <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
 </security-constraint>
 <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Administration</realm-name>
  </login-config>
 <security-role><role-name>administrator</role-name></security-role>
```

Security guideline for all URLs in the path `http://<host>/<webapp-path>/admin/`

Only users with the role "administrator" are authorized

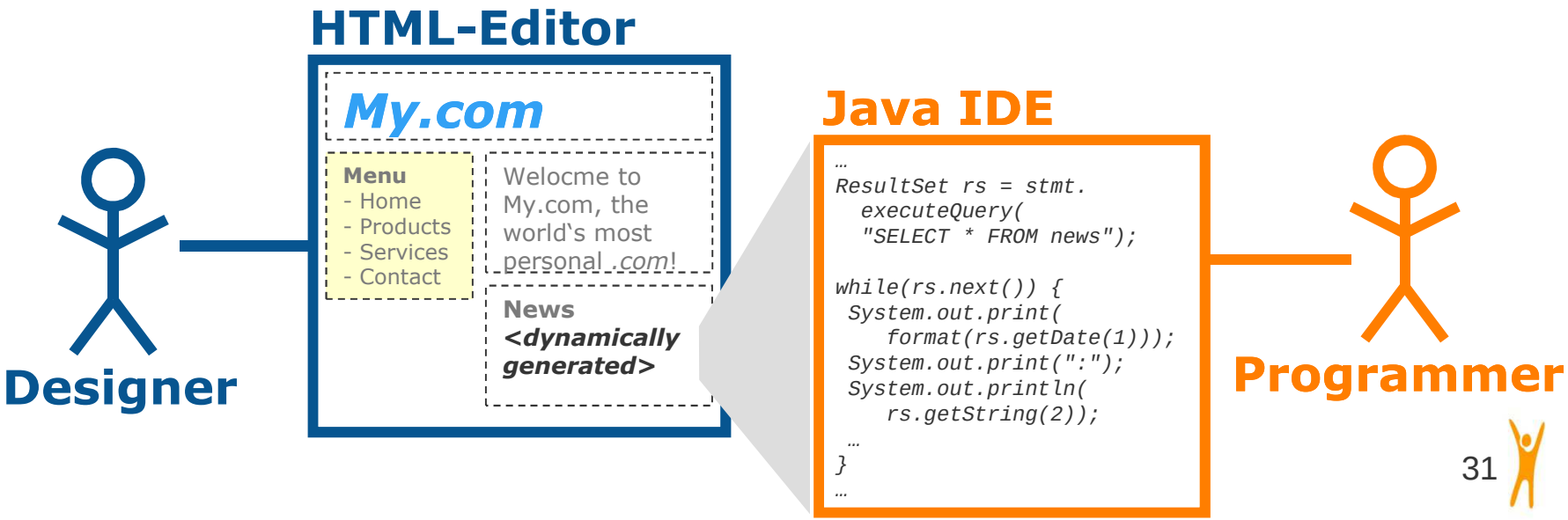
Data does not need to be encrypted (other options: INTEGRAL, CONFIDENTIAL)

Transfer of user data via HTTP-authentication (base64 encoded)
(other options: DIGEST, CLIENT-CERT, FORM)

Java Server Pages

Overview

- Problems with Servlets:
  - Unclear and complex (`out.print("<html>");`)
  - No direct use of HTML design tools possible
  - No support of parallel development of code and design
  - Maintenance of servlet pages demands a Java programmer
- Desired: Separation of tasks

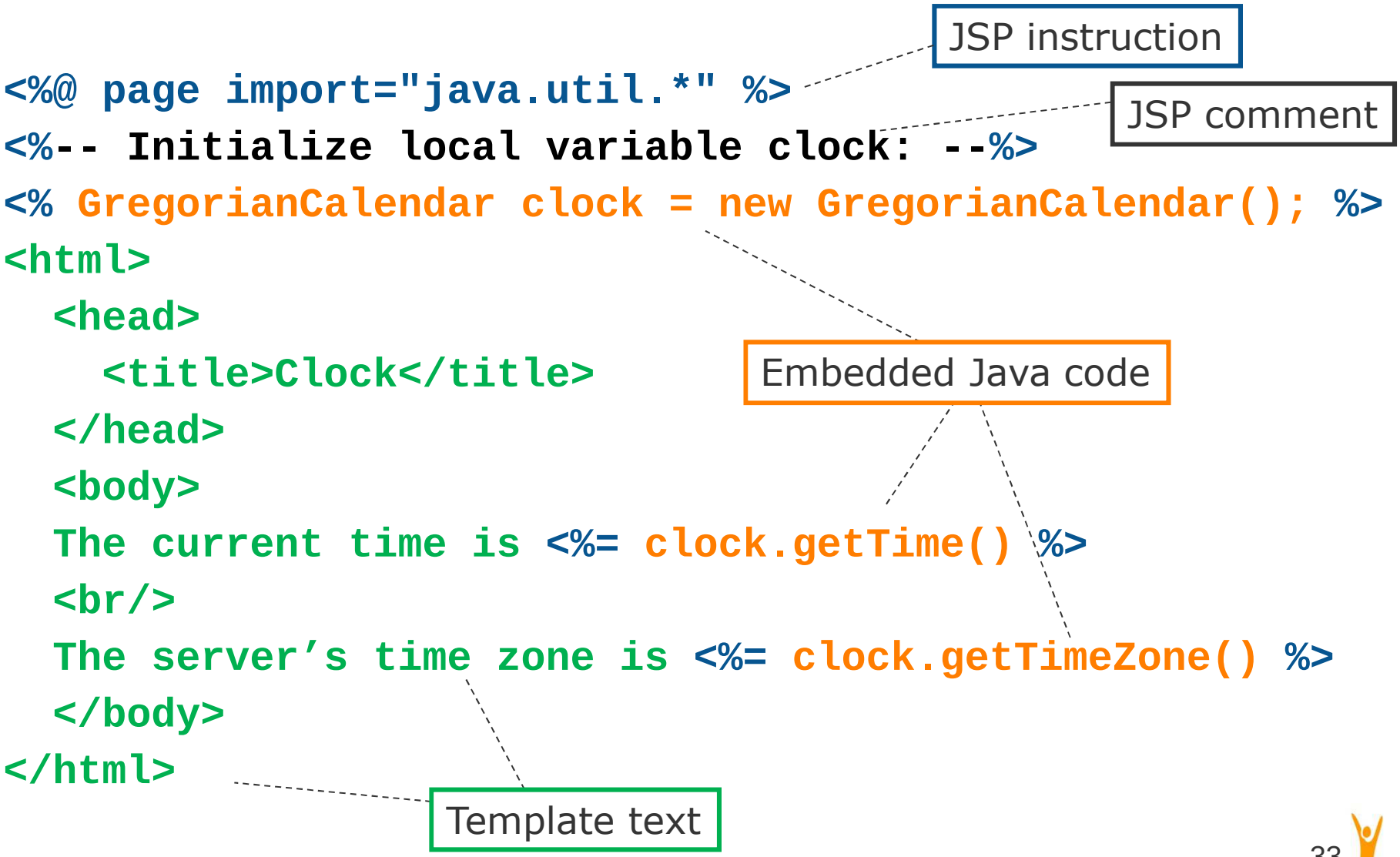


Java Server Pages

- Server-side script language
  - HTML with embedded scripts or software components
  - Preferable similar to HTML to reuse HTML tools
  - As simple as possible to realize dynamic page parts
  - Examples: Server-side JavaScript, ASP, JSP, PHP, etc.
- Java Server Pages (JSP)
  - Java as script language
  - Extensible by Java Beans and user-defined tags
  - Transparent translation to servlets

Java Server Pages

Beispiel: "clock.jsp"



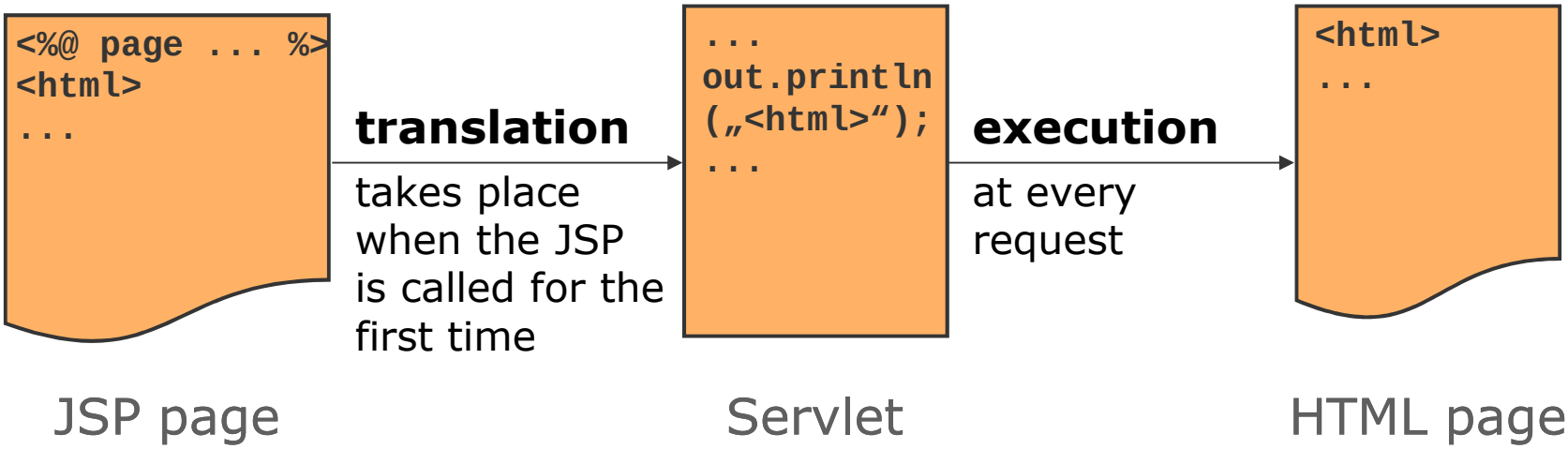

```
<%@ page import="java.util.*" %>
<%-- Initialize local variable clock: --%>
<% GregorianCalendar clock = new GregorianCalendar(); %>
<html>
  <head>
    <title>Clock</title>
  </head>
  <body>
  The current time is <%= clock.getTime() %>
  <br/>
  The server's time zone is <%= clock.getTimeZone() %>
  </body>
</html>
```

TUWEL course resources: JSPExamples (clock.jsp)

Java Server Pages

Life Cycle

- Transparent translation to Servlets
  - Changes in the JSP pages are performed automatically
  - No manual compilation is necessary



Java Server Pages

Template Text

- Template Text (= "standard" HTML, XML, or any text)
  - Not interpreted by the JSP engine
  - Directly passed to the client
  - Exception: to display "<%", "<\%" has to be used

Java Server Pages

Scripting Elements (1/4)

- **Expressions**

  e.g., `current time: <%= new java.util.Date() %>`

- **Predefined variables**
  `request`, `response`, `out`, `session`, `application`, `config`, …
  (→ Servlet API) can be used

  e.g., `Your hostname is <%= request.getRemoteHost() %>`

- **Scriptlets** represent complex expressions

  e.g.,
  ```
  <% String queryData = request.getQueryString();
  out.println("Attached GET data: " + queryData); %>
  ```

- **Declarations**

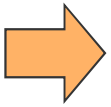
  e.g.,
  ```
  <%! private int accessCount = 0; %>
  Accesses to pages since reboot: <%= ++accessCount %>
  ```


Java Server Pages

Scripting Elements (2/4)

- Scripting elements become part of the servlet generated from the JSP
  - The **`service()`** method is composed from template text, expressions and scriptlets in the given order:

```
<% if( a > b ) %>
  A: <%= a %>

<% else %>
  B: <%= b %>
```



```
if(a > b)
  out.print("A:");
  out.print(a);
else
  out.print("B:");
  out.print(b);
```

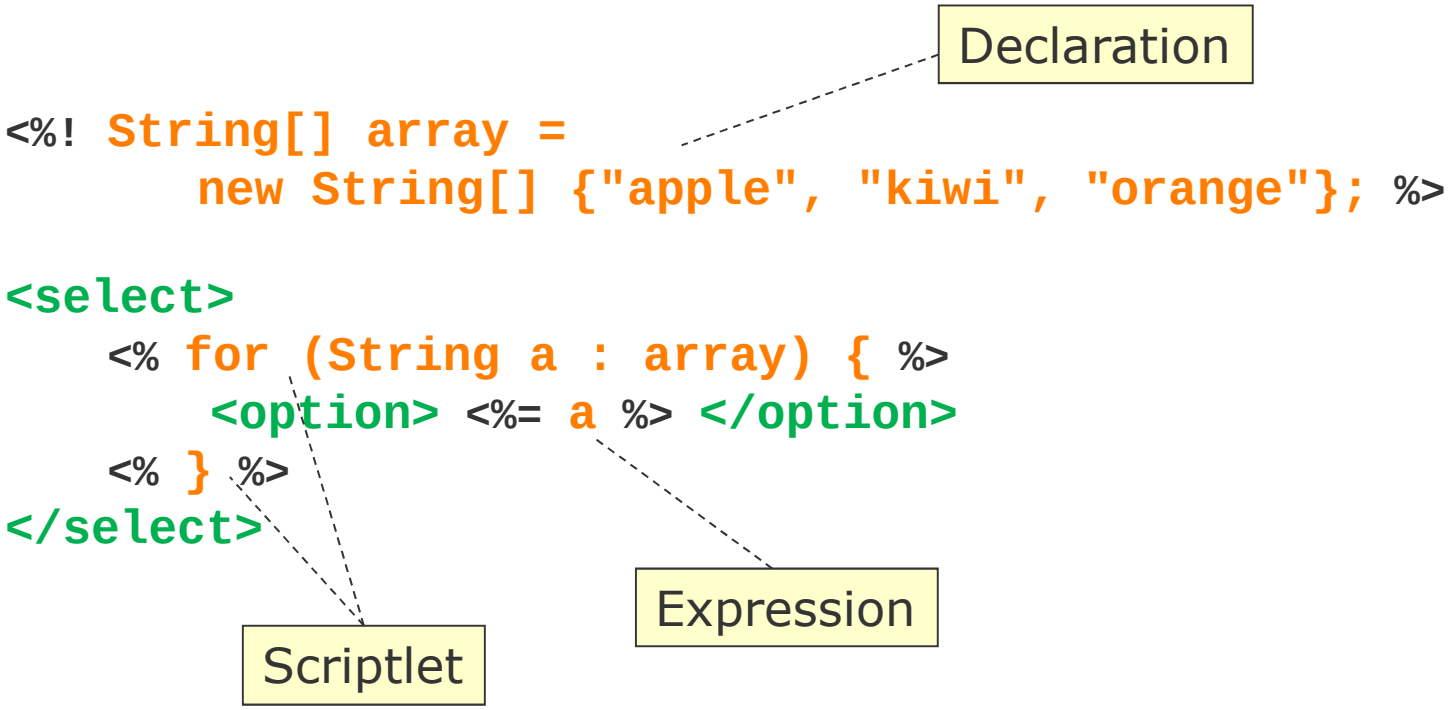
Error!

  - Declarations (variables, methods) become part of the servlet class, the order of declarations in the template text is not relevant

Java Server Pages

Scripting Elements (3/4)

- Scripting elements can be used to generate HTML-code dynamically
- Example:



TUWEL course resources: JSPExamples (fruits.jsp)

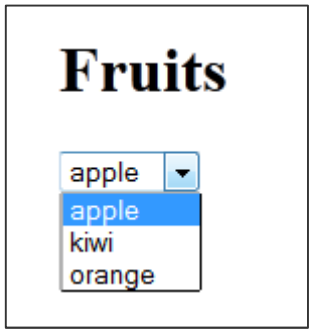
Java Server Pages

Scripting Elements (4/4)

- Generated HTML code:

```
<select>
        <option> apple </option>
        <option> kiwi </option>
        <option> orange </option>
</select>
```

- Rendered in the browser:



TUWEL course resources: JSPExamples (fruits.jsp)

Java Server Pages

Directives

- Directives control the translation to servlet code
- Basic types: page, tag library, and include:
  - `<%@ page import="java.util.*" %>`
  - `<%@ page contentType="text/html" %>`
  - `<%@ page session="true" %>`
  - `<%@ page errorPage="error.jsp" %>`
  - …
  - `<%@ taglib uri="http://java.sun.com/jsp/jstl/core“ prefix="c" %>`
  - …
  - `<%@ include file="footer.html" %>`

Default setting

Integration during translation time

Java Server Pages

Actions

- Actions use constructs in XML syntax to control the execution of JSP
  - Dynamically include page parts

```
<jsp:include page="menu.jsp" flush="true" />
```

  - Redirect request to another JSP or servlet

```
<jsp:forward page="another.jsp" />
```

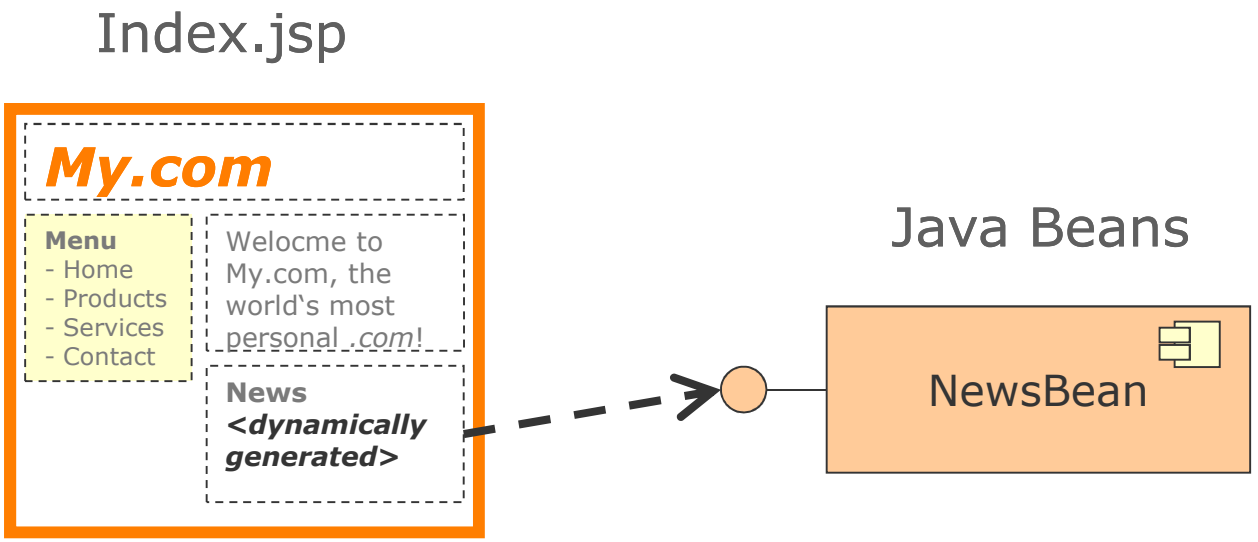
  - Actions for usage of Java Beans

```
<jsp:useBean id="name" class="package.class" />

<jsp:setProperty name="name" property="someProperty“
      value="value" />

<jsp:getProperty name="name" property="someProperty" />
```


Java Server Pages

Using Java Beans

- Java Code within JSP is hard to maintain
- Even simple changes demand for special knowledge
- Software components with defined interfaces (→ Java Beans) allow for a better separation



Java Server Pages

Declaration of Bean Variables

- Declaration of a Bean variable, e.g.

```
<jsp:useBean id="demoBean" class="big.DemoBean"
                scope="application" />
```

- Scope
  - `page` – temporary within a page (local variable of Servlet)
  - `request` – temporary within an HTTP-Request (HttpServletRequest)
  - `session` – within a user session (HttpSession)
  - `application` – global for the whole web application (ServletContext)
- Automatic construction: New Bean object is created on first use

Java Server Pages

Access to Bean Variables

- Access to local Bean variable in script code, e.g.:

```
<%= demoBean.getDescription() %>…
```

- Or access in XML notation with special actions, e.g.:

```
<jsp:getProperty name="demoBean"
                 property="description" />

<jsp:setProperty name="demoBean"
                 property="description"
                 value="I‘m a Bean." />
```

Actions to transfer request parameters, e.g.:

```
<jsp:setProperty name="demoBean"
                 property="name" param="firstname"/>

<jsp:setProperty name="demoBean" property="*" />
```

All properties are set where the name matches a form field

Java Server Pages

Example Java Beans (1/3)

UserData.java:

```
package myPackage;

public class UserData {
  String username, email;
  int age;

  public void setUsername( String value ) { username = value; }
  public void setEmail( String value ) { email = value; }
  public void setAge( int value ) { age = value;   }

  public String getUsername() { return username; }
  public String getEmail() { return email; }
  public int getAge() { return age; }
}
```

UserData.html

```
<html>
 <body>
   <form method="post" action="SaveName.jsp">
     <label for="name">What's your name?</label>
     <input id="name" type="text" name="username" size="20"/>
     <input type="submit"/>
   </form>
  </body>
</html>
```


Java Server Pages

Example Java Beans (2/3)

SaveName.jsp

```
<jsp:useBean id="user" class="myPackage.UserData"
        scope="session"/>
<jsp:setProperty name="user" property="*"/>
<html>
  <body>
    <a href="NextPage.jsp">Continue</a>
  </body>
</html>
```

NextPage.jsp

```
<jsp:useBean id="user" class="myPackage.UserData"
                scope="session"/>
<html>
  <body>
    You entered<br/>
    Name: <%= user.getUsername() %><br/>
    Email: <%= user.getEmail() %><br/>
    Age: <%= user.getAge() %><br/>
  </body>
</html>
```


Java Server Pages

Example Java Beans (3/3)

UserData.html

What's your name? Max

Send

SaveName.jsp

Continue

NextPage.jsp

You entered
Name: Max
Email: null
Age: 0

Java Server Pages

Pro and Contra

- Pro
  - Simple to learn
  - Allows the use of HTML design tools
- Contra
  - Scriptlets let the code become hard to maintain
  - Mixture of control and presentation code

► Combine Servlets and JSP

MVC Pattern

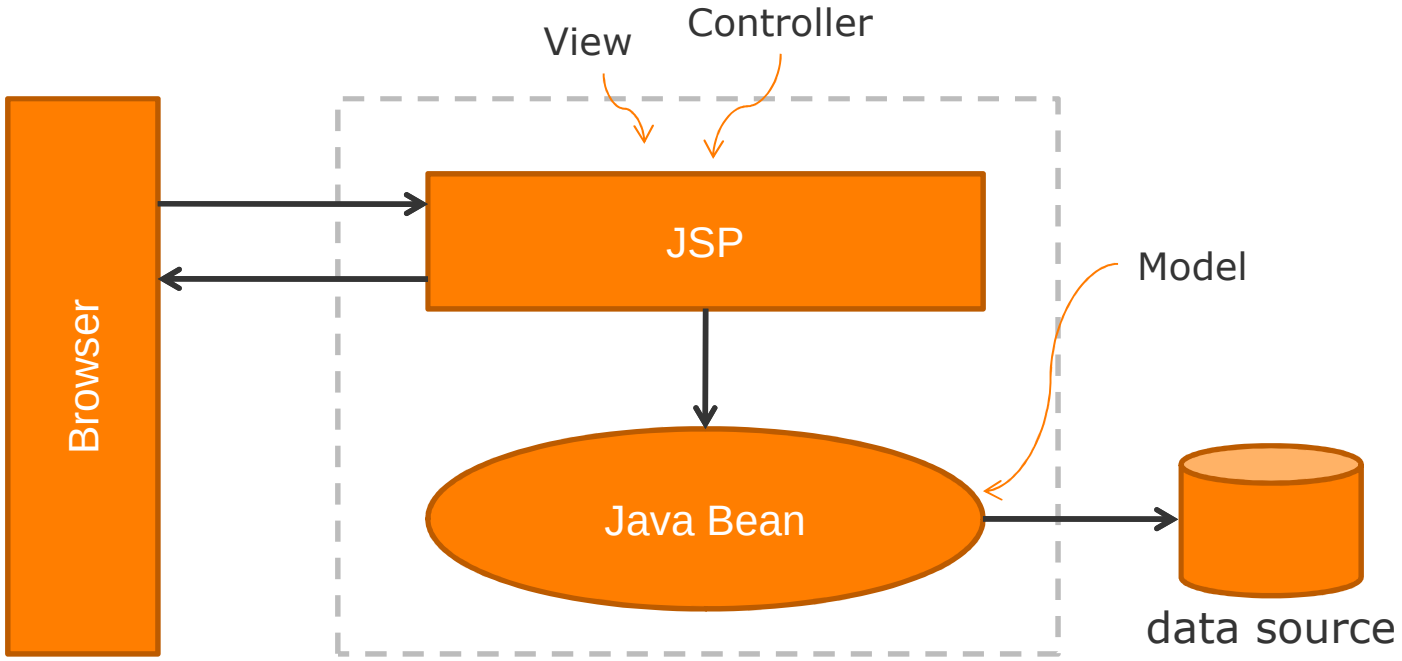
Model View Controller

- MVC stands for **M**odel – **V**iew – **C**ontroller
- Decouples data from its presentation

- Model
  - Contains state and/or data (and perhaps application logic)
- View
  - Renders the model's data and/or state
  - Requests for updates (by interactions)
  - Perhaps only a subset of the model's information is needed to view
- Controller
  - Interprets user input
  - Maps the input to the model

Realize MVC with JSP

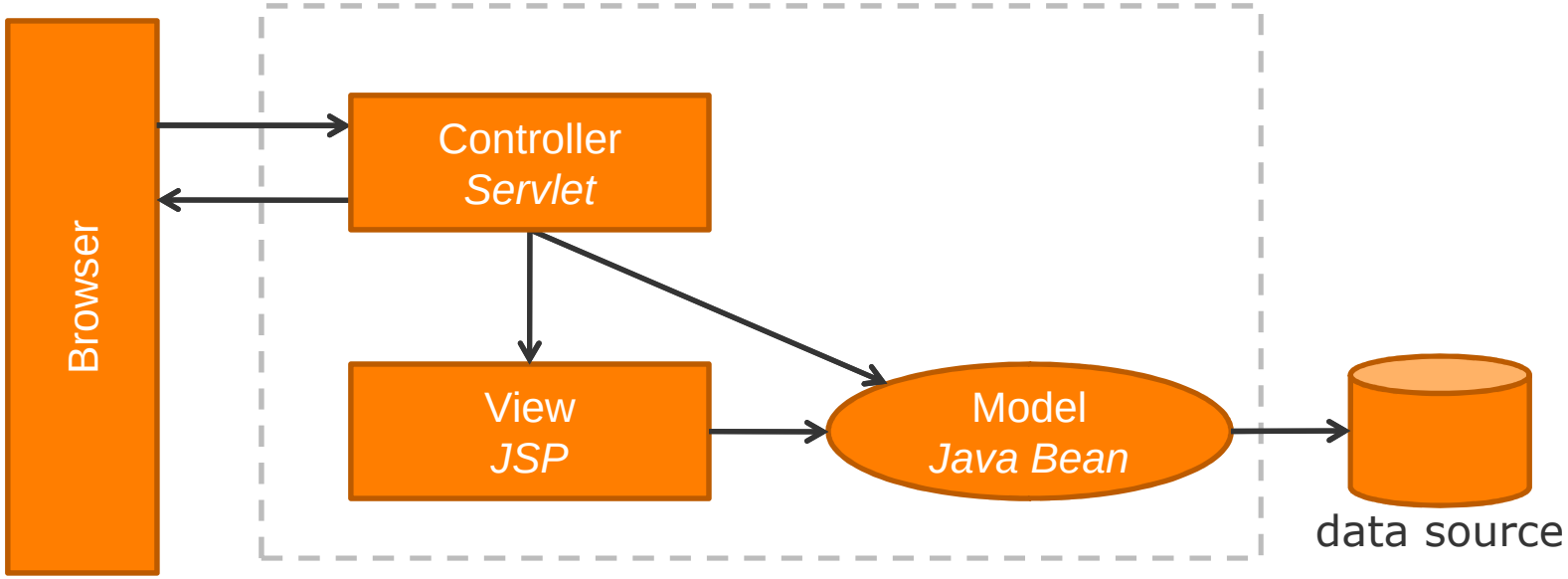
Model 1 architecture



- Attention:
  - JSP site manages requests **and** responses
  - Mixture of control- and presentation code

Realize MVC with JSP

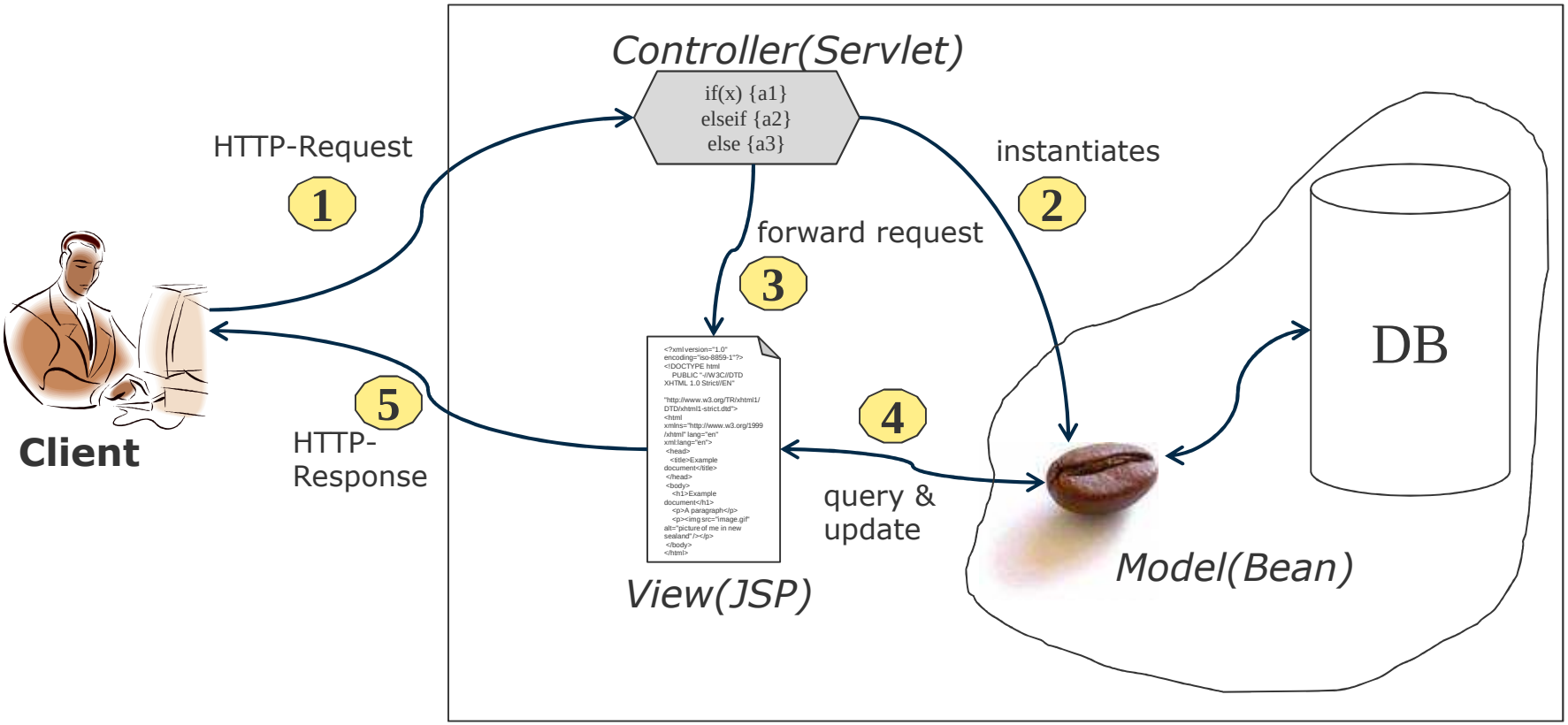
Model 2 architecture



- Differences to Model 1:
  - Servlet manages requests and responses (= Controller)
  - Servlet might select view
  - JSP handles presentation

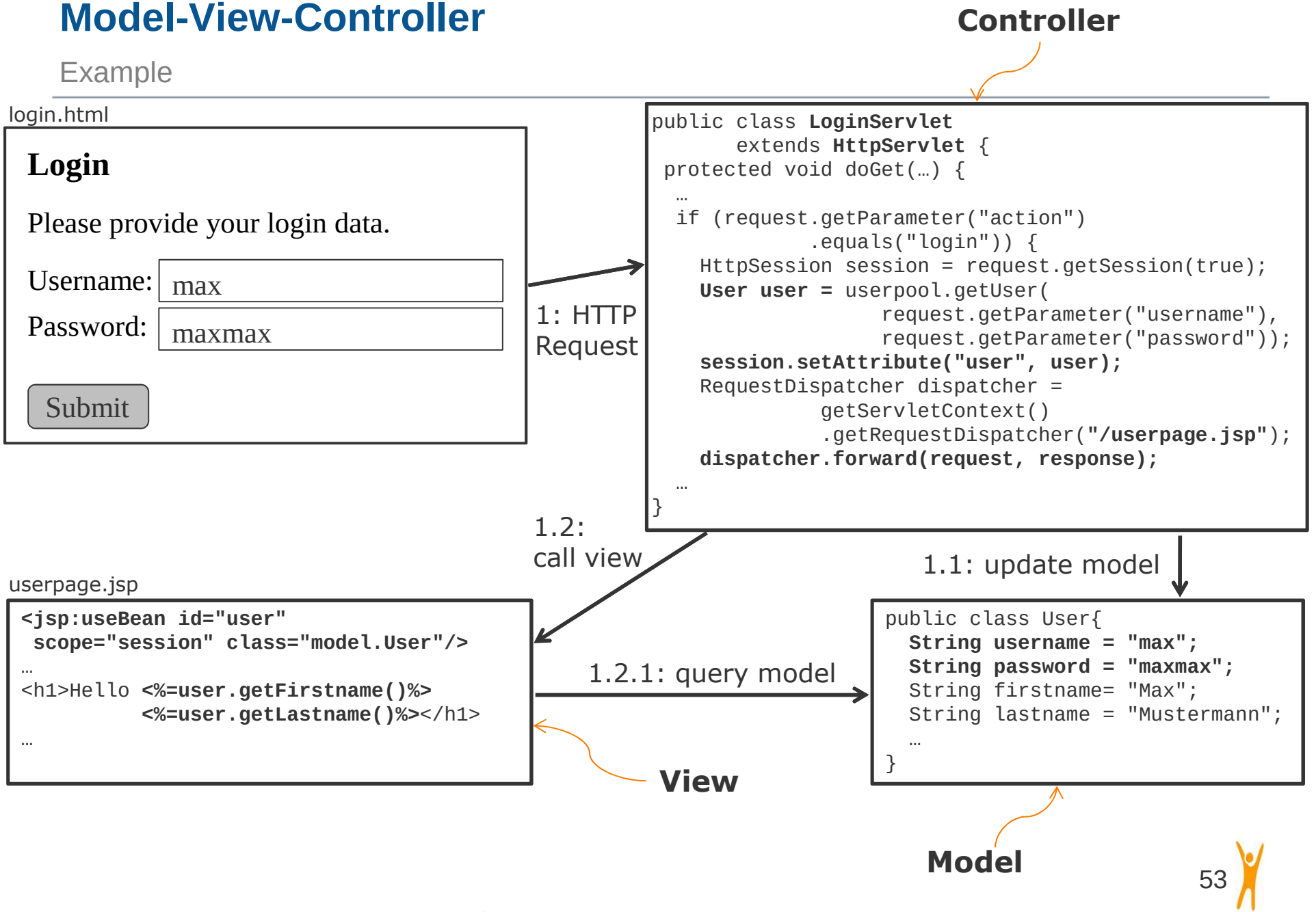
Architectural Patterns

MVC for the Web



Model-View-Controller

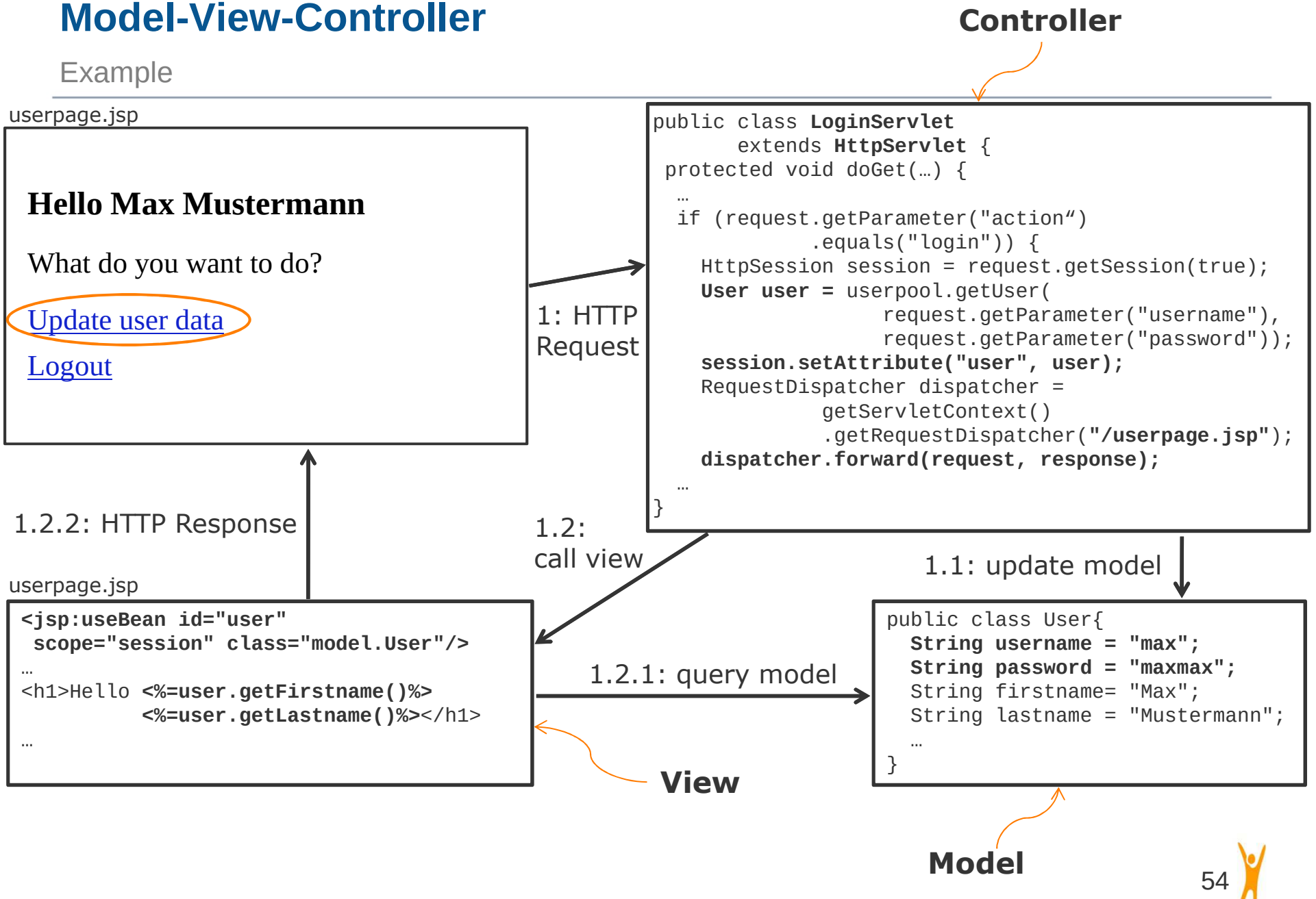
Example



TUWEL course resources: RegistrationMVC

Model-View-Controller

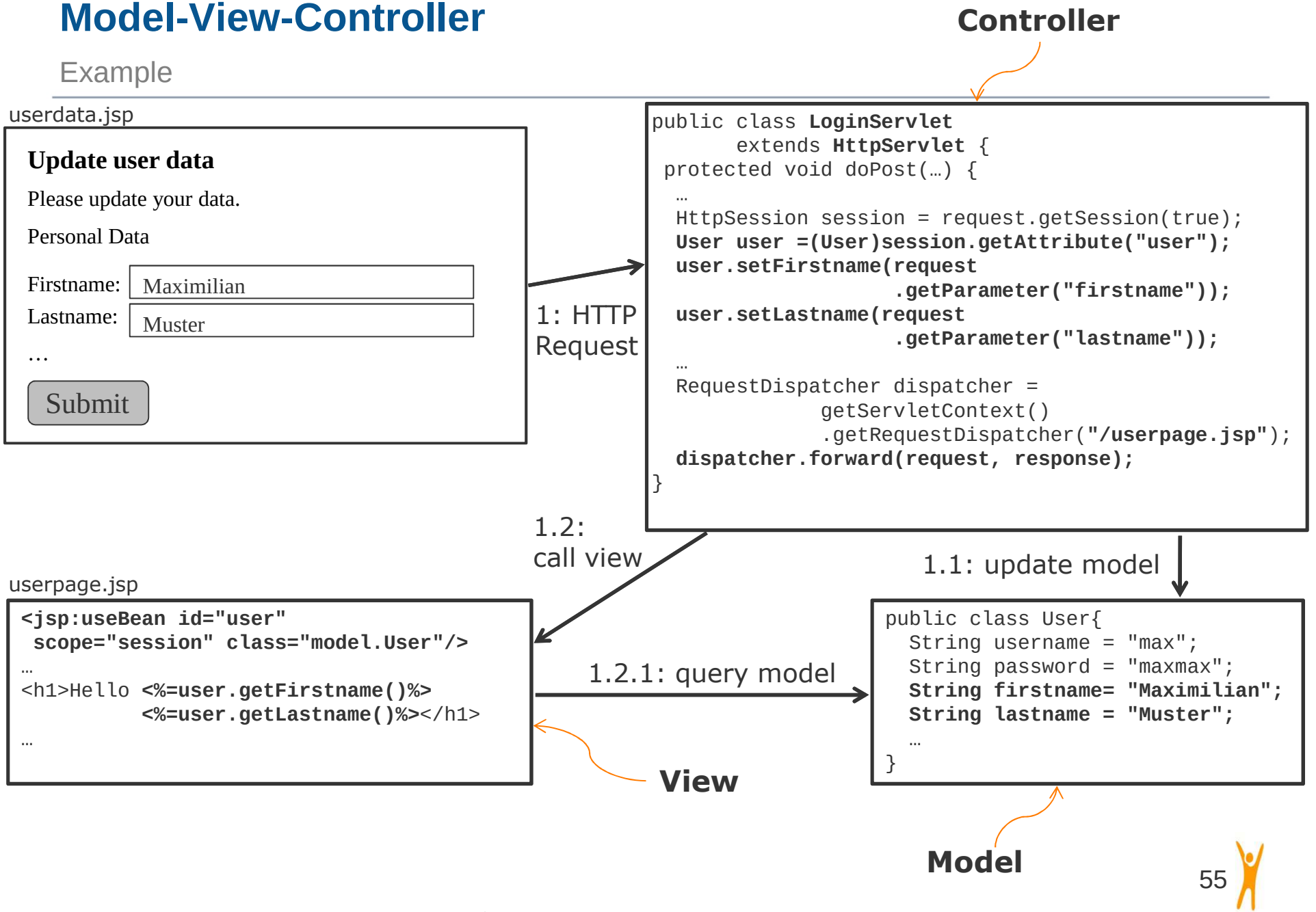
Example



TUWEL course resources: RegistrationMVC

Model-View-Controller

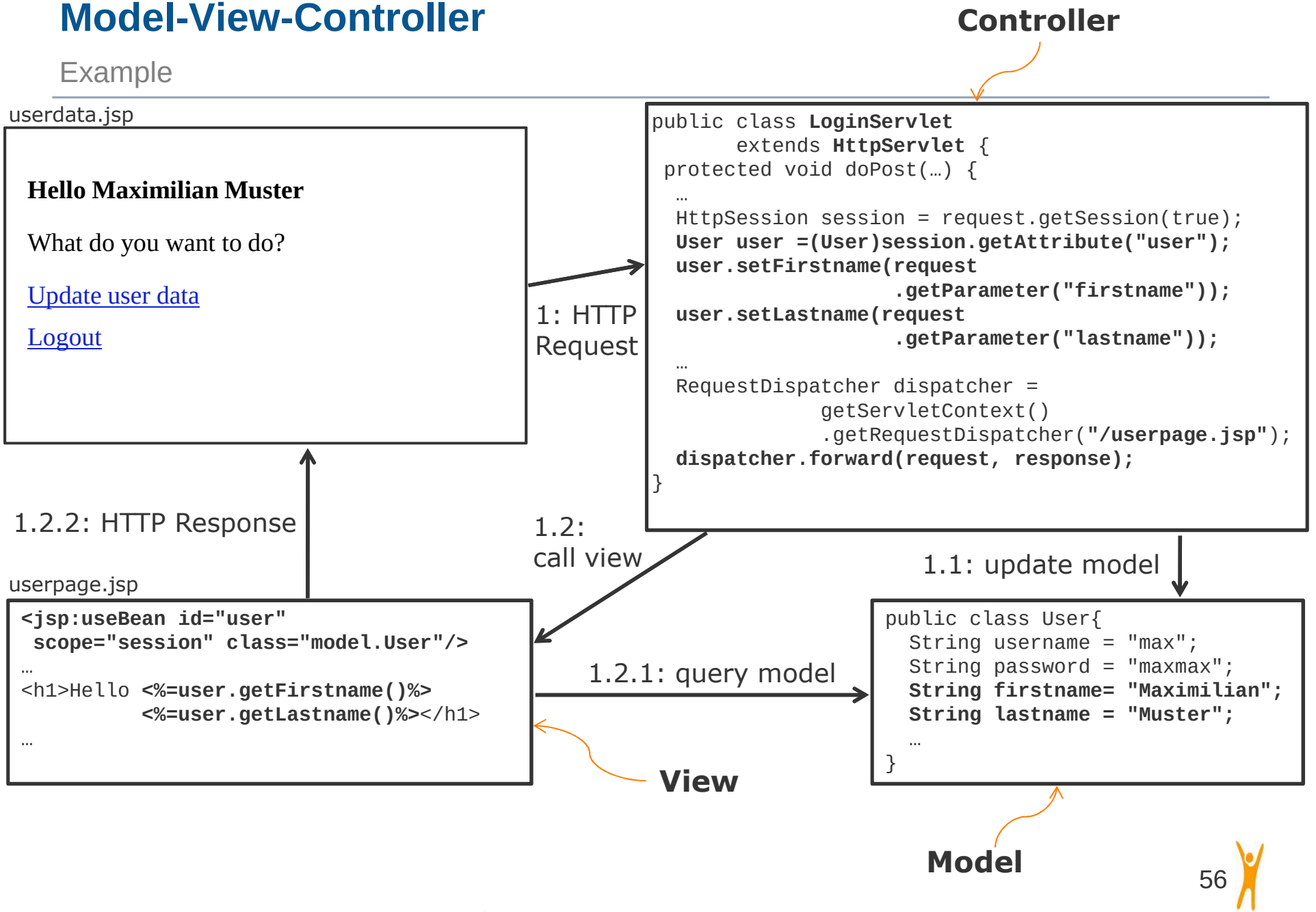
Example



TUWEL course resources: RegistrationMVC

Model-View-Controller

Example



TUWEL course resources: RegistrationMVC

Summary

- Advantages of Servlets/JSP
  - Suitable for complex applications
  - Java Server Pages as template- and scripting-mechanism
  - Availability of libraries (database access, XML, etc.) and frameworks (Struts, etc.)
  - Many tools and open source projects

- Alternatives
  - Interfaces and frameworks for other languages, e.g. PHP, Perl
  - Server specific interface plugins, e.g. Microsoft IIS API
  - Advanced Frameworks, e.g. JavaServer Faces

Further Literature

- B. Basham, K. Sierra, B. Bates. Servlets und JSP von Kopf bis Fuß. O'Reilly, 2009.
- M. Hall, L. Brown. Core Servlets and JavaServer Pages. Prentice Hall, 2003. Online available: http://pdf.coreservlets.com.
- Oracle Corporation. Java Platform, Enterprise Edition 6 API Specification. Online available: http://docs.oracle.com/javaee/6/api/.