Web-based information systems 1

Servlets: sessions, authentication & servlet details

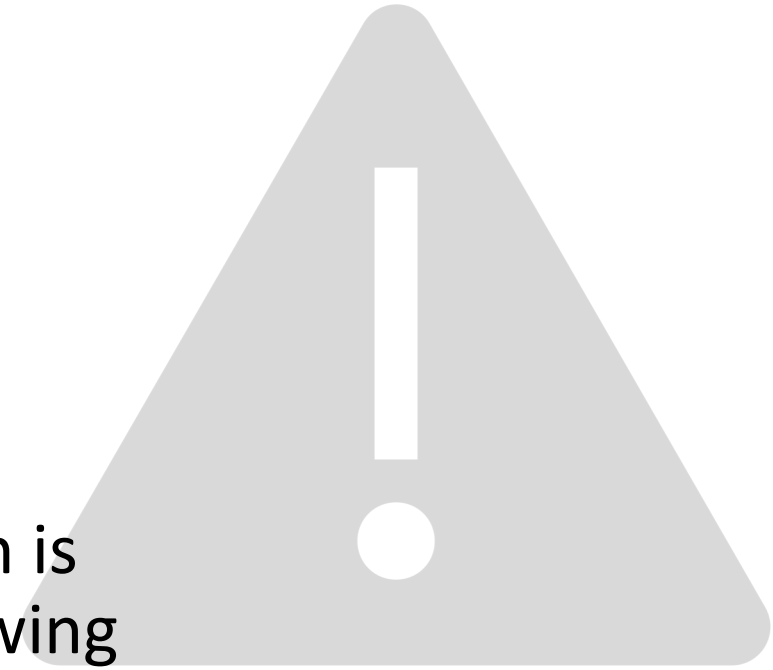
Michiel Noback (NOMI)
Institute for Life Sciences and Technology
Hanze University of Applied Sciences

Introduction

This presentation:

- application-scoped variables with web.xml
- user authentication
- Sessions
  - These are two very related subjects as you will see
- More on servlet configuration
- App lifecycle hooks

Disclaimer



Everything in this presentation is purely for the purpose of showing principles.

For real-world applications you will need to do much work on security!

Application scope

Adding a deployment descriptor

Go to Project Structure (Cmd + ; on Mac)

- Facets -> Web -> Web Gradle (select your project)
- Under pane Deployment Descriptors, click "+"
- Select "web.xml”
- In the path, put this: /src/main/webapp/WEB-INF/web.xml
- You’ll get something like this

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
               http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version=”4.0">

</web-app>
```

Adding an entry to web.xml

- In web.xml, put the following snippet

```
<context-param>
   <param-name>admin_email</param-name>
   <param-value>admin@example.com</param-value>
</context-param>
```

Accessing an entry of web.xml

- In your servlets, use the web app variable like this

```
getServletContext().getInitParameter("admin_email")
```

- or in your Thymeleaf template like this

```
th:text="${#servletContext.getInitParameter('admin_email')}"
```

Adding a session time-out

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                             http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">

    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

The session-timeout entry makes your session short-lived

Role of web.xml

- The web.xml file holds information that is relevant to your entire web application. It holds
  - information that is relevant to the whole application, such as the baseURL, admin email and contact information
  - servlet configuration
- To access web.xml init params from a servlet, use this code:

```
String param = servletConfig
      .getServletContext()
      .getInitParameter("param_name");
```

Scopes

- As in all programming environments, scopes are really important in web applications.
- We have seen the three main ones:

Application scope

- Specified in web.xml and accessed through
  - `${#servletContext.getInitParameter('admin_email')}`
    in the Thymeleaf files
  - `getServletContext().getInitParameter("<attrib_name>")`
    in the servlet code

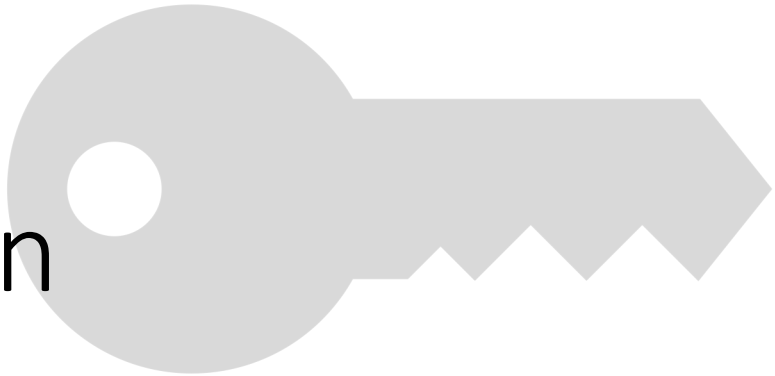
Session scope

- When a session object is available, accessed through
  - `${session.<attrib_name>}`
    in the Thymeleaf files
  - `getSession.getAttribute("<attrib_name>")`
    or
    `getSession.setAttribute("<attrib_name>", <attrib>)`
    in the servlet code

‘Request’ scope

- Each http request generates its own HttpRequest object.
- It can be accessed through
  - `${#request.<attrib_name>}` in the Thymeleaf files
  - `request.getParameter("<attrib_name>")`
  - `request.setAttribute("<attrib_name>", "<attrib>")`
- The ' WebContext ' object is of course used much more in Thymeleaf, simply accessed as
  - `${attrib_name>}`

Sessions & authentication



User authentication

- There are several reasons why you would like to know who it is that requests a certain page:
  - you want only authorized persons to have access
  - you want to track somebody's history on your site (e.g. for a shopping card)
- To be able to do this, you will need **session** information

http has no memory

- A server handling your request will have no memory of your visit after it is processed
- To be able to “keep in touch” you will need to have a tracking system: we call this a **session**
- The most well-known session implementation system is by means of **cookies**

Cookies

- “In computing, a cookie (also tracking cookie, browser cookie, and HTTP cookie) is a small piece of text stored on a user's computer by a web browser. A cookie consists of one or more name-value pairs containing bits of information such as … the identifier for a server-based session… .” (wikipedia.org)

Cookies and sessions

- In java, sessions are usually maintained by means of a JSessionID stored as a cookie
- You will probably never create a cookie directly
- You simply use this statement:

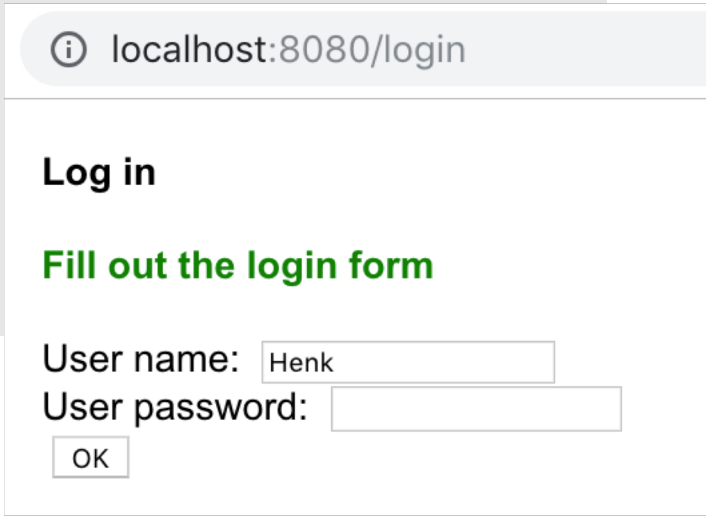
  HttpSession session = request.getSession();

- Let's look at some sample code to create a login system using sessions

A login form: login.html

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Log in</title>
</head>
<body>
<h4 th:text="${message}" th:class="${message_type}">_message_</h4>
<form action="login" method="POST">
 <label for="username_field"> User name: </label>
 <input id="username_field" type="text" name="username" required/> <br/>
 <label for="password_field"> User password: </label>
 <input id="password_field" type="password" name="password"
required/><br/>
 <label class="login_field"> </label>
 <input type="submit" value="OK"/>
</form>
</body>
</html>
```

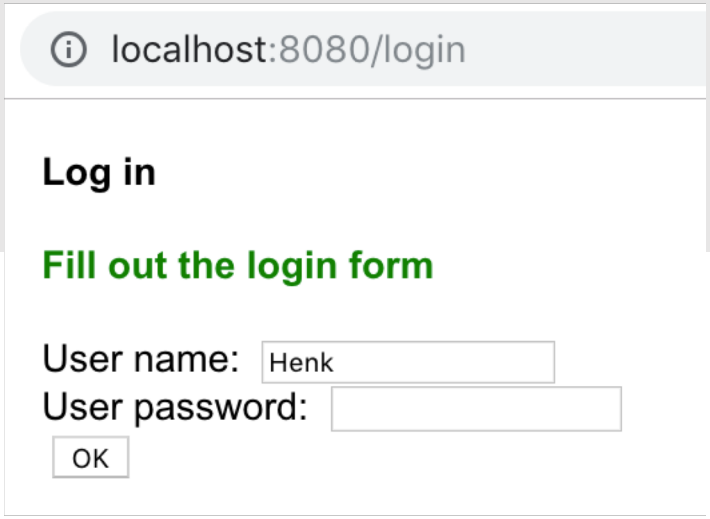
A very simple login form. Take notice of the form field names eg “username”



A login form: login.html

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Log in</title>
</head>
<body>
<h4 th:text="${message}" th:
<form action="login" method=
 <label for="username_field"
 <input id="username_field"
 <label for="password_field"
 <input id="password_field" type="pa
 <label class="login_field"> </label>
 <input type="submit" value="OK"/>
</form>
</body>
</html>
```

Actually, when you see a http:// protocol instead of an https:// protocol, you do NOT have safety on you connection, even though you cannot see the password in the form! But that issue is not within scope of this course.



Login servlet (part 1)

```
@WebServlet(name = "LoginServlet", urlPatterns = "/login")
public class LoginServlet extends HttpServlet {
    //code omitted...
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
                                                    throws ServletException, IOException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        WebContext ctx = //code omitted...
        HttpSession session = request.getSession();
        String nextPage;
        if (session.isNew() || session.getAttribute("user") == null) {
            boolean authenticated = authenticate(username, password);
            if (authenticated) {
                session.setAttribute("user",
                                     new User("Henk", "henk@example.com", Role.USER));
                nextPage = "mainSite";
            } else {
                ctx.setVariable("message",
                     "Your password and/or username are incorrect; please try again");
                ctx.setVariable("message_type", "error");
                nextPage = "login";
            }
        } else {
            nextPage = "mainSite";
        }
        templateEngine.process(nextPage, ctx, response.getWriter());
    }
}
```

Check the session status and proceed: if new, check user credentials

Set an error message if the user does not pass authentication

Login servlet (part 2)

```
@WebServlet(name = "LoginServlet", urlPatterns = "/login")
public class LoginServlet extends HttpServlet {
    //code omitted...

    private boolean authenticate(String username, String password) {
        return username.equals("Henk") && password.equals("henk");
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws ServletException, IOException {
        WebContext ctx = new WebContext(
                request,
                response,
                request.getServletContext(),
                request.getLocale());
        ctx.setVariable("message", "Fill out the login form");
        ctx.setVariable("message_type", "info");
        templateEngine.process("login", ctx, response.getWriter());
    }
}
```

Testing the pudding

- Start the server and run
  - login
  - Enter username and password
  - False
  - Correct
- What is the result, what do you see?
- Can you detect weaknesses and/or improve on the design

Servlet configuration

Servlet annotations

- The @WebServlet annotation offers several possibilities for configuration. Here are the interesting ones:

Name	Type	Req-uired	Description
urlPatterns *or* value	String[]	y	Specify one or more URL patterns of the servlet. Either of attribute can be used, but not both
asyncSupported	String	n	Specify whether the servlet supports asynchronous operation mode. Default is false.
initParams	WebInit Param[]	n	Specify one or more initialization parameters of the servlet. Each parameter is specified by @WebInitParam annotation type.
loadOnStartup	int	n	Specify load-on-startup order of the servlet. Default is -1

@Webservlet - urlPattern

- The urlPattern parameter specifies which urls will map on your servlet.
- This configuration:

```
@WebServlet(name = "WelcomeServlet",
            urlPatterns = {"/welcome", "/index", "/home"})
public class WelcomeServlet extends HttpServlet { ... }
```

maps to

- http://localhost:8080/index
- http://localhost:8080/welcome
- http://localhost:8080/home

@Webservlet - loadOnStartup

- loadOnStartup is an integer which specifies the servlet load order.
- The container loads servlets with lower values before servlets with higher values.
- If loadOnStartup is negative (it defaults to -1), the servlet will be loaded when the container decides to - typically on first access.

```
@WebServlet(urlPatterns = {"/welcome"},
     loadOnStartup = 1)
     //guarantees first loading at deployment
public class WelcomeServlet extends HttpServlet { ... }
```

@Webservlet – initParams

- They are defined as instances of @WebInitParam

```
@WebServlet(urlPatterns = {"/welcome"},
  initParams = {
    @WebInitParam(name = "tempDir", value = "/tmp"),
    @WebInitParam(name = "allowedFileTypes", value = "jpg,png")})
public class WelcomeServlet extends HttpServlet { ... }
```

@Webservlet - versus web.xml

- When configuration is done in both web.xml ánd in @WebServlet annotations, the web.xml configuration wins.
- Use case: development mode versus production mode.
- See next slide

@Webservlet - versus web.xml

- Here, the url will become ‘/extra’ and tempDir ‘/some/location’

```
@WebServlet(name = "WelcomeServlet",
  urlPatterns = {"/welcome", "/home"},
  initParams = {@WebInitParam(name = "tempDir", value = "/tmp")})
public class WelcomeServlet extends HttpServlet { }
```

```
<servlet-mapping>
    <servlet-name>WelcomeServlet</servlet-name>
    <url-pattern>/extra</url-pattern>
</servlet-mapping>
<servlet>
    <servlet-name>WelcomeServlet</servlet-name>
    <servlet-
class>nl.bioinf.wis_on_thymeleaf.servlets.WelcomeServlet</servlet-class>
    <init-param>
        <param-name>tempDir</param-name>
        <param-value>/some/location</param-value>
    </init-param>
</servlet>
```

@Webservlet – asyncSupported

- This allows the servlet to store incoming requests for later response.
- For practical purposes, with this configuration you can set up a servlet that will (in effect) **push** data to the client (after the client sends the initial request to the server).
- This is very interesting, but outside the scope of this course.

@Webservlet - versus web.xml

- Here, the url will become ‘/extra’ and tempDir ‘/some/location’

```
@WebServlet(name = "WelcomeServlet",
  urlPatterns = {"/welcome", "/home"},
  initParams = {@WebInitParam(name = "tempDir", value = "/tmp")})
public class WelcomeServlet extends HttpServlet { }
```

HOOKS

What are hooks

- Hooks are pieces of code that let you write code that will run at application life cycle events.
- The ones we will look at here are:
  - Application startup/shutdown hook
  - Servlet initialization/cleanup hooks

Application startup & shutdown

- If you want to do something at
  - startup – e.g. create a single template engine to service all servlets
  - shutdown – e.g. free DB resources
- you annotate with @WebListener and
- implement the ServletContextListener interface

App startup & shutdown hooks

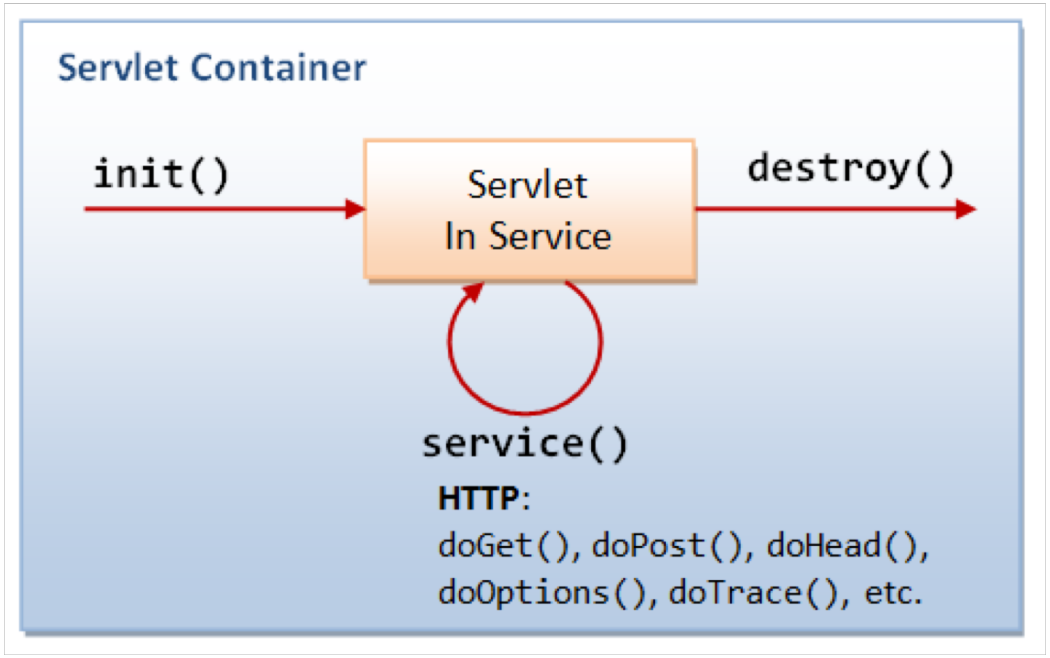
```
@WebListener
public class ThymeleafUtil implements ServletContextListener {

  @Override
  public void contextInitialized(ServletContextEvent servletContextEvent) {
      System.out.println("[ThymeleafUtil] Initializing template engine");
      createTemplateEngine(servletContextEvent.getServletContext());
      System.out.println("[ThymeleafUtil] Initializing database");
      initializeDatabase(servletContextEvent.getServletContext());
  }

  @Override
  public void contextDestroyed(ServletContextEvent servletContextEvent) {
      System.out.println("[ThymeleafUtil] Shutting down!");
  }
}
```

Servlet instantiation and destroy

- instantiation – creating or loading objects that are used by the servlet in the handling of its requests
- `init()` is guaranteed to be called before the first call to service()
- shutdown – freeing memory of these objects



Servlet lifecycle hooks

- `init(ServletConfig config)` - Called by the servlet container to indicate to a servlet that the servlet is being placed into service.
- `init()` - A convenience method which can be overridden so that there's no need to call `super.init(config)`.
- `init()` - Called by the servlet container to indicate to a servlet that the servlet is being taken out of service

Servlet lifecycle hooks

```
@WebServlet(name = "PhraseServlet", urlPatterns = "/give.phrase")
public class PhraseServlet extends HttpServlet {

    @Override
    public void init() throws ServletException {
        //no super.init() required
        System.out.println("[PhraseServlet] Running no-arg init()");
        this.templateEngine = ThymeleafUtil.getTemplateEngine();
    }

    @Override
    public void init(ServletConfig config) throws ServletException {
        super.init(config); //required in this implementation
        System.out.println("Running init(ServletConfig config)");
        config.getServletContext().getInitParameter("admin_email");
    }

    @Override
    public void destroy() {
        super.destroy();
        System.out.println("Shutting down servlet service");
    }
```

Review

What aspects have we seen in this presentation?

- web.xml deployment descriptor
- session object usage and scopes
- servlet contexts and hooks