95-702 Distributed Systems

Lecture 2: Server-Side Programming: An Introduction to Servlets

What is a Servlet?

- Created by Sun back in 1997
- A Java class that extends HttpServlet
- Responds to HTTP requests
- The response is usually XHTML or some other XML language
- May maintain state across several interactions (may use cookies or URL rewriting or hidden form fields)
- Live within a web container
- May be generated by a JSP compiler

Servlet Lifecycle

- The container loads the servlet class.
- The servlet's init() method is called exactly once.
- Upon each request, the container calls the servlet's service() method.
- The service() method selects the appropriate method to call and calls it.
- Finally, before the container shuts down, it calls the servlet's destroy() method.

What is an HTTP request?

/* From Core Servlets, Marty Hall

An HTTP **Request** header example

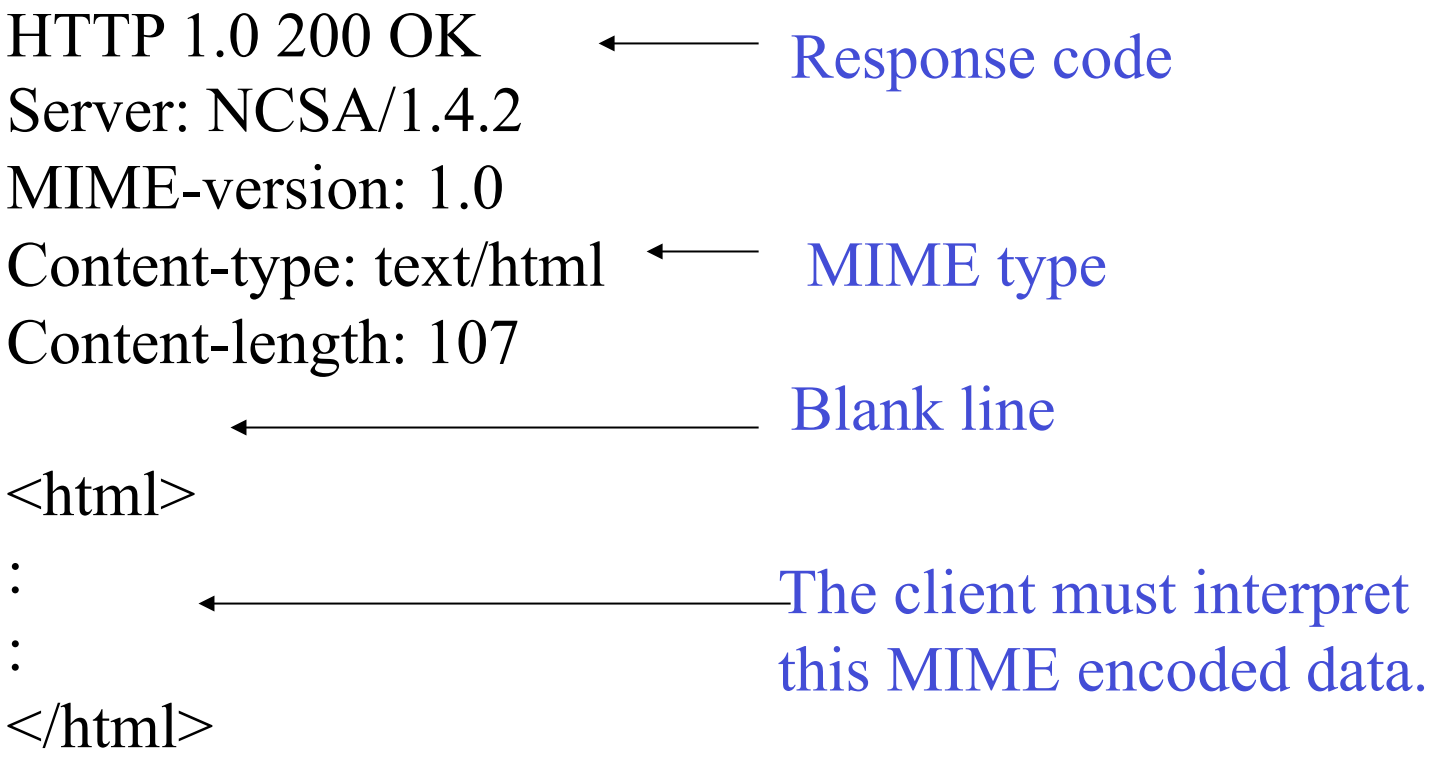
```
GET /path/file.html  HTTP/1.0
Accept:  text/html
Accept:  audio/x
User-agent:  MacWeb
A blank line followed by name
value pairs or an XML document
```

The whitespace is required.
Accept header fields tell the server MIME types (Multipurpose Internet Mail Extension) that are handled by the browser.

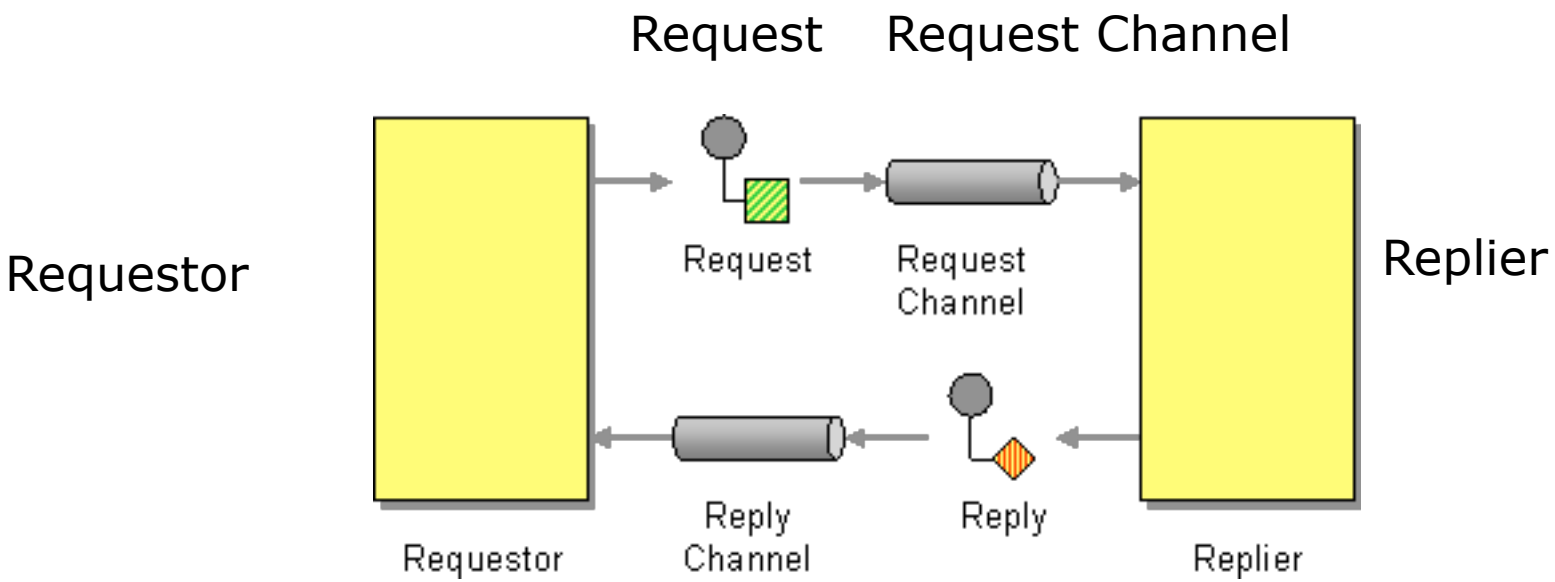
HTTP defines dozens of possible headers.

What is an HTTP Response?

An HTTP **Response** header example



Request Reply Pattern



Reply channel reply

The pattern applies in the asynchronous and synchronous cases. HTTP is synchronous request reply.

From “Enterprise Integration Patterns”.

HTTP General Form

<method> <resource identifier> <HTTP Version> <crlf>
[<Header> : <value>] <crlf>
: : :
[<Header> : <value>] <crlf>
a blank line
[entity body]

The resource identifier field specifies the name of the target resource; it's the URL stripped of the protocol and the server domain name. When using the GET method, this field will also contain a series of name=value pairs separated by '&'. When using a POST method, the entity body contains these pairs.

The HTTP version identifies the protocol used by the client.

Reading Form Data With Servlets Under a Web Server (Glassfish)

```
// QueryData.java -- Handle the voting form in radio.html
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class QueryData extends HttpServlet {

  public void doPost(HttpServletRequest req,
                     HttpServletResponse response)
                     throws ServletException,
                     IOException {
    doGet(req, response);
  }
```

```
public void doGet(HttpServletRequest req,
                  HttpServletResponse response)
                  throws ServletException,
                  IOException
  {
     String newPresident = req.getParameter("president");


     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     String docType = "<!DOCTYPE HTML PUBLIC \"//W3C//DTD” +
                      “HTML 4.0 ";
                      docType += "Transitional//EN\">\n";
```

```
out.println(docType + "<HTML>\n" +
        "<HEAD><TITLE>Presidential Servlet" + "</TITLE>" +
        "</HEAD>\n" +
        "<BODY>\n" +
        "<H1>The new president is  "+ newPresident + "</H1>\n" +
        "</BODY></HTML>");
   }
}
```

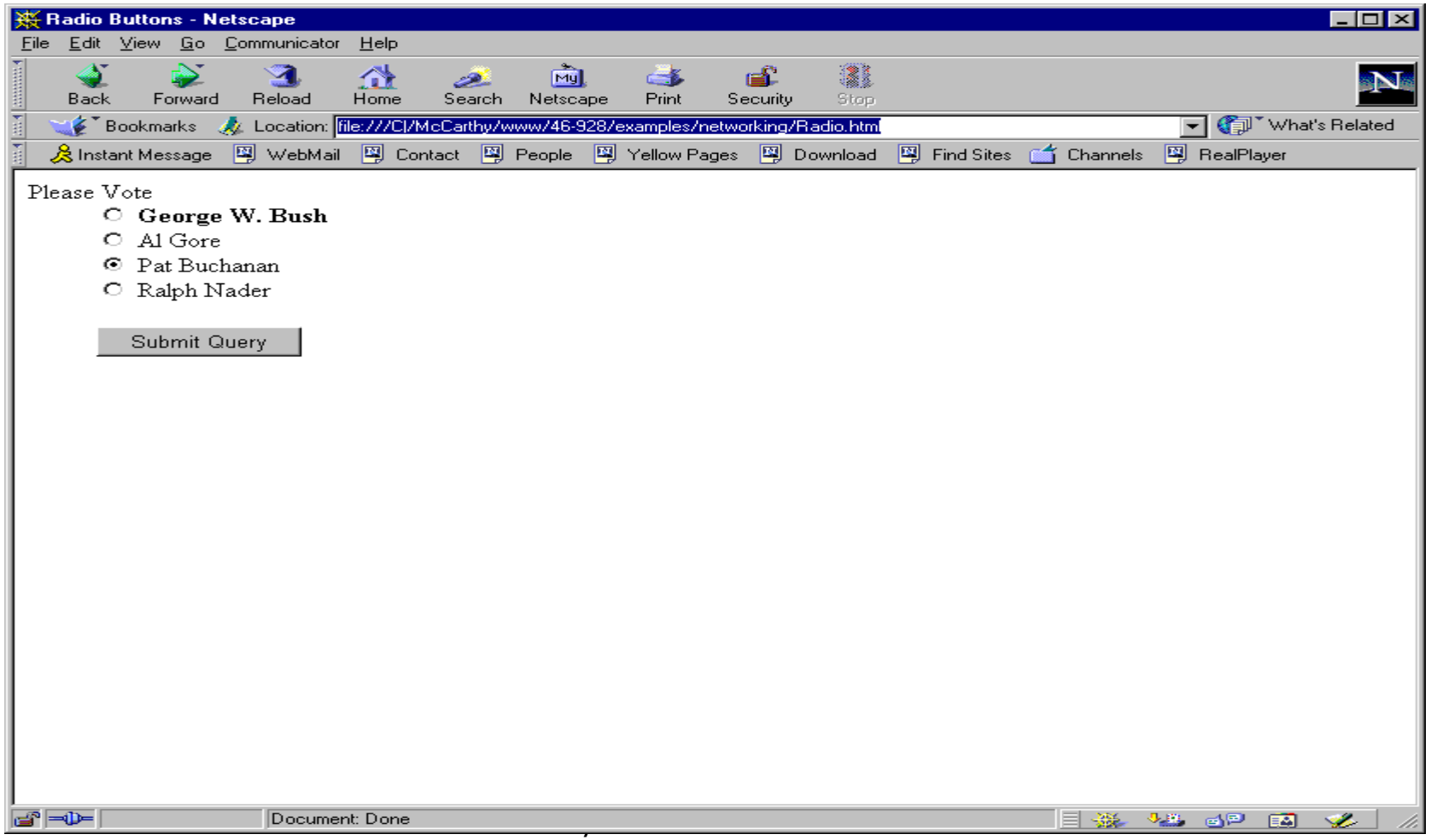
```
<!-- index.jsp -->
<html>
<head>
<title>Radio Buttons</title>
</head>
<body BGCOLOR="WHITE">
  <form action="http://localhost:8080/WeekTwoServlets/QueryData">
    <dl>
      <dt> Please Vote </dt>
      <dd><Input type = "Radio" name = "president" value= "Bush">
                                                  <b>George W. Bush</b>
      <dd><Input type = "Radio" name = "president" value = "Gore"> Al Gore
      <dd><Input type = "Radio" name = "president" value = "Buchanan"> Pat Buchanan
      <dd><Input type = "Radio" name = "president" value = "Nader"> Ralph Nader
      <p> <input type = "submit">
    </dl>
  </form>
</body>
</html>
```

Web server's port

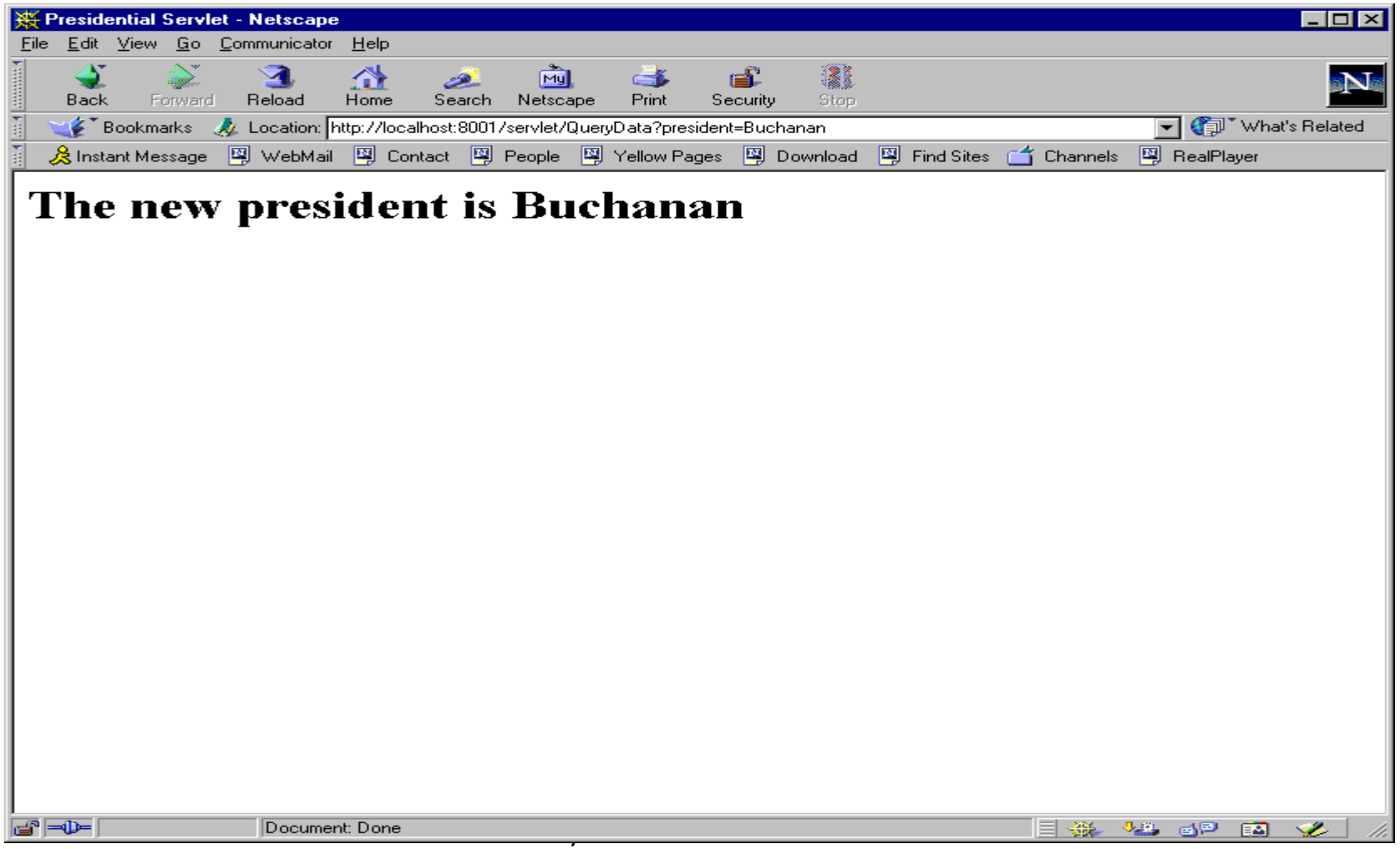
Project path

servlet

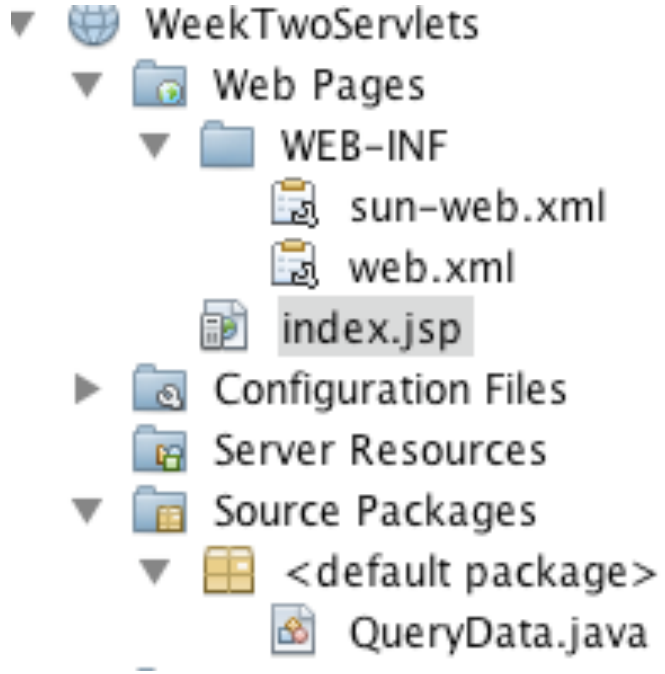
Radio HTML in the browser



The Servlet's Response



NetBeans Project List



Netbeans provides a development environment.

The software is deployed to Glassfish.

NetBeans Generated web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation=
           "http://java.sun.com/xml/ns/javaee
           http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>QueryData</servlet-name>
    <servlet-class>QueryData</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>QueryData</servlet-name>
    <url-pattern>/QueryData</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Note how the servlet's name is associated with a URL pattern.

"QueryData" is a user defined identifier for use only within this file.

Some Non-Functional Characteristics

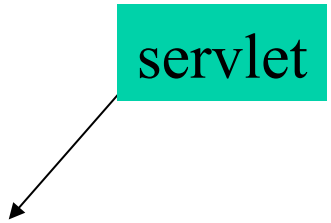
Interoperability ?
Concurrency?
Security? Eve? Mallory?

Suppose we were to configure the web server to do SSL.

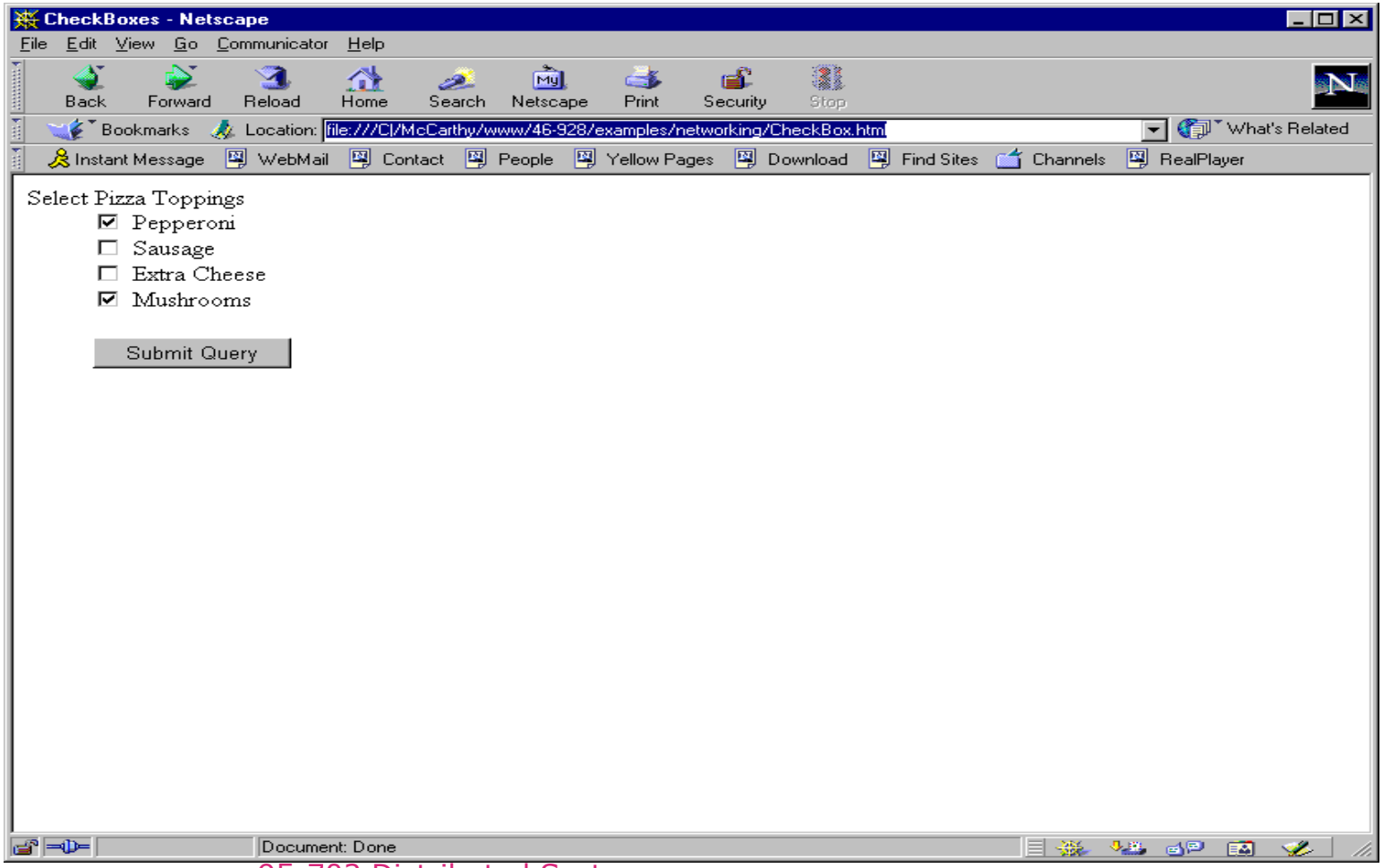
Interoperability ?
Concurrency?
Security? Eve? Mallory? Does SSL provide secure voting?

Handling CheckBoxes

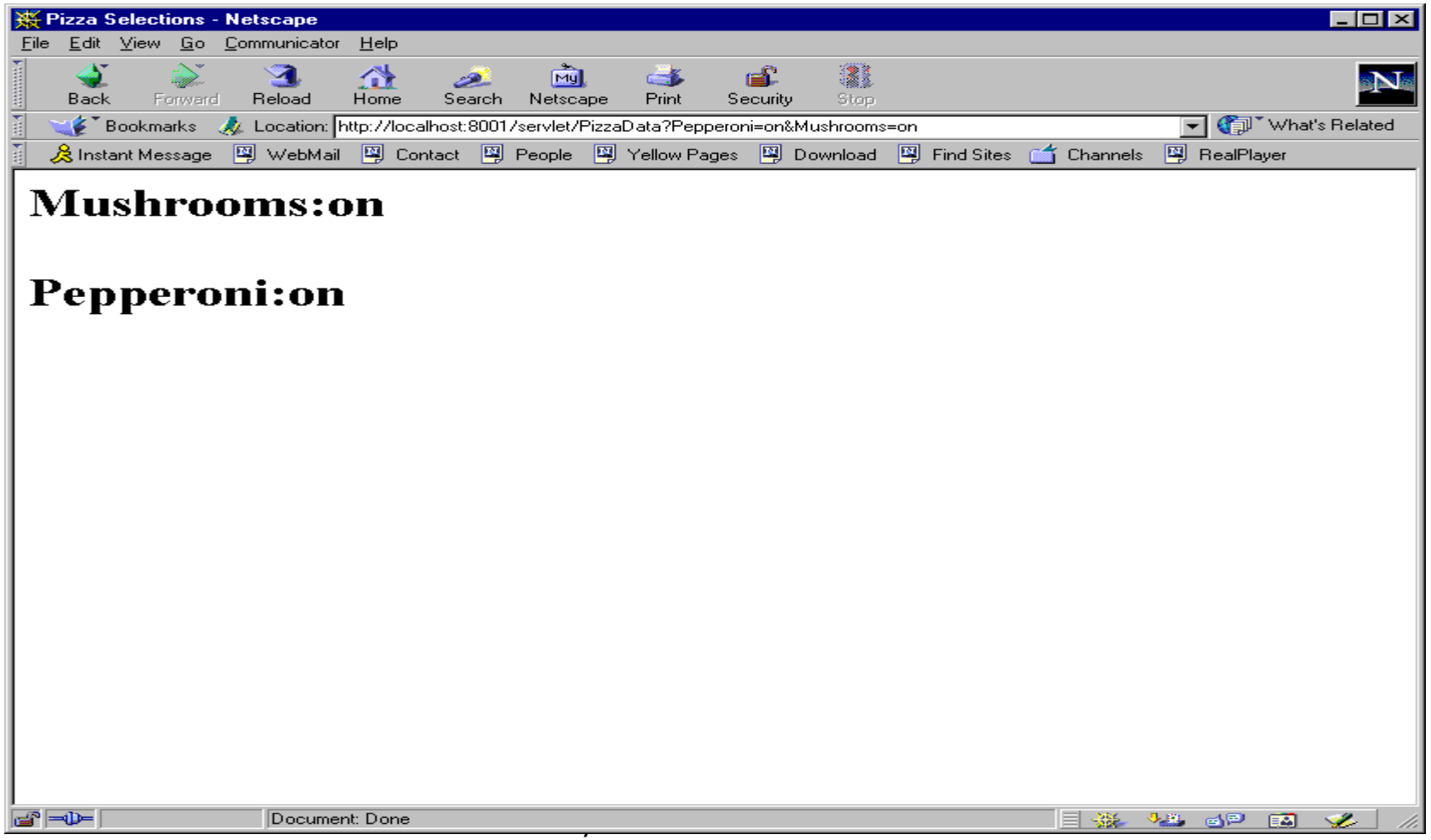
```
<!-- CheckBox.html -->
<html>
<head>
<title>CheckBoxes</title>
</head>
<body BGCOLOR="WHITE">
   <form action="http://localhost:8080/servlet/PizzaData">
     <dl>
       <dt> Select Pizza Toppings </dt>
       <dd><Input type = "CheckBox" name = "Pepperoni"> Pepperoni
       <dd><Input type = "CheckBox" name = "Sausage"> Sausage
       <dd><Input type = "CheckBox" name = "Extra Cheese"> Extra Cheese
       <dd><Input type = "CheckBox" name = "Mushrooms"> Mushrooms
       <p> <input type = "submit">
     </dl>
   </form>
</body>
</html>
```



Pizza Toppings



Servlet Response



PizzaData Servlet

```
// PizzaData.java -- Handle the toppings selection from pizza.html
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PizzaData extends HttpServlet {
    public void doPost(HttpServletRequest req,
                       HttpServletResponse response)
                       throws ServletException,
                       IOException {

        doGet(req, response);
    }
```

```
public void doGet(HttpServletRequest req,
                  HttpServletResponse response)
                  throws ServletException,
                  IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String finalString = "";

        Enumeration paramNames = req.getParameterNames();

        while(paramNames.hasMoreElements()) {
            String paramName = (String) paramNames.nextElement();
            finalString += paramName + ":" ;

            finalString += req.getParameter(paramName) + "<p>";
        }
```

Enumerate over the input.

```
String docType = "<!DOCTYPE HTML PUBLIC \"//W3C//DTD”
                    + “ HTML 4.0 ";
            docType += "Transitional//EN\">\n";

        out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>Pizza Selections" + "</TITLE>” +
                 “</HEAD>\n" +
                "<BODY>\n" +
                "<H1>" + finalString + "</H1>\n" +
                "</BODY></HTML>");
    }

}
```

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
    "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
    <web-app>
    <servlet>
        <servlet-name>NameInThisFile</servlet-name>
        <servlet-class>PizzaData</servlet-class>
        <load-on-startup/>
    </servlet>
    <servlet-mapping>
        <servlet-name>NameInThisFile</servlet-name>
        <url-pattern>/PizzaData/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Some Non-Functional Characteristics

Interoperability ?
Concurrency?
Security? Eve? Mallory?

Suppose we were to configure the web server to do SSL.

Interoperability ?
Concurrency?
Security? Eve? Mallory? Does SSL provide secure electronic commerce?

Part II Session Tracking and Servlet Collaboration

- First we will use a shared object.
- Then we'll use Java's Session Tracking API.

Session Tracking with Servlets

HTTP is a normally a stateless protocol. What does that mean?

Compare buying coffee at Starbucks with the act of eating a seven course meal at The Tavern On The Green.

We can add state to HTTP by having each user introduce themselves in some way.

We'll look at traditional session tracking and then look at the Session Tracking API.

Traditional Session Tracking

- User Authorization
- Hidden Form fields
- URL Rewriting
- Persistent cookies

We'll look at the first and last.

User Authorization

- The web server requests the user name and password. The information is available to any servlet that needs it.
- The browser resends the name and password with each subsequent request.
- Data about the user and the user's state can be saved in a shared object.

Shared Objects

- A convenient way to store data associated with a user.
- There are likely to be many servlets running.
- They can collaborate through a shared object.
- Only one instance of the shared object should exist.
- It has to be available (in the classpath) of the servlets that needs it.
- It will be used by several threads and therefore should protect itself against simultaneous access.
- We'll look at a shared object and two servlets that use it.

VisitTracker.java

```
// Servlet collaboration can be done through a shared object.
// Any servlet has access to this object and it only has one
// instance.
// It maintains a hash table of names and dates.

// Sections of code that must not be executed simultaneously
// are called critical sections. Java provides the synchronized
// keyword to protect these critical sections. For a synchronized
// instance method, Java obtains an exclusive lock on the class
// instance.

import java.util.*;
```

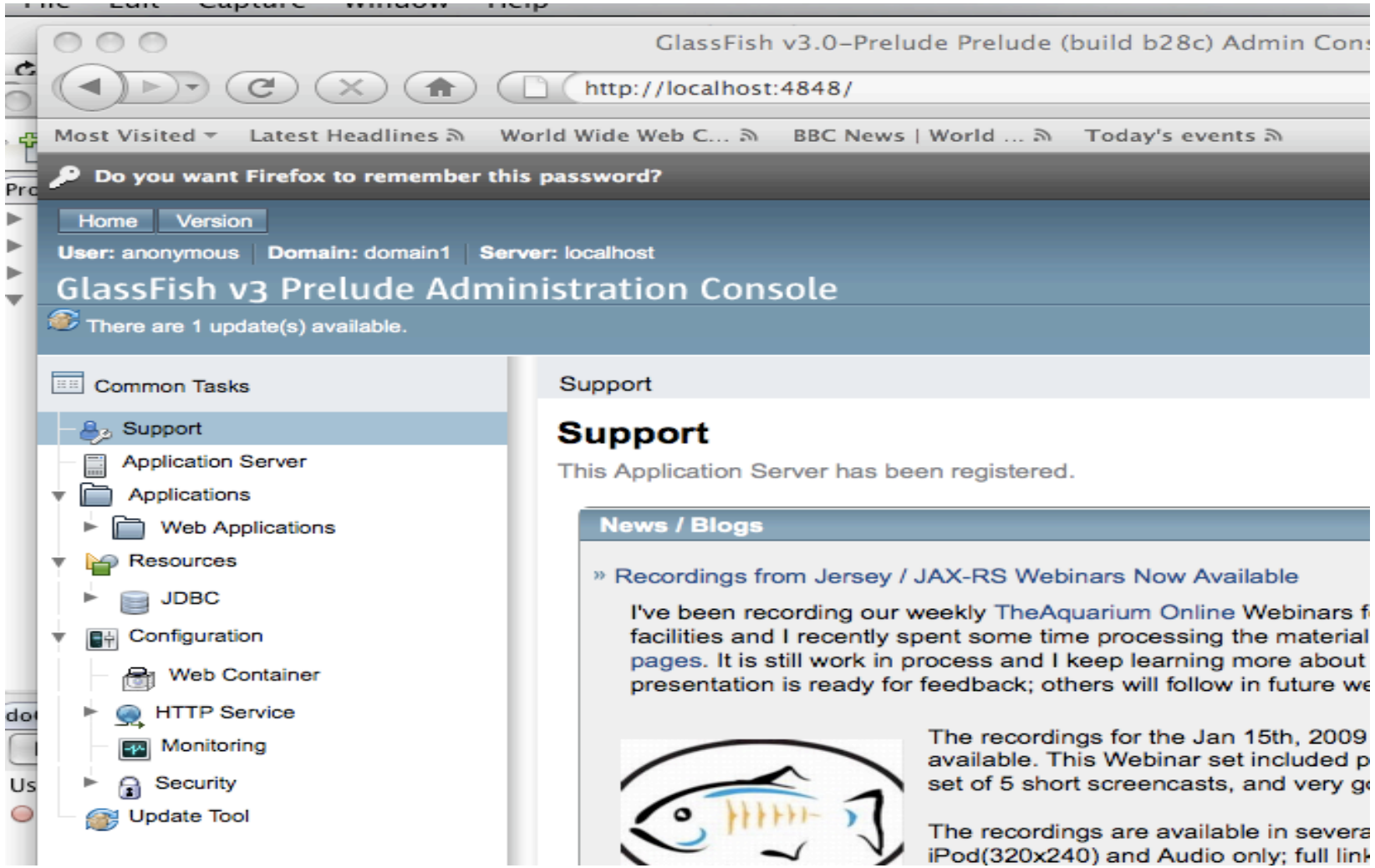
```
public class VisitTracker  {

    private Map nameDatePairs;
    private static VisitTracker instance = new VisitTracker();

    private VisitTracker() {          // private constructor
        nameDatePairs = new HashMap();
    }

    public static VisitTracker getInstance() { return instance; }

    synchronized public void addVisit(String userName) {

        nameDatePairs.put(userName, new Date());
    }
```

```
synchronized public Date lastVisit(String name) {

        Date d = (Date)nameDatePairs.get(name);
        return d;

    }
}
```

User Authorization

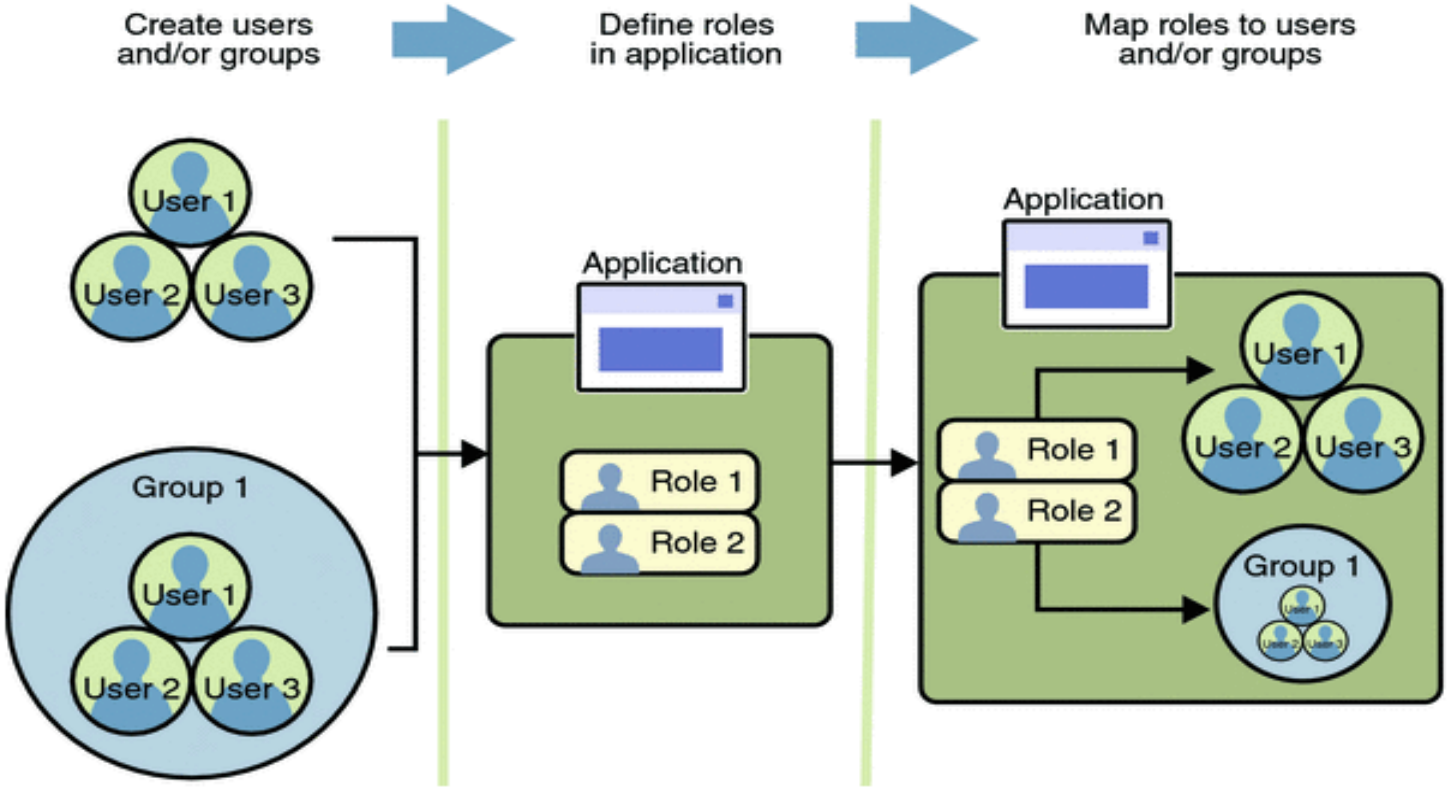
- Administered by the web server – Glassfish
- A realm is a set of name, password, role triples
- Different realms are possible - RDBMS or LDAP
- Use the GlassFish administrator tool at localhost:4848
- The GlassFish admin-realm is for the app server.
- Manage users under the file realm.
- Security requirements are defined in the application's web.xml.
- The role is specified in the web.xml.
- Those users, who know the password and are assigned the appropriate role, may use the service.
- From within the servlet use String name = req.getRemoteUser(); to access the user name.

Administer GlassFish at port 4848 Select security tag on left



From the J2EE Tutorial

Figure 28–6 Mapping Roles to Users and Groups



The following sections provide more information on realms, users, groups, and roles.

GlassFish Web.xml (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>NameInThisFile</servlet-name>
        <servlet-class>UserAuthorizationDemo</servlet-class>
    </servlet>
```

GlassFish Web.xml (2)

```
<servlet-mapping>
    <servlet-name>NameInThisFile</servlet-name>
    <url-pattern>/UserAuthorizationDemo/*</url-pattern>
</servlet-mapping>
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

GlassFish Web.xml (3)

```
<security-constraint>
   <web-resource-collection>
      <web-resource-name>SomeProtection</web-resource-name>
      <url-pattern>/UserAuthorizationDemo/*</url-pattern>
      <http-method>GET</http-method>
   </web-resource-collection>
   <auth-constraint>
      <role-name>student</role-name>
   </auth-constraint>
</security-constraint>
<login-config>
   <auth-method>BASIC</auth-method>
   <realm-name>file</realm-name>
</login-config>
<security-role>
   <role-name>student</role-name>
</security-role>
</web-app>
```

Sun-web.xml

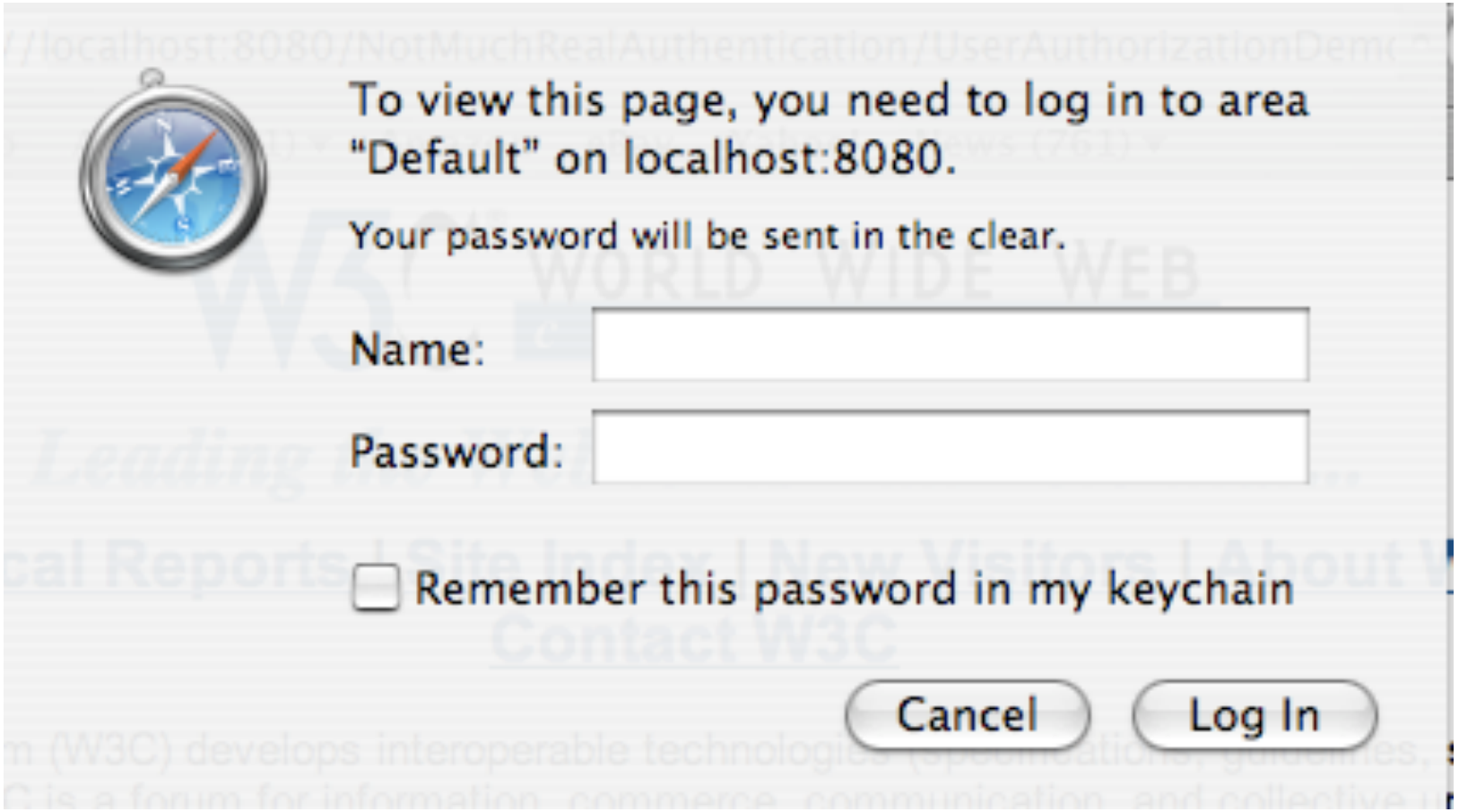
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- DOCTYPE NOT SHOWN -->
<sun-web-app error-url="">
  <context-root>/UserAuthorizationProject</context-root>
  <security-role-mapping>
    <role-name>student</role-name>
    <principal-name>Mike</principal-name>
    <principal-name>Jethro</principal-name>
  </security-role-mapping>
  <class-loader delegate="true"/>
  <jsp-config>
    <property name="keepgenerated" value="true">
      <description>Keep a copy of the generated servlet class' java code.</description>
    </property>
  </jsp-config>
</sun-web-app>
```

index.jsp

```
<html>
    <head>
      <meta http-equiv="Content-Type" content="text/html;
        charset=UTF-8">
      <title>The UserAuthorzationDemo index.jsp page</title>
    </head>
    <!-- GetForm.html -->
    <body>
      <form method="get" action="UserAuthorizationDemo">
        Only authorized visitors please<p>
      <input type = "submit">
     </form>
    </body>
</html>
```

```
// UserAuthorizationDemo.java
// This servlet reads from GlassFish and finds the name of the
// authorized user. It then adds it to a hash table storing
// the time of this visit. It makes use of VisitTracker.

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class UserAuthorizationDemo extends HttpServlet {

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

```
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    String name = req.getRemoteUser();    // ask the server
    if(name == null) {
        System.out.println("The system administrator should protect" +
                           " this page.");
    }
    else {

        out.println("This user was authorized by the server:" + name);
        VisitTracker visit = VisitTracker.getInstance();
        Date last = visit.lastVisit(name);
        if(last == null) out.println("Welcome, you were never here before");
        else out.println("Your last visit was on " + last);
        visit.addVisit(name);
      }
    }
}
```


To view this page, you need to log in to area "Default" on localhost:8080.
Your password will be sent in the clear.
Name:
Password:
Remember this password in my keychain
Cancel
Log In

Some Non-Functional Characteristics

Interoperability ?
Concurrency?
Security? Eve? Mallory?

Suppose we were to configure the web server to do SSL.

Interoperability ?
Concurrency?
Security? Eve? Mallory? If we are using SSL is user authentication still useful?

HTTP Cookies

- Perhaps we don't want to authenticate our users but would still like to interact with them using a stateful application level protocol. Can you give some examples?

- A cookie is a bit of information (name=value pair) sent by a web server to a browser. On subsequent visits, the cookie is sent back to the server.

- The server can use the information as a key to recover information about prior visits. This information may be in a database or a shared object.

- Cookies are read from the request object by calling getCookies() on the request object.

- Cookies are placed in the browser by calling addCookie() on the response object.

Using Cookies

```
// CookieDemo.java

// This servlet uses a cookie to determine when the
// last visit by this browser occurred. It makes use of
// the VisitTracker object.

// Cookies normally expire as soon as the browser exits.
// We want the cookie to last one year and so we use
// setMaxAge(seconds) on the cookie.

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class CookieDemo extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException {

        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();

        Cookie[] c = req.getCookies();
        // If this person has been here before then we should have
        // a cookiedemouser field assigned to a unique id.

        String id = null;
```

```
if (c!=null) {   // we may have the cookie we are after

    for (int i=0;i<c.length;i++) {

        if (c[i].getName().equals("cookiedemouser")) {

            id = c[i].getValue();
            break;
        }
    }
}
```

```
        if (id == null) {
                // They have not been here before and need a
                // cookie. We get a unique string (with respect
                // to this host)and make sure it is of the 'query string' form.
                // It uses the clock. Don't turn the clock back!
                String uid = new java.rmi.server.UID().toString();
                id = java.net.URLEncoder.encode(uid);
                Cookie oreo = new Cookie("cookiedemouser",id);
                oreo.setMaxAge(60*60*24*365);
                res.addCookie(oreo);
        }
        VisitTracker visit = VisitTracker.getInstance();
        Date last = visit.lastVisit(id);
        if(last == null) out.println("Welcome, you were never here before");
        else out.println("Your last visit was on " + last);
        visit.addVisit(id);
    }
}
```

The New Session Tracking API

- Support may vary depending on the server.
- Implemented with cookies or with URL rewriting if cookies fail (URL rewriting requires help from the servlet).
- Every user of the site is associated with a javax.servlet.http.HttpSession object
- The session object can hold any arbitrary set of Java objects.
- Servlets collaborate by accessing the session object.
- The following example abstracts away shared object concerns.
- All valid sessions are grouped together in a HttpSessionContext object

The Session Tracking API

```
// SessionDemo.java
// The session object associated with this user/browser is available
// to other servlets.

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class SessionDemo extends HttpServlet {
```

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

        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();

        // Get the current session object. Create one if none exists.
        HttpSession session = req.getSession(true);

        // Get the Date associated with this session
        Date d = (Date)session.getAttribute("dateofvisit");

        if(d == null)  out.println("Your first time, welcome!");

        else  out.println("Your last visit was on " + d);

        session.setAttribute("dateofvisit", new Date());
} }
```

Some Non-Functional Characteristics

Interoperability ?
Concurrency?
Security? Eve? Mallory?

Suppose we were to configure the web server to do SSL.

Interoperability ?
Concurrency?
Security? Eve? Mallory?