Mindestschutz für JSF-Anwendungen

Sicherheit von Web-Anwendungen

- Web-Anwendungen sind gefährdet!

- Unbekannte Benutzer in unbekanntem Umfeld
- Kriminelle Energie nicht auszuschließen

→ Jede Web-Anwendung muss geschützt werden!

- **Minimalschutz:**
  - **Eingabe-Validierung**
  - **Zugriffsschutz**
  - **Session-Timeout**

… Standardverfahren einsetzen, keine "Bastellösungen"!

Minimalschutz

- Eingabevalidierung:
  - Abfangen von Texteingaben, die Programme in den Server einschleusen (SQL-Injection)
  - *Bedrohung nicht nur durch Eingaben, auch z.B. durch attraktive kostenlose "Fertigteile"*
- Zugriffsschutz (Login)
  - eigentlich Authentifizierung, Autorisierung & Zugriffsschutz
  - Handlungen können registrierten Benutzern zugeordnet werden
  - *Oft eher zur Datensammlung als zum Schutz eingesetzt*
- Session-Timeout
  - Reduzierung angreifbarer "Dangling Sessions"
  - *Bei Nutzern unbeliebt…*

Eingabevalidierung

→ Sicherheit bietet nur die serverseitige Validierung!

- Frontend-Frameworks bieten (meist) flexible Validierungs-Möglichkeiten

→ unbedingt benutzen!

→hilft gegen Vergessen und Faulheit

- JSF bietet
  - Standard-Validatoren (oft kombiniert mit Konvertern)
  - Benutzerdefinierte Validatoren
    → *beliebige Überprüfungen möglich (z.B. auch mit Regex)*
    → *Jede Eingabe durch einen Validator oder Konverter prüfen!*
  - Bean-Validatoren ("unsichtbar")

Session-Timeout

- Gefahr durch Fremdnutzung einer inaktiven Sitzung
  - Vor Ort am eingeloggten Rechner
    "Rauchen gefährdet Ihre Datensicherheit"
  - Von fern durch "Session Hijacking"

- Alle Server bieten Timeout

  → nutzen

  anwendungsbezogen "vernünftigen" Wert finden.

Zugriffsschutz: JAAS

- Login programmieren?
- Nichts leichter als das:
  Registrieren, in der DB speichern, bei Login prüfen.

→ sehr leicht zu hacken!

JAAS – Java Authentication and Authorization Service

- API zum Festlegen und Überprüfen von Benutzern und Rechten
- Implementierung in sog. Provider-Modulen, die verschiedene Authentisierungsprotokolle bedienen (SQL, LDAP, PKI…)
- Authentifizierungs-Modell *außerhalb des Codes konfigurierbar*
- Application Server enthalten (mehrere) JAAS-Provider-Module "serverbasierte Authentifizierung"

Ebenen des Zugriffschutzes

- Authentifizierung
  (Authentification)

  WER IST DAS?
- Autorisierung
  (Authorization)

  WER DARF WAS?
- Zugriffsüberprüfung
  (Checkpoint)

  DARF DER DAS?

Zugriffschutz: Deklarations-Elemente

- Zugangsdaten (Credentials)
  - **Benutzer** als „Principals“
  - Evtl. Benutzergruppen
- Rechtemodell
  - Benutzerbasiert -- aufwändig, wenig flexibel
  - **Rollenbasiert**
- Geschützte Ressourcen
  - **Datei- oder verzeichnisbasiert**
  - Funktionsbasiert
  - Zustandsbasiert
  - Beliebig

Benutzer mit der Rolle <<Aufsicht>> dürfen die Stechuhrseiten aufrufen

Authentifizierung

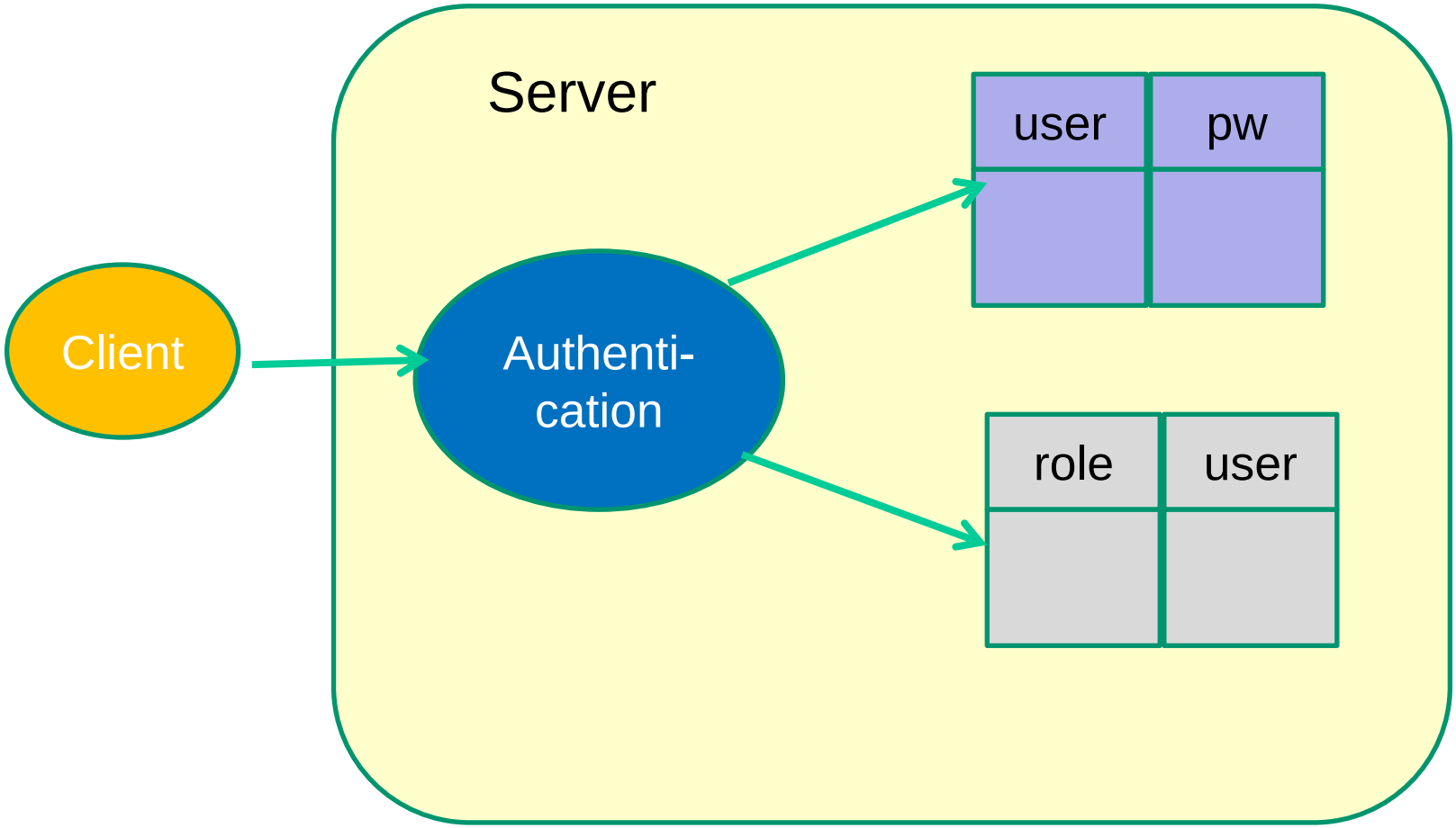
- Application-Server unterstützt Authentifizierung:
- Login setzt Identität und Rolle der Session.
- Identität und Rolle können aus dem Session-Kontext jederzeit abgefragt werden

- Technik ist *im Prinzip* egal (Fingerprint, Eyescan, …)
- Principal und Rolle werden ermittelt und gespeichert.

Wichtig:

- Credentials werden außerhalb der Anwendung (nämlich im Container) verwaltet – weniger leicht hackbar!

Zugriffsschutz-Schema:



Benutzer- und Rollen-Spezifikationen (Tomcat)

- In Tomcat Spezifikation von Benutzern und Rollen in der Datei tomcat-users.xml (unverschlüsselt)

```
<tomcat-users>
      <role rolename="aufsicht"/>
      <role rolename="mitarbeiter"/>
      <role rolename="manager"/>
      <role rolename="admin"/>
      <user username="ide" password="(generated password)"
                                 roles="manager,admin"/>
      <user username="schmidt" password="ctrlx" roles="aufsicht"/>
      <user username="meier" password="mxx" roles="mitarbeiter"/>
      <user username="mueller" password="xxm" roles="aufsicht, mitarbeiter"/>
</tomcat-users>
```

Single-Sign-On

- Tomcat-Server:

  Domain-Authentification
  - für alle Applikationen der Domain
  - umfasst Credentials und Rollen

- Glassfish-Server

  Realm-Authentification (Sun-Technologie)
  - für alle Applikationen eines Realms (Gültigkeitsbereichs)
  - beliebig viele Realms möglich
  - unterschiedliche Sicherung der Credentials:

    Typ File – Speicherung verschlüsselt in einer Datei

    Typ Certificate – Speicherung in einer zertifizierten Datenbank
  - drei Realms vorkonfiguriert: "file", "admin", "certificate"

  Rollen-Zuordnung applikations-spezifisch
  - Datei sun-web.xml

Benutzerspezifikation (Glassfish)

- Eingabe von Usernamen und Passwörtern über die Admin-Konsole des Servers:
- hier in das vorkonfigurierte Realm "file" (mit File-Sicherheit)

Rollen-Spezifikationen in der Applikation (Glassfish)

- Zuordnung von Usernamen zu Rollen in sun-web.xml (auch über einen Wizard)

```
<security-role-mapping>
    <role-name>aufsicht</role-name>
    <principal-name>schmidt</principal-name>
</security-role-mapping>
```

Autorisierung und Zugriffsprüfung

- Durch den Server:
  - applikationsspezifische Rechtezuweisung
  - zweistufige Zuordnung
    Rolle → Zugriffsberechtigungen
    (paradox "security constraint" genannt)
    Zugriffsberechtigung → Resourcen
    (Dateien, Verzeichnisse)
  - Spezifikation im Deployment Descriptor web.xml.

- Durch die Applikation ("händisch"):
  - Zugriff auf Principal und Rolle über den SessionContext,
    (Methoden getCallerPrincipal(), isCallerInRole())
  - Jede zu schützende Aktion muss Überprüfung enthalten

  → aspektorientierte Überprüfung durch JSF-PhaseListener

Rollenbasierte Ressourcensicherung (beide Container)

- Zuordnung von Ressourcen zu Rollen in web.xml (Wizard)

```
<security-constraint>
    <display-name>AufsichtBerechtigung</display-name>
    <web-resource-collection>
      <web-resource-name>AufsichtResource</web-resource-name>
      <url-pattern>/stechuhr/*</url-pattern>
      <http-method>GET</http-method>
       <http-method>GET</http-method>
      <http-method>DELETE</http-method>
    </web-resource-collection>
    <auth-constraint>
      <role-name>aufsicht</role-name>
    </auth-constraint>
</security-constraint>
```

In Glassfish zusätzlich:

- Zuordnung der Applikation zum Realm (Schutzbereich) in web.xml:

```
<login-config>
      <auth-method>BASIC</auth-method>
      <realm-name>file</realm-name>
 </login-config>
```

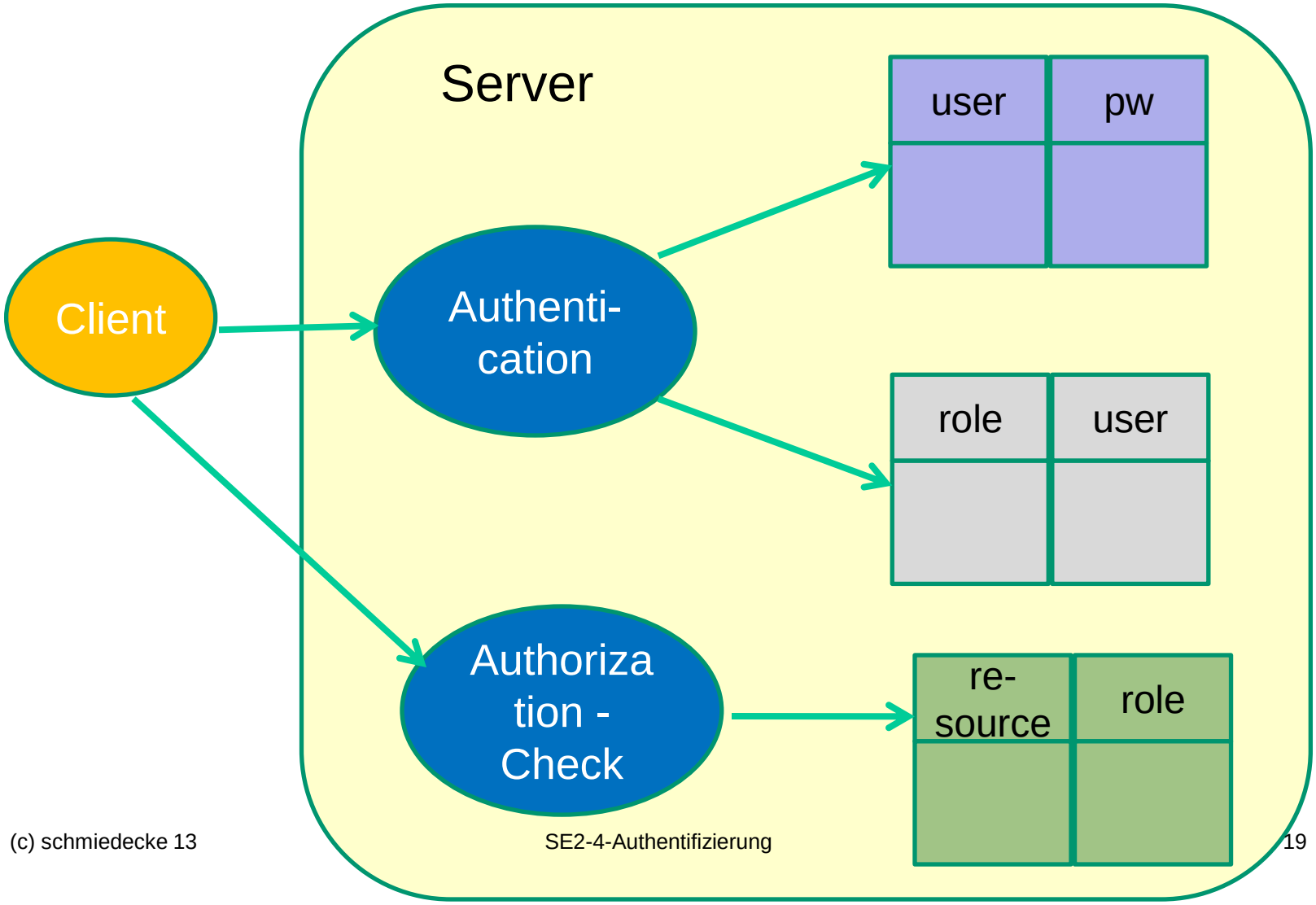
Sicherheits-Annotationen

- Nicht in JSF, aber in EJB 3
- Schutz von Klassen und Methoden per Annotation
- @DeclareRoles, @RolesAllowed, @PermitAll, @DenyAll

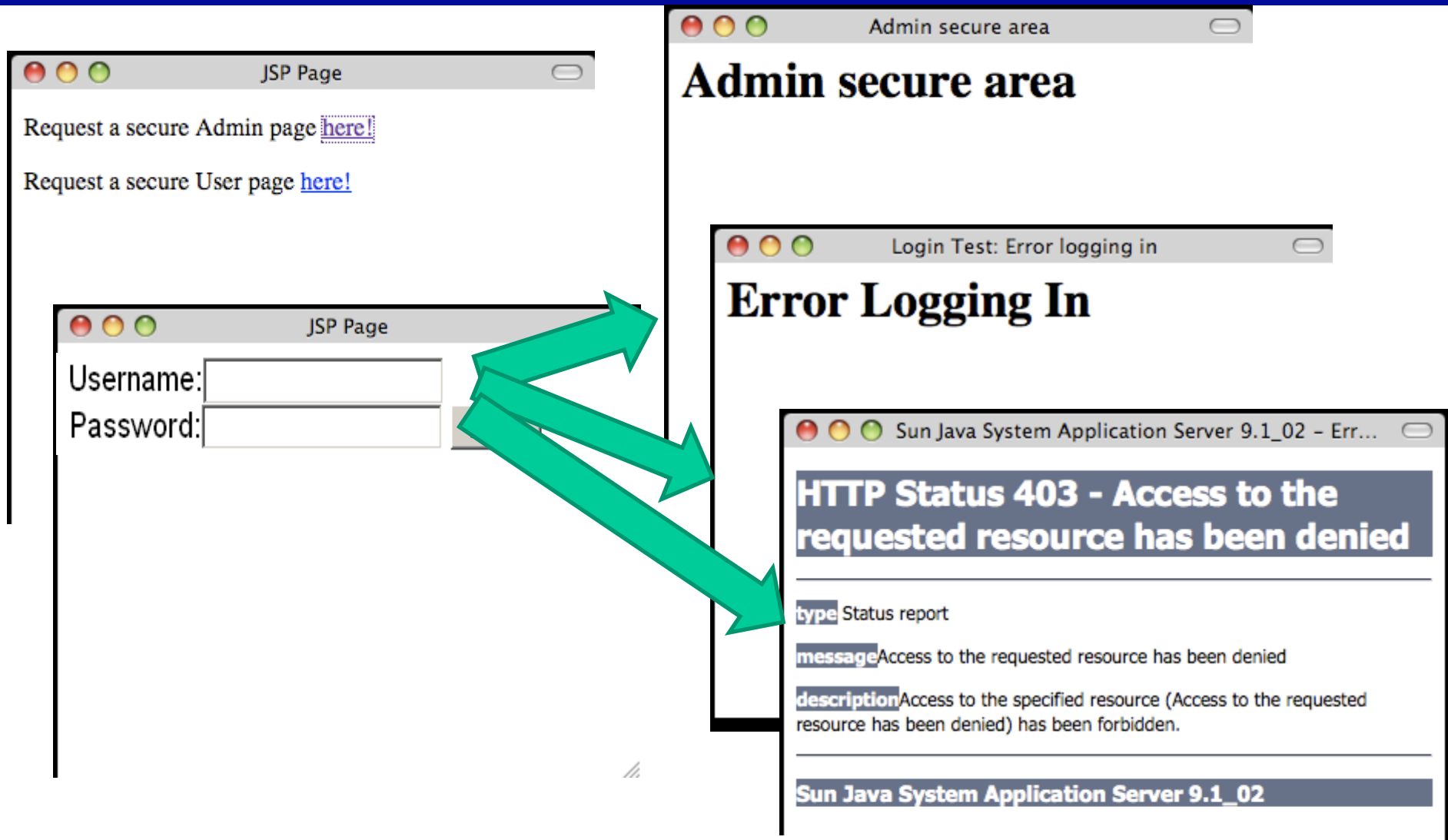
```
@Stateless
@DeclareRoles(value={"administrator", "kunde"})
@SecurityDomain(value="knaufsecurity")
public class SecuredBean implements Secured
{  // [ … ]

  @RolesAllowed(value={"administrator"} )
  public void forAdminOnly()
  {
    logger.info("forAdminOnly");
  }

  @RolesAllowed(value={"kunde"} )
  public void forKundeOnly()
  {
    logger.info("forKundeOnly");
  }  // [… ]
```

Bsp. aus http://www.cs.hs-rm.de/~knauf/KomponentenArchitekturen2008/security/index.html

Zugriffsschutz-Schema:



Anforderung einer gesicherten Seite
(Beispiel aus Tutorial, s.u.)



http://netbeans.org/kb/docs/web/security-webapps.html

Logout

- Die eingegebenen Login-Daten bleiben bis zum Beenden der Sitzung gültig.
- Zum Beenden muss die Session ungültig gemacht werden,
  etwa durch folgendes JSP-Script:

```
<% request.getSession(false).invalidate(); %>
```

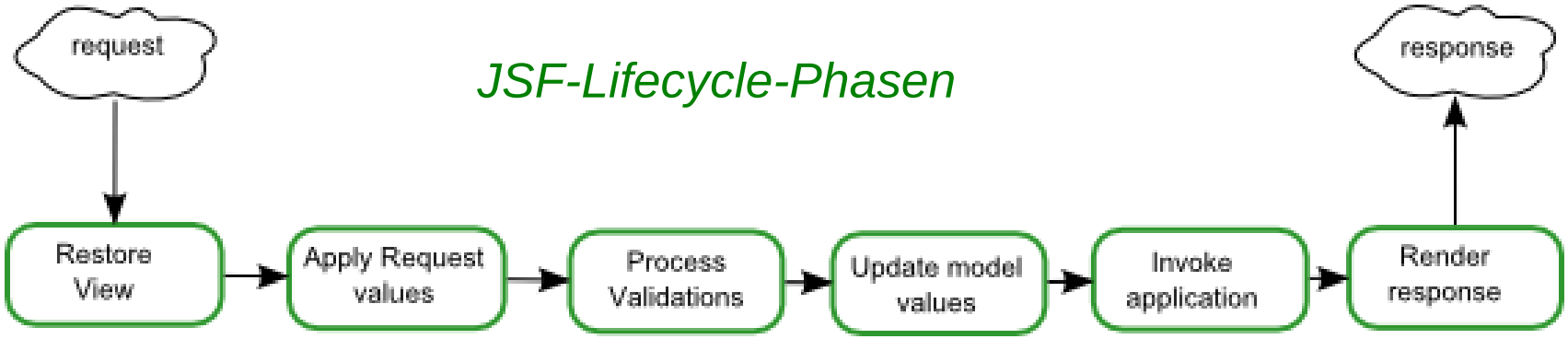
- **Logout**: zu einer JSP mit der obigen Zeile verzweigen.
- **Timeout**: im web.xml konfigurieren, führt zum Aufruf der invalidate-Methode durch den Server:

```
<servlet-mapping>
      <servlet-name>Faces Servlet</servlet-name>
      <url-pattern>/faces/*</url-pattern>
   </servlet-mapping>
   <session-config>
      <session-timeout>
         30
      </session-timeout>
   </session-config>
```

Aspektorientierte Autorisierungsprüfung mit PhaseListenern

- Autorisierung ist eine „Querschnittsaufgabe“ –
  das klingt nach aspektorientierter Programmierung

- PhaseListener bieten ein verwandtes Konzept:
  - Der JSF-Lifecycle ist in abgegrenzte Phasen eingeteilt
  - Vor und nach jeder Phase wird ein Ereignis erzeugt, für das ein PhaseListener registriert werden kann
  - Damit werden PhaseListener querschnittsartig vor oder nach bestimmten Programmteilen ausgeführt



Phasengebundene Autorisierungsprüfung: Prinzip

- PhaseListener für die Autorisierungsprüfung:
  - Zu Beginn einer Seitenanfrage Login-Status-Prüfung
  - Typischerweise am Ende der ersten Phase:
    „after phase“ --- PhaseId.RESTORE_VIEW
  - In der Methode afterPhase(PhaseEvent ev) wird dann eine beliebige Überprüfung aufgerufen.
  - Bei Misserfolg wird zu einer Fehlerseite navigiert,
    entweder fest per „redirect“
    oder mit dem NavigationHandler entsprechend einer konfigurierten Navigation (flexibler)

Phasengebundene Autorisierungsprüfung: PhaseListener-Implementierung

```
import javax.faces.event.PhaseListener;
[…]
public class Authentificator implements PhaseListener {
   public PhaseId getPhaseId ()
     { return PhaseId.RESTORE_VIEW;   }

   public void afterPhase(PhaseEvent evt) {
      FacesContext ctx = evt.getFacesContext();
      if (! loginOK(ctx)) {
         NavigationHandler nav =
            ctx.getApplication().getNavigationHandler();
        nav.handleNavigation(ctx, null, „not authorized“);
      }
    }

   public boolean loginOK(FacesContext ctx) {….}
}
```

Phasengebundene Autorisierungsprüfung : Konfiguration in faces-config.xml

- Registrierung des PhaseListeners

```
<lifecycle>
    <phase-listener>
            com.example.event.Authenticator
    <phase-listener>
</lifecycle>
```

- Navigationsregel

```
<navigation-rule>
  <navigation-case>
      <from-outcome>not authorized</from-outcome>
       <to-view-id>/autherror.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
```

Zusammenfassung Authentifizierung und Autorisierung

- Login - Mindestabsicherung gegen Missbrauch
  - Eigenschaft der Session
  - Verwaltung durch den Server (dringend empfohlen!)
  - Logout und Timeout nicht vergessen!

- Rechteverwaltung:
  - Benutzerrollen
  - Zugriffsrechte auf Resourcen rollenabhängig
  - Einfachste Form: Resource = Seite oder Verzeichnis

- Verschiedene Techniken
  - Basis-Autorisierung des Seitenzugriffs durch den Server
  - Annotationsbasierte Autorisierung von bis auf Methodenebene (EJB)
  - Frei programmierbare Überprüfung durch JSF-PhaseListener
  - Händische Programmierung in allen zu schützenden Aktionen
    - nicht zu empfehlen!

Das war's (weitestgehend) in Sachen Frontend