15.–18. 09. 2008
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Verhext!

Wicket und seine Konzepte

Carl-Eric Menzel - Olaf Siefart

Senacor

Motivation - Warum Wicket?

- Motivation
  - Warum Wicket?
- Wicket
  - Grundlagen
  - Eigene Komponenten
  - Templating
  - Security
- Ausblick
  - Wicket 1.4

Die Webframework-Frage

- die Auswahl des Webframeworks ist meistens eine unternehmensweite Entscheidung
- der Auswahlprozess ist schwierig durch firmenpolitisch motivierte Restriktionen
  - Entscheidung wird nicht auf Basis vernünftiger Evaluierung getroffen
  - Kriterium „etablierte Technologien“ einzusetzen wird in erster Linie unter dem Blickwinkel der Verbreitung gesehen

Die Webframework-Frage

- Es gibt eine Vielzahl JEE Servlet-basierter Frameworks mit zum Teil sehr unterschiedlichen Ansätzen bzgl.
  - Anwendungsarchitektur
  - View-Technologien
  - Zustandsverwaltung
  - Seitenmodell (Single Page, Multi-Page oder Mischformen)

- Akzeptanz in der Entwickler-Community
- Stabilität der Framework-API
- Roadmap und Releasezyklen

Wer die Wahl hat, hat die Qual ...

	Architektur-muster	Programmier modell	State-Management	Conversation-Unterstützung	Ajax-Unterstützung	Trennung UI-Logik und Design	Unternehmen-seinsatz ?
Struts	MVC2	-	-	-	-	-	-
Spring MVC	MVC2	0	-	-	-	-	+
JSF	Component/Event	0	+	-	-	-	-
Tapestry	Component/Event	+	+	-	+	0	-
Wicket	Component/Event	++	++	+	++	++	+
GWT	Component/Event	+	0		++	+	0
Spring Webflow	MVC2	+	0	++	n/a	n/a	+
JBoss Seam	Component/Event	0	+	++	+	+	-
Shale	Component/Event	0	+	+	0	+	-

Warum Wicket?

- **Komponenten-orientierte Entwicklung:** erlaubt die einfache Erstellung und (Wieder-)Verwendung von UI-Elementen.
- **Wartbarkeit:** Durch die komplette Umsetzung der UI-Logik in Java existiert eine gute Vorraussetzung für Erweiterungen, Wiederverwendung und Refactorings
- XHTML-Templates anstatt JSP, Umsetzung von statischem HTML
- **kostengünstige Entwicklung:** Einfacher Technologiestack, Designer für HTML-Templates, Abstraktion von Web-Technologie-Problemen, Verwendung fertiger Komponenten
- **Betrieb:** Unterstützung für Cluster, leichtgewichtige Sessions für die Replikation, viefältige Diagnose + Monitoringmöglichkeiten
- Ajax-Fallback-Komponenten, zur Unterstützung vielfältiger Szenarien (Online-Banking)

Wicket Schwachstellen

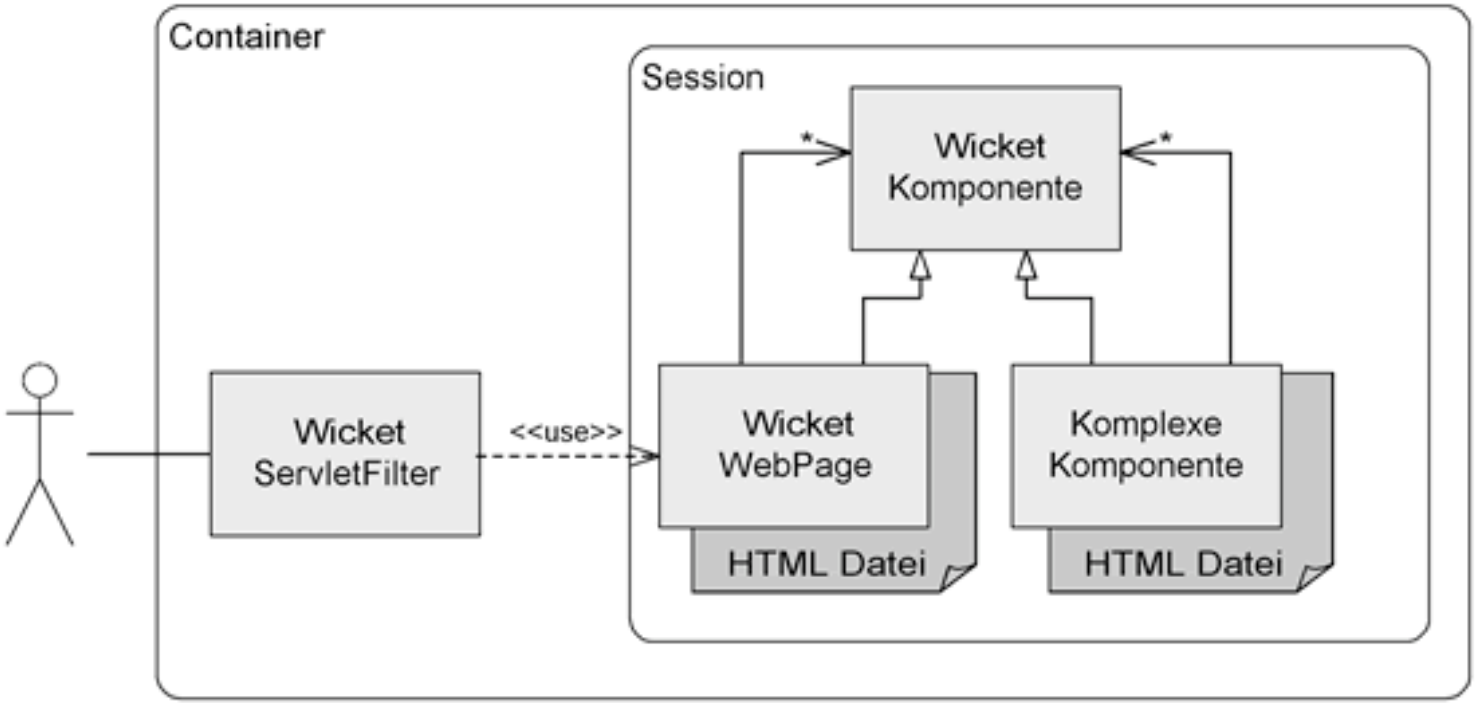
- Dokumentation: Lückenhafte Online-Dokumentation im WIKI-Style gefährdet die Akzeptanz. Besser seit „Wicket in Action“
- Stabilität/Reifegrade/ Dokumentation der Third-Party-Komponenten
- Support/KnowHow am Markt
- Stabilität, Reifegrade und Dokumentation der Third-Party-Komponenten
- Intensive Nutzung von (Anonymous) Inner Classes ist gewöhnungsbedürftig
- Wicket ist grundsätzlich für andere Markup-Typen (WML) geeignet, bietet aber selbst keine alternative Implementierung

Wicket - Grundlagen

- Motivation
  - Warum Wicket?
- Wicket
  - Grundlagen
  - Eigene Komponenten
  - Templating
  - Security
- Ausblick
  - Wicket 1.4

Wicket Grundlagen

- Wie sieht eine Wicket Komponente aus?
  - Java + HTML: Code-Beispiel (Login-Seite)



Wicket Grundlagen

- HelloWorld.html

```
<html>
<body>
  <span wicket:id=”hello”>Hi!</span>
</body>
</html>
```

Wicket Grundlagen

- HelloWorld.java

```
public class HelloWorld extends WebPage {
  public HelloWorld() {
    this.add(new Label(“hello”, “Hallo,
Welt!”);
  }
}
```

Eigenschaften von Komponenten

- Seiten sind Komponenten und bilden die Wurzel des Komponentenbaumes
- Komponenten werden mit new erzeugt und der Vaterkomponente mit add() hinzugefügt
- Komponenten können das Ziel eines Benutzerevents sein (z.B. Klick auf einen Link)
- Events werden in Listener-Methoden der Komponenten behandelt

Eigenschaften von Komponenten

- Event-Listener sind abstrakte Methoden und werden meist unter Verwendung von anonymen Klassen implementiert:

```
Link addToBasket = new Link("add") {
        public void onClick() {
                // ...
                setResponsePage(BasketPage.class);
        }
};
```

- Seitenwechsel erfolgt über:
  - *setResponsePage(Class pageClass)*
  - *setResponsePage(Page page)*

Wicket Model

- Werden für die Darstellung von Domain-Daten und zur Aktualisierung von Domain-Daten (z.B. über Formularkomponenten) verwendet:

```
public interface IModel extends IDetachable{
  Object getObject();
  void setObject(final Object object);
}
```

- Implementierungen als Wrapper oder Delegate
  - Verwendung von Model als Wrapper

    TextField name = new TextField("name", new Model(""));
  - PropertyModels: über EL-ähnliche Ausdrücke

```
TextField email = new TextField("email", new PropertyModel(this,
"registration.email"));
```

Wicket Model

- Abbildung von attached und detached state
  - Models können am Ende jedes Requests aufgeräumt bzw. komprimiert werden
    - Aufruf von detach() durch Wicket
  - Laden des Model-Objects über Lazy-Implementierung von getObject().

  - Im detached state wird das Minimum an Information in der Session behalten, das notwendig ist, um das Model zu rekonstruieren
    - z.B. contactID im detached State
    - contactID und Contact-Objekt im attached State

Zentrale Wicket-Klassen

- WicketFilter-Deklaration in der web.xml
- WebApplikation-Klasse
  - legt die Startseite der Anwendung fest
  - anwendungsweite Einstellungen des Frameworks (z.B. Encoding, URL-Strategie, ...)
  - ComponentInstantiationListener --> Spring Integration
  - Security-Settings
  - Definition Homepage
- WicketSession: Abstraktion, Subclassing
  - Locale und Style-Einstellung
  - Flash Messages
  - Eigene Erweiterungen möglich (typisch: User-Attribute)

Behaviors

- Wiederverwendbare, gekapselte Verhaltensvorlagen, die durch Komposition zu jeder Komponente hinzugefügt werden können
- mit onComponentTag(Component c, ComponentTag tag) kann das der Komponente zugeordenete Html-Tag manipuliert werden:
  - z.B. Einhängen von Javascript-Eventhandlern (onClick, onBlur, onModify ...)
  - z.B. Modifizieren von class oder style

Behaviors

- SimpleAttributeModifier

```
c.add(new SimpleAttributeModifier("class",
"cssName"));

public void onComponentTag(final Component
  component, final ComponentTag tag) {
  if (isEnabled(component)) {
    tag.getAttributes().put(attribute, value);
  }
 }
```

Ajax-Integration

- Wicket beinhaltet eine Cross-Browser-fähige Ajax-JavaScript-Bibliothek, der Ajax-Support basiert auf Behaviors
- eigene JavaScript-Bibliotheken können über das IHeaderContributor-Interface eingebunden werden
- JavaScript-Eventlistener können über AjaxEventBehavior bequem eingebunden werden:

```
component.add(new AjaxEventBehavior("onClick") {
public void onEvent(AjaxRequestTarget target)
{...}
};
```

- Wichtig: `setOutputMarkupId(true)`

AjaxSelfUpdatingTimerBehavior

```
add(new Form("chatForm") {
 {
 final Vector<String> nachrichten = ...;
 final TextArea textArea = new TextArea
  ("nachrichten", new Model(nachrichten)) {
  {
   add(new AjaxSelfUpdatingTimerBehavior
    (Duration.milliseconds(1500)));
  }
 };
 textArea.setOutputMarkupId(true);
 add(textArea);
 }
});
```

AjaxButton

```
TextField textField = new TextField
 ("neueNachricht", new Model(""));
 textField.setOutputMarkupId(true);
 add(textField);

 add(new AjaxButton("senden") {
  protected void onSubmit(AjaxRequestTarget
   target, Form form) {
     ...
     target.addComponent(textArea);
     target.addComponent(textField);
  }
 });
```

Eigene Komponenten

- Motivation
  - Warum Wicket?
- Wicket
  - Grundlagen
  - Eigene Komponenten
  - Templating
  - Security
- Ausblick
  - Wicket 1.4

Erstellung einer eigenen Komponente

- Häufiges Vorgehen:
  - “Normale” Page erstellen
  - Wiederverwendbare Teile identifizieren
  - Diese Teile herausschneiden, in eigenes HTML und eigene Klasse packen
  - Fertig. Auslieferung ggf. auch als .jar, inklusive allen zusätzlichen Dateien

Komponentisieren einer Seite - HTML

```
<html>
 <body>
  <form wicket:id=”userform”>
  Username: <input type=”text”
wicket:id=”username”/>
  Email: <input type=”text” wicket:id=”email”/>
  Passwort:<input type=”password”
wicket:id=”passwort”/>
                  <input type=”password”
wicket:id=”passwort2”/>
  <input type=”submit” wicket:id=”submit”/>
  </form>
 </body>
</html>
```

Komponentisieren einer Seite - Java

```
public class CreateUserSimplePage extends WebPage {
 private User user = new User();
 public CreateUserSimplePage() {
  Form form = new Form("userform", new
CompoundPropertyModel(user)) {
     @Override protected void onSubmit() {
       // speichern...
     }
    };
  form.add(new TextField("username"));
  form.add(new TextField("email"));
  form.add(new PasswordTextField("passwort"));
  form.add(new PasswordTextField("passwort2"));
  form.add(new Button("submit"));
  this.add(form);
 }
}
```

Komponentisieren einer Seite

Komponentisieren – HTML neue Seite

```
<html>
 <body>
  <form wicket:id=”userform”>
  <div wicket:id=”userdetails”/>
  <div wicket:id=”userpassword”/>
  <input type=”submit” wicket:id=”submit”/>
  </form>
 </body>
</html>
```

Komponentisieren – Java neue Seite

```
public class CreateUserPanelizedPage extends WebPage {
 private User user = new User();
 public CreateUserPanelizedPage() {
  Form form = new Form("userform", new
CompoundPropertyModel(user)) {
     @Override protected void onSubmit() {
       // speichern...
     }
    };
  form.add(new UserDetailsPanel("userdetails", false));
  form.add(new UserPasswordPanel("userpassword",
false));
  this.add(form);
 }
}
```

Komponentisieren – HTML Panel 2

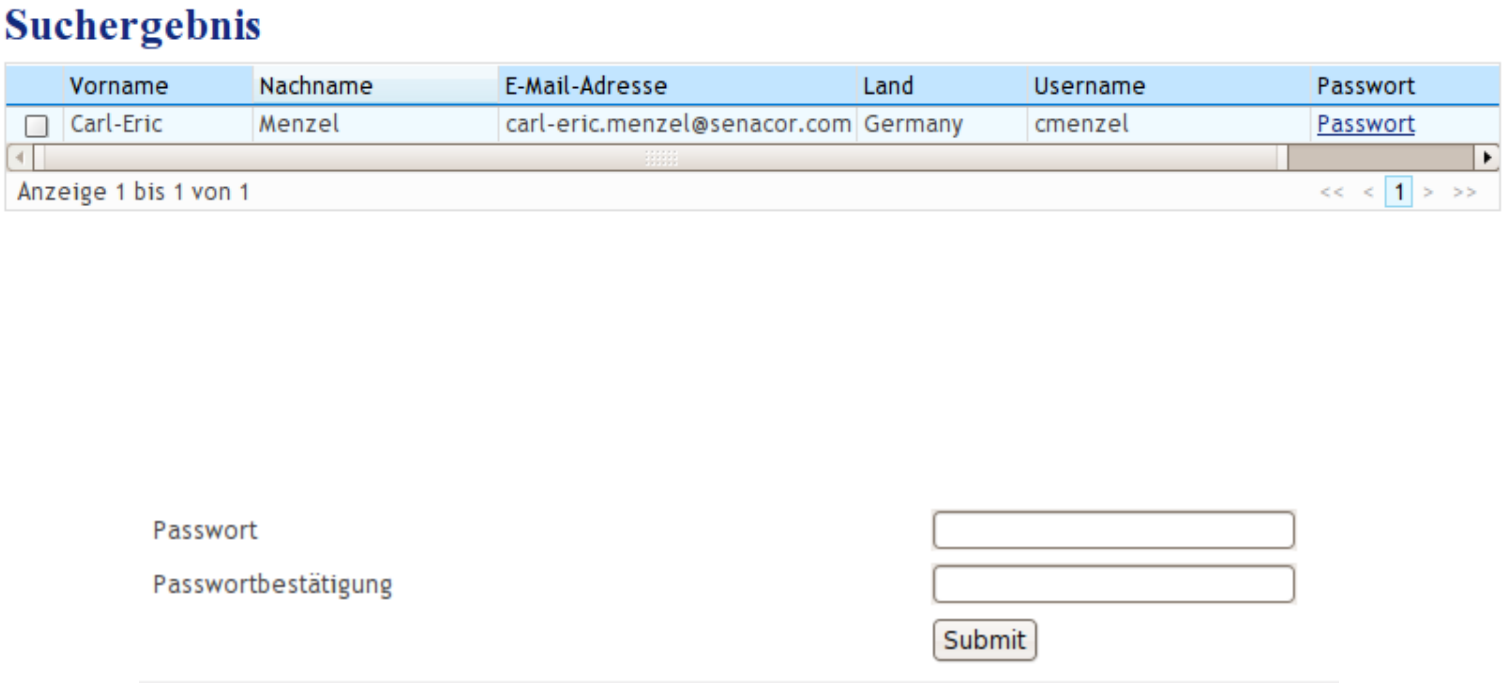
```
<html>
 <body>
  <wicket:panel>
  <form wicket:id=”userform”>
  Passwort:<input type=”password”
wicket:id=”passwort”/>
                  <input type=”password”
wicket:id=”passwort2”/>
  <input type=”submit” wicket:id=”submit”/>
  </form>
  </wicket:panel>
 </body>
</html>
```

Komponentisieren – Java Panel 2

```
public class UserPasswordPanel extends Panel {
 public UserPasswordPanel(String id, final boolean
showButton) {
  super(id);
  Form form = new Form("passwordform") {
   @Override protected void onSubmit() {
    UserPasswordPanel.this.onSubmit();
   }
  };
  form.add(new PasswordTextField("passwort"));
  form.add(new PasswordTextField("passwort2"));
  form.add(new Button("submit") {
   @Override public boolean isVisible() {
    return showButton;
   }});
  this.add(form);
 }
 protected void onSubmit() { }}
```

Komponentisieren einer Seite

Komponentisieren einer Seite



Komponentisieren – Suchseite

```
new AjaxLabelLink(componentId, "Passwort") {
     @Override
     protected void onClick(AjaxRequestTarget target) {
      setResponsePage(new EditPasswordPage(
        (User) rowModel.getObject(), getPage()));
     }
    };
```

Komponentisieren – Passwortseite

```
public class EditPasswordPage extends WebPage {
 public EditPasswordPage(User user, final Page returnPage)
{
  this.setModel(new CompoundPropertyModel(user));
  this.add(new UserPasswordPanel("userpassword", true) {
   @Override protected void onSubmit() {
    // update password here
    setResponsePage(returnPage);
   }
  });
 }
}
```

Header Contributions

- Komponenten brauchen oft eigenes CSS oder Javascript, das im Header der Seite landen soll.

- Im Template:

```
<wicket:head>
  <style ... />
</wicket:head>
<wicket:panel> ... </wicket:panel>
```

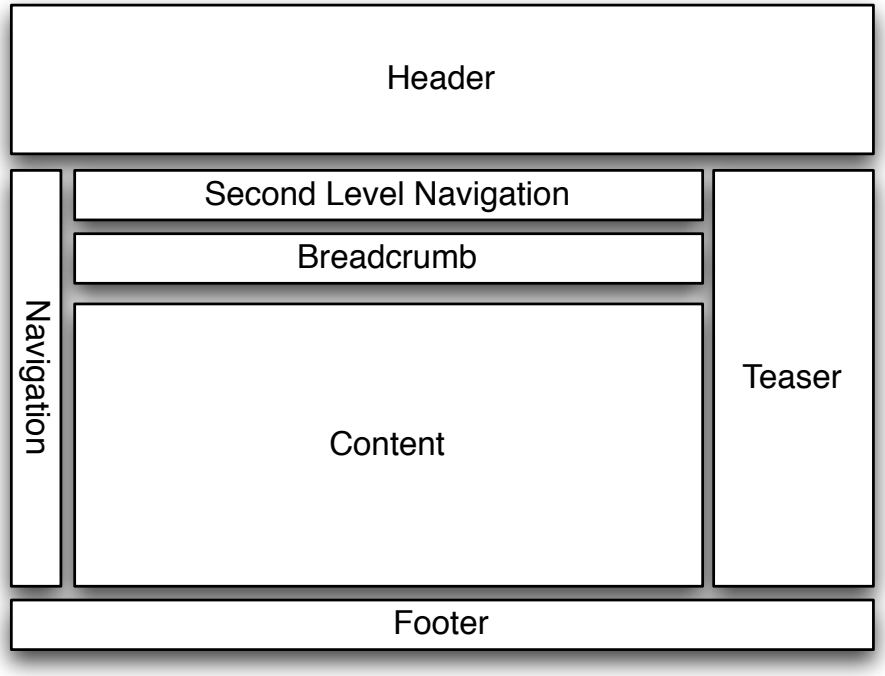
- Im Code: via IHeaderContributor

Wicket - Templates

- Motivation
  - Warum Wicket?
- Wicket
  - Grundlagen
  - Eigene Komponenten
  - Templating
  - Security
- Ausblick
  - Wicket 1.4

Wofür?

- Aufteilung des Seitenlayouts in versch. Bereiche



- Früher: verschachtelte Tabellen
- Heute: CSS-Frameworks wie YUI, YAML, Mollio, ...

Basis: Pages, Panels, Fragments, Repeater

- **Pages:** können allein gerendert werden.
- **Panel:** Kapselung und Layouting von anderen Komponenten (im Markup über divs)
- **Fragments:** Inline Panel, für die keine eigenen HTML-Dteien benötigt werden
- **Repeater:** RepeatingView vs. ListView

Fragments

-
```
<wicket:panel>
  <div wicket:id="login"></div>
  <wicket:fragment wicket:id="loginFragment">
    ...
  </wicket:fragment>
</wicket:panel>
```
-
```
add(loginFragment);
```
-
```
private class LoginFragment extends Fragment {
  public LoginFragment(String id, String mId) {
    super(id,mID,AuthenticationStatePanel.this);
  }
}
```

Markup Inheritance

- Seitenaufbau über Vererbung innerhalb von Page-Klassen (Oberklassen sind abstrakt)
- der Content-Bereich wird von der konkreten Seitenklasse implementiert
- Definition des zu vererbenden Markup-Bereichs:

```
<!--  Main Content / Middle  -->
<div id="content"><wicket:child /></div>
```

- Deklaration der Implementierung

```
<wicket:extend> .... </wicket:extend>
```

- Nur eine Vererbung pro Komponente (analog Java) möglich
- pro Seite eine HTML-Layout-Datei notwendig

Composition

- Anstatt Vererbung wird die Seite komplett aus einzelnen Panels zusammengesetzt
- Häufiges Pattern: Panel-Repeater-Komponente, die alle Panels nacheinander rendert

```
new RepeatingView(id) {
  protected void onPopulate() {
    removeAll();
    for (Panel panel : panels) {
     add(panel);
   }
  }
};
```

Composition

- Ermöglicht Pages ohne HTML-Layout-Dateien

```
public ProjectDetailsPage
  (PageParameters pageParameters) {
  long projectId = pageParameters.getLong
    ("projectId");
  Project project = projectManager.retrieve
    (projectId);
  middleColumn.getListPanel().addPanel(
    new EditableTreeTablePanel("panelItem",
      project));
 }
}
```

Bewertung

- sehr gute Templating-Mechanismen, CSS-Frameworks können problemlos als Basis verwendet werden
- für YUI existiert im wicket-extension Projekt eine Umsetzung
- Durch geschickte Wahl der Mechanismen kann die Erzeugung von HTML-Layout-Dateien auf die notwendigen beschränkt werden (Steigerung der Entwicklungseffizienz)

Wicket - Security

- Motivation
  - Warum Wicket?
- Wicket
  - Grundlagen
  - Eigene Komponenten
  - Templating
  - Security
- Ausblick
  - Wicket 1.4

Wicket Security Konzepte

- Authentifizierung vs. Autorisierung
- Authentifizierung
  - Der Benutzer authentifiziert sich mit UserName + Password
  - Diese Informationen werden geprüft, die Rolle identifiziert, die Informationen in der Session gespeichert
  - Unsere Anwendung passt sich an die Rolle des Users an

- Autorisierung: IAuthorizationStrategy
  - `boolean isInstantiationAuthorized(Class componentClass);`
  - `boolean isActionAuthorized(Component component, Action action);`

IAuthorizationStrategy

- Actions: Render, Enable
- Konfiguration in der Applikation-Klasse
- Eigene Implementierung von isActionAuthorized

```
if (action.equals(Component.ENABLE)) {
 Class<? extends Component> c = component.getClass();
 if (c.getAnnotation(Admin.class) != null) {
  User user = WBSSession.get().getUser();
  return (user != null && user.isAdmin());
 }
 return true;
}
```

Wicket Abstract Security Platform (WASP)

- Stellt eine API für Wicket-Security zur Verfügung
- Erweitert die Wicket Actions: Access, Render, Enable, Inherit
- Über eine ActionFactory können eigene Actions definiert werden
- stellt Implementierung von SecureComponents (SecurePanel, SecureForm, SecureLink, ...) zur Verfügung
- Eigene SecureComponents über:
  - Hinzufügen eines ISecurityChecks
  - Verwendung eines ISecureModels
  - Implementierung von ISecureComponent

SWARM

- Implementierung der WASP-API
- Basis-Applikationklasse: SwarmWebApplication
  - Konfiguration
  - Definition von Aliassen
- Berechtigungen werden über Policy-File konfiguriert:

```
grant principal ${SimplePrincipal} "PROJECT_ADMIN"
{
permission ${ComponentPermission} "${user}",
  "inherit, render";
permission ${ComponentPermission} "${user}", "enable";
};
```

SWARM

- Session-Basisklasse: WaspSession
  - public void login(Object context) throws LoginException
- SWARM: Der Context ist ein LoginContext sein
  - public abstract Subject login() throws LoginException;

```
DefaultSubject userSubject = new DefaultSubject();
user = authenticationManager.login(username, password);
for (Principal principal : user.getRole.getPrincipals()) {
 userSubject.addPrincipal(new SimplePrincipal
  (principal.getName().name()));
}
```

SecureMenu

- Menu wird über Verlinkung realisiert
- Anforderung: MenuItems dürfen nur angezeigt werden, wenn die Zielseite auch gerendert werden kann.
- Realisierung über
  - SecurePanel

```
public class TopLevelMenuItem<T extends
WebPage> extends  SecurePanel
```

  - und LinkSecurityCheck

```
BookmarkablePageLink pageLink = new
BookmarkablePageLink("link", pageClass);
setSecurityCheck(new LinkSecurityCheck(this,
pageClass));
```

Wicket - Security

- Motivation
  - Warum Wicket?
- Wicket
  - Grundlagen
  - Eigene Komponenten
  - Templating
  - Security
- Ausblick
  - Wicket 1.4

Wicket 1.4 und weiter?

- Generics: Diskussion + aktueller Stand
- Wicket 1.5
  Wishlist: ssl, Portal 286
- Wicket-Extensions
- IDE-Unterstützung: Laughing Panda Wicket Bench
- Wicket-Portal: Ein Rückblick auf verschiedene Ansätze

Pax (OSGI)

- Pax ist eine OSGI-basierte Erweiterung zu Wicket
- Erlaubt den Austausch von Teilen der Wicket-Anwendung zur Laufzeit durch Laden/Entladen von OSGI-Bundles
- Der Pax Wicket Service ist ein OSGi-Service der die Erzeugung von Wicket-Applikationen, die auf der OSGI-Platform laufen, unterstützt.
- http://www.ops4j.org/projects/pax/wicket/

Wicket Web Beans

- Wicket Web Beans (WWB) ist ein Komponenten-Framework zum automatischen Anzeigen und Ändern von Java POJOs
- Webseiten werden automatisch auf Basis der POJO-Attribute generiert, können aber auch angepasst werden.
- WWB's BeanForm Komponenten beinhalten Ajax-Funktionalität, die Felder der Form werden im Hintergrund zum Server gesendet und erzeugen so eine Rich Client Illusion.

- http://wicketwebbeans.sourceforge.net

Wicket Databinder

- Databinder ist ein Framework für Wicket-Applikationen, die hauptsächlich mit persistenten Daten arbeiten.
- Das Databinder-Framework unterstützt für die Persistierung Hibernate, ActiveObjects und Cayenne.
- Domänen-Objekte werden mit der ausgewählten Technologie persistiert und können direkt über DataForms angezeigt und editiert werden.

- http://databinder.net/site/show/overview

15.–18. 09. 2008
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Vielen Dank!

Carl-Eric Menzel - Olaf Siefart

Senacor

Senacor

Senacor ist als unabhängiger Berater für IT-Transformationen Überregional präsent

Wesentliche Daten

Größe
- Mitarbeiterzahl 2008: ca. 100
- Davon ca. 85 Professionals

Wachstum
- 20-25% organisches Wachstum pro Jahr
- Unternehmensgröße seit 2002 verdreifacht

Historie
- Gründung 1999 unter dem früheren Namen 100world
- Umbenennung Anfang 2007 im Zusammenhang mit der Übernahme des SOA-Beratungsteams der Deutschen Post

Anteils-eigner
- Management und private Investoren
- Uneingeschränkte Unabhängigkeit