DOWNSHIFTING JSF SECURITY

Thorsten Kisner and Helge Hemmer
Management and Engineering
AHT GROUP AG
Essen, Germany
email: \{t.kisner, h.hemmer\}@aht-group.com

ABSTRACT

Security concepts are an inherent component of web applications. A security module is required to authenticate users and to authorize access to available functions and information of the system, following a set of permission rules defining users and user groups. There are many existing mechanisms for authenticating users in web applications to be included, but the process of authorizing users depends highly on the business processes of the application. In most cases it is not enough to grant or deny access to individual web pages of the applications frontend. Depending on the role based access control model most applications require a permission control on individual User Interface (UI) components such as menu items or action buttons. For more complex applications a single level role model is not able to depict the business processes needs and therefore a multi level orthogonal role model is required which is not provided by many common security concepts. The security module presented in this paper includes such a model and is designed for web applications with a presentation layer implemented in Java Server Faces (JSF) with an Enterprise Java Bean (EJB) backend, both part of the Java Enterprise Edition (EE) standard specification, and utilizes the Context and Dependency Injection (CDI) of Java EE 6. A high customizable and fine granular multi level role based access model is provided with additional features out of the box. The security module provides a seamless integration with tools and concepts for different scopes ranging from page authorization, menu creation and UI component authorization. Complex enterprise applications can also benefit from the documentation tools provided for the security concept itself as well as different reports about the role based access control model. The concept is implemented in a GPL-licensed Java library available for free download and via Maven dependency management and is currently used in several enterprise web applications.

KEY WORDS

Web application, security, framework, JavaEE, JSF

1 Introduction

Keeping data and operations safe and secure is a key requirement to all productive software systems. This paper is focusing on the security aspect of Java Enterprise Edition (EE) based web-applications, that are implemented using the Java Server Faces (JSF) user front end technology. The presented security module targets the same objectives described in [1] and is a part of the AhtUtils Java EE toolset. AhtUtils aims at simplifying the Java EE software development process by offering tools and solutions for demands like creating complex reports in various formats (using the JasperReports engine), multi-language support for Java Persistence API (JPA) entities and database initialization/export of EJBs representing a type or status, along with a set of JSF components.

The security module of AhtUtils is based upon the third version of Seam Security provided by RedHats’s Seam framework. By extending Seam Security with additional data structures and concepts plus the option to combine security features from different scopes into single points of configuration, an orthogonal multi level and fine granular role based access control model [2][3] is easier to integrate in JSF-based enterprise applications.

It benefits from many other AhtUtils elements, for example through the integration with implementation independent menu rendering capabilities directly available as JSF components as a meta model, built-in multi language support, a documentation toolbox or database tools.

The remainder of this paper discusses the related work and positions the proposed approach in Section 2. Section 3 introduces the Java Server Faces (JSF) framework in which domain our proposal is implemented. Before discussing the core implementation itself from a technical point of view in Section 4 we describe in 5 important additional features. Section 6 presents results and ends with an overview on future work.

2 Related Work

The Open Web Application Security project (OWASP) is a global non-profit organisation that aims on presenting all vulnerable points of web application environments available today, hoping that this increases the sensibility of developers and therefore helps to make software more secure. The OWASP is the home of many security related projects, like the Enterprise Security API (ESAPI), a security concept available for different programming languages including PHP and Java. The OWASP analysis of JSF specific security demands (available online at: [4]) is a good start-
ing point to learn about possible security issues within JSF development. Besides research and development of special security issues like the problem of a secure way of serialization of Java objects in [5], the work that is related to this paper is concerning full featured security frameworks offering a range of functionality from authentication, authorization to permission rules and more. There is a small number of major players in the Java EE world, all trying to offer these features in a simple but yet powerful solution that is easy to be integrated in other enterprise software projects. The individual optimization and extension of these frameworks is mostly done within closed source business projects or more academic focused works like in [6].

AhtUtils security is targeted to be Open Source and to fit the needs of real world application development. The solutions to challenges encountered in new development projects can therefore be transferred back to the security module due to its Open Source character. When searching for a basis to develop an enhanced security concept, first of all there is the Java programming language’s own basic security concept. The Java Authentication and Authorization Service (JAAS) [7] is included in the recent Standard Edition and Enterprise Edition of the Java platform. Being a framework aimed to be very flexible in terms of authentication methods and chaining of these, the application developer needs to implement the concrete authentication method(s) on its own. Most Java EE application servers are shipping with JAAS compatible authentication methods like file-based user/password lists, database- or LDAP-connections. Furthermore, there are ambitioned approaches like Seam Security [8], Spring Security [9] (formally Acegi) or Apache Shiro [10]. While Seam Security and Spring Security are focusing on Java EE web application development, Shiro is also designed to be used in a Java SE environment, providing a single concept to be used in both environments. It is easy to use and integrate and the provided 10-minute tutorial helps to lower the entry borders of using it. Spring Security is targeted to support Spring enabled projects and is therefore a good choice within these types of application, as it integrates naturally with them. When it comes to standard Java EE only projects, Seam Security 3 (prior versions are depending on a fully Seam-enabled project and are therefore not a good choice, when dependencies should be limited) is a good enhancement compared to JAAS. All of these frameworks are a good choice in respect to their targeted environment(s). Because of these well-established and mighty products available today, we decided to make use of one of these pieces of software and extending it with simplifying concepts and bringing it together with other features of the AHTUtils toolset.

To reduce transient dependencies, we chose Seam Security 3 as basis for our developments.

3 Java Server Faces

The Java Server Faces (JSF) specification is a framework for developing web applications. It is focused on the server-side implementation the presentation layer but many component libraries also include client side components implemented in JavaScript (JS). Developers benefit from reusable component libraries, can connect client JS events to event handler processed on the server and can link data structures (like lists of objects) with components on the client side to represent them (like tables).

There is a reference implementation Java Specification Request JSR-127 available for JSF. This implementation includes simple GUI elements which have direct equivalents to HTML tags.

3.1 JSF Architecture

The architecture of a JSF application follows strictly the Model-View-Controller (MVC) pattern and is outlined in Figure 1. The FacesServlet Controller processes the complete user interaction and communication between the View and the Model.

The Java part of JSF of JSF is Model and consists of “Managed Beans” and should not be confused with the “data model” of an application. The Beans are implemented as Plain Old Java Object (POJOs) and annotated with to be recognized as Managed Beans. Available sophisticated component libraries as well as the JSP Standard Tag Library (JSTL) build with the page definitions from the developer the View of an application.

3.2 JSF Lifecycle

A typical request and response cycle demonstrated the lifecycle of a JSF request and is shown in Figure 2.

"The state of the component tree is restored from the previous request in the Restore View Phase, in the next phase Apply Request Values all new values are applied to the corresponding components. Before the model values are updated, validations and converters are invoked in phase 3. After updated model values, the phase Invoke Application will be entered and after processing of all events, the response is rendered and sent back to the client. There are shortcuts available from the first four phases to Render"
3.3 JSF Expression Language

The JSF Expression Language (JSF-EL) is processed in different phases of the JSF lifecycle and mainly responsible for updating the model values and later on rendering the response (see Figure 2). The expression itself is embedded in the syntax `#{}`, for example `#{managedBean.datafield}`, or `#{managedBean.method()}` and includes logical or arithmetic expressions which can be combined with method and object-value bindings [11].

4 Applying AhtUtils-Security

4.1 Security Concept

A multi-level security concept is implemented to define all access restrictions to a JSF web application. The core components are views, actions, use cases and roles.

- A view is a single page identified by a unique JSF view identifier.
- Inside a view, the user can perform different actions, for example in a CRUD application this may be creating, updating or deleting an entity.
- To lighten the definition and management of complex system, a use case can combine different views and actions.
- The privileges of a user are described by the granted roles in the system. A role can aggregate different use cases and actions or the aggregation of these defined in a use case.

The components role, use case and view described above can be part of distinct categories to enable development of several independent security domains. Contrary to other security frameworks we don’t allow the nesting of roles to reduce the complexity and error-rate in configuration of the model [12]. The only relationship between an existing model and the proposed security model is the connection between User and Role.

The relationship of all entity components of the security framework can be demonstrated by a concrete implementation and the resulting entity relationship diagram like shown in Figure 3(a). A black dot indicates the many edge in a one-to-many relationship, e.g. one User can belong to many Roles.

![ER diagram](a) ER diagram  ![Orthogonal Roles](b) Orthogonal Roles

Figure 3. Security concept

Attaching a role to different categories allows the definition of independent role domains which can be described as a context specific orthogonal role concept. While all users belong to one or more system role, they may additionally belong to another context specific role domain. The example in Figure 3(b) demonstrates this with project specific roles which can vary for different projects.

4.2 Definition of Security Settings

The security settings are defined in XML files, at least the files views.xml and roles.xml are required. All elements support XML lang and description child elements for the self documentation system (see Section 5.2).

An example for a minimal roles.xml is shown in Listing 1. For the system administrator there are two views allowed: The admin and the adminUsers page. It is not relevant in this stage if both pages are on the same hierarchy level or if the adminUsers is on the next hierarchy level compared to admin, hierarchies are handled for menu creation in Section 5.4.

Listing 1. Example of roles.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<security>
  <category code="system">
    <roles>
      <role code="systemAdmin">
        <views>
          <view code="admin"/>
          <view code="adminUsers"/>
        </views>
      </role>
    </roles>
  </category>
</security>
```
Each role refers to one or more view definitions which are shown in Listing 2. Beside viewPattern or urlMappings there are some additional attributes for a view available:

- **code** for referencing this view from other definition files.
- **public** indicating that this view is available for everyone.
- **onlyLogin** indicates that no special role is required, but the user has to be logged in.

Listing 2. Example or security binding definition

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<security>
  <category code="admin">
    <view code="admin" public="false">
      <navigation package="my.web.admin">
        <viewPattern>/jsf/admin/admin.xhtml</viewPattern>
      </navigation>
    </view>
    <view code="adminRoles" public="false">
      <navigation package="my.web.admin">
        <viewPattern>/jsf/admin/roles.xhtml</viewPattern>
      </navigation>
    </view>
  </category>
</security>
```

4.3 Integrated URL Rewriting

The navigation element of a view defines navigation specific settings. The location of related annotations automatically generated in a later stage (see Section 4.4) is defined with the attribute package. Then, the user friendly URL defined in urlMapping maps the URL to a specific JSF view identifier viewPattern. The rewriting engine is pluggable, but currently PrettyFaces [13] is used.

The urlMapping allows additional arguments using the JSF Expression Language (JSF-EL) like /admin/user/{id} or /admin/user/{myBean.id}. In the first case the expression would be mapped to the corresponding viewPattern with ?id=123 which can be evaluated with CDI injected RequestParam or the second case where the variable id of the bean myBean is set.

4.4 Type-Safe Security Bindings

While it’s always easy to refer to a security setting in Java by a String, it’s hard to keep complex systems developed over years with a large developer team consistent with this concept.

Type-safe security bindings can be achieved by Java annotations with @Target and @Retention CDI qualifiers. These annotations are Java classes and their correct usage in the web application is enforced by the compiler and even CDI-capable IDEs warn the user if he wants to use a non-existing (maybe just due to a typing error) security binding. Compared to a String-based solution, the usage of type-safe security binding results in a much more robust refactoring.

Listing 3. Example or security binding for admin view

```
@SecurityBindingType
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE, ElementType.METHOD})
public @interface ErpViewAdmin {}
```

A security binding can be attached to a view and enforces that the page rendered for users of roles to which this view is assigned. Depending on the current security context of the user, a forward is triggered to a login page (if the user is not logged in) or to an access-denied page (if the user is logged in, but does not have the permissions to view the page).

The concept is not limited to views, even methods in managed beans can be secured with these security bindings. If a button is attached to method call, this method can be annotated with @RoleAdmin public void deleteNews() and only members of the admin role are allowed to call this method.

4.5 Security in Facelet View Handler

With the role definitions it is easy to restrict access to a complete view, but for many applications it’s necessary to render parts of a page differently for different roles. A normal user may see only public news while a administrator has an additional button available where he can delete a news item. This conditional rendering can easily integrated with the JSF component restrictTo like shown in listing 4.

Listing 4. Security in Facelet View Handler

```
<view code="adminRoles" public="false">
  <navigation package="my.web.admin">
    <viewPattern>/jsf/admin/roles.xhtml</viewPattern>
  </navigation>
</view>
```

The attributes for the component as:restrictTo refer to the concept outlined in Figure 3(a): (i) role, (ii) action and (iii) usecase.
Defining a system wide security model with a couple of views, different roles and users being member of these roles allows implementing of a web application with a one dimensional security concept. Figure 4 demonstrates this with a simple example of a web page showing news. Three different roles are defined, a public role, an admin role and an internal role.

All three roles are allowed to view the page news, but the internal news (outlined in red) are only shown to the roles admin and internal. Additional to only viewing the news, users of the role admin are allowed to edit news.

Figure 4. Security concept of a web page

For complex Management Information Systems (MIS) or Enterprise Resource Planning (ERP) solutions such one dimensional role base access control models are not sufficient [14]. The example above can be extended to a project information system where all members of the role internal are working on different projects. In each project they may have different functions like Project Manager, Quality Control, Accountant or Marketing. It is obvious that these functions can be interpreted as a role. This example cannot be mapped to the traditional many-to-many relationship between User and Role, an additional entity (e.g.) Staff is required with relations to User, Role and Project.

These definitions are much more related to the business processes of a web application rather than to traditional security frameworks like JAAS, Seam security or Spring security. Provided mechanisms like “Has the current user the role xy?” are not enough to answer the question in our example: “Has the current user the role xy for the project z?” Developers facing this issue realize that this cannot be solved with the existing security framework and tend to introduce a new entity ProjectRole to solve their problem. Thus everything related to ProjectRole is decoupled from the security framework and the developer is responsible for (re-) implementing sufficient security features.

Our proposal has introduced categories for security settings and allow the definition of independent security domains, e.g. a global domain and a project domain. The same concept can now be used for both domains and the developer does not loose the existing methods for handling security features (following the well-known obligatory DRY development pattern: Don’t repeat yourself). This is shown in Figure 6: The entity Staff is connected the already existing SecurityRole.

5  AhtUtils Addons

5.1 Multi Language Support

For enterprise web application internationalization and localization (supporting different languages and regional differences) are often key features. While it’s easy to provide support for different languages in JSF for static labels or plain UI components by using ResourceBundles and a JSF-EL syntax like #{msg.label} it’s slightly more difficult to provide this to entities the applications are dealing with. These entities can be added, deleted or modified during application lifecycle and not all translations may be available. The framework ensures that at least placeholders are available for all required languages and not NullPointerException will occur.

The library provides multi language support in two domains: Java Persistence API (JPA) and Java Architecture for XML Binding (JAXB). The JAXB implementation is realized by lang elements like shown in the next section while the JPA implementation combines the ability advanced JAP mappings of and the JSF-EL.

Listing 5. JPA multi-language features

```java
@OneToMany (cascade = CascadeType.ALL,
            fetch=FetchType.EAGER)
@MapKey(name = "key")
protected Map<String, UtilsLang> name;
```

This mapping stores all translations (even for different entities) in a single table and provides the correct translation to the JSF components with a expression like #{bean.entity.name[lang-key].translation}.

5.2 Self Documentary

A easy to use security framework is only one step to a successful deployment and handover of a complex web application. In most cases a detailed documentation is required – especially for the security settings and the user/roles model.

In some cases a documentation in multiple languages is required. The integrated documentation module benefits from the multi language support described in Section 5.1 and is able to generate such a documentation for all configured languages. The documentation shown in Listing 6 can be added to all elements described in Figure 3(a).

Listing 6. Documentation in XML

```xml
<documentation>
  <langs>
    <lang key="de" translation="Systemrollen" />  
    <lang key="fr" translation="Roles des usagers" />  
    <lang key="en" translation="System roles" />  
  </langs>
</documentation>
```
The documentation module in the AhtUtils library is able to generate ER diagrams directly from Entity EJB classes. The user has to annotate all entities which should be included in the diagram with a single @EjbErNode qualifier. Classes with these annotations are inspected using the Java Reflection API for additional (and already existing) annotations of the Java Persistence API (JPA) and the processing follows these relationships:

- @javax.persistence.ManyToOne
- @javax.persistence.OneToOne
- @javax.persistence.ManyToMany

The result is represented by a graph and can be exported to the DOT graph description which can be directly visualized like shown in Figure 6.

### 5.4 Menu Creation

A menu with a tree like structure is the most common implementation of navigation in JSF web applications. While it’s easy to implement a simple menu for sophisticated component libraries (e.g. PrimeFaces \texttt{menubar}), plain HTML/CSS style menu mostly relies on nested \texttt{ul/li} elements. If developers have to deal with different web applications for different customers at the same time, a generic menu abstraction (see Listing 7) integrated in the security framework described in Section 4.1 will drastically reduce the implementation effort.

Listing 7. Example of \texttt{menu.xml}

```xml
<menu>
  <item code="admin"/>
  <item code="adminUsers"/>
  <item code="adminRoles"/>
</menu>
```

AHTUtils offers JSF components for rendering the menu to widely used \texttt{ul/li} structure or directly to complex structures like Primefaces \texttt{p:menubar}. Developers can define the menu in the shown XML format and are even able to add dynamic entries to menu, using the underlying objects.

### 5.5 Database Tools

The definition of all security settings is done in with a few XML files which can directly be mapped to the JPA model. For this task import/export tools are provided which can easily initialize the database with given values or export the current settings of the web application by a direct connection or the business logic layer via EJB.

The assignment of users to roles is done in the web application itself dynamically. Typically implemented as an administration page for assigning roles to users and another one for assigning users to roles. These pages together reflect the many-to-many relationship between \texttt{User} and \texttt{Role} in the ER-diagram.

### 6 Conclusion and Future Work

The AhtUtils Security framework extends the solid industry-standard solution JBoss Seam Security with a fine granulated access model for flexible definition of roles, views and categories, making it possible to describe even complex security scenarios containing different levels of access like project- and system-related realms. It allows to define rules for page access of web applications, as well as restrictions to be applied to certain actions within the application. AHTUtils AhtUtils integrates perfect with other parts of the especially with the JSF related aspects. Due
to a integration into other modules of the AhtUtils Java EE toolkit additional features like navigation or security aware menu generation are available and do not need to be programmatically defined on own implementations. Furthermore, multi-language support is already built-in to our solutions. Because it depends on the basic architecture of Java EE and the JBoss Seam standard extension, experienced developers can get familiar with it very quickly. Available as a Maven artifact, adding it to projects is easy and learning the concepts is also supported by the tutorial, available on the project web presence soon. Using the Seam/AhtUtils approach, developers also save time, because they can concentrate on domain level security and do not have to deal with JSF/Servlet security implementations on their own. Talking about maintenance, another important aspects like documentation comes up. In most cases of enterprise software development a extensive and up to date documentation of the current active security model is required. AhtUtils can automatically generate these type of information automatically using the integrated report module, providing reports in PDF and Excel format. The described security is proven by productive usage in various active web applications. Still it lacks a formal evaluation in terms of performance aspects and a detailed comparison to other frameworks available today, but his has been targeted for future work.

References


