SECURE - Java based Authentication, Authorization & Session Management Framework

SRS Document

Disclaimer

This Software Requirements Specification document is a guideline. The document details all the high level requirements. The document should be used as a guideline by the students to design the Solution Architecture for the project. The document also describes the broad scope of the project and high level logical object model. But while developing the solution if the developer has a valid point to add more details being within the scope specified then it can be accommodated after consultation.

Java based Authentication, Authorization & Session Management Framework - SECURE

Introduction

The purpose of this document is to define scope and requirements of a Java based Authentication, Authorization & Session Management Framework for a for a leading software house focussed on web based application development. The proposed system will provide a mechanism to quickly and easily secure any target web application with zero coding effort.

This document should be used by the development team to architect the solution the project.

Management Summary

A leading software development house involved in large number of web based application wanted to eliminate the wasteful repeated development of login and security code required with every application. The primary objective of SECURE is to quickly and easily secure any target web application with zero coding effort:

1. Ability simply integrate SECURE with the target web application by defining the SECURE filter in the web.xml.
2. Flexibility for user driven or administrator driven user registration.
3. Allow configuration of password policy including minimum password length, type of characters, and common passwords dictionary check.
4. Easy configuration of role based permissions of application URIs.
5. Change and forgot password functionality.
6. Allow easy upload of common password dictionary. This is used to prevent the user from selecting a commonly used password - a mechanism to avoid dictionary attacks.

It was proposed to develop a comprehensive Java based authentication, authorization and session management framework. SECURE will be designed & developed to run on IBM WebSphere Application Server and IBM DB2 Universal Database using Java EE Filter mechanism.

Key Assumptions

1. The developer familiarizes with concept of filters on web applications
2. To test this application setup a dummy application or use application being developed by peers to test the authentication framework implementation.

High Level Architecture

SECURE's high level logical architecture is illustrated through the diagram shown below. It will have following categories of users:

1. Web application developers.
2. Application administrator (and the end-users) of the target application.

The framework will provide a very simple and loosely coupled mechanism to add authentication and authorization to any new web application. It will leverage the Java EE standard "filters" to build this framework.

Authentication

The overall high level flow for authentication is outlined below:

1. The end-user attempts to access a "secured" page (URI) in the web application;
2. SECURE's "filter" intercepts the end-user's request;
3. SECURE's filter now retrieves the end-user's "user object" from the session;
4. If valid "user object" exists, filter allows end-user to access the secured page; otherwise it redirects the end-user to the "login page". A SECURE "login servlet" authenticates the end-user using entered credentials against the user table in the database. If the authentication is successful, GAUTH stores the "user object" in the session and redirects user to the "welcome page". In the event of authentication failure, SECURE redirects the end-user to the "login page" again.

Authorization

The overall high level flow is outlined below:

1. The logged-in user attempts to access a page (URI) in the web application;
2. SECURE "filter" intercepts the logged-in user's request;
3. SECURE filter now retrieves the logged-in user's "user object" from the session; and obtains the list of all the Roles(s) that the user is assigned.
4. A lookup using the URI is performed in Permissions table to extract the list of mapped Role(s). User's Role(s) is searched in the mapped Roles. If it is not found, then a HTTP 401 error is raised. Otherwise (i.e. User's role(s) found in the mapped roles) the logged-in user is allowed to access the requested page.

While this approach works for any web application, a much finer grain access control can be exercised for RESTful applications.

Functional Requirements

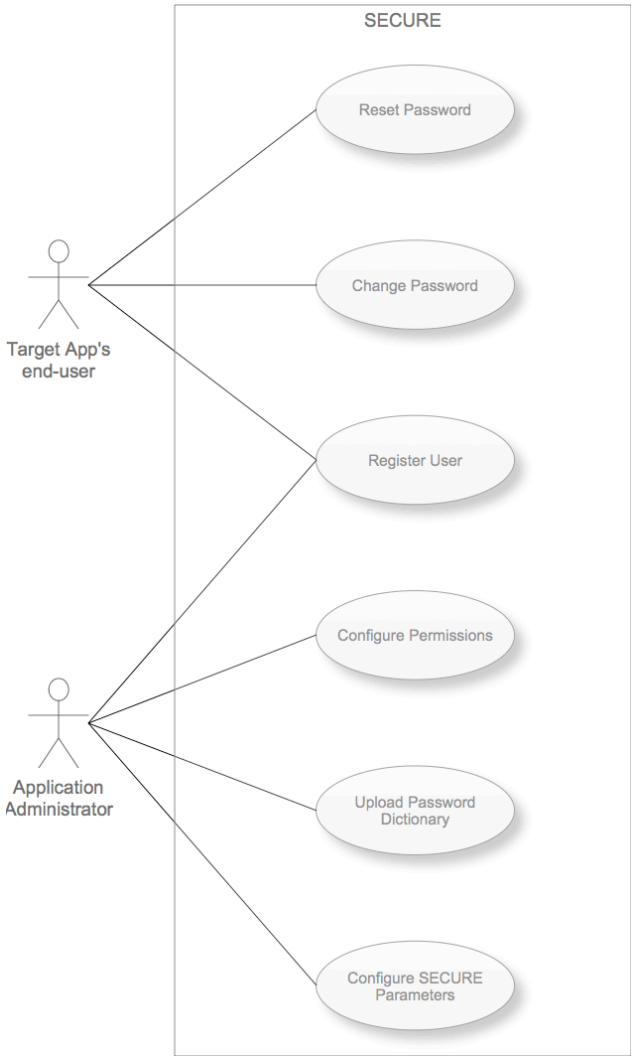
The high level functional requirements for the SECURE framework are outlined in the Use Case diagram described in this section.

SECURE will have pre-designed forms for user registration/sign-up, password change, and reset password (in event of forgetting the password) operations. The UI of these forms will be controlled via their respective CSS files. SECURE will provide a set of default CSS files that the application developer can edit to match UI of these forms with the target application being developed.

SECURE also provides URLs for operations like “sign-up”, “change password”, and “reset password”. Clicking on one of these URLs opens the corresponding form (as outlined previously) for the required operation.

Use Case Diagrams

The following figure illustrates the Use Case diagram for the system.



Use Case Diagram

Use Cases

Register User

Use Case Element	Description
Number	UC.01
Application	To register an user for the target application
Use Case Name	Register User
Primary Actor	Application Administrator
Secondary Actor	Target Application's end-user (driven by configuration option for "self registration" as explained in the Use Case Number UC.06)
Pre-condition	None
Trigger	Application Administrator clicks on the **Register User** link on the admin interface page or end-user clicks on the **sign-up/registe**r link on the target application page
Basic Flow	• System displays the user registration form. The form has essential user information fields like e-mail id (will be used as the login id), password and confirm password. • Up on clicking the "register" button, the user is registered provided s/he has not been previously registered. Only the MD5 hash of the user password will be stored in the database and NOT the password.
Alternate Flow	None
Output	System displays the details of successful operations. In event of duplicate user registration attempt, a suitable error message is displayed.

Change Password

Use Case Element	Description
Number	UC.02
Application	To allow the target application's end-user to change his/her password
Use Case Name	Change Password
Primary Actor	Target Application's end-user
Secondary Actor	None
Pre-condition	None
Trigger	Target Application's end-user clicks on the **change password** link inside the target application.
Basic Flow	• System displays the change password form. The form has fields for "current password", "new password" and "confirm new password". • Up on clicking the "change" button, the logged in target application's end user password is changed provided s/he has entered the correct current password; and the new password & reconfirm password match and the password meets the strength criteria set up during the use case UC.06. • Cancel button cancels the change password operation.
Alternate Flow	• In event of any error, it is clearly displayed and user is asked to renter data or perform operation again.

Use Case Element	Description
Output	System displays the details of successful operations. In event of an error (e.g. mismatched new & reconfirm password or incorrect current password or low password strength), a suitable error message is displayed; and the operation is cancelled.

Reset Password

Use Case Element	Description
Number	UC.03
Application	Resets the password for the target application's end-user, if s/he has forgotten the password.
Use Case Name	Reset Password
Primary Actor	Target Application's end-user
Secondary Actor	None
Pre-condition	None
Trigger	Target Application's end-user clicks on the **reset password** link inside the target application.
Basic Flow	• System displays the reset password form, where the target application's end-user is prompted to enter his/her registered e-mail id. • User receives an e-mail containing a link to password reset operation. • Upon clicking the link, reset password form appears. It has fields for new password and reconfirm password. Clicking on "reset" button resets the password to the new password, provided the "new password" and "confirm password" match and the password meets the strength criteria defined in use case UC.06.
Alternate Flow	In event of error (new & confirm password mismatch or low password strength), a suitable error message is displayed.
Output	If the registered e-mail is found, then the e-mail containing reset password link is sent to the registered e-mail. If it is not found, the request is ignored without any feedback to the end user. Upon successful reset password, a mail is sent informing password reset attempt was successful.

Configure Permissions

Use Case Element	Description
Number	UC.04
Application	To define the list of URIs of the target application that are permitted to be accessed by a "role"
Use Case Name	Configure Permissions
Primary Actor	Application Administrator
Secondary Actor	None

Use Case Element	Description
Pre-condition	None
Trigger	Application Administrator clicks on the **Configure Permissions** link on the admin interface page
Basic Flow	• System displays the form for configuring the permissions. • Application selects the role from a drop down list for which permissions need to be configured. At this stage, if there are already configured URIs then they are displayed. • Application Administrator can add and/or delete target application URIs that the selected role holder is permitted to access. • Upon clicking the “save” button, permissions are saved in the database and are ready to be used by the “filter”. Cancel button cancels the operations.
Alternate Flow	None
Output	None

Upload Password Dictionary

Use Case Element	Description
Number	UC.05
Application	To upload the commonly used password dictionary. This is used to during the password strength checking (if enabled) to ensure that the user does not use a common password.
Use Case Name	Upload Password Dictionary
Primary Actor	Application Administrator
Secondary Actor	None
Pre-condition	None
Trigger	Application Administrator clicks on the **Upload Password Dictionary** link on admin interface page
Basic Flow	• System initiates an upload dialog. • Once a password dictionary file is selected and uploaded, the previous contents are erased. System builds the “bit vector” (using bloom filter algorithm) for the passwords in the uploaded dictionary. During the password change operation, this bit vector travels to client side in form of a JSON [Refer to UC.01, UC.02, and UC.03]. The client side Java Script performs the negative look up using this bit vector.
Alternate Flow	None
Output	System displays the details of successful operations. In the event of any upload error, previous dictionary & its bit vectors remains unchanged and a suitable error message is displayed.

Configure SECURE Parameters

Use Case Element	Description
Number	UC.06
Application	To configure SECURE parameters that control the overall security except permissions
Use Case Name	Configure SECURE parameters
Primary Actor	Application Administrator
Secondary Actor	None
Pre-condition	None
Trigger	Application Administrator clicks on the **Configure SECURE Parameters** link on admin interface page
Basic Flow	• System displays the SECURE parameter configuration page. • It allows following configurations: 1. Allow self registration by target application's end-user: Yes or No. "No" is the default value. 2. Minimum password length: any numeral > 8. Default value is "8". 3. Include number and special characters: Yes or No. Default is "Yes". • "Save" button saves the configuration; "Cancel" button exits the operation without saving anything.
Alternate Flow	None
Output	None

About Bloom Filter

Bloom Filter is a probabilistic data structure. It is in form of a "bit vector" or an array of bits with all bits initialized to "0". To add an element, it uses 2 (or more) hash functions that map the element to 2 (or more) of the array positions with a uniform random distribution. These mapped bit positions are set to "1".

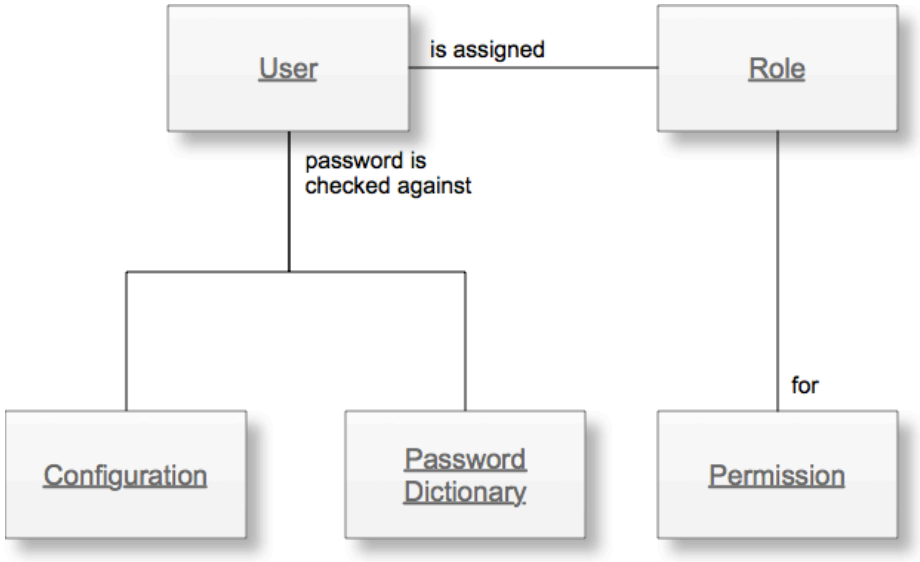
To perform a look up for an element, find the hash of that element using the same set of hash functions used during addition to get 2 (or more) array positions. If any of the bits at these positions are 0, then the element is not in the set.

Therefore, selection of hash functions will be critical, as the same set will be implemented in Java for server side operation to build the bit vector; and in Java Script to perform the negative look up on that bit vector.

The memory efficiency results from the bit vector usage. It will not require several bytes to store each password. For example in form of a bit vector, 10000 passwords can fit in to an array of about 1.5KB.

Logical Object Model

A high level logical object model of the system is shown below. During technical design it will be transformed into a physical model covering all system entities. Such a diagram will include their relationship and its cardinality.



Logical Object Model

1. User is an authorized user of the target application. Each user has his/her e-mail id and password. The password's MD5 hash is stored instead of the password.
2. User password (new or change or reset) are checked against the configuration and password dictionary as outlined in respective use cases earlier.
3. Each user is assigned one or more role in the target application.
4. URI wise permission(s) are set up for Role(s).

Database Design Guidelines

This involves the transformation of the use cases, state diagrams, and logical object model into detailed and optimized physical database table designs.

Typically persistent classes will map to table(s) with their attributes as columns of the table. In some cases a high level object may map in to a master-child table. Invoice is one such example where it maps in to "invoice_header" and "invoice_line_item" table.

Associations between two persistent objects are realized as foreign keys to the associated objects. A foreign key is a column in one table that contains the primary key value of the associated object.

Similarly, a standard technique in relational modeling is to use an intersection entity to represent many-to-many associations. Following is a broad checklist for physical database database design:

1. Database must be properly normalized except those instances where de-normalization help improves performance. This option must be used with special care.
2. All persistent classes that use the database for persistency must map to database structures.
3. Many-to-many relationships must have an intersecting table.
4. Primary keys should be defined for each table, unless there is a performance reason not to define a primary key.

5. Indexes should be defined to optimize access.

6. Data and referential integrity constraints should be defined.

Testing Approach

Quality of the software can be achieved with basic hygiene and consistency followed during design and development of User Interface(UI), Navigation, Validations as per the business process requirement.

To ensure the project delivers acceptable quality to the customer, its important to create a checklist of the conventions to be followed across. Common checks as below are for your reference during design and development:

Common Checks	Validation Type
Page Title is valid for the feature being provided on the page	UI
Order of the Data Entry Fields is logical as per the functionality being provided by the feature	UI
Order of the Display only Fields makes viewing and understanding easy for the user	UI
Spellings and Correctness of Label for the Data Entry and Display fields	UI
The labels are not wrapping onto another row thereby adding a blank row on the page	UI
The fields with drop down are displayed in single row instead of drop down coming on the next row	UI
Data Entry field basic validations are working i.e Text field /Numbers / Dates allow data for their type only	Functional
The dates are following a standard format dd/mmm/yyyy on all forms	UI
The color scheme of all forms i.e headers labels , alerts, entry fields are uniform throughout the application	UI
The action buttons for a New Data Entry Form are uniform for all forms that is allowing data entry	UI
The action buttons are performing the desired action e.g. "submit" is creating a new record if there are no errors and recording all the input fields, whereas 'cancel' is not creating a new record in the database	Functional
The links provided on the forms are opening correctly.	Functional
The data feed mechanism for Read and Write files is generating a log with count of entries.	Navigation

SECURE testing will also involve creating a dummy target application and integrating the same with SECURE in order to perform comprehensive testing.

Suggested Technical Reading

The project is aimed at making the student understand concepts of Design and Development using IBM Rational tools, Web Sphere Application Server and DB2 Database. The following reading reference is easy to understand and should be read to get a clear understanding of capabilities of the tools and how you would leverage them to execute a project.

Technical Reference	URL to access
RAD - Tackling challenges of software development with Rational Application Developer for WebSphere Software	http://www.ibm.com/developerworks/rational/library/08/0926_ackerman-mahate/index.html

Technical Reference	URL to access
IBM Education Assistant - Rational Application Developer 7.5	http://publib.boulder.ibm.com/infocenter/ieduasst/rtnv1r0/index.jsp?topic=/com.ibm.iea.rad_v7/rad/rad75.html
RSA-Overview of Rational Software Architect for WebSphere Software Version 7.5	http://www.ibm.com/developerworks/rational/library/08/0926_arnold/index.html
Using the new features of UML Modeler in IBM Rational Software Architect Version 7.5	http://www.ibm.com/developerworks/rational/library/08/0926_diu/index.html
Rational Technical Library	http://www.ibm.com/developerworks/rational/library/
Java EE Filters	http://docs.oracle.com/javaee/6/tutorial/doc/bnagb.html
RESTFul Web Applications	http://en.wikipedia.org/wiki/Representational_state_transfer
Bloom Filter and Hash Functions	http://en.wikipedia.org/wiki/Bloom_filter and http://www.serve.net/buz/hash.adt/java.001.html