

Lesson 1 Authentication

This lab requires that you use **TrainingApp**, **ExternalApp**, **Guidewire Studio™**, and a supported web browser. Start **Guidewire Studio** for **TrainingApp**. Start the server as Debug 'Server'.

Start **ExternalApp** using the **Start ExternalApp** shortcut.

The default URL for **TrainingApp** is: `http://localhost:8880/ab/ContactManager.do`. Log in to **TrainingApp** as Super User whose login/password is `su/gw`.

1.1 Configure external authentication service

Succeed Insurance wants to replace the base application's user authentication integration with a custom plugin and associated class that validates authentication information from an external system. The external system in this lab is **ExternalApp**. **ExternalApp** will function as an external authentication system like an LDAP system.

1.1.1 Requirements

Spec 1 Create an authentication package named **authentication**.

- Employ Guidewire recommended naming conventions for your package.

Spec 2 Create a Gosu class in the authentication package, named **SIAuthenticationServicePlugin** that implements `gw.plugin.security.AuthenticationServicePlugin`.

- The **authenticate** method must:
 - Take a `UsernamePasswordAuthenticationSource` that contains the http request's username and password parameters (key value pair). If the source is not valid, throw a `gw.pl.exception.GWConfigurationException`.
 - Supply these values to **ExternalApp**'s User Authentication web service. The web service contains one method — `authenticate(username, password)` — that returns a Boolean value of true when the values match.
 - The credentials for the web service are:
 - Username = `externalappuser`
 - Password = `gw`
- The `setCallback` method simply sets the callback handler.
- If external authentication is successful, returns the public ID for the user. If unsuccessful, throws a `javax.security.auth.login.FailedLoginException`.

Spec 3 Register new SIAuthenticationServicePlugin class in the AuthenticationServicePlugin registry.

Spec 4 ExternalApp's User web service WSDL URL is:

<http://localhost:8890/ab/ws/externalapp/webservice/UserAuthenticationAPI?WSDL>

Spec 5 ExternalApp's User Authentication web service authenticates the following key value pairs:

User name	Password
su	guidewire
aapplegate	guidewire
bbaker	guidewire

1.1.2 Tasks

1. Create a new webservice.authentication package.
2. In the package, create a web service collection that connects to ExternalApp's UserAuthentication web service.
3. Create a new plugin.authentication package.
4. Create a new implementation of the AuthenticationServicePlugin class.
5. Edit the existing AuthenticationServicePlugin registry file.
6. Deploy code changes.
7. Perform verification steps.

1.1.3 Verification steps

1. Login into TrainingApp using the username/ password key value of su/guidewire.
 - a) Confirm successful login.
 - b) Log out.
2. Login into TrainingApp using the username/ password key value of aapplegate /guidewire.

- a) Confirm successful login.
- b) Log out.
- 3. **Login into TrainingApp using the username/ password key value of su/gw**
 - o Confirm failed login.
- 4. **Login into TrainingApp using the username/ password key value of afakename/guidewire.**
 - o Confirm failed login.
- 5. **Restore TrainingApp to its original AuthenticationServicePlugin.**
 - a) Close **AuthenticationServicePlugin** tab.
 - b) Right-click on **AuthenticationServicePlugin** registry in Project view.
 - c) Select **Revert to Base**.
 - d) From the **Studio** menu, **Restart** the server.
 - e) Verify **su/gw** log in is successful.



1.1.4 Solution

1. **Create a new webservice.authentication package.**
 - a) Right-click on **gsrc** package and select **New → Package**.
 - b) Enter **si.ta.webservice.authentication** as the new package name.
2. **In the package, create a web service collection that connects to ExternalApp's UserAuthentication web service.**
 - a) Right-click on the **authentication** package and select **New → Web Service Collection**.
 - b) Enter **userauthwsc** as the new Web Service Collection name.
 - c) Click the **Add New Resource** button.
 - d) Enter the resource URL:
<http://localhost:8890/ab/ws/externalapp/webservice/UserAuthenticationAPI?WSDL>
 - e) Click the **Fetch** button.
3. **Create a new plugin.authentication package.**

a) Right-click on **si.ta** package and select **New → Package**.

b) Enter **plugin.authentication** as the new package name.

4. Create a new implementation of the **AuthenticationServicePlugin** class.

a) Right-click on authentication package and select **New → Gosu Class**.

b) Enter **SIAuthenticationServicePlugin** as the new class name.

c) Implement the **AuthenticationServicePlugin** interface.

- Implement the **authenticate** method.
- Implement the **Callback** property.

```
package si.ta.plugin.authentication

uses gw.api.system.PLLoggerCategory
uses gw.pl.exception.GWConfigurationException
uses gw.plugin.security.AuthenticationServicePlugin
uses gw.plugin.security.AuthenticationServicePluginCallbackHandler

uses gw.plugin.security.AuthenticationSource
uses gw.plugin.security.UserNamePasswordAuthenticationSource
uses si.ta.webservice.authentication.userauthwsc.userauthenticationapi.UserAuthenticationAPI
uses javax.security.auth.login.FailedLoginException
uses com.guidewire.pl.web.controller.UserDisplayableException

class SIAuthenticationServicePlugin implements AuthenticationServicePlugin {

    private var _handler : AuthenticationServicePluginCallbackHandler

    /**
     * Authenticate user to external authentication system.
     */
    @Param("source", "AuthenticationSource")
    @Returns("Public ID of the authenticated user")
    @Throws(FailedLoginException, "Thrown for a variety of user-failed login reasons")
    @Throws(UserDisplayableException, "Thrown for a variety of system-failed login reasons")
    override function authenticate(source : AuthenticationSource) : String {
        // retrieve username and password from AuthenticationSource
        if (!(source.typeis UserNamePasswordAuthenticationSource)) {
            throw new GWConfigurationException("Invalid AuthenticationSource: " + source.getClass())
        } else if (!source.determineSourceComplete()) {
            // if the auth source does not contain all required info
            throw new FailedLoginException("Incomplete AuthenticationSource")
        }
        var castSource = source as UserNamePasswordAuthenticationSource
        var username = castSource.Username
        var password = castSource.Password

        try{
            // query external system for user authentication
            var api = new UserAuthenticationAPI()
            api.Config.Http.Authentication.Basic.Username = "externalappuser"
            api.Config.Http.Authentication.Basic.Password = "gw"
```

```

api.Config.CallTimeout = 30000
var authenticated = api.authenticate(username, password)

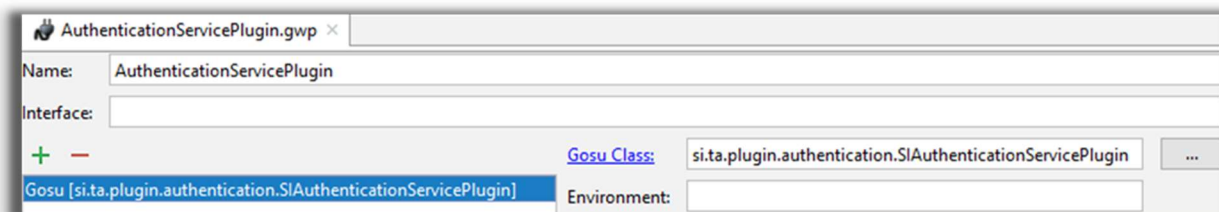
// react to external system results
if (authenticated) {
    if (this._handler == null) {
        throw new GWConfigurationException("Invalid AuthenticationServicePluginCallbackHandler: null")
    }
    // lookup the Public ID
    var publicID = _handler.findUser(username)
    if (publicID == null) {
        throw new FailedLoginException("Unauthorized User: " + username)
    } else {
        return publicID
    }
} else {
    throw new FailedLoginException("Unauthorized User: " + username)
}
} catch (f: FailedLoginException) {
    // log (info level for training purposes) & rethrow FailedLoginException
    PLLoggerCategory.SECURITY.info(f.Message)
    throw f
} catch (t: Throwable) {
    // catch, log as error and wrap all other exceptions
    PLLoggerCategory.SECURITY.error(t.Message)
    throw new UserDisplayableException("Authentication Plugin Failure!\n Please remain calm, the
authorities are on their way.")
}
}

override property set Callback(callbackHandler : AuthenticationServicePluginCallbackHandler) {
    _handler = callbackHandler
}
}

```

5. Edit the existing AuthenticationServicePlugin registry file.

- Navigate to **configuration.config.Plugins.registry**.
- Open AuthenticationServicePlugin registry.
- Remove the existing configured plugin by clicking the minus symbol..
- Add a Gosu Plugin by clicking the plus symbol.
- Define the Gosu plugin.



6. **Add the web credentials to the plugin as parameters.**
7. **Deploy code changes.**
 - a) From the **Studio** menu, **Restart** the server.
8. **Perform verification steps.**