AETF Mobile Sample Security

Contents

Overview

AutoEdge|TheFactory uses an OpenEdge application table - called ApplicationUser - for authentication. This table contains user login names, domains and passwords, as well as some extra information. Login and Logout services are provided via the SecurityTokenService, which is accessed via a REST interface (as installed above).

The OE Web Server has multiple authentication models available. This mobile sample currently uses the anonymous method.

The application uses a STS for its authentication because there are a number of users in the realm, and it's not desirable or practical to duplicate the users in both the Web Server and application. The other (GUI etc) clients use the application authentication realm and don't access the application via the OE Web Server, making it impractical to move the entire app's authentication to the web server.

Server-side implementation

OE Web Server

The sample app uses the anonymous model for the Web Server security, which is the default method.

Snippet from RESTContent/WEB-INF/web.xml

```
12          <param-name>contextConfigLocation</param-name>
13          <param-value>
14              <!-- USER EDIT: Select which application security model to employ
15              /WEB-INF/appSecurity-basic-local.xml
16              /WEB-INF/appSecurity-anonymous.xml
17              /WEB-INF/appSecurity-form-local.xml
18              /WEB-INF/appSecurity-container.xml
19              -->
20              /WEB-INF/appSecurity-anonymous.xml
21          </param-value>
22
23 ]]>
```

Security Token Service

The Security Token Service is implemented as an ABL class. This service is used for Login, Logout and session validation purposes; it's a trimmed-down implementation, although sufficient for sample purposes.

SecurityTokenService.cls method headers

```
class SecurityTokenService:
    @openapi.openedge.export(type="REST", useReturnValue="false", writeDataSetBeforeImage="false").
    @progress.service.resourceMapping(type="REST", operation="invoke", URI="/Login", alias="",
mediaType="application/json").
    method public void Login (input  pcUserName as character,
                              input  pcDomain as character,
                              input  pcPassword as character,
                              output pcUserContext as character):

    end method.

    @openapi.openedge.export(type="REST", useReturnValue="false", writeDataSetBeforeImage="false").
    @progress.service.resourceMapping(type="REST", operation="invoke", URI="/Logout", alias="",
mediaType="application/json").
    method public void Logout (input  pcUserContextId as character,
                               output pcResult as character):
    end method.

    @openapi.openedge.export(type="REST", useReturnValue="false", writeDataSetBeforeImage="false").
    @progress.service.resourceMapping(type="REST", operation="invoke", URI="/ValidateSession",
alias="", mediaType="application/json").
    method public void ValidateSession(input  pcUserContextId as character,
                                       output pcResult as character):
    end method.
end class.
```

Login() sample output


```
{
    "error": false,
    "CCID": "b32f68c9-25f0-ce8a-e211-cd6f6b87617a",
    "TenantId": "8a57828a-13a7-2dac-e011-c00e5f9417fb",
    "TenantName": "fjord",
    "LoginDomain": "customer.fjord",
    "UserType": "Customer",
    "CustNum": 2,
    "CreditLimit": 20091,
    "Name": "Amy L.Robinson",
    "CustomerEmailHome": "Amy@customer.aetf"
}
```


Logout() sample output


```
{"error":false}
```


Client-side implementation

Session initialisation

When the first page of the app loads, the progress sessions are started, via the following code. The JSDO has a requirement for a session before it can retrieve data, and so before anything else happens, we initialise sessions to the Web Service.

home page Load event handler

```
//start the session
var user, pw = '';
var settings = [VehicleOrderService_ShoppingCartService_Settings,
VehicleOrderService_DealerService_Settings, VehicleOrderService_BrandDataService_Settings,
SecurityTokenService_SecurityTokenService_Settings];
for ( var i = 0; i < settings.length; i++) {
 AETF.context.initSession(settings[i].RESTapplicationURL, user, pw, settings[i].catalogURL);
};
```

Notes:

1. The username and passwords used are blank, as befits an anonymous authentication model
2. The anonymous authentication model is the default for a session on the client.

Session Defaults (snippet from progress.session.js)

```
progress.data.Session.LOGIN_SUCCESS = 1;
progress.data.Session.LOGIN_AUTHENTICATION_FAILURE = 2;
progress.data.Session.LOGIN_GENERAL_FAILURE = 3;

progress.data.Session.AUTH_TYPE_ANON = "anonymous";
progress.data.Session.AUTH_TYPE_BASIC= "basic";
progress.data.Session.AUTH_TYPE_FORM = "form";

this.userName = null;
this.loginTarget= '/static/home.html';
this.serviceURI = null;
this.catalogURIs = [];
this.services = [];
this.loginResult = null;
this.loginHttpStatus = null;
this.clientContextId = null;
this.authenticationModel = progress.data.Session.AUTH_TYPE_ANON;
```

3. An array of settings objects is used: there are 4 JSDO services that this app uses. Each unique service has its own session, which is cached/stored in the `AETF.context.sessions` object (see code below)
4. The first 3 services (starting with `VehicleOrderService`) all happen to use the same JSDO Catalogue, so only one session is created for these. The `SecurityTokenService` service uses a different catalog; this all means that there are 2 sessions created for the app.

AETF.context.initSession in aetf.js

```
AETF.context.initSession = function(applicationURL, userName, password, catalogURL) {
    /* if the first time, or not logged in, try to log in */
    if (AETF.context.sessions == undefined
        || AETF.context.sessions[applicationURL] == undefined
        || AETF.context.sessions[applicationURL].loginResult != progress.data.Session.LOGIN_SUCCESS) {

      if (applicationURL == undefined || applicationURL == '')
        console.log("applicationURL was not specified. catalogURL: " + catalogURL);

      // putting the session object into our AETF namespace so that it can be
      // accessed later, if needed
      if (AETF.context.sessions == undefined) {AETF.context.sessions = {};};

      if (AETF.context.sessions[applicationURL] == undefined) {
        AETF.context.sessions[applicationURL] = new progress.data.Session();
      };

      var pdsession = AETF.context.sessions[applicationURL];
      var loginResult = pdsession.login(applicationURL, userName, password);

      if (loginResult != progress.data.Session.LOGIN_SUCCESS) {
        var msg;
        console.log('ERROR: Login failed with code: ' + loginResult);
        switch (loginResult) {
        case progress.data.Session.LOGIN_AUTHENTICATION_FAILURE:
          msg = 'Invalid userid or password';
          break;
        case progress.data.Session.LOGIN_GENERAL_FAILURE:
        default:
          msg = 'Service is unavailable';
          break;
        };

       // Optionally handle the login failure – the code depends on your
       // application design
        alert(msg);

        // this would prevent a click of a button (i.e. login) from continuing, for instance
        return false;
      }
      // load catalog once login succeeded
      pdsession.addCatalog(catalogURL);
    };
  };
```

Customer / user login

Customers need to log in to the application before they can add items to their cart. This is done via the Options page, accessed using the 'gear' icon at top-right. Before customers can log in, a brand **must** be selected. This is because there is a one-to-one relationship between a brand and a tenant in the OE database.

Handy hint
All passwords in the system are **letmein**.
The user **amy** is valid for all brands.
User lists are in /AETF Mobile Sample/doc/applicationusers.txt

If login suceeds, the context returned by the Login() call is cached in the client. On successful logout, the cache is cleared.

Snippet from aetf.js

```
AETF.context.loginComplete = function(usrContext) {
    var ctx = JSON.parse(usrContext);
    if (!ctx.error) {
      if (ctx.UserType === 'Customer') {
          localStorage.setItem('custNum', ctx.CustNum);
      };
      AETF.context.userContext = ctx;
    } else {
     AETF.context.logoutComplete();
    };
  };

  AETF.context.logoutComplete = function() {
      if (AETF.context.userContext.UserType === 'Customer') {
          localStorage.removeItem('custNum');
      };

      AETF.context.userContext = null;
  };
```

Futures: OE Web Server security

- The `SecurityTokenService` should require heightened security. This can be implemented by using form-local security, in a manner similar to the below snippet. Note that the code below is untested.

Snippet from /WEB-INF/appSecurity-form-local.xml

```
01        <!-- OpenEdge ClientPrincipal SSO Filter -->
02        <custom-filter after="SESSION_MANAGEMENT_FILTER" ref="OEClientPrincipalFilter"/>
03
04        <!-- OpenEdge CORS Filter -->
05        <custom-filter ref="OECORSFilter" before="FILTER_SECURITY_INTERCEPTOR"/>
06
07        <!-- URL access controls -->
08
09        <!-- HTTP REST to AppServer gateway -->
10
11        <intercept-url pattern="/rest/Dealer/**" access="permitAll()"/>
12
13        <intercept-url pattern="/rest/BrandData/**" access="permitAll()"/>
14
15        <intercept-url pattern="/rest/Cart/**" access="permitAll()"/>
16
17        <intercept-url pattern="/rest/SecurityTokenService/**" access=
          "hasAnyRole('ROLE_AETF_Client')"/>
18
19        <!-- Standard web-app root for public data like index.html
20             DO NOT grant /** permitAll() access  -->
21        <intercept-url access="permitAll()" pattern="/*" method="GET"/>
22 ]]>
```

This would require changes to the Spring authentication realm, aka user.properties, via adding the `ROLE_AETF_Client`

/RESTContent/WEB-INF/users.properties file

```
restmgr=password,ROLE_PSCAdmin,ROLE_PSCOper,ROLE_PSCUser,enabled
restoper=password,ROLE_PSCOper,ROLE_PSCUser,enabled
restuser=password,ROLE_PSCUser,ROLE_PSCGuest,enabled
guest=letmein,ROLE_AETF_Client,enabled
```

It would also require a change to the home page Load event handler. note that the `VehicleOrderService` services still effectively use an anonymous user.

home page Load event handler

```
//start the session
var user, pw = '';
var settings = [VehicleOrderService_ShoppingCartService_Settings,
VehicleOrderService_DealerService_Settings, VehicleOrderService_BrandDataService_Settings];
for ( var i = 0; i < settings.length; i++) {
 AETF.context.initSession(settings[i].RESTapplicationURL, user, pw, settings[i].catalogURL);
};

user = 'guest';
pw = 'letmein'
AETF.context.initSession(
      SecurityTokenService_SecurityTokenService_Settings.RESTapplicationURL,
      user,
      pw,
      SecurityTokenService_SecurityTokenService_Settings.catalogURL);
```