Document Information

Parent Service: **Identity**

Component Name: External User Migration

Document Type: Developer Manual – Addendum for the main API document

Document Version: 0.2

Version History:

Author	Date	Changes
Behzad Boostanchi Behzad Boostanchi	Saturday, August 26, 2017 Saturday, September 2, 2017	Initial Draft Corrections for API call logic

Identity External Applications Migration Developer Document

1. Introduction

Since there are applications which implement their own authentication system, ‘Identity’ provides a mechanism to make the process of migrating from the legacy systems seamless as possible.

The intended applications should provide a simple REST API, which will be called by ‘Identity’ to determine the validity of an email-password combination. The url of this API should be registered in ‘Identity’ application repository. This repository was created upon the application’s developer request for integrating ‘Identity’.

During the email-authentication process, if ‘Identity’ can’t find the user-provided e-mail in its central account repository, it looks into a special repository for ‘app-specific-accounts’ and when no match found, it tries to call the developer-provided external-authentication-url; supplying the user email and password as parameters.

Inside this app-provided REST API, the email and password should be checked against the existing application account repository and inform the caller if those two user items are matched in the context of the application.

When ‘Identity’ gets a confirmation from the external app, it creates an account in its central repository and links it to the app-validated email-password pair. Since ‘Identity’ can’t trust of the validity of the email itself, it treats these externally validated accounts just in the context of the specified app and it doesn’t consider them as its ‘native’ accounts, but it strongly encourages to transform these semi-native ones to the ‘real’ Identity accounts.

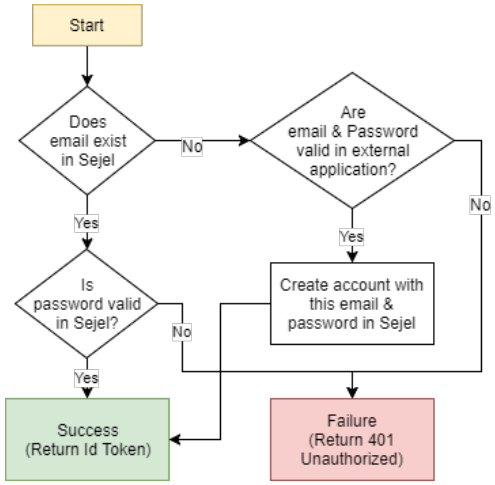
This transformation happens when the owner of these type of accounts tries to change or reset his/her password. In order for this process to take place correctly, it is important that the application disables its own logic of password change/reset for these accounts.

It is recommended that all applications, which intends to migrate to ‘Identity’, during the call to their provided password-validation API, flag the account which is related to the input email in their legacy account repository as ‘password-change-disabled’.

Obviously, the application could maintain its own specific attributes of the migrated accounts and relates them via the secure token, which provides by ‘Identity’ upon successful authentication.

1.1. Migration process workflow:

The overall workflow of the ‘migration’ process is illustrated here:



2. Steps to implement the migration in an application

2.1. Implement the REST API

For an application, the first step to integrate it legacy authentication data with ‘Identity’ is to implement a REST web service. The signature of this API is as follows:

```
LoginResponce CheckLogin( LoginRequest );
```

`LoginRequest` and `LoginResponce` are ‘json’ objects which are defined below:

```
LoginRequest: {
      “Email”: “the_user_provided_emai”,
      “Password: “the_user_provided_pass”
}
LoginResponce: {
      “IsAuthenticated”: “true/false”,
      “IsEmailValid”: “true/false”
}
```

'Identity' calls `CheckLogin` API twice, in the first call, it asks the external application to just validate the user-provided email and if it was successful, it calls the API for the second time and asks it to check if the user-provided email and password are matched or not.

The `CheckLogin` API is called with POST method providing the input user email and/or password as a json object. If `Password` field of the `LoginRequest` json object is empty, then web service should just check the validity of the provided email in its internal database and set the `IsEmailValid` field of the `LoginResponce` json object based on its internal check and returns this object to the called. If both `Email` and `Password` fields of the input json object are filled, the web service should consult its parent application's legacy user database and returns another json objects as its output. On this object, `IsAuthenticated` property indicates whether the provided email and password are matched or not.

Important: *Upon call to this API, if the external application detects a 'match' for email-pass pair, then it should flag the corresponding account as 'password change/reset'* ***disabled****, so these actions would be redirected to 'Identity' and its subsystems.*

2.2 Register the API with 'Identity'

The 'API' FQDN URL should be handed over the 'Identity' team along with the application's specific AppID in order to flag the application as a 'migratory' one, so activates the external authentication logic.

Note: It is assumed that the application has implemented the required 'Identity' integrations like 'Web SDK' or its low-level API.

3. Sample Code

To make the implementation process of user migration easier, here a sample code of REST API for external authentication is provided:

This sample code is written in C# and is adhering to ASP.NET Web API and it's intended to be called as `http://[base_url]/api/login`

WebApiConfig.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;

namespace ExternalLoginApi
{
    public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {
            // Web API configuration and services

            // Web API routes
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
                defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

Controller

logincontroller.cs

```
using System;
using System.Web.Http;
using ExternalLoginAPI.Models;

namespace ExternalLoginAPI.Controllers
{
    public class LoginController : ApiController
    {
        public LoginResponse Login(LoginRequest req)
        {

            LoginResponse theResponse = new LoginResponse();


            if (string.IsNullOrWhiteSpace(req.Password))
            {
                // 1st Call: Just validate the provided email
                //  against the internal database

                if (req.Email == "test@example.com")
                {
                    theResponse = new LoginResponse
                    {
                        IsAuthenticated = false,
                        IsEmailValid = true
                    };

                }

            }
            else
            {
                // Here we should check the input email and pass against
                //  the internal legacy database and returns the
                //  appropriate responce

                if (req.Email == "test@example.com" && req.Password == "123")
                {
                    theResponse = new LoginResponse
                    {
                        IsAuthenticated = true,
                        IsEmailValid = true
                    };
                }
            }


            return theResponse;
        }
    }
}
```

Model

loginrequest.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace ExternalLoginAPI.Models
{
    public class LoginRequest
    {
        public string Email { get; set; }
        public string Password { get; set; }
    }
}
```

loginresponse.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace ExternalLoginAPI.Models
{
    public class LoginResponse
    {
        public bool IsAuthenticated { get; set; }
        public bool IsEmailValid { get; set; }
    }
}
```