# CONTENTS

1. [Introduction](#_Introduction)
2. [Changes to the projects](#_Changes)
   1. [Prometheus CRM Base Web Services](#_Prometheus_CRM_Base)
   2. [SilverSurfer Frontend](#_SilverSurfer)
   3. [Prometheus CRM Service Wrapper](#_Prometheus_CRM_ServiceWrapper)
   4. [Prometheus CRM Email Templates](#_Prometheus_CRM_Emailtemplates)
   5. [Third Party](#_Third_Party_Websites)

# Introduction

* 1. This document serves as a technical specification to allow Authentication and Authorization of the Prometheus CRM Base Web Service using Basic Authentication. The Document will outline all the work required to implement Authorization and Authentication of requests coming from both SilverSurfer and the Third party websites.

# Changes

* 1. Changes to the following projects will be done to accommodate Authentication and Authorization using Basic Authentication
     + - Prometheus CRM Base WebServices
       - SilverSurfer Frontend
       - Prometheus CRM ServiceWrapper
       - Email Template
       - Third Party Websites

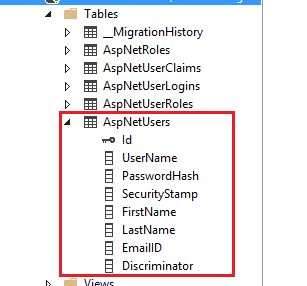
# Prometheus CRM Base Web Services

## Implementing Basic Authentication

### Adding the ASP.NET Identity Provider

Before implementing the Basic Authentication we will create the data store in our database, we will run this Microsoft.AspNet.Identity.EntityFramework nugget package. The migration folder will be created in this project. The migration folder contains the InitialCreate class which contains the UP and Down methods. The UP method will create the necessary tables when the Add-Migration command is run against the Database. The Add-Migration command will be executed on the package manager console.

The following are the tables which will be added to the database.



The table names can be changed to match our naming convection.

Reference Links

<https://blogs.msdn.microsoft.com/webdev/2013/06/27/introducing-asp-net-identity-a-membership-system-for-asp-net-applications/>

<https://github.com/rustd/AspnetIdentitySample>

### Implementing the Basic Authentication

After the database has been set up, then we can implement the Basic Authentication by creating the class which implements IHttpModule,IDisposable interfaces. And in the class’s init method we will need to wire up two events from the HttpApplication object passed to this method. The method we will attach to the AuthenticateRequest event will be called when the client’s credentials are presented. We will also wire up the EndRequest method in order to generate the message that will cause the client to send its credentials. We will also need a Dispose method, but we don’t need to put anything. The class which we will implement will look as follows:

public class SecurityHttpAuthenticationModule : IHttpModule, IDisposable

{

  public void Init(HttpApplication context)

  {

    context.AuthenticateRequest += AuthenticateRequests;

    context.EndRequest += TriggerCredentials;

  }

  public void Dispose()

  {

  }

Any Http Client we send credentials in response to a WWW—Authenticate header that we will include in the HTTP response, We should include that header when a request generates a 401 status code. The header must provide a hint as to the authentication method being used and the realm in which the authentication will apply (the realm can be any arbitrary string and is used to flag to the browser different areas on the server).The code to send that message is what you put in the method wired to the EndRequest event. This example generates a message that specifies that Basic Authentication is being used within the PHVIS realm:

private static void TriggerCredentials(object sender, EventArgs e)

{

  HttpResponse resp = HttpContext.Current.Response;

  if (resp.StatusCode == 401)

  {

    resp.Headers.Add("WWW-Authenticate", @"Basic realm='PHVIS'");

  }

}

Within the method we have wired up to the AuthenticateRequest method, we will need to retrieve the Authorization headers the client will send as a result of receiving a 401/WWW-Authenticate message:

private static void AuthenticateRequests(object sender,

  EventArgs e)

{

  string authHeader =

HttpContext.Current. Request.Headers["Authorization"];

  if (authHeader != null)

  {

Once we have determined that the client has passed the Authorization header, we need to parse out the data holding the username and password. The username and password are base64-encoded and separated by a colon. This code retrieves the username and password into a two-position string array:

AuthenticationHeaderValue authHeaderVal =

  AuthenticationHeaderValue.Parse(authHeader);

if (authHeaderVal.Parameter != null)

{

  byte[] unencoded = Convert.FromBase64String(

    authHeaderVal.Parameter);

  string userpw =

    Encoding.GetEncoding("iso-8859-1").GetString(unencoded);

  string[] creds = userpw.Split(':');

The next step is to validate the username and password using ASPNET.Identity.EntityFramework we set up earlier.Then the final step is to create an identity for the user that will be used in the authorization processes later in the ASP.NET pipeline.

ApplicationUser user = await userManager.FindAsync(context.UserName, context.Password);

To pass that identity information through the pipeline, we will create a Genericidentity object with the name of the identity we want to assign to the user.Once we have created the GenericIdentity object. We must put it in the Thread class’s CurrentPrincipal property. Then we will assign Thread.CurrentPrincipal to the User Property in the HttpContext’s Current property:

if (user == null)

{

context.SetError("invalid\_grant", "The user name or password is incorrect.");

return;

}

GenericIdentity gi = new GenericIdentity(creds[0]);

  Thread.CurrentPrincipal = new GenericPrincipal(gi, null);

  HttpContext.Current.User = Thread.CurrentPrincipal;

We can also add roles to the generic principal

string[] roles = "manager,admin".Split(',');

Thread.CurrentPrincipal = new GenericPrincipal(gi, roles);

To integrate our HTTP module into our site’s processing, in the Base WebService’s web.config file, we will use the add tag within the modules element. The add tag’s type attribute must be set to a string consisting of the fully qualified class name followed by the assembly name of your module:

<modules>

  <add name="myCustomerAuth"

    type="SecureWebAPI. SecurityHttpAuthenticationModule, SecureWebAPI"/>

</modules>

The GenericIdentity object we have created will work with the ASP.NET Authorize attribute. We can also access the GenericIdentity from inside a service method to perform authorization activities

[Authorize]

Public HttpResponseMessage Get(int id)

{

}

Reference Links

<https://msdn.microsoft.com/en-us/magazine/dn201748.aspx>

https://www.asp.net/web-api/overview/security/basic-authentication

## Adding APIUsersController And APIUserManager

APIUsersController and APIUsersManager will be created. The APIUsersController will expose the API methods. The methods which will be exposed are CreateAPIUsers,GetAPIUsers and DeleteAPIUser.The APIUsersManager will encapsulate the UserManager of the identity framework from Microsoft. The UserManager will expose the method which we will use to connect to the database. This Manager will give us access to the following methods update, create and delete.

## Adding APIRolesController and APIRoleManager

APIRolersController and APIRoleManager will be created. The APIRolesController will expose the API methods. The methods which will be exposed are CreateAPIRole,GetAPIRoles ,AssignRoles and DeleteAPIRole.The APIRoleManager will encapsulate the RoleManager of the identity framework from Microsoft. The RoleManager will expose the method which will connect to the database. This Manager will give us access to the following methods AddToRole,AddToRoles and GetRoles.

## Overview of Authentication Request

Whenever a user tries to access the Base WebService from the SilverSurfer or any other third party application. The Custom HttpModule will intercept the request pipeline and grab the username and the password contained in the Authentication Header. Then the UserManager of AspNet.identity.EntityFramework will authenticate the user against the database.if the users credentials are valid the GenericPrincipal object will be created and assigned to the HttpContext.User.Current property,if it does not then we will return the error message to the user.

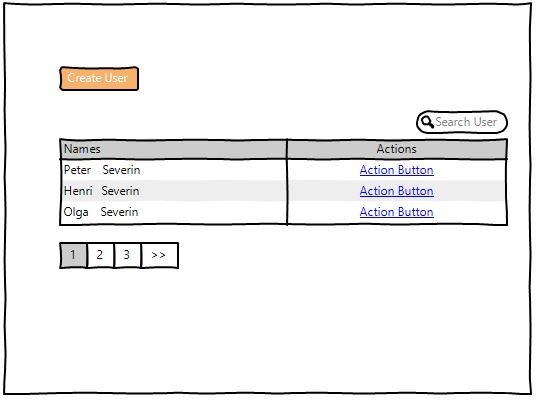
# SilverSurfer

## Changes to UI

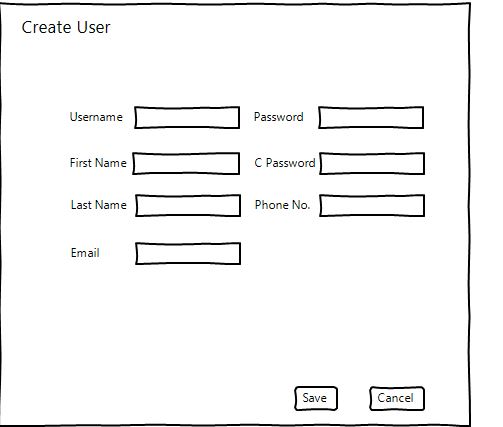
### API User Page

We will create the APIUser management screen. The screen will contain a grid which will display the API Users in the system. It will also contain the Create User and the Search button. The Create User button will show a dialog box when it is clicked. The dialog box will give us an opportunity to add new API Users. The screen will also give us access to the Edit,Delete and Manage Roles buttons upon clicking the Action button. The Edit Button will enable us edit the APIUser.When the Edit Button is clicked the Dialog Box will pop-up containing the details of the APIUser.When the Delete button is clicked the pop-up window with Yes and No button will appear asking whether the APIUser should be deleted or not. The Dialog Box containing two columns of Available roles and Assigned roles will show up when the Manage Role button is clicked.

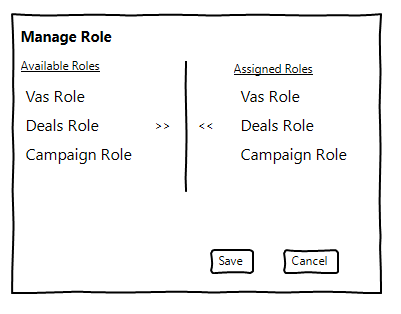
The following is the APIUser Management Screen



Create and Edit User Dialog Screen



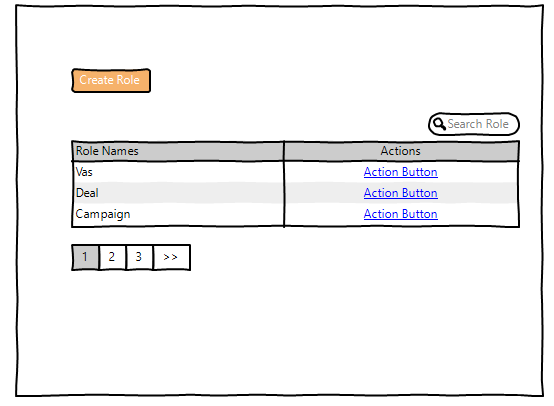
Manage Role Dialog Screen



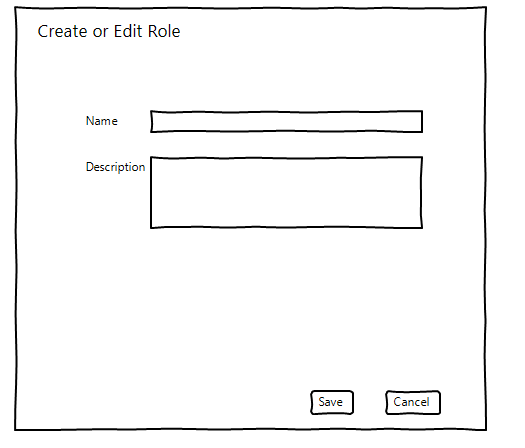
Roles Page

We will also create the APIRole management screen.The screen will contain a grid which will display the API Roles in the system. It will also contain the Create Role button. This button will show a dialog box when clicked. The dialog box will give us an option of creating a new Role. Individual’s rows on the grid will give us access to the Edit and Delete Buttons. When the Edit Button is clicked the Dialog Box will pop-up containing the details of the APIRole.When the Delete button is clicked the pop-up window with Yes and No button will appear asking whether the APIRole should be deleted or not.

Manage Role Page Screen



Edit or Create Role Dialog Screen



## Changes to the Backend

### User Page

The APIUsersController will be created. This controller will expose three methods CreateUser,DeleteUser and GetAPIUsers.The CreateUser will enable us to add a new APIUser and edit an existing APIUser.This is the method which will be invoked when the save button of the Create Or Edit Dialog box is clicked. This method will validate the APIUser model before accepting it.The DeleteUser method will be invoked when the Yes button of the warning dialog box is clicked. The GetAPIUsers is invoked when the APIUsers Screen is accessed. The APIUsersController will transfer the responsibilities of creating, deleting and getting APIUser to the APIUsersService class.

### Roles Page

The APIRolesController will be created. This controller will expose four methods CreateRole,DeleteRole,GetAPIRoles,GetAvailableUnAssignedRoles and AssignRoles.The CreateRole will enable us create new role and edit the existing role. This is the method which will be invoked when the save button of the Create Or Edit Dialog box is clicked. This method will validate the APIRole model before it is accepted. The DeleteRole method will be invoked when the Yes button of the warning dialog box is clicked. The GetAPIRoles is invoked when the APIRoles Screen is accessed. The GetAvailableUnAssignedRoles will be invoked when the manage roles button is clicked on the APIUser’s screen. The AssignRoles method will be invoked when the save button of the Manage Roles Dialog Box is clicked. The APIRolesController will transfer the responsibilities of creating ,deleting ,getting APIRole s and managing Roles to the APIRolesService class.

## Authentication Users

Authentication will be achieved by passing the username and password to the Base WebService .we can store it in the web config .

# Prometheus CRM ServiceWrapper

## Base Service Changes

The BaseService class properties of Username and Password should be set to the actual value of the Password and Username and we would want to authenticate

## Adding IAPIUsersService and IAPIRolesService interfaces

The IAPIUsersService will expose the following public methods CreateUser,DeleteUser and GetAPIUsers.The IAPIRoleService interface will expose these public methods CreateRoles,DeleteRole,GetAPIRoles,GetAvailableUnAssignedRoles and AssignRoles.

## Adding APIUsersService and APIRolesService classes

The APIUsersService and APIRolesService will implement the IAPIUsersService and IAPIRolesService interfaces respectively. The APIUsersService will get and pass APIUser details to Base WebServices.And The APIRolesService will get and pass APIRoles to Base WebService.

# Prometheus CRM Emailtemplates

This project will not change significantly. We will only hard-code the usernames and Password in the web config file. The login credentials will be passed along for every request.

# Third Party Websites

Any web request from the third party should contain the username and password in its Http Request Authorization Header.

Sign Off