**Account Controller**

using System;

using System.Globalization;

using System.Linq;

using System.Security.Claims;

using System.Threading.Tasks;

using System.Web;

using System.Web.Mvc;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.Owin;

using Microsoft.Owin.Security;

using ArizonaSignCompany.Models;

namespace ArizonaSignCompany.Controllers

{

[Authorize]

public class AccountController : Controller

{

private ApplicationSignInManager \_signInManager;

private ApplicationUserManager \_userManager;

public AccountController()

{

}

public AccountController(ApplicationUserManager userManager, ApplicationSignInManager signInManager )

{

UserManager = userManager;

SignInManager = signInManager;

}

public ApplicationSignInManager SignInManager

{

get

{

return \_signInManager ?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>();

}

private set

{

\_signInManager = value;

}

}

public ApplicationUserManager UserManager

{

get

{

return \_userManager ?? HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();

}

private set

{

\_userManager = value;

}

}

//

// GET: /Account/Login

[AllowAnonymous]

public ActionResult Login(string returnUrl)

{

ViewBag.ReturnUrl = returnUrl;

return View();

}

//

// POST: /Account/Login

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)

{

if (!ModelState.IsValid)

{

return View(model);

}

// This doesn't count login failures towards account lockout

// To enable password failures to trigger account lockout, change to shouldLockout: true

var result = await SignInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, shouldLockout: false);

switch (result)

{

case SignInStatus.Success:

using (var db = new ArizonaSignCompanyEntities())

{

var customer = db.Customer\_Information.SingleOrDefault(c => c.Email == model.Email);

if(customer == null || !customer.isApproved)

{

SignInManager.AuthenticationManager.SignOut(DefaultAuthenticationTypes.ApplicationCookie);

return RedirectToAction("RegistrationPending", "Account");

}

}

return RedirectToLocal(returnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.RequiresVerification:

return RedirectToAction("SendCode", new { ReturnUrl = returnUrl, RememberMe = model.RememberMe });

case SignInStatus.Failure:

default:

ModelState.AddModelError("", "Invalid login attempt.");

return View(model);

}

}

//

// GET: /Account/VerifyCode

[AllowAnonymous]

public async Task<ActionResult> VerifyCode(string provider, string returnUrl, bool rememberMe)

{

// Require that the user has already logged in via username/password or external login

if (!await SignInManager.HasBeenVerifiedAsync())

{

return View("Error");

}

return View(new VerifyCodeViewModel { Provider = provider, ReturnUrl = returnUrl, RememberMe = rememberMe });

}

//

// POST: /Account/VerifyCode

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> VerifyCode(VerifyCodeViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

// The following code protects for brute force attacks against the two factor codes.

// If a user enters incorrect codes for a specified amount of time then the user account

// will be locked out for a specified amount of time.

// You can configure the account lockout settings in IdentityConfig

var result = await SignInManager.TwoFactorSignInAsync(model.Provider, model.Code, isPersistent: model.RememberMe, rememberBrowser: model.RememberBrowser);

switch (result)

{

case SignInStatus.Success:

return RedirectToLocal(model.ReturnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.Failure:

default:

ModelState.AddModelError("", "Invalid code.");

return View(model);

}

}

//

// GET: /Account/Register

[AllowAnonymous]

public ActionResult Register()

{

return View();

}

//

// POST: /Account/Register

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

var user = new ApplicationUser { UserName = model.Email, Email = model.Email };

var result = await UserManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

var customer = new Customer\_Information

{

Email = model.Email,

FirstName = model.FirstName,

LastName = model.LastName,

Customer\_ID = user.Id,

Address = model.Address,

City = model.City,

Company = model.Company,

Phone = model.Phone,

State = model.State,

Zip = (int)model.Zip,

isApproved = false

};

using(var db = new ArizonaSignCompanyEntities())

{

//try

//{

db.Customer\_Information.Add(customer);

db.SaveChanges();

//}

/\*catch(Exception)

{

throw;

}\*/

}

//await SignInManager.SignInAsync(user, isPersistent:false, rememberBrowser:false);

// For more information on how to enable account confirmation and password reset please visit https://go.microsoft.com/fwlink/?LinkID=320771

// Send an email with this link

// string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);

// var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

// await UserManager.SendEmailAsync(user.Id, "Confirm your account", "Please confirm your account by clicking <a href=\"" + callbackUrl + "\">here</a>");

return RedirectToAction("RegistrationPending", "Account");

}

AddErrors(result);

}

// If we got this far, something failed, redisplay form

return View(model);

}

//

// GET: /Account/ConfirmEmail

[AllowAnonymous]

public async Task<ActionResult> ConfirmEmail(string userId, string code)

{

if (userId == null || code == null)

{

return View("Error");

}

var result = await UserManager.ConfirmEmailAsync(userId, code);

return View(result.Succeeded ? "ConfirmEmail" : "Error");

}

//

// GET: /Account/ForgotPassword

[AllowAnonymous]

public ActionResult ForgotPassword()

{

return View();

}

//

// POST: /Account/ForgotPassword

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> ForgotPassword(ForgotPasswordViewModel model)

{

if (ModelState.IsValid)

{

var user = await UserManager.FindByNameAsync(model.Email);

if (user == null || !(await UserManager.IsEmailConfirmedAsync(user.Id)))

{

// Don't reveal that the user does not exist or is not confirmed

return View("ForgotPasswordConfirmation");

}

// For more information on how to enable account confirmation and password reset please visit https://go.microsoft.com/fwlink/?LinkID=320771

// Send an email with this link

// string code = await UserManager.GeneratePasswordResetTokenAsync(user.Id);

// var callbackUrl = Url.Action("ResetPassword", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

// await UserManager.SendEmailAsync(user.Id, "Reset Password", "Please reset your password by clicking <a href=\"" + callbackUrl + "\">here</a>");

// return RedirectToAction("ForgotPasswordConfirmation", "Account");

}

// If we got this far, something failed, redisplay form

return View(model);

}

//

// GET: /Account/ForgotPasswordConfirmation

[AllowAnonymous]

public ActionResult ForgotPasswordConfirmation()

{

return View();

}

//

// GET: /Account/ResetPassword

[AllowAnonymous]

public ActionResult ResetPassword(string code)

{

return code == null ? View("Error") : View();

}

//

// POST: /Account/ResetPassword

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> ResetPassword(ResetPasswordViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

var user = await UserManager.FindByNameAsync(model.Email);

if (user == null)

{

// Don't reveal that the user does not exist

return RedirectToAction("ResetPasswordConfirmation", "Account");

}

var result = await UserManager.ResetPasswordAsync(user.Id, model.Code, model.Password);

if (result.Succeeded)

{

return RedirectToAction("ResetPasswordConfirmation", "Account");

}

AddErrors(result);

return View();

}

//

// GET: /Account/ResetPasswordConfirmation

[AllowAnonymous]

public ActionResult ResetPasswordConfirmation()

{

return View();

}

//

// POST: /Account/ExternalLogin

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult ExternalLogin(string provider, string returnUrl)

{

// Request a redirect to the external login provider

return new ChallengeResult(provider, Url.Action("ExternalLoginCallback", "Account", new { ReturnUrl = returnUrl }));

}

//

// GET: /Account/SendCode

[AllowAnonymous]

public async Task<ActionResult> SendCode(string returnUrl, bool rememberMe)

{

var userId = await SignInManager.GetVerifiedUserIdAsync();

if (userId == null)

{

return View("Error");

}

var userFactors = await UserManager.GetValidTwoFactorProvidersAsync(userId);

var factorOptions = userFactors.Select(purpose => new SelectListItem { Text = purpose, Value = purpose }).ToList();

return View(new SendCodeViewModel { Providers = factorOptions, ReturnUrl = returnUrl, RememberMe = rememberMe });

}

//

// POST: /Account/SendCode

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> SendCode(SendCodeViewModel model)

{

if (!ModelState.IsValid)

{

return View();

}

// Generate the token and send it

if (!await SignInManager.SendTwoFactorCodeAsync(model.SelectedProvider))

{

return View("Error");

}

return RedirectToAction("VerifyCode", new { Provider = model.SelectedProvider, ReturnUrl = model.ReturnUrl, RememberMe = model.RememberMe });

}

//

// GET: /Account/ExternalLoginCallback

[AllowAnonymous]

public async Task<ActionResult> ExternalLoginCallback(string returnUrl)

{

var loginInfo = await AuthenticationManager.GetExternalLoginInfoAsync();

if (loginInfo == null)

{

return RedirectToAction("Login");

}

// Sign in the user with this external login provider if the user already has a login

var result = await SignInManager.ExternalSignInAsync(loginInfo, isPersistent: false);

switch (result)

{

case SignInStatus.Success:

return RedirectToLocal(returnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.RequiresVerification:

return RedirectToAction("SendCode", new { ReturnUrl = returnUrl, RememberMe = false });

case SignInStatus.Failure:

default:

// If the user does not have an account, then prompt the user to create an account

ViewBag.ReturnUrl = returnUrl;

ViewBag.LoginProvider = loginInfo.Login.LoginProvider;

return View("ExternalLoginConfirmation", new ExternalLoginConfirmationViewModel { Email = loginInfo.Email });

}

}

//

// POST: /Account/ExternalLoginConfirmation

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> ExternalLoginConfirmation(ExternalLoginConfirmationViewModel model, string returnUrl)

{

if (User.Identity.IsAuthenticated)

{

return RedirectToAction("Index", "Manage");

}

if (ModelState.IsValid)

{

// Get the information about the user from the external login provider

var info = await AuthenticationManager.GetExternalLoginInfoAsync();

if (info == null)

{

return View("ExternalLoginFailure");

}

var user = new ApplicationUser { UserName = model.Email, Email = model.Email };

var result = await UserManager.CreateAsync(user);

if (result.Succeeded)

{

result = await UserManager.AddLoginAsync(user.Id, info.Login);

if (result.Succeeded)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

return RedirectToLocal(returnUrl);

}

}

AddErrors(result);

}

ViewBag.ReturnUrl = returnUrl;

return View(model);

}

//

// POST: /Account/LogOff

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult LogOff()

{

AuthenticationManager.SignOut(DefaultAuthenticationTypes.ApplicationCookie);

return RedirectToAction("Index", "Home");

}

//

// GET: /Account/ExternalLoginFailure

[AllowAnonymous]

public ActionResult ExternalLoginFailure()

{

return View();

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

if (\_userManager != null)

{

\_userManager.Dispose();

\_userManager = null;

}

if (\_signInManager != null)

{

\_signInManager.Dispose();

\_signInManager = null;

}

}

base.Dispose(disposing);

}

#region Helpers

// Used for XSRF protection when adding external logins

private const string XsrfKey = "XsrfId";

private IAuthenticationManager AuthenticationManager

{

get

{

return HttpContext.GetOwinContext().Authentication;

}

}

private void AddErrors(IdentityResult result)

{

foreach (var error in result.Errors)

{

ModelState.AddModelError("", error);

}

}

private ActionResult RedirectToLocal(string returnUrl)

{

if (Url.IsLocalUrl(returnUrl))

{

return Redirect(returnUrl);

}

return RedirectToAction("Index", "Home");

}

internal class ChallengeResult : HttpUnauthorizedResult

{

public ChallengeResult(string provider, string redirectUri)

: this(provider, redirectUri, null)

{

}

public ChallengeResult(string provider, string redirectUri, string userId)

{

LoginProvider = provider;

RedirectUri = redirectUri;

UserId = userId;

}

public string LoginProvider { get; set; }

public string RedirectUri { get; set; }

public string UserId { get; set; }

public override void ExecuteResult(ControllerContext context)

{

var properties = new AuthenticationProperties { RedirectUri = RedirectUri };

if (UserId != null)

{

properties.Dictionary[XsrfKey] = UserId;

}

context.HttpContext.GetOwinContext().Authentication.Challenge(properties, LoginProvider);

}

}

#endregion

[AllowAnonymous]

public ActionResult RegistrationPending()

{

return View();

}

}

}

**Manage Controller**

using System;

using System.Linq;

using System.Threading.Tasks;

using System.Web;

using System.Web.Mvc;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.Owin;

using Microsoft.Owin.Security;

using ArizonaSignCompany.Models;

namespace ArizonaSignCompany.Controllers

{

[Authorize]

public class ManageController : Controller

{

private ApplicationSignInManager \_signInManager;

private ApplicationUserManager \_userManager;

public ManageController()

{

}

public ManageController(ApplicationUserManager userManager, ApplicationSignInManager signInManager)

{

UserManager = userManager;

SignInManager = signInManager;

}

public ApplicationSignInManager SignInManager

{

get

{

return \_signInManager ?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>();

}

private set

{

\_signInManager = value;

}

}

public ApplicationUserManager UserManager

{

get

{

return \_userManager ?? HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();

}

private set

{

\_userManager = value;

}

}

//

// GET: /Manage/Index

public async Task<ActionResult> Index(ManageMessageId? message)

{

ViewBag.StatusMessage =

message == ManageMessageId.ChangePasswordSuccess ? "Your password has been changed."

: message == ManageMessageId.SetPasswordSuccess ? "Your password has been set."

: message == ManageMessageId.SetTwoFactorSuccess ? "Your two-factor authentication provider has been set."

: message == ManageMessageId.Error ? "An error has occurred."

: message == ManageMessageId.AddPhoneSuccess ? "Your phone number was added."

: message == ManageMessageId.RemovePhoneSuccess ? "Your phone number was removed."

: "";

var userId = User.Identity.GetUserId();

var model = new IndexViewModel

{

HasPassword = HasPassword(),

PhoneNumber = await UserManager.GetPhoneNumberAsync(userId),

TwoFactor = await UserManager.GetTwoFactorEnabledAsync(userId),

Logins = await UserManager.GetLoginsAsync(userId),

BrowserRemembered = await AuthenticationManager.TwoFactorBrowserRememberedAsync(userId)

};

return View(model);

}

//

// POST: /Manage/RemoveLogin

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> RemoveLogin(string loginProvider, string providerKey)

{

ManageMessageId? message;

var result = await UserManager.RemoveLoginAsync(User.Identity.GetUserId(), new UserLoginInfo(loginProvider, providerKey));

if (result.Succeeded)

{

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

message = ManageMessageId.RemoveLoginSuccess;

}

else

{

message = ManageMessageId.Error;

}

return RedirectToAction("ManageLogins", new { Message = message });

}

//

// GET: /Manage/AddPhoneNumber

public ActionResult AddPhoneNumber()

{

return View();

}

//

// POST: /Manage/AddPhoneNumber

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> AddPhoneNumber(AddPhoneNumberViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

// Generate the token and send it

var code = await UserManager.GenerateChangePhoneNumberTokenAsync(User.Identity.GetUserId(), model.Number);

if (UserManager.SmsService != null)

{

var message = new IdentityMessage

{

Destination = model.Number,

Body = "Your security code is: " + code

};

await UserManager.SmsService.SendAsync(message);

}

return RedirectToAction("VerifyPhoneNumber", new { PhoneNumber = model.Number });

}

//

// POST: /Manage/EnableTwoFactorAuthentication

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> EnableTwoFactorAuthentication()

{

await UserManager.SetTwoFactorEnabledAsync(User.Identity.GetUserId(), true);

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", "Manage");

}

//

// POST: /Manage/DisableTwoFactorAuthentication

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> DisableTwoFactorAuthentication()

{

await UserManager.SetTwoFactorEnabledAsync(User.Identity.GetUserId(), false);

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", "Manage");

}

//

// GET: /Manage/VerifyPhoneNumber

public async Task<ActionResult> VerifyPhoneNumber(string phoneNumber)

{

var code = await UserManager.GenerateChangePhoneNumberTokenAsync(User.Identity.GetUserId(), phoneNumber);

// Send an SMS through the SMS provider to verify the phone number

return phoneNumber == null ? View("Error") : View(new VerifyPhoneNumberViewModel { PhoneNumber = phoneNumber });

}

//

// POST: /Manage/VerifyPhoneNumber

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> VerifyPhoneNumber(VerifyPhoneNumberViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

var result = await UserManager.ChangePhoneNumberAsync(User.Identity.GetUserId(), model.PhoneNumber, model.Code);

if (result.Succeeded)

{

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", new { Message = ManageMessageId.AddPhoneSuccess });

}

// If we got this far, something failed, redisplay form

ModelState.AddModelError("", "Failed to verify phone");

return View(model);

}

//

// POST: /Manage/RemovePhoneNumber

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> RemovePhoneNumber()

{

var result = await UserManager.SetPhoneNumberAsync(User.Identity.GetUserId(), null);

if (!result.Succeeded)

{

return RedirectToAction("Index", new { Message = ManageMessageId.Error });

}

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", new { Message = ManageMessageId.RemovePhoneSuccess });

}

//

// GET: /Manage/ChangePassword

public ActionResult ChangePassword()

{

return View();

}

//

// POST: /Manage/ChangePassword

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> ChangePassword(ChangePasswordViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

var result = await UserManager.ChangePasswordAsync(User.Identity.GetUserId(), model.OldPassword, model.NewPassword);

if (result.Succeeded)

{

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", new { Message = ManageMessageId.ChangePasswordSuccess });

}

AddErrors(result);

return View(model);

}

//

// GET: /Manage/SetPassword

public ActionResult SetPassword()

{

return View();

}

//

// POST: /Manage/SetPassword

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> SetPassword(SetPasswordViewModel model)

{

if (ModelState.IsValid)

{

var result = await UserManager.AddPasswordAsync(User.Identity.GetUserId(), model.NewPassword);

if (result.Succeeded)

{

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user != null)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

}

return RedirectToAction("Index", new { Message = ManageMessageId.SetPasswordSuccess });

}

AddErrors(result);

}

// If we got this far, something failed, redisplay form

return View(model);

}

//

// GET: /Manage/ManageLogins

public async Task<ActionResult> ManageLogins(ManageMessageId? message)

{

ViewBag.StatusMessage =

message == ManageMessageId.RemoveLoginSuccess ? "The external login was removed."

: message == ManageMessageId.Error ? "An error has occurred."

: "";

var user = await UserManager.FindByIdAsync(User.Identity.GetUserId());

if (user == null)

{

return View("Error");

}

var userLogins = await UserManager.GetLoginsAsync(User.Identity.GetUserId());

var otherLogins = AuthenticationManager.GetExternalAuthenticationTypes().Where(auth => userLogins.All(ul => auth.AuthenticationType != ul.LoginProvider)).ToList();

ViewBag.ShowRemoveButton = user.PasswordHash != null || userLogins.Count > 1;

return View(new ManageLoginsViewModel

{

CurrentLogins = userLogins,

OtherLogins = otherLogins

});

}

//

// POST: /Manage/LinkLogin

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult LinkLogin(string provider)

{

// Request a redirect to the external login provider to link a login for the current user

return new AccountController.ChallengeResult(provider, Url.Action("LinkLoginCallback", "Manage"), User.Identity.GetUserId());

}

//

// GET: /Manage/LinkLoginCallback

public async Task<ActionResult> LinkLoginCallback()

{

var loginInfo = await AuthenticationManager.GetExternalLoginInfoAsync(XsrfKey, User.Identity.GetUserId());

if (loginInfo == null)

{

return RedirectToAction("ManageLogins", new { Message = ManageMessageId.Error });

}

var result = await UserManager.AddLoginAsync(User.Identity.GetUserId(), loginInfo.Login);

return result.Succeeded ? RedirectToAction("ManageLogins") : RedirectToAction("ManageLogins", new { Message = ManageMessageId.Error });

}

protected override void Dispose(bool disposing)

{

if (disposing && \_userManager != null)

{

\_userManager.Dispose();

\_userManager = null;

}

base.Dispose(disposing);

}

#region Helpers

// Used for XSRF protection when adding external logins

private const string XsrfKey = "XsrfId";

private IAuthenticationManager AuthenticationManager

{

get

{

return HttpContext.GetOwinContext().Authentication;

}

}

private void AddErrors(IdentityResult result)

{

foreach (var error in result.Errors)

{

ModelState.AddModelError("", error);

}

}

private bool HasPassword()

{

var user = UserManager.FindById(User.Identity.GetUserId());

if (user != null)

{

return user.PasswordHash != null;

}

return false;

}

private bool HasPhoneNumber()

{

var user = UserManager.FindById(User.Identity.GetUserId());

if (user != null)

{

return user.PhoneNumber != null;

}

return false;

}

public enum ManageMessageId

{

AddPhoneSuccess,

ChangePasswordSuccess,

SetTwoFactorSuccess,

SetPasswordSuccess,

RemoveLoginSuccess,

RemovePhoneSuccess,

Error

}

#endregion

}

}