Add a BirthDate Property to ApplicationUser

Modify RegisterViewModel to Include BirthDate

Modify the Register Action in AccountController to Save BirthDate

ApplicationUser

RegisterViewModel

Register.cshtml

AccountController

ApplicationUser

using System.Data.Entity;

using System.Security.Claims;

using System.Threading.Tasks;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.EntityFramework;

namespace WebApplication1.Models

{

// You can add profile data for the user by adding more properties to your ApplicationUser class, please visit https://go.microsoft.com/fwlink/?LinkID=317594 to learn more.

public class ApplicationUser : IdentityUser

{

public async Task<ClaimsIdentity> GenerateUserIdentityAsync(UserManager<ApplicationUser> manager)

{

// Note the authenticationType must match the one defined in CookieAuthenticationOptions.AuthenticationType

var userIdentity = await manager.CreateIdentityAsync(this, DefaultAuthenticationTypes.ApplicationCookie);

// Add custom user claims here

return userIdentity;

}

}

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public ApplicationDbContext()

: base("DefaultConnection", throwIfV1Schema: false)

{

}

public static ApplicationDbContext Create()

{

return new ApplicationDbContext();

}

}

RegisterViewModel

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace WebApplication1.Models

{

public class ExternalLoginConfirmationViewModel

{

[Required]

[Display(Name = "Email")]

public string Email { get; set; }

}

public class ExternalLoginListViewModel

{

public string ReturnUrl { get; set; }

}

public class SendCodeViewModel

{

public string SelectedProvider { get; set; }

public ICollection<System.Web.Mvc.SelectListItem> Providers { get; set; }

public string ReturnUrl { get; set; }

public bool RememberMe { get; set; }

}

public class VerifyCodeViewModel

{

[Required]

public string Provider { get; set; }

[Required]

[Display(Name = "Code")]

public string Code { get; set; }

public string ReturnUrl { get; set; }

[Display(Name = "Remember this browser?")]

public bool RememberBrowser { get; set; }

public bool RememberMe { get; set; }

}

public class ForgotViewModel

{

[Required]

[Display(Name = "Email")]

public string Email { get; set; }

}

public class LoginViewModel

{

[Required]

[Display(Name = "Email")]

[EmailAddress]

public string Email { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[Display(Name = "Remember me?")]

public bool RememberMe { get; set; }

}

public class RegisterViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm password")]

[Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

}

public class ResetPasswordViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 6)]

[DataType(DataType.Password)]

[Display(Name = "Password")]

public string Password { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm password")]

[Compare("Password", ErrorMessage = "The password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

public string Code { get; set; }

}

public class ForgotPasswordViewModel

{

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

}

}

}

AccountController

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 WebApplication1.Models;

namespace WebApplication1.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:

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)

{

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("Index", "Home");

}

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

}

}

Register.cshtml

@model WebApplication1.Models.RegisterViewModel

@{

ViewBag.Title = "Register";

}

<h2>@ViewBag.Title.</h2>

@using (Html.BeginForm("Register", "Account", FormMethod.Post, new { @class = "form-horizontal", role = "form" }))

{

@Html.AntiForgeryToken()

<h4>Create a new account.</h4>

<hr />

@Html.ValidationSummary("", new { @class = "text-danger" })

<div class="form-group">

@Html.LabelFor(m => m.Email, new { @class = "col-md-2 control-label" })

<div class="col-md-10">

@Html.TextBoxFor(m => m.Email, new { @class = "form-control" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-label" })

<div class="col-md-10">

@Html.PasswordFor(m => m.Password, new { @class = "form-control" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.ConfirmPassword, new { @class = "col-md-2 control-label" })

<div class="col-md-10">

@Html.PasswordFor(m => m.ConfirmPassword, new { @class = "form-control" })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" class="btn btn-default" value="Register" />

</div>

</div>

}

@section Scripts {

@Scripts.Render("~/bundles/jqueryval")

}