Graphical user interface, application

Description automatically generated

Graphical user interface, application, Word

Description automatically generated

BundleConfig.cs

using System.Web;

using System.Web.Optimization;

namespace MVCDemo05Books

{

public class BundleConfig

{

// For more information on bundling, visit https://go.microsoft.com/fwlink/?LinkId=301862

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new ScriptBundle("~/bundles/jquery").Include(

"~/Scripts/jquery-{version}.js"));

bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(

"~/Scripts/jquery.validate\*"));

// Use the development version of Modernizr to develop with and learn from. Then, when you're

// ready for production, use the build tool at https://modernizr.com to pick only the tests you need.

bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(

"~/Scripts/modernizr-\*"));

bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(

"~/Scripts/bootstrap.js"));

bundles.Add(new StyleBundle("~/Content/css").Include(

"~/Content/bootstrap.css",

"~/Content/site.css"));

}

}

}

FilterConfig.cs

using System.Web;

using System.Web.Mvc;

namespace MVCDemo05Books

{

public class FilterConfig

{

public static void RegisterGlobalFilters(GlobalFilterCollection filters)

{

filters.Add(new HandleErrorAttribute());

}

}

}

IdetityConfig.cs

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Security.Claims;

using System.Threading.Tasks;

using System.Web;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.EntityFramework;

using Microsoft.AspNet.Identity.Owin;

using Microsoft.Owin;

using Microsoft.Owin.Security;

using MVCDemo05Books.Models;

namespace MVCDemo05Books

{

public class EmailService : IIdentityMessageService

{

public Task SendAsync(IdentityMessage message)

{

// Plug in your email service here to send an email.

return Task.FromResult(0);

}

}

public class SmsService : IIdentityMessageService

{

public Task SendAsync(IdentityMessage message)

{

// Plug in your SMS service here to send a text message.

return Task.FromResult(0);

}

}

// Configure the application user manager used in this application. UserManager is defined in ASP.NET Identity and is used by the application.

public class ApplicationUserManager : UserManager<ApplicationUser>

{

public ApplicationUserManager(IUserStore<ApplicationUser> store)

: base(store)

{

}

public static ApplicationUserManager Create(IdentityFactoryOptions<ApplicationUserManager> options, IOwinContext context)

{

var manager = new ApplicationUserManager(new UserStore<ApplicationUser>(context.Get<ComputerDbContext>()));

// Configure validation logic for usernames

manager.UserValidator = new UserValidator<ApplicationUser>(manager)

{

AllowOnlyAlphanumericUserNames = false,

RequireUniqueEmail = true

};

// Configure validation logic for passwords

manager.PasswordValidator = new PasswordValidator

{

RequiredLength = 6,

RequireNonLetterOrDigit = false,

RequireDigit = false,

RequireLowercase = false,

RequireUppercase = true,

};

// Configure user lockout defaults

manager.UserLockoutEnabledByDefault = true;

manager.DefaultAccountLockoutTimeSpan = TimeSpan.FromMinutes(5);

manager.MaxFailedAccessAttemptsBeforeLockout = 5;

// Register two factor authentication providers. This application uses Phone and Emails as a step of receiving a code for verifying the user

// You can write your own provider and plug it in here.

manager.RegisterTwoFactorProvider("Phone Code", new PhoneNumberTokenProvider<ApplicationUser>

{

MessageFormat = "Your security code is {0}"

});

manager.RegisterTwoFactorProvider("Email Code", new EmailTokenProvider<ApplicationUser>

{

Subject = "Security Code",

BodyFormat = "Your security code is {0}"

});

manager.EmailService = new EmailService();

manager.SmsService = new SmsService();

var dataProtectionProvider = options.DataProtectionProvider;

if (dataProtectionProvider != null)

{

manager.UserTokenProvider =

new DataProtectorTokenProvider<ApplicationUser>(dataProtectionProvider.Create("ASP.NET Identity"));

}

return manager;

}

}

// Configure the application sign-in manager which is used in this application.

public class ApplicationSignInManager : SignInManager<ApplicationUser, string>

{

public ApplicationSignInManager(ApplicationUserManager userManager, IAuthenticationManager authenticationManager)

: base(userManager, authenticationManager)

{

}

public override Task<ClaimsIdentity> CreateUserIdentityAsync(ApplicationUser user)

{

return user.GenerateUserIdentityAsync((ApplicationUserManager)UserManager);

}

public static ApplicationSignInManager Create(IdentityFactoryOptions<ApplicationSignInManager> options, IOwinContext context)

{

return new ApplicationSignInManager(context.GetUserManager<ApplicationUserManager>(), context.Authentication);

}

}

}

RouteConfig.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using System.Web.Routing;

namespace MVCDemo05Books

{

public class RouteConfig

{

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

);

}

}

}

Startup.Auth.cs

using System;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.Owin;

using Microsoft.Owin;

using Microsoft.Owin.Security.Cookies;

using Microsoft.Owin.Security.Google;

using Owin;

using MVCDemo05Books.Models;

namespace MVCDemo05Books

{

public partial class Startup

{

public void ConfigureAuth(IAppBuilder app)

{

app.CreatePerOwinContext(ComputerDbContext.Create);

app.CreatePerOwinContext<ApplicationUserManager>(ApplicationUserManager.Create);

app.CreatePerOwinContext<ApplicationSignInManager>(ApplicationSignInManager.Create);

app.UseCookieAuthentication(new CookieAuthenticationOptions

{

AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,

LoginPath = new PathString("/Account/Login"),

Provider = new CookieAuthenticationProvider

{

OnValidateIdentity = SecurityStampValidator.OnValidateIdentity<ApplicationUserManager, ApplicationUser>(

validateInterval: TimeSpan.FromMinutes(30),

regenerateIdentity: (manager, user) => user.GenerateUserIdentityAsync(manager))

}

});

app.UseExternalSignInCookie(DefaultAuthenticationTypes.ExternalCookie);

app.UseTwoFactorSignInCookie(DefaultAuthenticationTypes.TwoFactorCookie, TimeSpan.FromMinutes(5));

app.UseTwoFactorRememberBrowserCookie(DefaultAuthenticationTypes.TwoFactorRememberBrowserCookie);

}

}

}

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 MVCDemo05Books.Models;

namespace MVCDemo05Books.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, FirstName = model.FirstName, LastName = model.LastName, Phone = model.Phone, 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

}

}

ComputerController

using MVCDemo05Books.Models;

using System;

using System.Collections.Generic;

using System.Data.Entity;

using System.Linq;

using System.Net;

using System.Web;

using System.Web.Mvc;

namespace MVCDemo05Books.Controllers

{

public class ComputerController : Controller

{

// GET: Book

public ActionResult Index()

{

return RedirectToAction("List");

}

public ActionResult List()

{

using (var database = new ComputerDbContext())

{

var computers = database.Computers

.Include(a => a.Author)

.ToList();

return View(computers);

}

}

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

using (var database = new ComputerDbContext())

{

var computer = database.Computers

.Where(a => a.Id == id)

.Include(a => a.Author)

.First();

return View(computer);

}

}

[Authorize]

public ActionResult Create()

{

return View();

}

[HttpPost]

[Authorize]

public ActionResult Create(Computer computer)

{

if (ModelState.IsValid)

{

using (var database = new ComputerDbContext())

{

var authorId = database.Users

.Where(u => u.UserName == this.User.Identity.Name)

.First()

.Id;

computer.AuthorId = authorId;

object p = database.Computers.Add(computer);

database.SaveChanges();

return RedirectToAction("Index");

}

}

return View(computer);

}

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

using (var database = new ComputerDbContext())

{

var computer = database.Computers

.Where(a => a.Id == id)

.Include(a => a.Author)

.First();

if (! IsUserAuthorizedToEdit(computer))

{

return new HttpStatusCodeResult(HttpStatusCode.Forbidden);

}

if (computer == null)

{

return HttpNotFound();

}

return View(computer);

}

}

[HttpPost]

[ActionName("Delete")]

public ActionResult DeleteConfirmed(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

using (var database = new ComputerDbContext())

{

var computer = database.Computers

.Where(a => a.Id == id)

.Include(a => a.Author)

.First();

if (computer == null)

{

return HttpNotFound();

}

database.Computers.Remove(computer);

database.SaveChanges();

return RedirectToAction("Index");

}

}

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

using (var database = new ComputerDbContext())

{

var computer = database.Computers

.Where(a => a.Id == id)

.First();

if(computer == null)

{

return HttpNotFound();

}

var model = new ComputerViewModel();

model.Id = computer.Id;

model.Name = computer.Name;

model.Brand = computer.Brand;

model.Description = computer.Description;

model.Price = computer.Price;

model.Warranty = computer.Warranty;

return View(model);

}

}

[HttpPost]

public ActionResult Edit(ComputerViewModel model)

{

if (ModelState.IsValid)

{

using (var database = new ComputerDbContext())

{

var computer = database.Computers

.FirstOrDefault(a => a.Id == model.Id);

computer.Id = model.Id;

computer.Name = model.Name;

computer.Brand = model.Brand;

computer.Description = model.Description;

computer.Price = model.Price;

computer.Warranty = model.Warranty;

database.Entry(computer).State = EntityState.Modified;

database.SaveChanges();

return RedirectToAction("Index");

}

}

return View(model);

}

private bool IsUserAuthorizedToEdit(Computer computer)

{

bool isAdmin = this.User.IsInRole("Admin");

bool isAuthor = computer.IsAuthor(this.User.Identity.Name);

return isAdmin || isAuthor;

}

}

}

HomeController

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace MVCDemo05Books.Controllers

{

public class HomeController : Controller

{

public ActionResult Index()

{

return RedirectToAction("List", "Computer");

}

}

}

ManageController

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 MVCDemo05Books.Models;

namespace MVCDemo05Books.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

}

}

Migrations

namespace MVCDemo05Books.Migrations

{

using System;

using System.Data.Entity;

using System.Data.Entity.Migrations;

using System.Linq;

public sealed class Configuration : DbMigrationsConfiguration<MVCDemo05Books.Models.ComputerDbContext>

{

public Configuration()

{

AutomaticMigrationsEnabled = true;

AutomaticMigrationDataLossAllowed = true;

}

protected override void Seed(MVCDemo05Books.Models.ComputerDbContext context)

{

// This method will be called after migrating to the latest version.

// You can use the DbSet<T>.AddOrUpdate() helper extension method

// to avoid creating duplicate seed data.

}

}

}

AccountViewModels

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace MVCDemo05Books.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]

[StringLength(50)]

[Display(Name = "First Name")]

public string FirstName { get; set; }

[Required]

[StringLength(50)]

[Display(Name = "Last Name")]

public string LastName { get; set; }

[Required]

[StringLength(30)]

[Display(Name = "Phone")]

public string Phone { get; set; }

[Required]

[EmailAddress]

[Display(Name = "Email")]

public string Email { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 5)]

[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 = 5)]

[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; }

}

}

ApplicationUser

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.EntityFramework;

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Security.Claims;

using System.Threading.Tasks;

using System.Web;

namespace MVCDemo05Books.Models

{

public class ApplicationUser : IdentityUser

{

[Required]

[StringLength(50)]

public string FirstName { get; set; }

[Required]

[StringLength(50)]

public string LastName { get; set; }

[Required]

[StringLength(30)]

public string Phone { get; set; }

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;

}

}

}

Computer.cs

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace MVCDemo05Books.Models

{

public class Computer

{

[Key]

public int Id { get; set; }

[Required]

public string Name { get; set; }

[Required]

public string Brand { get; set; }

[Required]

public string Description { get; set; }

[Required]

public string Price { get; set; }

[Required]

[DataType(DataType.Date)]

public string Warranty { get; set; }

[ForeignKey("Author")]

public string AuthorId { get; set; }

public virtual ApplicationUser Author { get; set; }

public bool IsAuthor(string name)

{

return this.Author.UserName.Equals(name);

}

}

}

ComputerDbContext

using System.Collections.Generic;

using System.Data.Entity;

using System.Security.Claims;

using System.Threading.Tasks;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.EntityFramework;

namespace MVCDemo05Books.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 ComputerDbContext : IdentityDbContext<ApplicationUser>

{

public ComputerDbContext()

: base("DefaultConnection", throwIfV1Schema: false)

{

}

public virtual IDbSet<Computer> Computers { get; set; }

public IEnumerable<object> Computer { get; internal set; }

public static ComputerDbContext Create()

{

return new ComputerDbContext();

}

}

}

ComputerViewModel

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Linq;

using System.Web;

namespace MVCDemo05Books.Models

{

public class ComputerViewModel

{

public int Id { get; set; }

[Required]

public string Name { get; set; }

[Required]

public string Brand { get; set; }

[Required]

public string Description { get; set; }

[Required]

public string Price { get; set; }

[Required]

[DataType(DataType.Date)]

public string Warranty { get; set; }

[ForeignKey("Author")]

public string AuthorId { get; set; }

}

}

ManageViewModels

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using Microsoft.AspNet.Identity;

using Microsoft.Owin.Security;

namespace MVCDemo05Books.Models

{

public class IndexViewModel

{

public bool HasPassword { get; set; }

public IList<UserLoginInfo> Logins { get; set; }

public string PhoneNumber { get; set; }

public bool TwoFactor { get; set; }

public bool BrowserRemembered { get; set; }

}

public class ManageLoginsViewModel

{

public IList<UserLoginInfo> CurrentLogins { get; set; }

public IList<AuthenticationDescription> OtherLogins { get; set; }

}

public class FactorViewModel

{

public string Purpose { get; set; }

}

public class SetPasswordViewModel

{

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 5)]

[DataType(DataType.Password)]

[Display(Name = "New password")]

public string NewPassword { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm new password")]

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

public string ConfirmPassword { get; set; }

}

public class ChangePasswordViewModel

{

[Required]

[DataType(DataType.Password)]

[Display(Name = "Current password")]

public string OldPassword { get; set; }

[Required]

[StringLength(100, ErrorMessage = "The {0} must be at least {2} characters long.", MinimumLength = 5)]

[DataType(DataType.Password)]

[Display(Name = "New password")]

public string NewPassword { get; set; }

[DataType(DataType.Password)]

[Display(Name = "Confirm new password")]

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

public string ConfirmPassword { get; set; }

}

public class AddPhoneNumberViewModel

{

[Required]

[Phone]

[Display(Name = "Phone Number")]

public string Number { get; set; }

}

public class VerifyPhoneNumberViewModel

{

[Required]

[Display(Name = "Code")]

public string Code { get; set; }

[Required]

[Phone]

[Display(Name = "Phone Number")]

public string PhoneNumber { get; set; }

}

public class ConfigureTwoFactorViewModel

{

public string SelectedProvider { get; set; }

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

}

}

Views/Computer/Create.cshtml

@model MVCDemo05Books.Models.Computer

@{

ViewBag.Title = "Create";

}

<div class="container">

<div class="well">

<h2>Create Computer</h2>

@using (Html.BeginForm("Create", "Computer", FormMethod.Post, new { @class = "form-horizontal" }))

{

@Html.AntiForgeryToken()

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

<div class="form-group">

@Html.LabelFor(m => m.Name, new { @class = "control-label col-sm-4" })

<div class="col-sm-4">

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

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Brand, new { @class = "control-label col-sm-4" })

<div class="col-sm-4">

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

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Description, new { @class = "control-label col-sm-4" })

<div class="col-sm-4">

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

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Price, new { @class = "control-label col-sm-4" })

<div class="col-sm-4">

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

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Warranty, new { @class = "control-label col-sm-4" })

<div class="col-sm-4">

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

</div>

</div>

<div class="form-group">

<div class="col-sm-4 col-sm-offset-4">

@Html.ActionLink("Cancel", "Index", "Computer", null, new { @class = "btn btn-default" })

<input type="submit" value="Create" class="btn btn-success" />

</div>

</div>

}

</div>

</div>

Delete.cshtml

@model MVCDemo05Books.Models.Computer

@{

ViewBag.Title = "Delete";

}

<div class="container">

<div class="well">

<h2>Delete Computer</h2>

@using (Html.BeginForm("Delete", "Computer", FormMethod.Post, new { @class = "form-horizontal" }))

{

@Html.AntiForgeryToken()

<div class="form-group">

@Html.LabelFor(m => m.Name, new { @class = "control-label col-sm-4" })

<div class="col-sm-4">

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

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Brand, new { @class = "control-label col-sm-4" })

<div class="col-sm-4">

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

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Description, new { @class = "control-label col-sm-4" })

<div class="col-sm-4">

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

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Price, new { @class = "control-label col-sm-4" })

<div class="col-sm-4">

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

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Warranty, new { @class = "control-label col-sm-4" })

<div class="col-sm-4">

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

</div>

</div>

<div class="form-group">

<div class="col-sm-4 col-sm-offset-4">

@Html.ActionLink("Cancel", "Index", "Computer", null, new { @class = "btn btn-default" })

<input type="submit" value="Delete" class="btn btn-danger" />

</div>

</div>

}

</div>

</div>

Details.cshtml

@model MVCDemo05Books.Models.Computer

@{

ViewBag.Title = "Details";

}

<div class="container">

<book>

<header>

<h2>

@Model.Name

</h2>

</header>

<p>

Name: @Model.Name<br />

Brand: @Model.Brand<br />

Description: @Model.Description<br />

Price: @Model.Price<br />

Warranty: @Model.Warranty<hr />

@Model.Name

</p>

<small class="author">

--author @Model.Author.FirstName

</small>

<footer class="pull-right">

@if (User.IsInRole("Admin") || Model.IsAuthor(User.Identity.Name))

{

@Html.ActionLink("Edit", "Edit", "Computer", new { @id = Model.Id }, new { @class = "btn btn-success btn-xs" })

@Html.ActionLink("Delete", "Delete", "Computer", new { @id = Model.Id }, new { @class = "btn btn-danger btn-xs" })

}

@Html.ActionLink("Back", "Index", "Computer", null, new { @class = "btn btn-default btn-xs" })

</footer>

</book>

</div>

<hr />

Edit.cshtml

@model MVCDemo05Books.Models.ComputerViewModel

@{

ViewBag.Title = "Edit";

}

<div class="container">

<div class="well">

<h2>Edit Computer</h2>

@using (Html.BeginForm("Edit", "Computer", FormMethod.Post, new { @class = "form-horizontal" }))

{

@Html.AntiForgeryToken()

@Html.HiddenFor(m => m.Id)

@Html.HiddenFor(m => m.AuthorId)

<div class="form-group">

@Html.LabelFor(m => m.Name, new { @class = "control-label col-sm-4" })

<div class="col-sm-4">

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

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Brand, new { @class = "control-label col-sm-4" })

<div class="col-sm-4">

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

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Description, new { @class = "control-label col-sm-4" })

<div class="col-sm-4">

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

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Price, new { @class = "control-label col-sm-4" })

<div class="col-sm-4">

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

</div>

</div>

<div class="form-group">

@Html.LabelFor(m => m.Warranty, new { @class = "control-label col-sm-4" })

<div class="col-sm-4">

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

</div>

</div>

<div class="form-group">

<div class="col-sm-4 col-sm-offset-4">

@Html.ActionLink("Cancel", "Index", "Computer", null, new { @class = "btn btn-default" })

<input type="submit" value="Edit" class="btn btn-success" />

</div>

</div>

}

</div>

</div>

List.cshtml

@model List<MVCDemo05Books.Models.Computer>

@{

ViewBag.Title = "List";

}

<div class="container">

<div class="row">

@foreach (var computer in Model)

{

<div class="col-sm-6">

<book>

<header>

<h2>

@Html.ActionLink(computer.Name, "Details", "Computer", new { @id = computer.Id}, null)

</h2>

</header>

</book>

</div>

}

</div>

</div>

Shared/\_Layout.cshtml

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title - Computers and components</title>

@Styles.Render("~/Content/css")

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

</head>

<body>

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

@Html.ActionLink("Computers and components", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })

</div>

<div class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li>@Html.ActionLink("Home", "Index", "Home")</li>

<li>@Html.ActionLink("About", "About", "Home")</li>

<li>@Html.ActionLink("Contact", "Contact", "Home")</li>

</ul>

@Html.Partial("\_LoginPartial")

</div>

</div>

</div>

<div class="container body-content">

@RenderBody()

<hr />

<footer>

<p>&copy; @DateTime.Now.Year - Computers and components</p>

</footer>

</div>

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

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

@RenderSection("scripts", required: false)

</body>

</html>

\_LoginPartial.cshtml

@using Microsoft.AspNet.Identity

@if (Request.IsAuthenticated)

{

using (Html.BeginForm("LogOff", "Account", FormMethod.Post, new { id = "logoutForm", @class = "navbar-right" }))

{

@Html.AntiForgeryToken()

<ul class="nav navbar-nav navbar-right">

<li>

@Html.ActionLink("Create Computer", "Create", "Computer")

</li>

<li>

@Html.ActionLink("Hello " + User.Identity.GetUserName() + "!", "Index", "Manage", routeValues: null, htmlAttributes: new { title = "Manage" })

</li>

<li><a href="javascript:document.getElementById('logoutForm').submit()">Log off</a></li>

</ul>

}

}

else

{

<ul class="nav navbar-nav navbar-right">

<li>@Html.ActionLink("Register", "Register", "Account", routeValues: null, htmlAttributes: new { id = "registerLink" })</li>

<li>@Html.ActionLink("Log in", "Login", "Account", routeValues: null, htmlAttributes: new { id = "loginLink" })</li>

</ul>

}

Error.cshtml

@model System.Web.Mvc.HandleErrorInfo

@{

ViewBag.Title = "Error";

}

<h1 class="text-danger">Error.</h1>

<h2 class="text-danger">An error occurred while processing your request.</h2>

Lockout.cshtml

@model System.Web.Mvc.HandleErrorInfo

@{

ViewBag.Title = "Locked Out";

}

<hgroup>

<h1 class="text-danger">Locked out.</h1>

<h2 class="text-danger">This account has been locked out, please try again later.</h2>

</hgroup>