**Phase End Project**

**ASP.NET MVC Ecommerce Site to Sell Laptops**

**Source code:**

**BAL.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace BAL

{

public class Laptop

{

public int laptop\_id { get; set; }

public string laptop\_name { get; set; }

public int Cost { get; set; }

public string Descr { get; set; }

}

public class Credit

{

public int Cardno { get; set; }

public string name { get; set; }

public int CVV { get; set; }

}

}

**DAL1.cs:**

using BAL;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DAL

{

public class DAL1

{

public bool Insert(Laptop school)

{

SqlConnection cn = new SqlConnection("Data Source=DESKTOP-5GL4B5D\\SQLEXPRESS1;Initial Catalog=Ecommerce;Integrated Security=True");

SqlCommand cmdInsert = new SqlCommand("insert into laptop(laptop\_id,laptop\_name,cost,Descr) values(@laptop\_id,@laptop\_name,@cost,@Descr)", cn);

cmdInsert.Parameters.AddWithValue("@laptop\_id", school.laptop\_id);

cmdInsert.Parameters.AddWithValue("@laptop\_name", school.laptop\_name) ;

cmdInsert.Parameters.AddWithValue("@cost", school.Cost);

cmdInsert.Parameters.AddWithValue("@Descr", school.Descr);

cn.Open();

int i = cmdInsert.ExecuteNonQuery();

bool status = false;

if (i == 1)

{

status = true;

}

cn.Close();//finally

cn.Dispose();//finally

return status;

}

public bool Update(Laptop school)

{

SqlConnection cn = new SqlConnection("Data Source=DESKTOP-5GL4B5D\\SQLEXPRESS1;Initial Catalog=Ecommerce;Integrated Security=True");

SqlCommand cmdUpdate = new SqlCommand("[dbo].[Update]", cn);

cmdUpdate.CommandType = System.Data.CommandType.StoredProcedure;

cmdUpdate.Parameters.AddWithValue("@p\_id", school.laptop\_id);

cmdUpdate.Parameters.AddWithValue("@p\_name", school.laptop\_name);

cmdUpdate.Parameters.AddWithValue("@p\_cost", school.Cost);

cmdUpdate.Parameters.AddWithValue("@p\_des", school.Descr);

cn.Open();

int s = cmdUpdate.ExecuteNonQuery();

bool statusd = false;

if (s == 1)

{

statusd = true;

}

cn.Close();//finally

cn.Dispose();//finally

return statusd;

}

public Laptop Find(int id)

{

SqlConnection cn = new SqlConnection("Data Source=DESKTOP-5GL4B5D\\SQLEXPRESS1;Initial Catalog=Ecommerce;Integrated Security=True");

SqlCommand cmdSelect = new SqlCommand("[dbo].sp\_Find", cn);

cmdSelect.CommandType = System.Data.CommandType.StoredProcedure;

cmdSelect.Parameters.AddWithValue("@p\_id", id);

SqlParameter p1 = new SqlParameter();

p1.ParameterName = "@p\_name";

p1.SqlDbType = System.Data.SqlDbType.NVarChar;

p1.Size = 10;

p1.Direction = System.Data.ParameterDirection.Output;

cmdSelect.Parameters.Add(p1);

SqlParameter p2 = new SqlParameter();

p2.ParameterName = "@p\_cost";

p2.SqlDbType = System.Data.SqlDbType.Int;

p2.Size = 20;

p2.Direction = System.Data.ParameterDirection.Output;

cmdSelect.Parameters.Add(p2);

SqlParameter p3 = new SqlParameter();

p3.ParameterName = "@p\_des";

p3.SqlDbType = System.Data.SqlDbType.NVarChar;

p3.Size = 120;

p3.Direction = System.Data.ParameterDirection.Output;

cmdSelect.Parameters.Add(p3);

cn.Open();

cmdSelect.ExecuteNonQuery();

Laptop found = new Laptop();

found.laptop\_name = p1.Value.ToString();

found.Cost = Convert.ToInt32(p2.Value);

found.Descr=p3.Value.ToString();

cn.Close();

cn.Dispose();

return found;

}

public List<Laptop> List()

{

SqlConnection cn = new SqlConnection("Data Source=DESKTOP-5GL4B5D\\SQLEXPRESS1;Initial Catalog=Ecommerce;Integrated Security=True");

SqlCommand cmdlist = new SqlCommand("select laptop\_id,laptop\_name,cost,Descr from laptop", cn);

cn.Open();

SqlDataReader dr = cmdlist.ExecuteReader();

List<Laptop> emplist = new List<Laptop>();

if (dr.HasRows)

{

while (dr.Read())

{

Laptop bal = new Laptop();

bal.laptop\_id = Convert.ToInt32(dr["laptop\_id"]);

bal.laptop\_name = dr["laptop\_name"].ToString();

bal.Cost = Convert.ToInt32(dr["cost"]);

bal.Descr= dr["Descr"].ToString();

emplist.Add(bal);

}

}

cn.Close();

cn.Dispose();

return emplist;

}

public bool Delete(int stuid)

{

SqlConnection cn = new SqlConnection("Data Source=DESKTOP-5GL4B5D\\SQLEXPRESS1;Initial Catalog=Ecommerce;Integrated Security=True");

SqlCommand cmdDelete = new SqlCommand("[dbo].sp\_Delete", cn);

cmdDelete.CommandType = System.Data.CommandType.StoredProcedure;

cmdDelete.Parameters.AddWithValue("@p\_id", stuid);

cn.Open();

int i = cmdDelete.ExecuteNonQuery();

bool status = false;

if (i == 1)

{

status = true;

}

cn.Close();//finally

cn.Dispose();//finally

return status;

}

public bool Insert1(Credit school)

{

SqlConnection cn = new SqlConnection("Data Source=DESKTOP-5GL4B5D\\SQLEXPRESS1;Initial Catalog=Ecommerce;Integrated Security=True");

SqlCommand cmdInsert = new SqlCommand("insert into Payment(Cardno,name,CVV) values(@Cardno,@name,@CVV)", cn);

cmdInsert.Parameters.AddWithValue("@Cardno", school.Cardno);

cmdInsert.Parameters.AddWithValue("@name", school.name);

cmdInsert.Parameters.AddWithValue("@CVV", school.CVV);

cn.Open();

int i = cmdInsert.ExecuteNonQuery();

bool status = false;

if (i == 1)

{

status = true;

}

cn.Close();//finally

cn.Dispose();//finally

return status;

}}}

**Helperclass.cs:**

using BAL;

using DAL;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Helper

{

public class Helperclass

{

DAL1 dal = null;

public Helperclass()

{

dal = new DAL1();

}

public bool Add(Credit school1)

{

return dal.Insert1(school1);

}

public bool AddE(Laptop school)

{

return dal.Insert(school);

}

public bool Edit(Laptop school)

{

return dal.Update(school);

}

public Laptop search(int id)

{

return dal.Find(id);

}

public List<Laptop> List()

{

return dal.List();

}

public bool remove(int id)

{

return dal.Delete(id);

}

}

}

**AccountController.cs:**

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 mvcccc.Models;

namespace mvcccc.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);

}

}

}

}

**HomeController.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace mvcccc.Controllers

{

public class HomeController : Controller

{

public ActionResult Index()

{

return View();

}

public ActionResult About()

{

ViewBag.Message = "Have an enjoyment with Lots of offers Amazon today announced it plans to create 1 million new jobs in India by 2025 through continued investments in technology, infrastructure, and its logistics network. The jobs – created both directly and indirectly – will be across industries, including information technology, skill development, content creation, retail, logistics, and manufacturing, and are in addition to the 700,000 jobs Amazon’s investments have enabled over the last six years in India. On Wednesday, Amazon Founder and CEO Jeff Bezos also announced that the company plans to invest US $1B to help bring 10 million traders and micro, small, and medium-sized businesses (MSMEs) across India online, enabling US $10B in cumulative exports by 2025 and supporting India’s economic diversification.";

return View();

}

public ActionResult Contact()

{

ViewBag.Message = "Make Money with Us\r\nSell on Amazon\r\nSell under Amazon Accelerator\r\nAmazon Global Selling\r\nBecome an Affiliate\r\nFulfilment by Amazon\r\nAdvertise Your Products\r\nAmazon Pay on Merchants";

return View();

}

}

}

**LaptoController.cs:**

using BAL;

using Helper;

using mvcccc.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace mvcccc.Controllers

{

public class LaptoController : Controller

{

// GET: Lapto

Helperclass helper = null;

public LaptoController()

{

helper = new Helperclass();

}

public ActionResult Index()

{

var stulist = helper.List();

List<laptop> modelsList = new List<laptop>();

foreach (var item in stulist)

{

modelsList.Add(new laptop

{

laptop\_id=item.laptop\_id,

laptop\_name=item.laptop\_name,

Cost=item.Cost,

Descr=item.Descr

});

}

return View(modelsList);

}

public ActionResult Pay()

{

return View();

}

public ActionResult Details(int id)

{

var data = helper.search(id);

laptop emp = new laptop();

emp.laptop\_id = id;

emp.laptop\_name = data.laptop\_name;

emp.Cost = data.Cost;

emp.Descr = data.Descr;

return View(emp);

}

public ActionResult Create()

{

return View();

}

[HttpPost]

public ActionResult Create(FormCollection collection)

{

Laptop bal = new Laptop();

bal.laptop\_id = Convert.ToInt32(Request["laptop\_id"]);

bal.laptop\_name = Request["laptop\_name"].ToString();

bal.Cost = Convert.ToInt32(Request["Cost"]);

bal.Descr = Request["Descr"].ToString();

bool ans = helper.AddE(bal);

if (ans)

{

return RedirectToAction("Index");

}

else

{

return View();

}

}

public ActionResult Edit(int id)

{

var emp = helper.search(id);

laptop model = new laptop();

model.laptop\_id = id;

model.laptop\_name = emp.laptop\_name;

model.Cost = emp.Cost;

model.Descr = emp.Descr;

return View(model);

}

[HttpPost]

public ActionResult Edit(int id, FormCollection collection)

{

try

{

var emp = helper.search(id);

emp.laptop\_id = Convert.ToInt32(Request["laptop\_id"]);

emp.laptop\_name = Request["laptop\_name"].ToString();

emp.Cost = Convert.ToInt32(Request["Cost"]);

emp.Descr = Request["Descr"].ToString();

bool ans = helper.Edit(emp);

if (ans)

{

return RedirectToAction("Index");

}

else

{

return View();

}

}

catch

{

return View();

}

}

public ActionResult Delete(int id)

{

var emp = helper.search(id);

laptop model = new laptop();

model.laptop\_id = id;

model.laptop\_name = emp.laptop\_name;

model.Cost = emp.Cost;

model.Descr = emp.Descr;

return View(model);

}

[HttpPost]

public ActionResult Delete(int id, FormCollection collection)

{

try

{

var dataFound = helper.search(id);

if (dataFound != null)

{

bool ans = helper.remove(id);

if (ans)

{

return RedirectToAction("Index");

}

else

{

return View();

}

}

return RedirectToAction("Index");

}

catch

{

return View();

}

}

}

}

**ManageController.cs:**

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 mvcccc.Models;

namespace mvcccc.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

}

}

**PayController.cs:**

using BAL;

using Helper;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Remoting.Messaging;

using System.Web;

using System.Web.Mvc;

namespace mvcccc.Controllers

{

public class PayController : Controller

{

// GET: Pay

Helperclass helper = null;

public PayController()

{

helper = new Helperclass();

}

public ActionResult Index()

{

return View();

}

public ActionResult Create()

{

return View();

}

[HttpPost]

public ActionResult Create(FormCollection collection)

{

Credit bal = new Credit();

bal.Cardno = Convert.ToInt32(Request["Cardno"]);

bal.name = Request["name"].ToString();

bal.CVV = Convert.ToInt32(Request["CVV"]);

bool ans = helper.Add(bal);

if (ans)

{

return RedirectToAction("Index");

}

else

{

return View();

}

}

}

}

**Laptop.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace mvcccc.Models

{

public class laptop

{

public int laptop\_id { get; set; }

public string laptop\_name { get; set; }

public int Cost { get; set; }

public string Descr { get; set; }

}

}

**Credit.cs:**

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Web;

namespace mvcccc.Models

{

public class Credit

{

[Required]

public int Cardno { get; set; }

public string name { get; set; }

[Required]

[DataType(DataType.Password)]

public int CVV { get; set; }

}

}

**IdentityModels.cs:**

using System.Data.Entity;

using System.Security.Claims;

using System.Threading.Tasks;

using Microsoft.AspNet.Identity;

using Microsoft.AspNet.Identity.EntityFramework;

namespace mvcccc.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();

}

}

}

**ManageViewsModels.cs:**

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using Microsoft.AspNet.Identity;

using Microsoft.Owin.Security;

namespace mvcccc.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 = 6)]

[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 = 6)]

[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; }

}

}

**Account:**

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

namespace mvcccc.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; }

}

}

**Views:**

**Create.cshtml:**

@model mvcccc.Models.laptop

@{

ViewBag.Title = "Create";

}

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>SAVE TO CART</h4>

<hr />

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

<div class="form-group">

@Html.LabelFor(model => model.laptop\_id, htmlAttributes: new { @class = "control-label col-md-2" })

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

@Html.EditorFor(model => model.laptop\_id, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.laptop\_id, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.laptop\_name, htmlAttributes: new { @class = "control-label col-md-2" })

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

@Html.EditorFor(model => model.laptop\_name, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.laptop\_name, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Cost, htmlAttributes: new { @class = "control-label col-md-2" })

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

@Html.EditorFor(model => model.Cost, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Cost, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Descr, htmlAttributes: new { @class = "control-label col-md-2" })

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

@Html.EditorFor(model => model.Descr, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Descr, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

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

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

</div>

</div>

</div>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

@section Scripts {

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

}

**Delete.cshtml:**

@model mvcccc.Models.laptop

@{

ViewBag.Title = "Delete";

}

<h2>Delete Product</h2>

<h3>Are you sure you want to delete this from cart?</h3>

<div>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.laptop\_id)

</dt>

<dd>

@Html.DisplayFor(model => model.laptop\_id)

</dd>

<dt>

@Html.DisplayNameFor(model => model.laptop\_name)

</dt>

<dd>

@Html.DisplayFor(model => model.laptop\_name)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Cost)

</dt>

<dd>

@Html.DisplayFor(model => model.Cost)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Descr)

</dt>

<dd>

@Html.DisplayFor(model => model.Descr)

</dd>

</dl>

@using (Html.BeginForm()) {

@Html.AntiForgeryToken()

<div class="form-actions no-color">

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

@Html.ActionLink("Back to List", "Index")

</div>

}

</div>

**Details.cshtml:**

@model mvcccc.Models.laptop

@{

ViewBag.Title = "Details";

}

<h2>Details</h2>

<div>

<h4>Laptop purchase with details</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.laptop\_id)

</dt>

<dd>

@Html.DisplayFor(model => model.laptop\_id)

</dd>

<dt>

@Html.DisplayNameFor(model => model.laptop\_name)

</dt>

<dd>

@Html.DisplayFor(model => model.laptop\_name)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Cost)

</dt>

<dd>

@Html.DisplayFor(model => model.Cost)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Descr)

</dt>

<dd>

@Html.DisplayFor(model => model.Descr)

</dd>

</dl>

</div>

<p>

@Html.ActionLink("Back to List", "Index")

</p>

**Index.cshtml:**

@model IEnumerable<mvcccc.Models.laptop>

@{

ViewBag.Title = "Index";

}

<h2>Cart details</h2>

<p>

@Html.ActionLink("Add to Cart", "Create")

</p>

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.laptop\_id)

</th>

<th>

@Html.DisplayNameFor(model => model.laptop\_name)

</th>

<th>

@Html.DisplayNameFor(model => model.Cost)

</th>

<th>

@Html.DisplayNameFor(model => model.Descr)

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.laptop\_id)

</td>

<td>

@Html.DisplayFor(modelItem => item.laptop\_name)

</td>

<td>

@Html.DisplayFor(modelItem => item.Cost)

</td>

<td>

@Html.DisplayFor(modelItem => item.Descr)

</td>

<td>

@Html.ActionLink("BUY", "Pay", new { id=item.laptop\_id }) |

@Html.ActionLink("Details", "Details", new { id=item.laptop\_id }) |

@Html.ActionLink("Delete", "Delete", new { id=item.laptop\_id })

</td>

</tr>

}

</table>