**Приложение Д. Листинг контроллеров приложения**

**Контроллер Account**

[Authorize]

public class AccountController : Controller

{

private ApplicationSignInManager \_signInManager;

private ApplicationUserManager \_userManager;

private OffersContext prdb = new OffersContext();

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;

} }

[AllowAnonymous]

public ActionResult Login(string returnUrl)

{

ViewBag.ReturnUrl = returnUrl;

return View();

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

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

{

if (!ModelState.IsValid)

{

return View(model);

}

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("", "Ошибка входа. Проверьте логин или пароль");

return View(model); } }

[AllowAnonymous]

public ActionResult AccountType()

{

return View();

}

[AllowAnonymous]

public ActionResult Register()

{

return View();

}

[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 UserManager.AddToRoleAsync(user.Id, "RegularUser");

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

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

}

AddErrors(result);

}

return View(model);

}

[AllowAnonymous]

public ActionResult RegisterProvider()

{

return View();

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> RegisterProvider(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 UserManager.AddToRoleAsync(user.Id, "RegularUser");

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

prdb.ProvidersRequests.Add(new ProvidersRequests { Email = user.Email, isConfirmed = false });

prdb.SaveChanges();

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

}

AddErrors(result);

}

return View(model);

}

[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");

}

[AllowAnonymous]

public ActionResult ForgotPassword()

{

return View();

}

[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)))

{

return View("ForgotPasswordConfirmation");

} }

return View(model);

}

[AllowAnonymous]

public ActionResult ForgotPasswordConfirmation()

{

return View();

}

[AllowAnonymous]

public ActionResult ResetPassword(string code)

{

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

}

[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)

{

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();

}

[AllowAnonymous]

public ActionResult ResetPasswordConfirmation()

{

return View();

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult ExternalLogin(string provider, string returnUrl)

{

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

}

[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 });

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> SendCode(SendCodeViewModel model)

{

if (!ModelState.IsValid)

{

return View();

}

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

{

return View("Error");

}

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

}

[AllowAnonymous]

public async Task<ActionResult> ExternalLoginCallback(string returnUrl)

{

var loginInfo = await AuthenticationManager.GetExternalLoginInfoAsync();

if (loginInfo == null)

{

return RedirectToAction("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:

ViewBag.ReturnUrl = returnUrl;

ViewBag.LoginProvider = loginInfo.Login.LoginProvider;

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

} }

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

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

{

if (User.Identity.IsAuthenticated)

{

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

}

if (ModelState.IsValid)

{

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);

}

[AllowAnonymous]

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

{

if (!await SignInManager.HasBeenVerifiedAsync())

{

return View("Error");

}

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

}

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> VerifyCode(VerifyCodeViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

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);

} }

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult LogOff()

{

AuthenticationManager.SignOut(DefaultAuthenticationTypes.ApplicationCookie);

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

}

[AllowAnonymous]

public ActionResult ExternalLoginFailure()

{

return View();

}

**Контроллер Account**

public class AdminController : Controller

{

private OffersContext db = new OffersContext();

private OffersContext db2 = new OffersContext();

private ApplicationDbContext adb = new ApplicationDbContext();

public ActionResult AdministratorPage(int? page)

{

if (User.IsInRole("Admin"))

{

var users = adb.Users.ToList();

ViewBag.Roles = adb.Roles.ToList();

int pageSize = 5;

int pageNumber = (page ?? 1);

return View(users.ToPagedList(pageNumber, pageSize));

}

else

return View("AccessDenied");

}

public async Task<ActionResult> SaveRole(string id, string roleName)

{

if (User.IsInRole("Admin"))

{

var user = adb.Users.Find(id);

var role = adb.Roles.Where(r => r.Name.Equals(roleName));

var rid = user.Roles.First().RoleId;

var oldRole = adb.Roles.Where(r => r.Id == rid);

var userManager = HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();

if (role!=null)

{

var result = await userManager.RemoveFromRoleAsync(id, oldRole.First().Name);

var result1 = await userManager.AddToRoleAsync(id, role.First().Name);

adb.SaveChanges();

}

return RedirectToAction("AdministratorPage", "Admin");

}

else

return View("AccessDenied");

}

[HttpGet]

public ActionResult DeleteUser (string id)

{

if (User.IsInRole("Admin"))

{

var user = adb.Users.Find(id);

if(user!= null)

{

return View(user);

}

else

return HttpNotFound();

}

else

return View("AccessDenied");

}

[HttpPost, ActionName("DeleteUser")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(string id)

{

if (User.IsInRole("Admin"))

{

var user = adb.Users.Find(id);

if (user != null)

adb.Users.Remove(user);

adb.SaveChanges();

return RedirectToAction("AdministratorPage");

}

else

return View("AccessDenied");

}

public ActionResult DeleteOffer(int? id)

{

if (User.IsInRole("Moderator"))

{

List<String> Images = new List<string>();

string[] splitResult;

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Offers offers = db.Offers.Find(id);

if (offers == null)

{

return HttpNotFound();

}

else

{

splitResult = offers.Images.Split(';');

Images = splitResult.ToList();

Images.RemoveAt(Images.Count - 1);

}

ViewBag.Images = Images;

return View(offers);

}

else

return View("AccessDenied"); }

[HttpPost, ActionName("DeleteOffer")]

[ValidateAntiForgeryToken]

public ActionResult DeleteOfferConfirmed(int? id)

{

if (User.IsInRole("Moderator"))

{

Offers offers = db.Offers.Find(id);

ModeratedOffers offer = db.ModeratedOffers.FirstOrDefault(o => o.OfferId == id);

if (offer != null)

db.ModeratedOffers.Remove(offer);

db.Offers.Remove(offers);

db.SaveChanges();

return RedirectToAction("ModeratorPage");

}

else

return View("AccessDenied");

}

public ActionResult ModeratorPage(int? page)

{

if (User.IsInRole("Moderator"))

{

int pageSize = 5;

int pageNumber = (page ?? 1);

ViewBag.AllOffers = db.Offers.ToList();

if(db.ProvidersRequests.ToList().Count != 0)

ViewBag.ProviderRequests = db.ProvidersRequests.ToList();

List<Offers> NonModered = new List<Offers>();

foreach( var q in db.Offers)

{

var item = db2.ModeratedOffers.FirstOrDefault(c => c.OfferId == q.Id);

if(item==null)

{

NonModered.Add(q);

} }

return View(NonModered.ToPagedList(pageNumber, pageSize));

}

else

return View("AccessDenied");

}

public ActionResult ModerateOffer(int? id)

{

if (User.IsInRole("Moderator"))

{

List<String> Images = new List<string>();

string[] splitResult;

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Offers offers = db.Offers.Find(id);

if (offers == null)

{

return HttpNotFound();

}

else

{

splitResult = offers.Images.Split(';');

Images = splitResult.ToList();

Images.RemoveAt(Images.Count - 1);

}

ViewBag.Images = Images;

return View(offers);

}

else

return View("AccessDenied");

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult ModerateOffer(Offers offers)

{

if (User.IsInRole("Moderator"))

{

var item = db.Offers.FirstOrDefault(c => c.Id == offers.Id);

item.Id = offers.Id;

item.Adress = offers.Adress;

item.catalogID = offers.catalogID;

item.CatalogName = offers.CatalogName;

item.Category = offers.Category;

item.City = offers.City;

item.Contacts = offers.Contacts;

item.Description = offers.Description;

item.Provider = offers.Provider;

item.Rating = offers.Rating;

item.ShortDescription = offers.ShortDescription;

item.WorkingHours = offers.WorkingHours;

item.isModerated = true;

db.ModeratedOffers.Add(new ModeratedOffers { OfferId = item.Id, isModerated = true });

db.SaveChanges();

return Redirect("ModeratorPage");

}

else

return View("AccessDenied");

}

public async Task<ActionResult> ConfirmProvider(string email)

{

if (User.IsInRole("Moderator"))

{

var user = adb.Users.FirstOrDefault(u => u.Email == email);

if (user != null)

{

var rid = user.Roles.First().RoleId;

var oldRole = adb.Roles.Where(r => r.Id == rid);

var userManager = HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();

var result = await userManager.RemoveFromRoleAsync(user.Id, oldRole.First().Name);

var result1 = await userManager.AddToRoleAsync(user.Id, "Provider");

var request = db.ProvidersRequests.FirstOrDefault(pr => pr.Email == email);

if(request!=null)

{

request.isConfirmed = true;

db.ProvidersRequests.Remove(request);

db.SaveChanges();

}

adb.SaveChanges();

}

return RedirectToAction("ModeratorPage", "Admin");

}

else

return View("AccessDenied");

}

public ActionResult DeleteRequest(int? id)

{

if (User.IsInRole("Moderator"))

{

var req = db.ProvidersRequests.Find(id);

if (req != null)

{

db.ProvidersRequests.Remove(req);

db.SaveChanges();

}

return RedirectToAction("ModeratorPage", "Admin");

}

else

return View("AccessDenied");

}

public ActionResult SendConfirmation (string email)

{

if (User.IsInRole("Moderator"))

{

MailAddress from = new MailAddress("LifeboyuModerator@yandex.ru", "Lifebouy Moderator");

MailAddress to = new MailAddress(email);

MailMessage m = new MailMessage(from, to);

m.Subject = "Подтверждение роли провайдера";

m.Body = "Вы запросили подтверждение роли провайдера. Если вы получили это письмо по ошибке, просто игнорируйте его.\n " +

"Для завершения подтверждения вам необходимо выслать свои реквизиты на этот адрес: LifeboyuModerator@yandex.ru";

SmtpClient smtp = new SmtpClient("smtp.yandex.ru", 587);

smtp.Port = 587;

smtp.DeliveryMethod = SmtpDeliveryMethod.Network;

smtp.EnableSsl = true;

smtp.Credentials = new NetworkCredential("LifeboyuModerator", "1716141q");

smtp.Send(m);

return RedirectToAction("ModeratorPage", "Admin");

}

else

return View("AccessDenied");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

**Контроллер Catalog**

public class CatalogController : Controller

{

private OffersContext db = new OffersContext();

private OffersContext db1 = new OffersContext();

private ApplicationDbContext adb = new ApplicationDbContext();

public ActionResult Index(string categories, string providers, string cities)

{

IQueryable<Offers> offers = db.Offers;

List<Offers> moderatedOffers = new List<Offers>();

List<int> offIds = new List<int>();

var mOffers = db.ModeratedOffers;

foreach ( var q in mOffers)

{

offIds.Add(q.OfferId);

}

foreach (var q in offIds)

{

var offer = offers.FirstOrDefault(o => o.Id == q);

if(offer!=null)

moderatedOffers.Add(offer);

}

offers = moderatedOffers.AsQueryable();

if (!String.IsNullOrEmpty(categories)&&!categories.Equals("Не выбрано"))

{

offers = offers.Where(o => o.Category == categories);

}

if (!String.IsNullOrEmpty(providers) && !providers.Equals("Не выбрано"))

{

offers = offers.Where(o => o.Provider == providers);

}

if (!String.IsNullOrEmpty(cities) && !cities.Equals("Не выбрано"))

{

offers = offers.Where(o => o.City == cities);

}

var city = db.Offers

.OrderBy(c => c.City)

.Select(c => c.City).Distinct().ToList();

var category = db.Offers

.OrderBy(c => c.Category)

.Select(c => c.Category).Distinct().ToList();

var provider = db.Offers

.OrderBy(c => c.Provider)

.Select(c => c.Provider).Distinct().ToList();

city.Insert(0, "Не выбрано");

provider.Insert(0, "Не выбрано");

category.Insert(0, "Не выбрано");

offers = offers.OrderByDescending(o => o.Rating);

OffersListView olv = new OffersListView

{

Offers = offers.ToList(),

Categories = new SelectList(category),

Providers = new SelectList(provider),

Cities = new SelectList(city)

};

return View(olv);

}

public ActionResult ShowOffer(int? id)

{

List<String> Images = new List<string>();

string[] splitResult;

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Offers offer = db.Offers.Find(id);

splitResult = offer.Images.Split(';');

Images = splitResult.ToList();

Images.RemoveAt(Images.Count - 1);

if (offer == null)

{

return HttpNotFound();

}

ViewBag.Images = Images;

return View(offer);

}

[HttpPost]

public ActionResult SendRating(int id, int rating)

{

if(User.Identity.IsAuthenticated)

{

var userVote = db.UsersVotes.FirstOrDefault(uv => uv.UserName == User.Identity.Name & uv.OfferId == id);

if (userVote == null)

{

var offer = db.Offers.Find(id);

if (offer != null)

{

if (offer.Rating == 0)

{

offer.Rating = rating;

}

else

offer.Rating = Convert.ToInt32(Math.Ceiling(Convert.ToDouble((offer.Rating + rating) / 2)));

db.UsersVotes.Add(new UsersVotes { OfferId = id, UserName = User.Identity.Name });

offer.VotesCount += 1;

db.SaveChanges();

return Json(new { success = true, responseText = "Ваш голос учтён." }, JsonRequestBehavior.AllowGet);

}

else

return Json(new { success = false, responseText = "Призошла ошибка." }, JsonRequestBehavior.AllowGet);

}

else

{

return Json(new { success = false, responseText = "Вы уже голосовали." }, JsonRequestBehavior.AllowGet);

}

}

else

return Json(new { success = false, responseText = "Авторизуйтесь для голосования" }, JsonRequestBehavior.AllowGet);

}

}

**Контроллер Constructor**

public class ConstructorController : Controller

{

private OffersContext db = new OffersContext();

public ActionResult Dashboard()

{

if(User.IsInRole("Provider"))

{

var userName = User.Identity.Name;

IQueryable<ProvidersOffers> providerOffers = db.ProviderOffers;

List<Offers> offers = new List<Offers>();

List<int> offIds = new List<int>();

providerOffers = providerOffers.Where(o => o.Owner == User.Identity.Name);

foreach (var q in providerOffers)

{

offIds.Add(q.OfferId);

}

foreach (var q in offIds)

{

var offer = db.Offers.FirstOrDefault(o => o.Id == q);

if (offer != null)

offers.Add(offer);

}

return View(offers);

}

else

return View("AccessDenied");

}

public ActionResult Details(int? id)

{

if (User.IsInRole("Provider"))

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Offers offers = db.Offers.Find(id);

if (offers == null)

{

return HttpNotFound();

}

return View(offers);

}

else

return View("AccessDenied");

}

public ActionResult Create()

{

if (User.IsInRole("Provider"))

{

return View();

}

else

return View("AccessDenied");

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create(IEnumerable<HttpPostedFileBase> files, Offers offers)

{

if (User.IsInRole("Provider"))

{

if (ModelState.IsValid)

{

string images = "";

if (files != null)

foreach (var file in files)

{

if (file != null && file.ContentLength > 0)

{

byte[] imageData = null;

using (var binaryReader = new BinaryReader(file.InputStream))

{

imageData = binaryReader.ReadBytes(file.ContentLength);

}

images += Convert.ToBase64String(imageData) + ";";

}

}

var id = offers.Provider.Split('@')[0] + "\_" + offers.Category+"\_" + offers.CatalogName;

offers.catalogID = Transliter.GetTranslit(id);

offers.Images = images;

db.Offers.Add(offers);

db.SaveChanges();

db.ProviderOffers.Add(new ProvidersOffers { OfferId = offers.Id, Owner = User.Identity.Name });

db.SaveChanges();

return RedirectToAction("Dashboard");

}

return View(offers);

}

else

return View("AccessDenied");

}

public ActionResult Edit(int? id)

{

if (User.IsInRole("Provider"))

{

List<String> Images = new List<string>();

string[] splitResult;

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Offers offers = db.Offers.Find(id);

if (offers == null)

{

return HttpNotFound();

}

else

{

splitResult = offers.Images.Split(';');

Images = splitResult.ToList();

Images.RemoveAt(Images.Count - 1);

}

ViewBag.Images = Images;

return View(offers);

}

else

return View("AccessDenied");

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit(IEnumerable<HttpPostedFileBase> files, Offers offers)

{

if (User.IsInRole("Provider"))

{

if (ModelState.IsValid)

{

var item = db.Offers.FirstOrDefault(c => c.Id == offers.Id);

string images = "";

if (files != null)

{

foreach (var file in files)

{

if (file != null && file.ContentLength > 0)

{

byte[] imageData = null;

using (var binaryReader = new BinaryReader(file.InputStream))

{

imageData = binaryReader.ReadBytes(file.ContentLength);

}

images += Convert.ToBase64String(imageData) + ";";

}

}

}

item.Id = offers.Id;

item.Adress = offers.Adress;

item.catalogID = offers.catalogID;

item.CatalogImg = offers.CatalogImg;

item.CatalogName = offers.CatalogName;

item.Category = offers.Category;

item.City = offers.City;

item.Contacts = offers.Contacts;

item.Description = offers.Description;

item.Provider = offers.Provider;

item.Prices = offers.Prices;

item.Rating = offers.Rating;

item.ShortDescription = offers.ShortDescription;

item.WorkingHours = offers.WorkingHours;

item.isModerated = false;

var tmp = db.ModeratedOffers.FirstOrDefault(o => o.OfferId == item.Id);

if(tmp!=null)

db.ModeratedOffers.Remove(tmp);

db.SaveChanges();

return RedirectToAction("Dashboard");

}

return View(offers);

}

else

return View("AccessDenied");

}

public ActionResult Delete(int? id)

{

if (User.IsInRole("Provider"))

{

List<String> Images = new List<string>();

string[] splitResult;

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Offers offers = db.Offers.Find(id);

if (offers == null)

{

return HttpNotFound();

}

else

{

splitResult = offers.Images.Split(';');

Images = splitResult.ToList();

Images.RemoveAt(Images.Count - 1);

}

ViewBag.Images = Images;

return View(offers);

}

else

return View("AccessDenied");

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

if (User.IsInRole("Provider"))

{

Offers offers = db.Offers.Find(id);

ModeratedOffers offer = db.ModeratedOffers.FirstOrDefault(o => o.OfferId == id);

if (offer != null)

db.ModeratedOffers.Remove(offer);

db.Offers.Remove(offers);

db.SaveChanges();

return RedirectToAction("Dashboard");

}

else

return View("AccessDenied");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}