IT2030 Assignment 16 Matt Brown

**BookstoreContext.cs**

using System;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.AspNetCore.Identity;

namespace Bookstore.Models

{

public class BookstoreContext : IdentityDbContext<User>

{

public BookstoreContext(DbContextOptions<BookstoreContext> options)

: base(options)

{ }

public DbSet<Author> Authors { get; set; }

public DbSet<Book> Books { get; set; }

public DbSet<BookAuthor> BookAuthors { get; set; }

public DbSet<Genre> Genres { get; set; }

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

// BookAuthor: set primary key

modelBuilder.Entity<BookAuthor>().HasKey(ba => new { ba.BookId, ba.AuthorId });

// BookAuthor: set foreign keys

modelBuilder.Entity<BookAuthor>().HasOne(ba => ba.Book)

.WithMany(b => b.BookAuthors)

.HasForeignKey(ba => ba.BookId);

modelBuilder.Entity<BookAuthor>().HasOne(ba => ba.Author)

.WithMany(a => a.BookAuthors)

.HasForeignKey(ba => ba.AuthorId);

// Book: remove cascading delete with Genre

modelBuilder.Entity<Book>().HasOne(b => b.Genre)

.WithMany(g => g.Books)

.OnDelete(DeleteBehavior.Restrict);

// seed initial data

modelBuilder.ApplyConfiguration(new SeedGenres());

modelBuilder.ApplyConfiguration(new SeedBooks());

modelBuilder.ApplyConfiguration(new SeedAuthors());

modelBuilder.ApplyConfiguration(new SeedBookAuthors());

}

public static async Task CreateAdminUser(IServiceProvider serviceProvider)

{

UserManager<User> userManager =

serviceProvider.GetRequiredService<UserManager<User>>();

RoleManager<IdentityRole> roleManager =

serviceProvider.GetRequiredService<RoleManager<IdentityRole>>();

string username = "admin";

string password = "Sesame";

string roleName = "Admin";

// if role doesn't exist, create it

if (await roleManager.FindByNameAsync(roleName) == null)

{

await roleManager.CreateAsync(new IdentityRole(roleName));

}

// if username doesn't exist, create it and add to role

if (await userManager.FindByNameAsync(username) == null)

{

User user = new User { UserName = username };

var result = await userManager.CreateAsync(user, password);

if (result.Succeeded)

{

await userManager.AddToRoleAsync(user, roleName);

}

}

}

}

}

**Program.cs**

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Hosting;

namespace Bookstore

{

public class Program

{

public static void Main(string[] args)

{

CreateHostBuilder(args).Build().Run();

}

public static IHostBuilder CreateHostBuilder(string[] args) =>

Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(webBuilder =>

{

webBuilder.UseStartup<Startup>()

.UseDefaultServiceProvider(options => options.ValidateScopes = false);

});

}

}

**Startup.cs**

using Microsoft.AspNetCore.Builder;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Microsoft.EntityFrameworkCore;

using Microsoft.AspNetCore.Identity; // add this

using Bookstore.Models;

namespace Bookstore

{

public class Startup

{

public Startup(IConfiguration configuration)

{

Configuration = configuration;

}

public IConfiguration Configuration { get; }

// Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddRouting(options => options.LowercaseUrls = true);

services.AddMemoryCache();

services.AddSession();

services.AddControllersWithViews().AddNewtonsoftJson();

services.AddDbContext<BookstoreContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("BookstoreContext")));

// add this

services.AddIdentity<User, IdentityRole>(options => {

options.Password.RequiredLength = 6;

options.Password.RequireNonAlphanumeric = false;

options.Password.RequireDigit = false;

}).AddEntityFrameworkStores<BookstoreContext>()

.AddDefaultTokenProviders();

}

// Use this method to configure the HTTP request pipeline.

public void Configure(IApplicationBuilder app)

{

app.UseDeveloperExceptionPage();

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication(); // add this

app.UseAuthorization(); // add this

app.UseSession();

app.UseEndpoints(endpoints =>

{

// route for Admin area

endpoints.MapAreaControllerRoute(

name: "admin",

areaName: "Admin",

pattern: "Admin/{controller=Book}/{action=Index}/{id?}");

// route for paging, sorting, and filtering

endpoints.MapControllerRoute(

name: "",

pattern: "{controller}/{action}/page/{pagenumber}/size/{pagesize}/sort/{sortfield}/{sortdirection}/filter/{author}/{genre}/{price}");

// route for paging and sorting only

endpoints.MapControllerRoute(

name: "",

pattern: "{controller}/{action}/page/{pagenumber}/size/{pagesize}/sort/{sortfield}/{sortdirection}");

// default route

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}/{slug?}");

});

BookstoreContext.CreateAdminUser(app.ApplicationServices).Wait();

}

}

}

**User.cs**

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations.Schema;

using Microsoft.AspNetCore.Identity;

namespace Bookstore.Models

{

public class User : IdentityUser

{

public string Firstname { get; set; }

public string Lastname { get; set; }

[NotMapped]

public IList<string> RoleNames { get; set; }

}

}

**RegisterViewModel.cs**

using System.ComponentModel.DataAnnotations;

namespace Bookstore.Models

{

public class RegisterViewModel

{

[Required(ErrorMessage = "Please enter a username.")]

[StringLength(255)]

public string Username { get; set; }

[Required(ErrorMessage = "Please enter a first name.")]

[StringLength(255)]

public string Firstname { get; set; }

[Required(ErrorMessage = "Please enter a last name.")]

[StringLength(255)]

public string Lastname { get; set; }

[Required(ErrorMessage = "Please enter an email address.")]

[DataType(DataType.EmailAddress)]

public string Email { get; set; }

[Required(ErrorMessage = "Please enter a password.")]

[DataType(DataType.Password)]

[Compare("ConfirmPassword")]

public string Password { get; set; }

[Required(ErrorMessage = "Please confirm your password.")]

[DataType(DataType.Password)]

[Display(Name = "Confirm Password")]

public string ConfirmPassword { get; set; }

}

}

**AccountController.cs**

using System.Threading.Tasks;

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Identity;

using Bookstore.Models;

namespace Bookstore.Controllers

{

public class AccountController : Controller

{

private UserManager<User> userManager;

private SignInManager<User> signInManager;

public AccountController(UserManager<User> userMngr,

SignInManager<User> signInMngr)

{

userManager = userMngr;

signInManager = signInMngr;

}

[HttpGet]

public IActionResult Register()

{

return View();

}

[HttpPost]

public async Task<IActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

var user = new Bookstore.Models.User

{

UserName = model.Username,

Firstname = model.Firstname,

Lastname = model.Lastname,

Email = model.Email

};

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

if (result.Succeeded)

{

await signInManager.SignInAsync(user, isPersistent: false);

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

}

else

{

foreach (var error in result.Errors)

{

ModelState.AddModelError("", error.Description);

}

}

}

return View(model);

}

[HttpPost]

public async Task<IActionResult> LogOut()

{

await signInManager.SignOutAsync();

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

}

[HttpGet]

public IActionResult LogIn(string returnURL = "")

{

var model = new LoginViewModel { ReturnUrl = returnURL };

return View(model);

}

[HttpPost]

public async Task<IActionResult> LogIn(LoginViewModel model)

{

if (ModelState.IsValid)

{

var result = await signInManager.PasswordSignInAsync(

model.Username, model.Password, isPersistent: model.RememberMe,

lockoutOnFailure: false);

if (result.Succeeded)

{

if (!string.IsNullOrEmpty(model.ReturnUrl) &&

Url.IsLocalUrl(model.ReturnUrl))

{

return Redirect(model.ReturnUrl);

}

else

{

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

}

}

}

ModelState.AddModelError("", "Invalid username/password.");

return View(model);

}

public ViewResult AccessDenied()

{

return View();

}

}

}

**Register.cshtml**

@model RegisterViewModel

@{

ViewBag.Title = "Register";

}

<h2>Register</h2>

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<form method="post" asp-action="Register">

<div class="form-group row">

<div class="col-sm-2"><label>Username:</label></div>

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

<input type="text" asp-for="Username"

class="form-control" />

</div>

<div class="col">

<span asp-validation-for="Username"

class="text-danger"></span>

</div>

</div>

<div class="form-group row">

<div class="col-sm-2"><label>First Name:</label></div>

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

<input type="text" asp-for="Firstname"

class="form-control" />

</div>

<div class="col">

<span asp-validation-for="Firstname"

class="text-danger"></span>

</div>

</div>

<div class="form-group row">

<div class="col-sm-2"><label>Last Name:</label></div>

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

<input type="text" asp-for="Lastname"

class="form-control" />

</div>

<div class="col">

<span asp-validation-for="Lastname"

class="text-danger"></span>

</div>

</div>

<div class="form-group row">

<div class="col-sm-2"><label>Email Address:</label></div>

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

<input type="text" asp-for="Email"

class="form-control" />

</div>

<div class="col">

<span asp-validation-for="Email"

class="text-danger"></span>

</div>

</div>

<div class="form-group row">

<div class="col-sm-2"><label>Password:</label></div>

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

<input type="password" asp-for="Password"

class="form-control" />

</div>

<div class="col">

<span asp-validation-for="Password"

class="text-danger"></span>

</div>

</div>

<div class="form-group row">

<div class="col-sm-2"><label>Confirm Password:</label></div>

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

<input type="password" asp-for="ConfirmPassword"

class="form-control" />

</div>

</div>

<div class="row">

<div class="offset-2 col-sm-4">

<button type="submit" class="btn btn-primary">Register</button>

</div>

</div>

<div class="row">

<div class="offset-2 col-sm-4">

Already registered? <a asp-action="LogIn">Log In</a>

</div>

</div>

</form>