Appsetting.json:

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=aspnet-MvcExample-014b4166-fb86-417c-b509-4313b550e82d;Trusted\_Connection=True;MultipleActiveResultSets=true";

}

-Models:

1.)ApplicationUser.cs

using Microsoft.AspNetCore.Identity;

namespace SecureShop.Models

{

public class ApplicationUser : IdentityUser

{

}

}

2.)Product.cs

using System.ComponentModel.DataAnnotations;

namespace SecureShop.Models

{

public class Product

{

public int Id { get; set; }

[Required, StringLength(80)]

public string Name { get; set; } = "";

[Required, Range(0.01, 999999)]

public decimal Price { get; set; }

[StringLength(500)]

public string? Description { get; set; } // Razor encodes on output -> prevents XSS

}

}

3.)Order.cs

using System.ComponentModel.DataAnnotations;

namespace SecureShop.Models

{

public class Order

{

public int Id { get; set; }

[Required] public string UserId { get; set; } = "";

[Required] public int ProductId { get; set; }

[Range(1, 9999)]

public int Quantity { get; set; } = 1;

public Product? Product { get; set; }

}

}

4.)RegisterViewModel.cs

using System.ComponentModel.DataAnnotations;

namespace SecureShop.Models

{

public class RegisterViewModel

{

[Required, EmailAddress]

public string Email { get; set; } = "";

// Min 8, at least 1 upper, 1 digit, 1 special

[Required, DataType(DataType.Password)]

[RegularExpression(@"^(?=.\*[A-Z])(?=.\*\d)(?=.\*[^A-Za-z0-9]).{8,}$",

ErrorMessage = "Password must be 8+ chars with uppercase, number & special char.")]

public string Password { get; set; } = "";

[DataType(DataType.Password)]

[Compare(nameof(Password))]

public string ConfirmPassword { get; set; } = "";

}

}

-DbContext

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using SecureShop.Models;

namespace SecureShop.Data

{

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) { }

public DbSet<Product> Products => Set<Product>();

public DbSet<Order> Orders => Set<Order>();

}

}

-Program.cs

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using SecureShop.Data;

using SecureShop.Models;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

// Identity (cookies, lockout = simple brute-force mitigation)

builder.Services.AddIdentity<ApplicationUser, IdentityRole>(options =>

{

options.Password.RequireDigit = true;

options.Password.RequireUppercase = true;

options.Password.RequireNonAlphanumeric = true;

options.Password.RequiredLength = 8;

options.Lockout.AllowedForNewUsers = true;

options.Lockout.MaxFailedAccessAttempts = 5;

options.User.RequireUniqueEmail = true;

})

.AddEntityFrameworkStores<ApplicationDbContext>()

.AddDefaultTokenProviders();

builder.Services.ConfigureApplicationCookie(opt =>

{

opt.LoginPath = "/Account/Login";

opt.AccessDeniedPath = "/Account/AccessDenied";

});

builder.Services.AddControllersWithViews();

var app = builder.Build();

// Auto-migrate and seed roles/users on startup (dev convenience)

using (var scope = app.Services.CreateScope())

{

var ctx = scope.ServiceProvider.GetRequiredService<ApplicationDbContext>();

ctx.Database.Migrate();

await SeedData.Run(scope.ServiceProvider);

}

// Basic secure headers (minimal)

app.Use(async (ctx, next) =>

{

ctx.Response.Headers["X-Content-Type-Options"] = "nosniff";

ctx.Response.Headers["X-Frame-Options"] = "DENY";

ctx.Response.Headers["X-XSS-Protection"] = "0"; // modern browsers rely on CSP/encoding

await next();

});

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

}

app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllerRoute(

name: "default",

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

app.Run();

-SeedData.cs:

using Microsoft.AspNetCore.Identity;

using Microsoft.Extensions.DependencyInjection;

using SecureShop.Models;

namespace SecureShop.Data

{

public static class SeedData

{

public static async Task Run(IServiceProvider services)

{

var roleMgr = services.GetRequiredService<RoleManager<IdentityRole>>();

var userMgr = services.GetRequiredService<UserManager<ApplicationUser>>();

foreach (var role in new[] { "Admin", "Customer" })

if (!await roleMgr.RoleExistsAsync(role))

await roleMgr.CreateAsync(new IdentityRole(role));

var adminEmail = "admin@shop.com";

var admin = await userMgr.FindByEmailAsync(adminEmail);

if (admin is null)

{

admin = new ApplicationUser { UserName = adminEmail, Email = adminEmail, EmailConfirmed = true };

await userMgr.CreateAsync(admin, "Admin@123!");

await userMgr.AddToRoleAsync(admin, "Admin");

}

}

}

}

-Controllers:

1.)ProductController.cs:

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using SecureShop.Data;

using SecureShop.Models;

namespace SecureShop.Controllers

{

public class ProductsController : Controller

{

private readonly ApplicationDbContext \_db;

public ProductsController(ApplicationDbContext db) => \_db = db;

[AllowAnonymous]

public async Task<IActionResult> Index(string? q, string? sort = "name")

{

// EF Core LINQ -> parameterized under the hood (prevents SQLi)

var items = \_db.Products.AsQueryable();

if (!string.IsNullOrWhiteSpace(q))

items = items.Where(p => p.Name.Contains(q));

items = sort == "price" ? items.OrderBy(p => p.Price) : items.OrderBy(p => p.Name);

return View(await items.AsNoTracking().ToListAsync());

}

[Authorize(Roles = "Admin")]

public IActionResult Create() => View();

[Authorize(Roles = "Admin"), HttpPost, ValidateAntiForgeryToken]

public async Task<IActionResult> Create(Product product)

{

if (!ModelState.IsValid) return View(product);

\_db.Products.Add(product);

await \_db.SaveChangesAsync();

TempData["msg"] = "Product created";

return RedirectToAction(nameof(Index));

}

// Example of SAFE raw SQL (still parameterized)

[Authorize(Roles = "Admin")]

public async Task<IActionResult> FastCount()

{

var count = await \_db.Products.FromSqlRaw("SELECT \* FROM Products").CountAsync();

return Content($"Total products: {count}");

}

}

}

2.) OrdersController.cs:

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using SecureShop.Data;

using SecureShop.Models;

using System.Security.Claims;

namespace SecureShop.Controllers

{

[Authorize(Roles = "Customer,Admin")]

public class OrdersController : Controller

{

private readonly ApplicationDbContext \_db;

public OrdersController(ApplicationDbContext db) => \_db = db;

public async Task<IActionResult> Create(int productId)

{

var product = await \_db.Products.FindAsync(productId);

if (product == null) return NotFound();

return View(new Order { ProductId = productId, Quantity = 1, Product = product });

}

[HttpPost, ValidateAntiForgeryToken]

public async Task<IActionResult> Create(Order order)

{

if (!ModelState.IsValid) return View(order);

order.UserId = User.FindFirstValue(ClaimTypes.NameIdentifier)!;

\_db.Orders.Add(order);

await \_db.SaveChangesAsync();

TempData["msg"] = "Order placed!";

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

}

}

}

3.) AccountController.cs:

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using SecureShop.Models;

namespace SecureShop.Controllers

{

public class AccountController : Controller

{

private readonly UserManager<ApplicationUser> \_userMgr;

private readonly SignInManager<ApplicationUser> \_signInMgr;

public AccountController(UserManager<ApplicationUser> u, SignInManager<ApplicationUser> s)

{ \_userMgr = u; \_signInMgr = s; }

[AllowAnonymous]

public IActionResult Register() => View();

[AllowAnonymous, HttpPost, ValidateAntiForgeryToken]

public async Task<IActionResult> Register(RegisterViewModel vm)

{

if (!ModelState.IsValid) return View(vm);

var user = new ApplicationUser { UserName = vm.Email, Email = vm.Email, EmailConfirmed = true };

var res = await \_userMgr.CreateAsync(user, vm.Password);

if (res.Succeeded)

{

await \_userMgr.AddToRoleAsync(user, "Customer");

await \_signInMgr.SignInAsync(user, isPersistent: false);

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

}

foreach (var e in res.Errors) ModelState.AddModelError("", e.Description);

return View(vm);

}

[AllowAnonymous]

public IActionResult Login(string? returnUrl = null) => View((object?)returnUrl);

[AllowAnonymous, HttpPost, ValidateAntiForgeryToken]

public async Task<IActionResult> Login(string email, string password, string? returnUrl = null)

{

var res = await \_signInMgr.PasswordSignInAsync(email, password, false, lockoutOnFailure: true);

if (res.Succeeded)

return !string.IsNullOrEmpty(returnUrl) ? Redirect(returnUrl) : RedirectToAction("Index", "Products");

ModelState.AddModelError("", "Invalid credentials or locked out");

return View(returnUrl);

}

[Authorize]

public async Task<IActionResult> Logout()

{

await \_signInMgr.SignOutAsync(); // invalidates cookie (secure logout)

return RedirectToAction(nameof(Login));

}

[AllowAnonymous]

public IActionResult AccessDenied() => View();

}

}

4.) AdminController.cs:

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using SecureShop.Data;

namespace SecureShop.Controllers

{

[Authorize(Roles = "Admin")]

public class AdminController : Controller

{

private readonly ApplicationDbContext \_db;

public AdminController(ApplicationDbContext db) => \_db = db;

public async Task<IActionResult> Dashboard()

{

var stats = new {

Products = await \_db.Products.CountAsync(),

Orders = await \_db.Orders.CountAsync()

};

return View(stats);

}

}

}

Views:

Views/Shared/\_Layout.cshtml

<li><a asp-controller="Products" asp-action="Index">Products</a></li>

<li><a asp-controller="Admin" asp-action="Dashboard">Admin</a></li>

@if (User.Identity!.IsAuthenticated)

{

<li><a asp-controller="Account" asp-action="Logout">Logout</a></li>

}

else

{

<li><a asp-controller="Account" asp-action="Login">Login</a></li>

<li><a asp-controller="Account" asp-action="Register">Register</a></li>

}

2.) Views/Products/Index.cshtml

@model IEnumerable<SecureShop.Models.Product>

@{

ViewData["Title"] = "Products";

}

<h2>Products</h2>

<form method="get" class="mb-3">

<input name="q" value="@Context.Request.Query["q"]" placeholder="Search..." />

<select name="sort">

<option value="name">Name</option>

<option value="price" @(Context.Request.Query["sort"]=="price"?"selected":"")>Price</option>

</select>

<button type="submit">Filter</button>

</form>

@if (TempData["msg"] is string m) { <div class="alert alert-success">@m</div> }

<table class="table">

<thead><tr><th>Name</th><th>Price</th><th></th></tr></thead>

<tbody>

@foreach (var p in Model)

{

<tr>

<td>@p.Name</td>

<td>@p.Price</td>

<td>

@if (User.IsInRole("Customer") || User.IsInRole("Admin"))

{

<a asp-controller="Orders" asp-action="Create" asp-route-productId="@p.Id">Buy</a>

}

</td>

</tr>

}

</tbody>

</table>

@if (User.IsInRole("Admin"))

{

<a asp-action="Create">+ Create Product</a>

}

3.) Views/Products/Create.cshtml:

@model SecureShop.Models.Product

<h2>Create Product</h2>

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

@Html.AntiForgeryToken()

<div>Name: <input asp-for="Name" /></div>

<div>Price: <input asp-for="Price" /></div>

<div>Description: <textarea asp-for="Description"></textarea></div>

<span asp-validation-summary="All"></span>

<button type="submit">Save</button>

</form>

@section Scripts { <partial name="\_ValidationScriptsPartial" /> }

4.) Views/Orders/Create.cshtml:

@model SecureShop.Models.Order

<h2>Place Order</h2>

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

@Html.AntiForgeryToken()

<input type="hidden" asp-for="ProductId" />

<div>Product: @Model.Product!.Name</div>

<div>Quantity: <input asp-for="Quantity" /></div>

<button type="submit">Confirm</button>

</form>

5.) Views/Account/Register.cshtml: @model SecureShop.Models.RegisterViewModel

<h2>Register</h2>

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

@Html.AntiForgeryToken()

<div>Email: <input asp-for="Email" /></div>

<div>Password: <input asp-for="Password" /></div>

<div>Confirm: <input asp-for="ConfirmPassword" /></div>

<span asp-validation-summary="All"></span>

<button type="submit">Register</button>

</form>

@section Scripts { <partial name="\_ValidationScriptsPartial" /> }

6.) Views/Account/Login.cshtml: @model string

<h2>Login</h2>

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

@Html.AntiForgeryToken()

<div>Email: <input name="email" type="email" required /></div>

<div>Password: <input name="password" type="password" required /></div>

<button type="submit">Login</button>

@if (!string.IsNullOrWhiteSpace(Model)) { <input type="hidden" name="returnUrl" value="@Model" /> }

</form>

7.) Views/Account/AccessDenied.cshtml: <h2>Access Denied</h2>

<p>You do not have permission to access this page.</p>

