**Asp .Net Core Secure Application**

**Controller Code :**

1.Account:

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Identity;

using SecureShoppingApp.Models;

using SecureShoppingApp.Services;

using Microsoft.AspNetCore.Html;

using System.Text.Encodings.Web;

namespace SecureShoppingApp.Controllers

{

    [AllowAnonymous]

    public class AccountController : Controller

    {

        private readonly IAuthService \_authService;

        private readonly ILogger<AccountController> \_logger;

        private readonly HtmlEncoder \_htmlEncoder;

        public AccountController(IAuthService authService, ILogger<AccountController> logger, HtmlEncoder htmlEncoder)

        {

            \_authService = authService;

            \_logger = logger;

            \_htmlEncoder = htmlEncoder;

        }

        [HttpGet]

        public IActionResult Register(string? returnUrl = null)

        {

            ViewData["ReturnUrl"] = returnUrl;

            return View();

        }

        [HttpPost]

        [ValidateAntiForgeryToken]

        public async Task<IActionResult> Register(RegisterViewModel model, string? returnUrl = null)

        {

            ViewData["ReturnUrl"] = returnUrl;

            if (!ModelState.IsValid)

            {

                return View(model);

            }

            // Additional server-side validation

            if (!IsValidEmail(model.Email))

            {

                ModelState.AddModelError("Email", "Please provide a valid email address");

                return View(model);

            }

            if (!IsStrongPassword(model.Password))

            {

                ModelState.AddModelError("Password", "Password does not meet security requirements");

                return View(model);

            }

            // Sanitize inputs

            model.FirstName = \_htmlEncoder.Encode(model.FirstName.Trim());

            model.LastName = \_htmlEncoder.Encode(model.LastName.Trim());

            model.Address = \_htmlEncoder.Encode(model.Address.Trim());

            try

            {

                var result = await \_authService.RegisterUserAsync(model);

                if (result.Succeeded)

                {

                    \_logger.LogInformation("User {Email} registered successfully", model.Email);

                    TempData["SuccessMessage"] = "Registration successful! Please log in.";

                    return RedirectToAction("Login", new { returnUrl });

                }

                foreach (var error in result.Errors)

                {

                    ModelState.AddModelError(string.Empty, error.Description);

                }

            }

            catch (Exception ex)

            {

                \_logger.LogError(ex, "Registration error for {Email}", model.Email);

                ModelState.AddModelError(string.Empty, "Registration failed. Please try again.");

            }

            return View(model);

        }

        [HttpGet]

        public IActionResult Login(string? returnUrl = null)

        {

            ViewData["ReturnUrl"] = returnUrl;

            return View();

        }

        [HttpPost]

        [ValidateAntiForgeryToken]

        public async Task<IActionResult> Login(LoginViewModel model, string? returnUrl = null)

        {

            ViewData["ReturnUrl"] = returnUrl;

            if (!ModelState.IsValid)

            {

                return View(model);

            }

            // Rate limiting check (simplified)

            var clientIP = HttpContext.Connection.RemoteIpAddress?.ToString() ?? "unknown";

            if (await IsRateLimitedAsync(clientIP))

            {

                ModelState.AddModelError(string.Empty, "Too many login attempts. Please try again later.");

                return View(model);

            }

            try

            {

                var result = await \_authService.LoginUserAsync(model);

                if (result.Succeeded)

                {

                    \_logger.LogInformation("User {Email} logged in successfully", model.Email);

                    if (!string.IsNullOrEmpty(returnUrl) && Url.IsLocalUrl(returnUrl))

                    {

                        return Redirect(returnUrl);

                    }

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

                }

                if (result.IsLockedOut)

                {

                    \_logger.LogWarning("User {Email} account locked out", model.Email);

                    ModelState.AddModelError(string.Empty, "Account locked due to multiple failed attempts.");

                }

                else

                {

                    ModelState.AddModelError(string.Empty, "Invalid login attempt.");

                }

            }

            catch (Exception ex)

            {

                \_logger.LogError(ex, "Login error for {Email}", model.Email);

                ModelState.AddModelError(string.Empty, "Login failed. Please try again.");

            }

            return View(model);

        }

        [HttpPost]

        [Authorize]

        [ValidateAntiForgeryToken]

        public async Task<IActionResult> Logout()

        {

            await \_authService.LogoutUserAsync();

            \_logger.LogInformation("User logged out");

            // Clear any additional cookies or session data

            HttpContext.Session.Clear();

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

        }

        [HttpGet]

        public IActionResult AccessDenied()

        {

            return View();

        }

        private static bool IsValidEmail(string email)

        {

            try

            {

                var addr = new System.Net.Mail.MailAddress(email);

                return addr.Address == email;

            }

            catch

            {

                return false;

            }

        }

        private static bool IsStrongPassword(string password)

        {

            if (password.Length < 8) return false;

            if (!password.Any(char.IsUpper)) return false;

            if (!password.Any(char.IsLower)) return false;

            if (!password.Any(char.IsDigit)) return false;

            if (!password.Any(ch => "!@#$%^&\*()\_+-=[]{}|;:,.<>?".Contains(ch))) return false;

            return true;

        }

        private async Task<bool> IsRateLimitedAsync(string clientIP)

        {

            // Simplified rate limiting - in production, use Redis or memory cache

            // This is a basic implementation

            var cacheKey = $"login\_attempts\_{clientIP}";

            var attempts = HttpContext.Session.GetInt32(cacheKey) ?? 0;

            if (attempts >= 5)

            {

                return true;

            }

            HttpContext.Session.SetInt32(cacheKey, attempts + 1);

            return false;

        }

    }

}

2.Admin :

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Authorization;

using Microsoft.EntityFrameworkCore;

using SecureShoppingApp.Data;

using SecureShoppingApp.Models;

using System.Text.Encodings.Web;

namespace SecureShoppingApp.Controllers

{

    [Authorize(Roles = "Admin")]

    public class AdminController : Controller

    {

        private readonly ApplicationDbContext \_context;

        private readonly ILogger<AdminController> \_logger;

        private readonly HtmlEncoder \_htmlEncoder;

        public AdminController(ApplicationDbContext context, ILogger<AdminController> logger, HtmlEncoder htmlEncoder)

        {

            \_context = context;

            \_logger = logger;

            \_htmlEncoder = htmlEncoder;

        }

        [HttpGet]

        public async Task<IActionResult> Dashboard()

        {

            try

            {

                var model = new AdminDashboardViewModel

                {

                    TotalUsers = await \_context.Users.CountAsync(),

                    TotalProducts = await \_context.Products.CountAsync(),

                    TotalOrders = await \_context.Orders.CountAsync(),

                    TotalRevenue = await \_context.Orders

                        .Where(o => o.Status == "Completed")

                        .SumAsync(o => o.TotalAmount),

                    RecentOrders = await \_context.Orders

                        .Include(o => o.User)

                        .OrderByDescending(o => o.OrderDate)

                        .Take(10)

                        .Select(o => new OrderSummaryViewModel

                        {

                            Id = o.Id,

                            UserEmail = o.User.Email ?? "Unknown",

                            TotalAmount = o.TotalAmount,

                            OrderDate = o.OrderDate,

                            Status = o.Status

                        })

                        .ToListAsync(),

                    PendingReviews = await \_context.ProductReviews

                        .Where(r => !r.IsApproved)

                        .CountAsync()

                };

                return View(model);

            }

            catch (Exception ex)

            {

                \_logger.LogError(ex, "Error loading admin dashboard");

                TempData["ErrorMessage"] = "Error loading dashboard data.";

                return View(new AdminDashboardViewModel());

            }

        }

        [HttpGet]

        public async Task<IActionResult> Products()

        {

            try

            {

                var products = await \_context.Products

                    .OrderBy(p => p.Name)

                    .ToListAsync();

                return View(products);

            }

            catch (Exception ex)

            {

                \_logger.LogError(ex, "Error loading products");

                TempData["ErrorMessage"] = "Error loading products.";

                return View(new List<Product>());

            }

        }

        [HttpGet]

        public IActionResult CreateProduct()

        {

            return View();

        }

        [HttpPost]

        [ValidateAntiForgeryToken]

        public async Task<IActionResult> CreateProduct(Product model)

        {

            if (!ModelState.IsValid)

            {

                return View(model);

            }

            try

            {

                // Sanitize inputs to prevent XSS

                model.Name = \_htmlEncoder.Encode(model.Name.Trim());

                model.Description = \_htmlEncoder.Encode(model.Description.Trim());

                model.Category = \_htmlEncoder.Encode(model.Category.Trim());

                // Additional validation

                if (await \_context.Products.AnyAsync(p => p.Name == model.Name))

                {

                    ModelState.AddModelError("Name", "A product with this name already exists.");

                    return View(model);

                }

                \_context.Products.Add(model);

                await \_context.SaveChangesAsync();

                \_logger.LogInformation("Product {ProductName} created by admin", model.Name);

                TempData["SuccessMessage"] = "Product created successfully!";

                return RedirectToAction(nameof(Products));

            }

            catch (Exception ex)

            {

                \_logger.LogError(ex, "Error creating product {ProductName}", model.Name);

                ModelState.AddModelError(string.Empty, "Error creating product. Please try again.");

                return View(model);

            }

        }

        [HttpGet]

        public async Task<IActionResult> Users()

        {

            try

            {

                var users = await \_context.Users

                    .Select(u => new UserSummaryViewModel

                    {

                        Id = u.Id,

                        FirstName = u.FirstName,

                        LastName = u.LastName,

                        Email = u.Email ?? string.Empty,

                        RegistrationDate = u.RegistrationDate,

                        IsActive = u.IsActive,

                        OrderCount = u.Orders.Count()

                    })

                    .OrderBy(u => u.LastName)

                    .ToListAsync();

                return View(users);

            }

            catch (Exception ex)

            {

                \_logger.LogError(ex, "Error loading users");

                TempData["ErrorMessage"] = "Error loading users.";

                return View(new List<UserSummaryViewModel>());

            }

        }

        [HttpGet]

        public async Task<IActionResult> Reviews()

        {

            try

            {

                var reviews = await \_context.ProductReviews

                    .Include(r => r.Product)

                    .Include(r => r.User)

                    .OrderByDescending(r => r.ReviewDate)

                    .Select(r => new ReviewSummaryViewModel

                    {

                        Id = r.Id,

                        ProductName = r.Product.Name,

                        UserName = $"{r.User.FirstName} {r.User.LastName}",

                        Rating = r.Rating,

                        Comment = r.Comment,

                        ReviewDate = r.ReviewDate,

                        IsApproved = r.IsApproved

                    })

                    .ToListAsync();

                return View(reviews);

            }

            catch (Exception ex)

            {

                \_logger.LogError(ex, "Error loading reviews");

                TempData["ErrorMessage"] = "Error loading reviews.";

                return View(new List<ReviewSummaryViewModel>());

            }

        }

        [HttpPost]

        [ValidateAntiForgeryToken]

        public async Task<IActionResult> ApproveReview(int id)

        {

            try

            {

                var review = await \_context.ProductReviews.FindAsync(id);

                if (review != null)

                {

                    review.IsApproved = true;

                    await \_context.SaveChangesAsync();

                    \_logger.LogInformation("Review {ReviewId} approved", id);

                }

                return RedirectToAction(nameof(Reviews));

            }

            catch (Exception ex)

            {

                \_logger.LogError(ex, "Error approving review {ReviewId}", id);

                TempData["ErrorMessage"] = "Error approving review.";

                return RedirectToAction(nameof(Reviews));

            }

        }

    }

    // Admin ViewModels

    public class AdminDashboardViewModel

    {

        public int TotalUsers { get; set; }

        public int TotalProducts { get; set; }

        public int TotalOrders { get; set; }

        public decimal TotalRevenue { get; set; }

        public int PendingReviews { get; set; }

        public List<OrderSummaryViewModel> RecentOrders { get; set; } = new();

    }

    public class OrderSummaryViewModel

    {

        public int Id { get; set; }

        public string UserEmail { get; set; } = string.Empty;

        public decimal TotalAmount { get; set; }

        public DateTime OrderDate { get; set; }

        public string Status { get; set; } = string.Empty;

    }

    public class UserSummaryViewModel

    {

        public string Id { get; set; } = string.Empty;

        public string FirstName { get; set; } = string.Empty;

        public string LastName { get; set; } = string.Empty;

        public string Email { get; set; } = string.Empty;

        public DateTime RegistrationDate { get; set; }

        public bool IsActive { get; set; }

        public int OrderCount { get; set; }

    }

    public class ReviewSummaryViewModel

    {

        public int Id { get; set; }

        public string ProductName { get; set; } = string.Empty;

        public string UserName { get; set; } = string.Empty;

        public int Rating { get; set; }

        public string Comment { get; set; } = string.Empty;

        public DateTime ReviewDate { get; set; }

        public bool IsApproved { get; set; }

    }

}

3.Product:

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Authorization;

using Microsoft.EntityFrameworkCore;

using SecureShoppingApp.Data;

using SecureShoppingApp.Models;

using System.Security.Claims;

using System.Text.Encodings.Web;

namespace SecureShoppingApp.Controllers

{

    public class ProductsController : Controller

    {

        private readonly ApplicationDbContext \_context;

        private readonly ILogger<ProductsController> \_logger;

        private readonly HtmlEncoder \_htmlEncoder;

        public ProductsController(ApplicationDbContext context, ILogger<ProductsController> logger, HtmlEncoder htmlEncoder)

        {

            \_context = context;

            \_logger = logger;

            \_htmlEncoder = htmlEncoder;

        }

        [HttpGet]

        public async Task<IActionResult> Index(string? category = null, string? search = null)

        {

            try

            {

                var query = \_context.Products.Where(p => p.IsActive);

                // Secure filtering with parameterized queries

                if (!string.IsNullOrEmpty(category))

                {

                    category = \_htmlEncoder.Encode(category.Trim());

                    query = query.Where(p => p.Category == category);

                }

                if (!string.IsNullOrEmpty(search))

                {

                    search = \_htmlEncoder.Encode(search.Trim());

                    query = query.Where(p => p.Name.Contains(search) || p.Description.Contains(search));

                }

                var products = await query

                    .OrderBy(p => p.Name)

                    .ToListAsync();

                ViewBag.Categories = await \_context.Products

                    .Where(p => p.IsActive)

                    .Select(p => p.Category)

                    .Distinct()

                    .OrderBy(c => c)

                    .ToListAsync();

                ViewBag.CurrentCategory = category;

                ViewBag.CurrentSearch = search;

                return View(products);

            }

            catch (Exception ex)

            {

                \_logger.LogError(ex, "Error loading products");

                TempData["ErrorMessage"] = "Error loading products.";

                return View(new List<Product>());

            }

        }

        [HttpGet]

        public async Task<IActionResult> Details(int? id)

        {

            if (id == null)

            {

                return NotFound();

            }

            try

            {

                var product = await \_context.Products

                    .Include(p => p.Reviews.Where(r => r.IsApproved))

                    .ThenInclude(r => r.User)

                    .FirstOrDefaultAsync(p => p.Id == id && p.IsActive);

                if (product == null)

                {

                    return NotFound();

                }

                var viewModel = new ProductDetailsViewModel

                {

                    Product = product,

                    AverageRating = product.Reviews.Any() ?

                        Math.Round(product.Reviews.Average(r => r.Rating), 1) : 0,

                    ReviewCount = product.Reviews.Count(),

                    UserReview = new ProductReview { ProductId = product.Id }

                };

                return View(viewModel);

            }

            catch (Exception ex)

            {

                \_logger.LogError(ex, "Error loading product details for ID {ProductId}", id);

                return NotFound();

            }

        }

        [HttpPost]

        [Authorize]

        [ValidateAntiForgeryToken]

        public async Task<IActionResult> AddReview(ProductReview model)

        {

            if (!ModelState.IsValid)

            {

                return RedirectToAction(nameof(Details), new { id = model.ProductId });

            }

            try

            {

                var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

                if (string.IsNullOrEmpty(userId))

                {

                    return Unauthorized();

                }

                // Check if user already reviewed this product

                var existingReview = await \_context.ProductReviews

                    .FirstOrDefaultAsync(r => r.ProductId == model.ProductId && r.UserId == userId);

                if (existingReview != null)

                {

                    TempData["ErrorMessage"] = "You have already reviewed this product.";

                    return RedirectToAction(nameof(Details), new { id = model.ProductId });

                }

                // Sanitize input to prevent XSS

                model.Comment = \_htmlEncoder.Encode(model.Comment.Trim());

                model.UserId = userId;

                model.ReviewDate = DateTime.UtcNow;

                model.IsApproved = false; // Admin approval required

                \_context.ProductReviews.Add(model);

                await \_context.SaveChangesAsync();

                \_logger.LogInformation("Review added for product {ProductId} by user {UserId}", model.ProductId, userId);

                TempData["SuccessMessage"] = "Review submitted successfully! It will be visible after admin approval.";

                return RedirectToAction(nameof(Details), new { id = model.ProductId });

            }

            catch (Exception ex)

            {

                \_logger.LogError(ex, "Error adding review for product {ProductId}", model.ProductId);

                TempData["ErrorMessage"] = "Error submitting review. Please try again.";

                return RedirectToAction(nameof(Details), new { id = model.ProductId });

            }

        }

        [HttpPost]

        [Authorize]

        [ValidateAntiForgeryToken]

        public async Task<IActionResult> AddToCart(int productId, int quantity = 1)

        {

            try

            {

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

                if (product == null || !product.IsActive)

                {

                    return NotFound();

                }

                if (quantity <= 0 || quantity > product.StockQuantity)

                {

                    TempData["ErrorMessage"] = "Invalid quantity selected.";

                    return RedirectToAction(nameof(Details), new { id = productId });

                }

                var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

                if (string.IsNullOrEmpty(userId))

                {

                    return Unauthorized();

                }

                // For simplicity, we'll use session to store cart items

                // In production, consider using database or Redis

                var cartKey = $"cart\_{userId}";

                var cart = HttpContext.Session.GetString(cartKey);

                var cartItems = string.IsNullOrEmpty(cart) ?

                    new List<CartItem>() :

                    System.Text.Json.JsonSerializer.Deserialize<List<CartItem>>(cart) ?? new List<CartItem>();

                var existingItem = cartItems.FirstOrDefault(c => c.ProductId == productId);

                if (existingItem != null)

                {

                    existingItem.Quantity += quantity;

                }

                else

                {

                    cartItems.Add(new CartItem

                    {

                        ProductId = productId,

                        ProductName = product.Name,

                        Price = product.Price,

                        Quantity = quantity

                    });

                }

                HttpContext.Session.SetString(cartKey, System.Text.Json.JsonSerializer.Serialize(cartItems));

                \_logger.LogInformation("Product {ProductId} added to cart for user {UserId}", productId, userId);

                TempData["SuccessMessage"] = "Product added to cart successfully!";

                return RedirectToAction(nameof(Details), new { id = productId });

            }

            catch (Exception ex)

            {

                \_logger.LogError(ex, "Error adding product {ProductId} to cart", productId);

                TempData["ErrorMessage"] = "Error adding product to cart. Please try again.";

                return RedirectToAction(nameof(Details), new { id = productId });

            }

        }

    }

    // ViewModels

    public class ProductDetailsViewModel

    {

        public Product Product { get; set; } = null!;

        public double AverageRating { get; set; }

        public int ReviewCount { get; set; }

        public ProductReview UserReview { get; set; } = null!;

    }

    public class CartItem

    {

        public int ProductId { get; set; }

        public string ProductName { get; set; } = string.Empty;

        public decimal Price { get; set; }

        public int Quantity { get; set; }

        public decimal Total => Price \* Quantity;

    }

}

**Model’s:**

1.Application User:

using System.ComponentModel.DataAnnotations;

using Microsoft.AspNetCore.Identity;

namespace SecureShoppingApp.Models

{

    public class ApplicationUser : IdentityUser

    {

        [Required]

        [StringLength(50, MinimumLength = 2)]

        [RegularExpression(@"^[a-zA-Z\s]+$", ErrorMessage = "First name can only contain letters and spaces")]

        public string FirstName { get; set; } = string.Empty;

        [Required]

        [StringLength(50, MinimumLength = 2)]

        [RegularExpression(@"^[a-zA-Z\s]+$", ErrorMessage = "Last name can only contain letters and spaces")]

        public string LastName { get; set; } = string.Empty;

        [StringLength(200)]

        public string Address { get; set; } = string.Empty;

        [DataType(DataType.DateTime)]

        public DateTime RegistrationDate { get; set; } = DateTime.UtcNow;

        public bool IsActive { get; set; } = true;

        // Navigation properties

        public virtual ICollection<Order> Orders { get; set; } = new List<Order>();

        public virtual ICollection<ProductReview> Reviews { get; set; } = new List<ProductReview>();

    }

    public enum UserRole

    {

        Admin,

        Customer

    }

}

2.LoginModel:

using System.ComponentModel.DataAnnotations;

namespace SecureShoppingApp.Models

{

    public class LoginViewModel

    {

        [Required(ErrorMessage = "Email is required")]

        [EmailAddress(ErrorMessage = "Invalid email format")]

        public string Email { get; set; } = string.Empty;

        [Required(ErrorMessage = "Password is required")]

        [DataType(DataType.Password)]

        public string Password { get; set; } = string.Empty;

        [Display(Name = "Remember me")]

        public bool RememberMe { get; set; }

        public string? ReturnUrl { get; set; }

    }

}

3.Order:

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace SecureShoppingApp.Models

{

    public class Order

    {

        [Key]

        public int Id { get; set; }

        [Required]

        public string UserId { get; set; } = string.Empty;

        [Required]

        [Column(TypeName = "decimal(18,2)")]

        public decimal TotalAmount { get; set; }

        [DataType(DataType.DateTime)]

        public DateTime OrderDate { get; set; } = DateTime.UtcNow;

        [Required]

        [StringLength(20)]

        public string Status { get; set; } = "Pending";

        [StringLength(200)]

        public string ShippingAddress { get; set; } = string.Empty;

        // Navigation properties

        [ForeignKey("UserId")]

        public virtual ApplicationUser User { get; set; } = null!;

        public virtual ICollection<OrderItem> OrderItems { get; set; } = new List<OrderItem>();

    }

    public class OrderItem

    {

        [Key]

        public int Id { get; set; }

        [Required]

        public int OrderId { get; set; }

        [Required]

        public int ProductId { get; set; }

        [Required]

        [Range(1, int.MaxValue)]

        public int Quantity { get; set; }

        [Required]

        [Column(TypeName = "decimal(18,2)")]

        public decimal UnitPrice { get; set; }

        // Navigation properties

        [ForeignKey("OrderId")]

        public virtual Order Order { get; set; } = null!;

        [ForeignKey("ProductId")]

        public virtual Product Product { get; set; } = null!;

    }

}

4.Product:

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace SecureShoppingApp.Models

{

    public class Product

    {

        [Key]

        public int Id { get; set; }

        [Required]

        [StringLength(100, MinimumLength = 2)]

        [RegularExpression(@"^[a-zA-Z0-9\s\-\.]+$", ErrorMessage = "Product name contains invalid characters")]

        public string Name { get; set; } = string.Empty;

        [Required]

        [StringLength(1000)]

        [DataType(DataType.MultilineText)]

        public string Description { get; set; } = string.Empty;

        [Required]

        [Range(0.01, 999999.99)]

        [Column(TypeName = "decimal(18,2)")]

        public decimal Price { get; set; }

        [Range(0, int.MaxValue)]

        public int StockQuantity { get; set; }

        [StringLength(50)]

        public string Category { get; set; } = string.Empty;

        [StringLength(500)]

        public string ImageUrl { get; set; } = string.Empty;

        [DataType(DataType.DateTime)]

        public DateTime CreatedDate { get; set; } = DateTime.UtcNow;

        public bool IsActive { get; set; } = true;

        // Navigation properties

        public virtual ICollection<OrderItem> OrderItems { get; set; } = new List<OrderItem>();

        public virtual ICollection<ProductReview> Reviews { get; set; } = new List<ProductReview>();

    }

}

5.ProductReview:

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace SecureShoppingApp.Models

{

    public class ProductReview

    {

        [Key]

        public int Id { get; set; }

        [Required]

        public int ProductId { get; set; }

        [Required]

        public string UserId { get; set; } = string.Empty;

        [Required]

        [Range(1, 5)]

        public int Rating { get; set; }

        [StringLength(1000)]

        [DataType(DataType.MultilineText)]

        public string Comment { get; set; } = string.Empty;

        [DataType(DataType.DateTime)]

        public DateTime ReviewDate { get; set; } = DateTime.UtcNow;

        public bool IsApproved { get; set; } = false;

        // Navigation properties

        [ForeignKey("ProductId")]

        public virtual Product Product { get; set; } = null!;

        [ForeignKey("UserId")]

        public virtual ApplicationUser User { get; set; } = null!;

    }

}

6.RegisterViewModel:

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace SecureShoppingApp.Models

{

    public class ProductReview

    {

        [Key]

        public int Id { get; set; }

        [Required]

        public int ProductId { get; set; }

        [Required]

        public string UserId { get; set; } = string.Empty;

        [Required]

        [Range(1, 5)]

        public int Rating { get; set; }

        [StringLength(1000)]

        [DataType(DataType.MultilineText)]

        public string Comment { get; set; } = string.Empty;

        [DataType(DataType.DateTime)]

        public DateTime ReviewDate { get; set; } = DateTime.UtcNow;

        public bool IsApproved { get; set; } = false;

        // Navigation properties

        [ForeignKey("ProductId")]

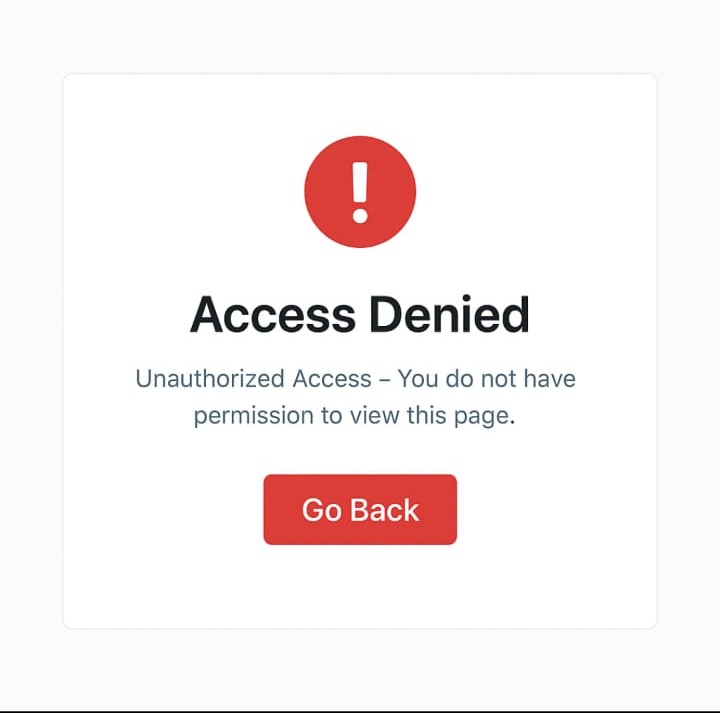
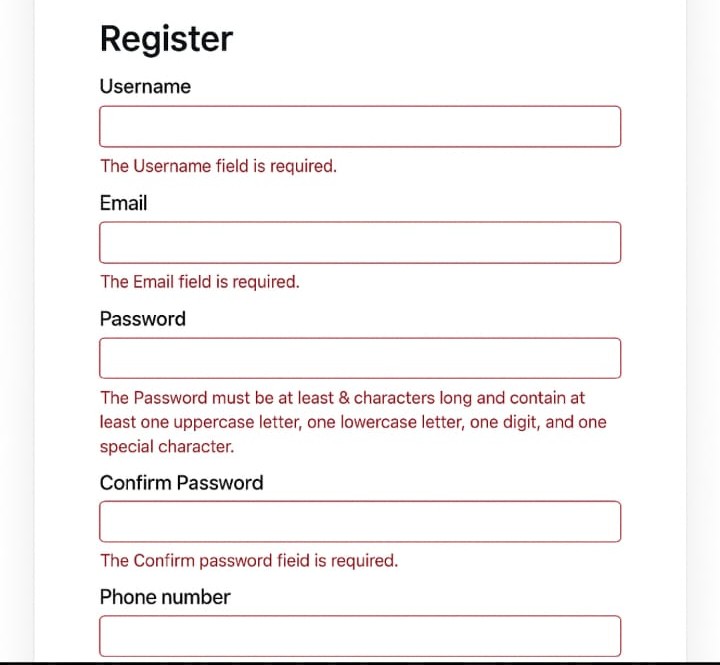
        public virtual Product Product { get; set; } = null!;

        [ForeignKey("UserId")]

        public virtual ApplicationUser User { get; set; } = null!;

    }

}

**Snapshots:**