Milestone 3 Report

Role-Based Product Management System in ASP.NET Core MVC

Submitted by: Ankit Dwivedi

# Problem Statement

The goal is to develop a Role-Based Product Management System using ASP.NET Core MVC. The system ensures secure authentication, authorization, and product management with role restrictions:  
- Admin: Full control (Create, Edit, Delete).  
- Manager: Limited access (View, Edit).  
- Both roles should have access to Product Listing.

# Requirements

1. User Authentication & Security  
- Secure registration and login using ASP.NET Identity.  
- Enforce password policies (min 8 chars, uppercase, special char).  
- Use SignInManager for login and UserManager for role assignment.  
- Ensure HTTPS and use Data Protection API for sensitive data.  
  
2. Role-Based Authorization  
- Admin → Create, Edit, Delete.  
- Manager → Edit, View only.  
- Implement using [Authorize(Roles="...")].  
  
3. Secure MVC Application  
- Use [Authorize] attributes properly.  
- Add CSRF protection (@Html.AntiForgeryToken()).  
- Show “Access Denied” for unauthorized users.  
  
4. Product Management Features  
- CRUD operations with role restrictions.  
- Proper redirects and success/error messages.  
- Store product prices securely using Data Protection.

# Implementation Steps

## Step 1: Project Creation

dotnet new mvc -o RoleBasedProductManagement\_MVC  
cd RoleBasedProductManagement\_MVC

## Step 2: Package Installation

dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore  
dotnet add package Microsoft.AspNetCore.Identity.UI  
dotnet add package Microsoft.EntityFrameworkCore.SqlServer  
dotnet add package Microsoft.EntityFrameworkCore.Tools  
dotnet add package Microsoft.AspNetCore.DataProtection

## Step 3: Folder Structure

Created Controllers, Models, Data, Views/Account, Views/Product, wwwroot

## Step 4: Database Context

public class ApplicationDbContext : IdentityDbContext<ApplicationUser>  
{  
 public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) { }  
 public DbSet<Product> Products { get; set; }  
}

## Step 5: Identity Setup

builder.Services.AddIdentity<ApplicationUser, IdentityRole>(options =>  
{  
 options.Password.RequiredLength = 8;  
 options.Password.RequireUppercase = true;  
 options.Password.RequireNonAlphanumeric = true;  
})  
.AddEntityFrameworkStores<ApplicationDbContext>()  
.AddDefaultTokenProviders()  
.AddDefaultUI();

## Step 6: Role Seeding

DbInitializer ensures Admin and Manager roles exist in the DB.

## Step 7: User Registration

Register.cshtml updated with Email, Password, ConfirmPassword, and Role dropdown (Admin/Manager).

## Step 8: Product Model

public class Product {  
 public int Id { get; set; }  
 public string Name { get; set; }  
  
 [NotMapped] public decimal Price { get; set; }  
 public string EncryptedPrice { get; set; }  
}

## Step 9: ProductController

Implemented CRUD with role restrictions.  
- Admin → Create, Edit, Delete.  
- Manager → Edit, View.  
Used Data Protection API to encrypt product prices.

## Step 10: Views

ProductList.cshtml, CreateProduct.cshtml, EditProduct.cshtml, and AccessDenied.cshtml created with Bootstrap.

## Step 11: UI Enhancements

Bootstrap navbar added in \_Layout.cshtml with role-aware links and success/error messages using TempData.

## Step 12: Database Migration

dotnet ef migrations add InitialCreate  
dotnet ef database update

# Testing & Outputs

Test Case 1: Admin Registration  
- Email: admin@test.com | Password: Admin@123 | Role: Admin  
- Can create, edit, delete products.  
  
Test Case 2: Manager Registration  
- Email: manager@test.com | Password: Manager@123 | Role: Manager  
- Can edit/view products but cannot delete.  
  
Test Case 3: Product Creation (Admin)  
- Creates Laptop (1200). Success message shown.  
  
Test Case 4: Product Edit (Manager)  
- Updates Laptop price to 1100. Success message shown.  
  
Test Case 5: Unauthorized Delete (Manager)  
- Manager tries to delete → Access Denied page shown.  
  
Test Case 6: Database Security  
- DB shows encrypted price, UI decrypts before displaying.

# Conclusion

This project successfully implements a secure, role-based product management system with:  
- Authentication & Authorization using ASP.NET Identity.  
- Admin/Manager roles enforced with [Authorize].  
- Data Protection API for secure price storage.  
- Password policies for strong authentication.  
- Bootstrap UI with feedback messages.