Program.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using SecureTaskManager.Data;
using SecureTaskManager.Models;
using SecureTaskManager.Services;

var builder = WebApplication.CreateBuilder(args);


builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddIdentity<ApplicationUser, IdentityRole>(options =>
{
    options.Password.RequiredLength = 8;
    options.Password.RequireUppercase = true;
    options.Password.RequireDigit = true;
    options.Password.RequireNonAlphanumeric = true;
    options.User.RequireUniqueEmail = true;
    options.Lockout.MaxFailedAccessAttempts = 5;
    options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(10);
})
.AddEntityFrameworkStores<AppDbContext>()
.AddDefaultTokenProviders();


builder.Services.ConfigureApplicationCookie(options =>
{
    options.Cookie.HttpOnly = true;
    options.Cookie.SecurePolicy = CookieSecurePolicy.Always;
    options.Cookie.SameSite = SameSiteMode.Lax;
    options.ExpireTimeSpan = TimeSpan.FromMinutes(15); // overall cookie lifetime
    options.SlidingExpiration = true; // renew on activity
    options.LoginPath = "/Account/Login";
    options.AccessDeniedPath = "/Account/AccessDenied";
});


builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("RequireAdminRole", policy => policy.RequireRole("Admin"));
    options.AddPolicy("CanEditTask", policy => policy.RequireClaim("CanEditTask",
"true"));
});

builder.Services.AddControllersWithViews();


builder.Services.AddScoped<IUserClaimService, UserClaimService>();

var app = builder.Build();
```

```
using (var scope = app.Services.CreateScope())
{
    var services = scope.ServiceProvider;
    var db = services.GetRequiredService<AppDbContext>();
    await db.Database.MigrateAsync();
    await SeedData.InitializeAsync(services);
}

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
    name: "areas",
    pattern: "{area:exists}/{controller=ManageTasks}/{action=Index}/{id?}");

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Tasks}/{action=TaskList}/{id?}");

app.Run();
```

Models/ApplicationUser.cs

```
using Microsoft.AspNetCore.Identity;
namespace SecureTaskManager.Models
{
    public class ApplicationUser : IdentityUser
    {
        public string? FullName { get; set; }
    }
}
```

Models/TaskItem.cs

```
using System.ComponentModel.DataAnnotations;

namespace SecureTaskManager.Models
{
    public class TaskItem
    {
        public int Id { get; set; }

        [Required, StringLength(200)]
        public string Title { get; set; } = string.Empty;

        [StringLength(2000)]
        public string Description { get; set; } = string.Empty;

        [Required]
        public string OwnerId { get; set; } = string.Empty;

        public bool IsCompleted { get; set; } = false;
    }
}
```

Data/AppDbContext.cs

```
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using SecureTaskManager.Models;

namespace SecureTaskManager.Data
{
    public class AppDbContext : IdentityDbContext<ApplicationUser>
    {
        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

        public DbSet<TaskItem> TaskItems => Set<TaskItem>();

        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);

        }
    }
}
```

Data/SeedData.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.DependencyInjection;
using SecureTaskManager.Models;

namespace SecureTaskManager.Data
{
    public static class SeedData
    {
        public static async Task InitializeAsync(IServiceProvider services)
        {
            var roleManager = services.GetRequiredService<RoleManager<IdentityRole>>();
            var userManager =
services.GetRequiredService<UserManager<ApplicationUser>>();


            var roles = new[] { "Admin", "User" };
            foreach (var r in roles)
                if (!await roleManager.RoleExistsAsync(r))
                    await roleManager.CreateAsync(new IdentityRole(r));

            var adminEmail = "admin@tasks.test";
            var admin = await userManager.FindByEmailAsync(adminEmail);
            if (admin == null)
            {
                admin = new ApplicationUser { UserName = adminEmail, Email =
adminEmail, EmailConfirmed = true, FullName = "Admin" };
                var res = await userManager.CreateAsync(admin, "Admin@123!");
                if (res.Succeeded)
                {
                    await userManager.AddToRoleAsync(admin, "Admin");

                    await userManager.AddClaimAsync(admin, new
System.Security.Claims.Claim("CanEditTask", "true"));
                }
            }


            var userEmail = "user@tasks.test";
            var user = await userManager.FindByEmailAsync(userEmail);
            if (user == null)
            {
                user = new ApplicationUser { UserName = userEmail, Email = userEmail,
EmailConfirmed = true, FullName = "Regular User" };
                var res = await userManager.CreateAsync(user, "User@123!");
                if (res.Succeeded)
                {
                    await userManager.AddToRoleAsync(user, "User");

                    await userManager.AddClaimAsync(user, new
System.Security.Claims.Claim("CanEditTask", "true"));
                }
            }
        }
    }
}
```

Services/UserClaimService.cs

```
using System.Security.Claims;
using Microsoft.AspNetCore.Identity;
using SecureTaskManager.Models;

namespace SecureTaskManager.Services
{
    public interface IUserClaimService
    {
        Task<bool> UserHasClaimAsync(string userId, string claimType);
    }

    public class UserClaimService : IUserClaimService
    {
        private readonly UserManager<ApplicationUser> _userManager;
        public UserClaimService(UserManager<ApplicationUser> userManager) =>
_userManager = userManager;

        public async Task<bool> UserHasClaimAsync(string userId, string claimType)
        {
            var user = await _userManager.FindByIdAsync(userId);
            if (user == null) return false;
            var claims = await _userManager.GetClaimsAsync(user);
            return claims.Any(c => c.Type == claimType && c.Value == "true");
        }
    }
}
```

Controllers/AccountController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Identity;
using SecureTaskManager.Models;
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.Authorization;

namespace SecureTaskManager.Controllers
{
    public class RegisterVm
    {
        [Required, EmailAddress] public string Email { get; set; } = string.Empty;
        [Required, StringLength(50)] public string FullName { get; set; } =
string.Empty;
        [Required, DataType(DataType.Password), StringLength(100, MinimumLength = 8)]
        [RegularExpression(@"^(?=.*[A-Z])(?=.*\d)(?=.*[^A-Za-z0-9]).+$")]
        public string Password { get; set; } = string.Empty;
        [Required, DataType(DataType.Password), Compare(nameof(Password))] public
string ConfirmPassword { get; set; } = string.Empty;
    }

    public class LoginVm
    {
        [Required, EmailAddress] public string Email { get; set; } = string.Empty;
        [Required, DataType(DataType.Password)] public string Password { get; set; } =
string.Empty;
        public bool RememberMe { get; set; }
    }

    public class AccountController : Controller
    {
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly SignInManager<ApplicationUser> _signInManager;

        public AccountController(UserManager<ApplicationUser> userManager,
SignInManager<ApplicationUser> signInManager)
        {
            _userManager = userManager;
            _signInManager = signInManager;
        }

        [HttpGet] public IActionResult Register() => View();
        [HttpPost, ValidateAntiForgeryToken]
        public async Task<IActionResult> Register(RegisterVm vm)
        {
            if (!ModelState.IsValid) return View(vm);

            var user = new ApplicationUser { UserName = vm.Email, Email = vm.Email,
FullName = vm.FullName };
            var res = await _userManager.CreateAsync(user, vm.Password);
            if (res.Succeeded)
            {
                await _userManager.AddToRoleAsync(user, "User");

                await _userManager.AddClaimAsync(user, new
System.Security.Claims.Claim("CanEditTask", "true"));
                await _signInManager.SignInAsync(user, isPersistent: false);
                return RedirectToAction("Dashboard", "Tasks");
            }
            foreach (var e in res.Errors) ModelState.AddModelError("", e.Description);
            return View(vm);
        }

        [HttpGet] public IActionResult Login(string? returnUrl = null) {
ViewData["ReturnUrl"]=returnUrl; return View(); }

        [HttpPost, ValidateAntiForgeryToken]
        public async Task<IActionResult> Login(LoginVm vm, string? returnUrl = null)
        {
            if (!ModelState.IsValid) return View(vm);
            var user = await _userManager.FindByEmailAsync(vm.Email);
            if (user == null) { ModelState.AddModelError("", "Invalid login."); return
View(vm); }
            var result = await _signInManager.PasswordSignInAsync(user, vm.Password,
vm.RememberMe, lockoutOnFailure: true);
            if (result.Succeeded)
            {
                if (!string.IsNullOrEmpty(returnUrl) && Url.IsLocalUrl(returnUrl))
return Redirect(returnUrl);
                return RedirectToAction("Dashboard", "Tasks");
            }
            if (result.IsLockedOut) ModelState.AddModelError("", "Account locked. Try
later.");
            else ModelState.AddModelError("", "Invalid login attempt.");
            return View(vm);
        }

        [Authorize, HttpPost, ValidateAntiForgeryToken]
        public async Task<IActionResult> Logout()
        {
            await _signInManager.SignOutAsync(); // clears cookie
            return RedirectToAction("Login", "Account");
        }

        public IActionResult AccessDenied() => View();
    }
}
```

Controllers/TasksController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using SecureTaskManager.Data;
using SecureTaskManager.Models;

namespace SecureTaskManager.Controllers
{
    [Authorize]
    public class TasksController : Controller
    {
        private readonly AppDbContext _db;
        private readonly UserManager<ApplicationUser> _userManager;

        public TasksController(AppDbContext db, UserManager<ApplicationUser>
userManager)
        {
            _db = db; _userManager = userManager;
        }

        public async Task<IActionResult> Dashboard()
        {
            var user = await _userManager.GetUserAsync(User);
            var tasks = await _db.TaskItems.AsNoTracking().Where(t => t.OwnerId ==
user.Id).ToListAsync();
            return View("TaskList", tasks);
        }

        [HttpGet]
        public IActionResult Create() => View();

        [HttpPost, ValidateAntiForgeryToken]
        public async Task<IActionResult> Create(TaskItem model)
        {
            if (!ModelState.IsValid) return View(model);
            var user = await _userManager.GetUserAsync(User);
            model.OwnerId = user.Id;
            _db.TaskItems.Add(model);
            await _db.SaveChangesAsync();
            return RedirectToAction("Dashboard");
        }

        [HttpGet]
        public async Task<IActionResult> Edit(int id)
        {
            var task = await _db.TaskItems.FindAsync(id);
            if (task == null) return NotFound();

            var user = await _userManager.GetUserAsync(User);
            if (task.OwnerId != user.Id && !User.IsInRole("Admin") &&
!User.HasClaim("CanEditTask","true"))
                return Forbid();
            return View(task);
        }

        [HttpPost, ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(TaskItem model)
        {
            if (!ModelState.IsValid) return View(model);
            var task = await _db.TaskItems.FindAsync(model.Id);
            if (task == null) return NotFound();
            var user = await _userManager.GetUserAsync(User);
            if (task.OwnerId != user.Id && !User.IsInRole("Admin") &&
!User.HasClaim("CanEditTask","true"))
                return Forbid();
            task.Title = model.Title.Trim();
            task.Description = model.Description.Trim();
            task.IsCompleted = model.IsCompleted;
            await _db.SaveChangesAsync();
            return RedirectToAction("Dashboard");
        }
    }
}
```

Areas/Admin/Controllers/ManageTasksController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using SecureTaskManager.Data;
using Microsoft.EntityFrameworkCore;

namespace SecureTaskManager.Areas.Admin.Controllers
{
    [Area("Admin")]
    [Authorize(Policy = "RequireAdminRole")]
    public class ManageTasksController : Controller
    {
        private readonly AppDbContext _db;
        public ManageTasksController(AppDbContext db) => _db = db;

        public async Task<IActionResult> Index()
        {
            var all = await _db.TaskItems.AsNoTracking().ToListAsync();
            return View(all);
        }
    }
}
```

Views/Shared/_Layout.cshtml

```
<!DOCTYPE html>
<html><head><meta charset="utf-8" /><meta name="viewport" content="width=device-width"
/></head>
<body>
    <nav>
        @if (User.Identity?.IsAuthenticated ?? false)
        {
            <span>@User.Identity.Name</span>
            <form asp-controller="Account" asp-action="Logout" method="post"
style="display:inline;">
                @Html.AntiForgeryToken()
                <button type="submit">Logout</button>
            </form>
        }
        else
        {
            <a asp-controller="Account" asp-action="Login">Login</a>
            <a asp-controller="Account" asp-action="Register">Register</a>
        }
    </nav>
    <div>@RenderBody()</div>
</body></html>
```

Views/Tasks/TaskList.cshtml

```
@model IEnumerable<SecureTaskManager.Models.TaskItem>
@{
    ViewData["Title"]="My Tasks";
}
<h2>@ViewData["Title"]</h2>
<a asp-action="Create">Create Task</a>
<table>
@foreach(var t in Model)
{
    <tr>
        <td>@Html.DisplayFor(m=>t.Title)</td>
        <td>@Html.DisplayFor(m=>t.Description)</td> <!-- encoded -->
        <td>@(t.IsCompleted ? "Done" : "Pending")</td>
        <td><a asp-action="Edit" asp-route-id="@t.Id">Edit</a></td>
    </tr>
}
</table>
```

Views/Tasks/Edit.cshtml

```
@model SecureTaskManager.Models.TaskItem
<form asp-action="Edit" method="post">
    @Html.AntiForgeryToken()
    <input asp-for="Id" type="hidden" />
    <div><label asp-for="Title"></label><input asp-for="Title" /></div>
    <div><label asp-for="Description"></label><textarea asp-
for="Description"></textarea></div>
    <div><label asp-for="IsCompleted"></label><input asp-for="IsCompleted"
type="checkbox" /></div>
    <button type="submit">Save</button>
</form>
```

```
builder.Services.AddDbContext<AppDbContext>(options =>
options.UseInMemoryDatabase("SecureTaskManagerInMem"));
```