Contents

[Adding Database 2](#_Toc130560950)

[Adding Seed Data to Database 4](#_Toc130560951)

[Creating Admin Area 5](#_Toc130560952)

[Creating Controller in Admin Areas 5](#_Toc130560953)

[Creating a select dropdown list 6](#_Toc130560954)

[Login Authentication/Authorization 6](#_Toc130560955)

[Identity 8](#_Toc130560956)

[Register New User 9](#_Toc130560957)

[Identity 2 9](#_Toc130560958)

[Adding Password validation in UserIdentity class 11](#_Toc130560959)

[Adding Login class 11](#_Toc130560960)

[Logout 14](#_Toc130560961)

[Edit Login 15](#_Toc130560962)

[Admin Users List 17](#_Toc130560963)

[Custom TagHelper to display Users in the Roles Index View 24](#_Toc130560964)

[Data Annotations 24](#_Toc130560965)

[Creating Custom notifications 25](#_Toc130560966)

[Creating Foreign Key Implementation 26](#_Toc130560967)

[On delete cascade 28](#_Toc130560968)

[Routing 29](#_Toc130560969)

[Layout Pages 30](#_Toc130560970)

[View Component 30](#_Toc130560971)

[Disabling an input tag 32](#_Toc130560972)

[Active Directory 33](#_Toc130560973)

[Adding Extra items into Active Directory Viewbag 34](#_Toc130560974)

[Adding Image 35](#_Toc130560975)

[Create a custom validation for files 36](#_Toc130560976)

[Posting to database 37](#_Toc130560977)

[Validating a foreign key select list 37](#_Toc130560978)

[Devices by Category 38](#_Toc130560979)

[Inline edit 40](#_Toc130560980)

[Select list 40](#_Toc130560981)

[Modal Pop Up 41](#_Toc130560982)

[Creating 41](#_Toc130560983)

[Cascading Dropdown 42](#_Toc130560984)

[Multi-level dropdown menus 44](#_Toc130560985)

[Multiplication of two textboxes on change 46](#_Toc130560986)

[Troubleshooting 48](#_Toc130560987)

## Adding Database

1. Create project without authentication
2. Add folder called Data
3. Install efcore tools and efcore sqlserver
4. View installed packages by right clicking project->Edit Project File
5. Add the following code for packages if working offline

<ItemGroup>

    <PackageReference Include="Codehaks.Pagination" Version="1.0.0" />

    <PackageReference Include="FluentAssertions" Version="6.8.0" />

    <PackageReference Include="Galactic.ActiveDirectory" Version="1.3.0.499" />

    <PackageReference Include="Microsoft.AspNet.Identity.Core" Version="2.2.3" />

    <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="6.0.11" />

    <PackageReference Include="Microsoft.AspNetCore.Identity.UI" Version="6.0.10" />

    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="6.0.12" />

    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="6.0.12">

      <PrivateAssets>all</PrivateAssets>

      <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>

    </PackageReference>

    <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration" Version="6.0.11" />

    <PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design" Version="6.0.10" />

    <PackageReference Include="System.Configuration.ConfigurationManager" Version="7.0.0" />

    <PackageReference Include="System.DirectoryServices" Version="7.0.0" />

    <PackageReference Include="System.DirectoryServices.AccountManagement" Version="7.0.0" />

  </ItemGroup>

1. Add class with name of context: ApplicationDbContext, and inherits from IdentityDbContext
2. Add constructor

public class ApplicationDbContext : IdentityDbContext

{

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

    {

    }

}

1. Add to Program.cs:

var connectionString = builder.Configuration.GetConnectionString("TIMBContextConnection") ?? throw new InvalidOperationException("Connection string 'TIMBContextConnection' not found.");

builder.Services.AddDbContext<ZltInventoryContext>(options =>options.UseSqlServer(connectionString));

1. Add to appsettings.json:

"ConnectionStrings":{"ZltContextConnection":"Server=(localdb)\\mssqllocaldb;Database=Zlt;Trusted\_Connection=True;MultipleActiveResultSets=true"},

1. Create a model

using System.ComponentModel.DataAnnotations;

namespace ZltInventory.Models

{

    public class Page

    {

        public int Id { get; set; }

        [Required]

        public string Title { get; set; }

        [Required]

        public string Slug { get; set; }

        [Required]

        public string Content { get; set; }

        public int Sorting { get; set; }

    }

}

1. Add Dbset property to context class
2. public DbSet<Page> Pages { get; set; }

### Adding Seed Data to Database

1. Create model SeedData

using Microsoft.EntityFrameworkCore;

using ZltInventory.Infrastructure;

namespace ZltInventory.Models

{

    public class SeedData

    {

        public static void Initialize(IServiceProvider serviceProvider)

        {

            using(var context = new ZltInventoryContext

                (serviceProvider.GetRequiredService<DbContextOptions<ZltInventoryContext>>()))

            {

                if (context.Pages.Any())

                {

                    return;

                }

                context.Pages.AddRange(

                    new Page

                    {

                        Title = "Home",

                        Slug = "home",

                        Content = "home page",

                        Sorting = 0

                    },

                    new Page

                    {

                        Title = "Devices",

                        Slug = "devices",

                        Content = "devices page",

                        Sorting = 0

                    }

                    );

                context.SaveChanges();

            }

        }

    }

}

1. In program.cs add these lines under var app = builder.Build();

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

{

    var services = scope.ServiceProvider;

    try

    {

        SeedData.Initialize(services);

    }

    catch (Exception ex)

    {

        throw ex;

    }

}

## Creating Admin Area

1. Create new Folder called Areas
2. Right click on the folder and add area called admin
3. Copy the routing given in the ScaffoldingReadMe.txt file as below to

app.UseEndpoints(endpoints =>

{

    endpoints.MapControllerRoute(

    name : "areas",

    pattern : "{area:exists}/{controller=Home}/{action=Index}/{id?}"

    );

});

1. Change the route to this and paste

app.MapControllerRoute(

  name: "areas",

  pattern: "{area:exists}/{controller=Home}/{action=Index}/{id?}"

);

1. Delete Data and Models folders

## Creating Controller in Admin Areas

[Area("Admin")]

    public class PagesController : Controller

    {

        private readonly ZltInventoryContext \_context;

        public PagesController(ZltInventoryContext context)

        {

            \_context = context;

        }

        // GET: Admin/Pages

        public async Task<IActionResult> Index()

        {

            return View(await \_context.Pages.OrderBy(x => x.Sorting).ToListAsync());

        }

    }

1. Add View
2. Copy Shared folder in Views, Viewstats and ViewImports and paste it into admin Views

## Creating a select dropdown list

In the Controller

public IActionResult Create()

{

ViewBag.CustomerId = new SelectList(\_context.Customer.OrderBy(x => x.Name), "Id", "Name");

return View();

}

public async Task<IActionResult> Create([Bind("Id,BaleNumber,Mass,PeelTime,ClientName")] BalePeeler balePeeler)

{

    ViewBag.CustomerId = new SelectList(\_context.Customer.OrderBy(x => x.Name), "Id", "Name");

    if (ModelState.IsValid)

    {

        \_context.Add(balePeeler);

        await \_context.SaveChangesAsync();

        return RedirectToAction(nameof(Index));

    }

    return View(balePeeler);

}

## Login Authentication/Authorization

1. In program.cs, add

builder.Services.AddAuthentication(

    CookieAuthenticationDefaults.AuthenticationScheme)

    .AddCookie(option =>

    {

        option.LoginPath = "/Acess/Login"; //Access is the controller, and Login is the View

        option.ExpireTimeSpan = TimeSpan.FromMinutes(20);

    });

1. Use authentication on top of use authorization

app.UseAuthentication();

app.UseAuthorization();

1. Create a new route

app.MapControllerRoute(

  name: "areas",

  pattern: "{area:exists}/{controller=Access}/{action=Login}/{id?}"

);

1. Add Model

public class VMLogin

{

    public string Email { get; set; }

    public string Password { get; set; }

    public bool KeepLoggedIn { get; set; }

}

1. Add Controller

using Microsoft.AspNetCore.Authentication;

using Microsoft.AspNetCore.Authentication.Cookies;

using Microsoft.AspNetCore.Mvc;

using System.Security.Claims;

using ZltInventory.Models;

namespace ZltInventory.Controllers

{

    public class AccessController : Controller

    {

        public IActionResult Login()

        {

            //check if the user is already logged in

            ClaimsPrincipal claimUser = HttpContext.User;

            if (claimUser.Identity.IsAuthenticated)

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

            return View();

        }

        [HttpPost]

        public async Task<IActionResult> Login(VMLogin modelLogin)

        {

            if (modelLogin.Email == "user@example.com" && modelLogin.Password == "123")

            {

                List<Claim> claims = new List<Claim>()

                {

                    new Claim(ClaimTypes.NameIdentifier, modelLogin.Email),

                    new Claim("OtherProperties","Example Role")

                };

                ClaimsIdentity claimsIdentity = new ClaimsIdentity(claims,CookieAuthenticationDefaults.AuthenticationScheme);

                AuthenticationProperties properties = new AuthenticationProperties()

                {

                    AllowRefresh = true,

                    IsPersistent = modelLogin.KeepLoggedIn

                };

                await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme,

                    new ClaimsPrincipal(claimsIdentity), properties);

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

            }

            ViewData["ValidateMessage"] = "user not found";

            return View();

        }

    }

}

1. In the desired controller add this method

public async Task<IActionResult> Logout()

{

    await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);

     return RedirectToAction("Login","Access");

}

1. Add Authorize

namespace ZltInventory.Controllers

{

    [Authorize]

    public class HomeController : Controller

## Identity

1. Ensure the context class inherits from IdentityDbContext

public class ZltInventoryContext:IdentityDbContext

1. Add identity services to Program.cs

builder.Services.AddIdentity<IdentityUser, IdentityRole>()

    .AddEntityFrameworkStores<ZltInventoryContext>();

The Identity User class contains a definition for UserName, Email, Password, etc but if there is a custom property like gender that we need in our system, we create a custom class that inherits from the IdentityUser class then plug that class in instead of the built in IdentityUser class.

1. Add authentication middleware before the Mvc Middleware that contains routing

app.UseAuthentication();

app.UseAuthorization();

app.MapControllerRoute(

  name: "areas",

  pattern: "{area:exists}/{controller=Access}/{action=Login}/{id?}"

);

1. Build solution
2. Add new migration and update database. New tables should be created in the database

### Register New User

1. Create a new class called Register

public class Register

{

    [Required]

    [EmailAddress]

    public string Email { get; set; }

    [Required]

    [DataType(DataType.Password)]

    public string Password { get; set; }

    [DataType(DataType.Password)]

    [Display(Name ="Confirm Password")]

    [Compare("Password", ErrorMessage ="Password and confirmation password do not match.")]

    public string ConfirmPassword { get; set; }

}

1. Add Controller AccountController

## Identity 2

1. Create a new class, App User that inherits from IdentityUser, it can be empty or have extra field e.g Occupation

public class AppUser:IdentityUser

{

    public string Occupation { get; set; }

}

1. Modify Context Class

 public class ZltInventoryContext:IdentityDbContext<AppUser>

{

    public ZltInventoryContext(DbContextOptions<ZltInventoryContext> options): base(options)

    {

    }

    public DbSet<Page> Pages { get; set; }

}

1. Add migration and update database

NB! NUMBER 1 TO 3 IS FOR TABLE CREATION

1. Create a new class User

public class User

{

    [Required, MinLength(2, ErrorMessage ="Minimum length is 2"), Key]

    public string? UserName { get; set; }

    public string Email { get; set; }

    [DataType(DataType.Password), Required, MinLength(4, ErrorMessage = "Minimum length is 4")]

    public string Password { get; set; }

    //Empty Constructor

    public User()

    {

    }

    //Consructor assigning required values

    User(AppUser appUser)

    {

        UserName = appUser.UserName;

        Email = appUser.Email;

        Password = appUser.PasswordHash;

    }

}

1. Add these lines to the program.cs

builder.Services.AddIdentity<AppUser, IdentityRole>()

    .AddEntityFrameworkStores<ZltInventoryContext>()

    .AddDefaultTokenProviders();

app.UseAuthentication();

app.UseAuthorization();

1. Create account controller

public class AccountController : Controller

{

    //The Db Context is not our default, it's user manager. Sign in manager is for signing in

        private readonly UserManager<AppUser> userManager;

        private readonly SignInManager<AppUser> signInManager;

        public AccountController(UserManager<AppUser> userManager, SignInManager<AppUser> signInManager)

        {

            this.userManager = userManager;

            this.signInManager = signInManager;

        }

    //GET/Account/Register

    //[AllowAnonymous]

    public IActionResult Register()

    {

        return View();

    }

    //POST/Account/Register

    [HttpPost]

    [AllowAnonymous]

    [ValidateAntiForgeryToken]

    public async Task<IActionResult> Register(User user)

    {

        if (ModelState.IsValid)

        {

            AppUser appUser = new AppUser

            {

                UserName = user.UserName,

                Email = user.Email

            };

            IdentityResult result = await userManager.CreateAsync(appUser, user.Password);

            if (result.Succeeded)

            {

                return RedirectToAction("Login");

            }

            else

            {

                foreach(IdentityError error in result.Errors)

                {

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

                }

            }

        }

        return View(user);

    }

}

1. Create Register View

### Adding Password validation in UserIdentity class

1. Change the Identity service in programs.cs

builder.Services.AddIdentity<AppUser, IdentityRole>(options=> {

    options.Password.RequiredLength = 4;

    options.Password.RequireNonAlphanumeric = false;

    options.Password.RequireDigit = false;

})

    .AddEntityFrameworkStores<ZltInventoryContext>()

    .AddDefaultTokenProviders();

### Adding Login class

1. Create Login class

public class Login

{

    [Required, EmailAddress]

    public string Email { get; set; }

    [DataType(DataType.Password),Required,MinLength(4,ErrorMessage ="Minimum length is 4")]

    public string Password { get; set; }

    public string? ReturnUrl { get; set; }

}

1. Add login methods to AccountController

//GET/Account/Login

//[AllowAnonymous]

public IActionResult Login(string returnUrl)

{

    Login login = new Login

    {

        ReturnUrl = returnUrl

    };

    return View(login);

}

//POST/Account/Login

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Login(Login login)

{

    if (ModelState.IsValid)

    {

        AppUser appUser = await userManager.FindByEmailAsync(login.Email);

        {

            SignInResult result1 = await signInManager.PasswordSignInAsync(appUser,login.Password,false,false)

        if (result1.Succeeded)

            {

                return Redirect(login.ReturnUrl ?? "/");

            }

        }

        ModelState.AddModelError("", "Login Failed, wrong credentials");

    }

    return View(login);

}

1. Create View for login

@model ZltInventory.Models.Login

@{

    //Layout = null;

}

<!DOCTYPE html>

<html>

<head>

    <meta name="viewport" content="width=device-width" />

    <title>Login</title>

</head>

<body>

<h4>Login</h4>

<hr />

<div class="row">

    <div class="col-md-4">

        <form asp-action="Login">

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

            <input hidden asp-for="ReturnUrl"/>

            <div class="form-group">

                <label asp-for="Email" class="control-label"></label>

                <input asp-for="Email" class="form-control" />

                <span asp-validation-for="Email" class="text-danger"></span>

            </div>

            <div class="form-group">

                <label asp-for="Password" class="control-label"></label>

                <input asp-for="Password" class="form-control" />

                <span asp-validation-for="Password" class="text-danger"></span>

            </div>

            <div class="form-group">

                <input type="Login" value="Create" class="btn btn-primary" />

            </div>

        </form>

    </div>

</div>

<div>

    <a asp-action="Index">Back to List</a>

</div>

</body>

</html>

1. Edit Layout Page

<header>

    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">

        <div class="container-fluid">

            <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">ZltInventory</a>

            <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-controls="navbarSupportedContent"

                    aria-expanded="false" aria-label="Toggle navigation">

                <span class="navbar-toggler-icon"></span>

            </button>

            <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">

                <ul class="navbar-nav flex-grow-1">

                    <li class="nav-item">

                        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>

                    </li>

                    <li class="nav-item">

                        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>

                    </li>

                    <li class="nav-item ml-1">

                        <a class="nav-link text-dark" asp-area="" asp-controller="Account" asp-action="Register">Register</a>

                    </li>

                </ul>

            </div>

            <div class="navbar-collapse collapse d-sm-inline-flex justify-content-end">

                <ul class="navbar-nav flex-grow-1justify-content-end ">

                    <li class="nav-item ml-1">

                        <a class="nav-link text-dark" asp-area="" asp-controller="Account" asp-action="Register">Register</a>

                    </li>

                    <li class="nav-item ml-1">

                        <a class="nav-link text-dark" asp-area="" asp-controller="Account" asp-action="Login">Login</a>

                    </li>

                </ul>

            </div>

        </div>

    </nav>

</header>

### Logout

1. Add Logout method in AccountController

//GET/Account/Logout

        public async Task<IActionResult> Logout()

        {

            await signInManager.SignOutAsync();

            return Redirect("/");

        }

1. Edit div in layout page

<div class="navbar-collapse collapse d-sm-inline-flex justify-content-end">

    <ul class="navbar-nav flex-grow-1justify-content-end ">

        @if (User?.Identity?.IsAuthenticated ?? false)

        {

            <li class="nav-item ml-1">

                <a class="nav-link text-dark" asp-area="" asp-controller="Account" asp-action="Logout">Logout</a>

            </li>

        }

        else

        {

            <li class="nav-item ml-1">

                <a class="nav-link text-dark" asp-area="" asp-controller="Account" asp-action="Register">Register</a>

            </li>

            <li class="nav-item ml-1">

                <a class="nav-link text-dark" asp-area="" asp-controller="Account" asp-action="Login">Login</a>

            </li>

        }

    </ul>

</div>

### Edit Login

1. Copy User class and create a new class called UserEdit

public class UserEdit

{

    [Required, EmailAddress]

    public string Email { get; set; }

    [Required, DataType(DataType.Password), MinLength(2, ErrorMessage = "Minimum length is 2")]

    public string Password { get; set; }

    //Empty Constructor

    public UserEdit()

    {

    }

    //Consructor assigning required values

    public UserEdit(AppUser appUser)

    {

        Email = appUser.Email;

        Password = appUser.PasswordHash;

    }

}

1. Alter Constructor in Account controller and add password Hasher

private readonly UserManager<AppUser> userManager;

private readonly SignInManager<AppUser> signInManager;

private IPasswordHasher<AppUser> passwordHasher;

public AccountController(UserManager<AppUser> userManager, SignInManager<AppUser> signInManager, IPasswordHasher<AppUser> passwordHasher)

{

    this.userManager = userManager;

    this.signInManager = signInManager;

    this.passwordHasher = passwordHasher;

}

1. Add the UserEdit methods to the AccountController

 //GET/Account/Edit

public async Task<IActionResult> Edit()

{

    AppUser appUser = await userManager.FindByNameAsync(User.Identity.Name);

    User user = new User(appUser);

    return View(user);

}

//POST/Account/Edit

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(User user)

{

    AppUser appUser = await userManager.FindByNameAsync(User.Identity.Name);

    if (ModelState.IsValid)

    {

        appUser.Email = user.Email;

        if (user.Password != null)

        {

            appUser.PasswordHash = passwordHasher.HashPassword(appUser, user.Password);

        }

        IdentityResult result = await userManager.UpdateAsync(appUser);

        if (result.Succeeded)

            TempData["Success"] = "Your information has been edited";

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

    }

    return View();

}

1. Add View

@model ZltInventory.Models.User

@{

}

<!DOCTYPE html>

<html>

<head>

    <meta name="viewport" content="width=device-width" />

    <title>Edit</title>

</head>

<body>

<h4>User</h4>

<hr />

<div class="row">

    <div class="col-md-4">

        <form asp-action="Edit">

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

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

            <div class="form-group">

                <label asp-for="Email" class="control-label"></label>

                <input asp-for="Email" class="form-control" />

                <span asp-validation-for="Email" class="text-danger"></span>

            </div>

            <div class="form-group">

                <label asp-for="Password" class="control-label"></label>

                <input asp-for="Password" class="form-control" />

                <span asp-validation-for="Password" class="text-danger"></span>

            </div>

            <div class="form-group">

                <input type="submit" value="Save" class="btn btn-primary" />

            </div>

        </form>

    </div>

</div>

<div>

    <a asp-action="Index">Back to List</a>

</div>

</body>

</html>

1. Add to layout page

<li class="nav-item ml-1">

    <a class="nav-link text-dark" asp-area="" asp-controller="Account" asp-action="Edit">Hi @User.Identity.Name</a>

</li>

### Admin Users List

1.Create new controller in Admin area for Users

public class UsersController : Controller

{

    private readonly UserManager<AppUser> userManager;

    public UsersController(UserManager<AppUser> userManager)

    {

        this.userManager = userManager;

    }

    // GET: Admin/Users

    public async Task<IActionResult> Index()

    {

            return View(userManager.Users);

    }

}

1. Create Index view

@model IEnumerable<AppUser>

@{

ViewData["Title"] = "Users";

}

<h1>Users</h1>

<p>

<a asp-action="Create"> Create Users</a>

</p>

<table class="table">

<thead>

    <tr>

        <th>

            UserName

        </th>

        <th>

            Email

        </th>

        <th></th>

    </tr>

</thead>

<tbody>

    @foreach (var item in Model)

    {

        <tr>

            <td>

                @item.UserName

            </td>

            <td>

                @item.Email

            </td>

        </tr>

    }

</tbody>

</table>

1. Add link to Layout page

<li class="nav-item">

    <a class="nav-link text-dark" asp-area="admin" asp-controller="Users" asp-action="Index">Users</a>

</li>

<li class="nav-item">

    <a class="nav-link text-dark" asp-area="admin" asp-controller="Roles" asp-action="Index">Roles</a>

</li>

1. Add new Controller for Roles

[Area("Admin")]

public class RolesController : Controller

{

    private readonly RoleManager<IdentityRole> roleManager;

    private readonly UserManager<AppUser> userManager;

    public RolesController(RoleManager<IdentityRole> roleManager, UserManager<AppUser> userManager)

    {

        this.roleManager = roleManager;

        this.userManager = userManager;

    }

    //GET /Admin/Roles

    public IActionResult Index()

    {

        return View(roleManager.Roles);

    }

}

1. Add View

@model IEnumerable<IdentityRole>

@{

    ViewData["Title"] = "Users";

}

<h1>Users</h1>

<p>

<a asp-action="Create"> Create Users</a>

</p>

<table class="table">

<thead>

    <tr>

        <th>

            Role

        </th>

        <th>

            users

        </th>

        <th></th>

    </tr>

</thead>

<tbody>

        @foreach (var role in Model)

        {

            <tr>

                <td>

                    @role.Name

                </td>

                <td>

                    users for this role

                </td>

                <td>

                    <a asp-action="Edit" asp-route-id="@role.Id" class="btn btn-primary">Edit</a> |

                    <a class="confirmDeletion btn btn-danger" asp-action="Delete" asp-route-id="@role.Id">Delete</a>

                </td>

            </tr>

        }

</tbody>

</table>

1. Add these methods into RolesController

//GET /Admin/Roles/Create

public IActionResult Create()

{

    return View();

}

//POST /Admin/Roles/Create

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create([MinLength(2), Required] string name)

{

    if (ModelState.IsValid)

    {

        name = name.ToLower();

        IdentityResult result = await roleManager.CreateAsync(new IdentityRole(name));

        if (result.Succeeded)

        {

            TempData["Success"] = "A role has been created!";

            return RedirectToAction("Index");

        }

        else

        {

            foreach(IdentityError error in result.Errors)

            {

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

            }

        }

    }

    ModelState.AddModelError("", "Minimum length is 2");

    return View(name);

}

1. Add Create Page

@model string

@{

    ViewData["Title"] = "Create a role";

}

<h1>Create a role</h1>

<div class="row">

    <div class="col-md-10">

        <form asp-action="Create">

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

            <div class="form-group">

                <label class="control-label">Role name</label>

                <input type="text" name="Name" class="form-control" />

            </div>

            <br />

            <div class="form-group">

                <input type="submit" value="Create" class="btn btn-primary" />

            </div>

        </form>

    </div>

</div>

<div>

    <a asp-action="Index">Back to List</a>

</div>

1. Create a new Model RoleEdit

using Microsoft.AspNetCore.Identity;

namespace ZltInventory.Models

{

    public class RoleEdit

    {

        public IdentityRole Role { get; set; }

        public IEnumerable<AppUser> Members { get; set; }

        public IEnumerable<AppUser> NonMembers { get; set; }

        public string RoleName { get; set; }

        public string[] AddIds { get; set; }

        public string[] DeleteIds { get; set; }

    }

}

1. Edit the Controller

//GET /Admin/Roles/Edit/5

public async Task<IActionResult> Edit(string id)

{

    IdentityRole role = await roleManager.FindByIdAsync(id);

    List<AppUser> members = new List<AppUser>();

    List<AppUser> nonmembers = new List<AppUser>();

    foreach(AppUser user in userManager.Users)

    {

        var list = await userManager.IsInRoleAsync(user, role.Name) ? members : nonmembers;

        list.Add(user);

    }

    return View(new RoleEdit

    {

        Role = role,

        Members = members,

        NonMembers = nonmembers

    });

}

//POST /Admin/Roles/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(RoleEdit roleEdit)

{

    IdentityResult result;

    foreach(string userId in roleEdit.AddIds ?? new string[] { })

    {

        AppUser user= await userManager.FindByIdAsync(userId);

        result = await userManager.AddToRoleAsync(user, roleEdit.RoleName);

    }

    foreach (string userId in roleEdit.DeleteIds ?? new string[] { })

    {

        AppUser user = await userManager.FindByIdAsync(userId);

        result = await userManager.RemoveFromRoleAsync(user, roleEdit.RoleName);

    }

    return Redirect(Request.Headers["Referer"].ToString());

}

1. Create Edit View

@model RoleEdit

@{

    ViewData["Title"] = "Edit Role";

}

<h1>Edit Roles</h1>

<div class="row">

    <div class="col-md-10">

        <form asp-action="Edit">

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

            <input type="hidden" name="RoleName" value="@Model.Role.Name" />

            <div class="row">

                <div class="col-6">

                    <h2>Add users to <span class="mybadge badge-primary">@Model.Role.Name</span> role</h2>

                    <table class="table table-bordered">

                        @if (Model.NonMembers.Count() == 0)

                        {

                            <tr>

                                <td>

                                    All users are members

                                </td>

                            </tr>

                        }

                        else

                        {

                            foreach (AppUser user in Model.NonMembers)

                            {

                                <tr>

                                    <td>

                                        @user.UserName

                                    </td>

                                    <td>

                                        <input type="checkbox" value="@user.Id" name="AddIds" />

                                    </td>

                                </tr>

                            }

                        }

                    </table>

                </div>

                <div class="col-6">

                    <h2>Remove users from <span class="mybadge badge-primary">@Model.Role.Name</span> role</h2>

                    <table class="table table-bordered">

                        @if (Model.Members.Count() == 0)

                        {

                            <tr>

                                <td>

                                    No users are members

                                </td>

                            </tr>

                        }

                        else

                        {

                            foreach (AppUser user in Model.Members)

                            {

                                <tr>

                                    <td>

                                        @user.UserName

                                    </td>

                                    <td>

                                        <input type="checkbox" value="@user.Id" name="DeleteIds" />

                                    </td>

                                </tr>

                            }

                        }

                    </table>

                </div>

            </div>

            <div class="form-group">

                <input type="submit" value="Save" class="btn btn-primary" />

            </div>

        </form>

    </div>

</div>

<br />

<div>

    <a asp-action="Index" class="btn btn-dark">Back to List</a>

</div>

### Custom TagHelper to display Users in the Roles Index View

1. Edit Roles Index

<td user-role="@role.Id">

    users for this role

</td>

1. In the Instrastructure Folder, add a class called RolesTagHelper

[HtmlTargetElement("td", Attributes = "user-role")]

public class RolesTagHelper : TagHelper

{

    private readonly RoleManager<IdentityRole> roleManager;

    private readonly UserManager<AppUser> userManager;

    public RolesTagHelper(RoleManager<IdentityRole> roleManager, UserManager<AppUser> userManager)

    {

        this.roleManager = roleManager;

        this.userManager = userManager;

    }

    [HtmlAttributeName("user-role")]

    public string RoleId { get; set; }

    public override async Task ProcessAsync(TagHelperContext context, TagHelperOutput output)

    {

        List<string> names = new List<string>();

        IdentityRole role = await roleManager.FindByIdAsync(RoleId);

        if (role != null)

        {

            foreach (var user in userManager.Users)

            {

                if (user != null && await userManager.IsInRoleAsync(user, role.Name))

                {

                    names.Add(user.UserName);

                }

            }

        }

        output.Content.SetContent(names.Count == 0 ? "No users" : String.Join(", ", names));

    }

}

## Data Annotations

1. Display Name

[Display(Name = "Brand")]

public int BrandId { get; set; }

2. Minimum Length

[MinLength(2, ErrorMessage = "Minimum length is 2")]

3. Maximum Length

[MaxLength(10, ErrorMessage = " Maximum length is 10")]

4. Regular Expression/Default characters

[RegularExpression(@"^[a-zA-Z-]+$", ErrorMessage = "Only Letters are allowed")]

5. Foreign Key

[ForeignKey("BrandId")]

6. Range

[Range(1, int.MaxValue, ErrorMessage = "You must choose a category")]

public int CategoryId { get; set; }

7. Not Mapped

[NotMapped]

public IFormFile? ImageUpload { get; set; }

1. File Extension

[FileExtension]

public IFormFile? ImageUpload { get; set; }

1. Compare

[Compare("Password", ErrorMessage ="Password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }

1. EmailAddress

[EmailAddress]

public string Email { get; set; }

1. AllowAnonymous-allows a method to be accessed by any one authorised or not in an authorised class

[Authorize]

    public class AccountController : Controller

    {

        [AllowAnonymous]

        public IActionResult Register()

        {

            return View();

        }

    }

}

## Creating Custom notifications

Create a Temp Data variable e.g success in the view page and assign it a value on a given condition

if (result.Succeeded)

    TempData["Success"] = "Your information has been edited";

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

Create a partial view(A regular view but without a layout page and can be inserted into another view) as a partial View in Shared folder called \_NotificationsPartial

@if(TempData["Success"] != null)

{

    <div class="alert alert-success">

        @TempData["Success"]

    </div>

}

@if (TempData["Error"] != null)

{

    <div class="alert alert-danger notification">

        @TempData["Error"]

    </div>

}

Edit layout page above @render body

<div class="container">

    <main role="main" class="pb-3">

        <partial name="\_NotificationPartial" />

        @RenderBody()

    </main>

</div>

## Creating Foreign Key Implementation

Create Model 1

namespace Zlt.Models

{

    public class Customer

    {

        public int Id { get; set; }

        public string Name { get; set; }

    }

}

Create Model 2

using System.ComponentModel;

using System.ComponentModel.DataAnnotations.Schema;

namespace Zlt.Models

{

    public class BalePeeler

    {

        public int Id { get; set; }

        [DisplayName("Bale Number")]

        public string BaleNumber { get; set; }

        public int Mass { get; set; }

        [DisplayName("Peel Time")]

        public DateTime PeelTime { get; set; }

        //Foreign Key Implementation

        [DisplayName("Customer Name")]

        public int CustomerId { get; set; }

        [ForeignKey("CustomerId")]

        public Customer Customer { get; set; }

    }

}

In the Controller

 // GET: BalePeelers/Create

        public IActionResult Create()

        {

            ViewBag.CustomerId = new SelectList(\_context.Customer.OrderBy(x => x.Name), "Id", "Name");

            return View();

        }

        // POST: BalePeelers/Create

        // To protect from overposting attacks, enable the specific properties you want to bind to.

        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

        [HttpPost]

        [ValidateAntiForgeryToken]

        public async Task<IActionResult> Create([Bind("Id,BaleNumber,Mass,PeelTime,ClientName")] BalePeeler balePeeler)

        {

            ViewBag.CustomerId = new SelectList(\_context.Customer.OrderBy(x => x.Name), "Id", "Name");

            if (ModelState.IsValid)

            {

                \_context.Add(balePeeler);

                await \_context.SaveChangesAsync();

                return RedirectToAction(nameof(Index));

            }

            return View(balePeeler);

        }

In the Create View

<div class="form-group">

    <label asp-for="CustomerId" class="control-label"></label>

    <select asp-for="CustomerId" asp-items="@ViewBag.CustomerId" class="form-control">

        <option value = 0>Choose Category</option>

    </select>

    <span asp-validation-for="CustomerId" class="text-danger"></span>

</div>

### On delete cascade

public class Blog

{

    public int Id { get; set; }

    public string Name { get; set; }

    public IList<Post> Posts { get; } = new List<Post>();

}

public class Post

{

    public int Id { get; set; }

    public string Title { get; set; }

    public string Content { get; set; }

    public int BlogId { get; set; }

    public Blog Blog { get; set; }

}

## Routing

Done in the program.cs file

app.MapControllerRoute(

    name: "default",

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

## Layout Pages

You can create many layout pages and then specify the layout page to use in a view in the \_ViewStart.cshtml

@{

    Layout = "\_Layout";

}

### View Component

1. Create a class in the Infrastructure folder

public class SerialsViewComponent : ViewComponent

{

    private readonly ApplicationDbContext \_context;

    public SerialsViewComponent(ApplicationDbContext context)

    {

        \_context = context;

    }

    public async Task<IViewComponentResult> InvokeAsync()

    {

        var serials = await GetSerialsAsync();

        return View(serials);

    }

    private Task<List<SerialNumber>> GetSerialsAsync()

    {

        return \_context.SerialNumbers.OrderBy(x => x.Id).ToListAsync();

    }

}

1. Create a folder with path: Shared/Components/Serials

@model IEnumerable<SerialNumber>

<ul class="list-group">

    <li class="list-group-item">

        <a class="nav-link text-dark" asp-area="" asp-controller="SerialNumbers">All Serial Numbers</a>

    </li>

    @foreach(var item in Model){

        <li class="list-group-item">

            <a class="nav-link text-dark" asp-area="" asp-controller="SerialNumbers" >@item.Name</a>

        </li>

    }

</ul>

1. Create a new layout page and assign it in the Views

@{

    Layout = "\_LayoutSerials";

}

1. In the Layout page

<div class="container">

    <main role="main" class="pb-3">

        <partial name="\_NotificationPartial" />

        <div class="row">

            <div class="col-3">

                @await Component.InvokeAsync("Serials")

            </div>

            <div class="col">

                @RenderBody()

            </div>

        </div>

    </main>

</div>

## Disabling an input tag

<form asp-action="Create">

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

    <!--Use disabled keyword-->

    <div class="form-group">

        <label asp-for="DeviceId" class="control-label"></label>

        <select asp-for="DeviceId" class="form-control" asp-items="ViewBag.DeviceId" disabled></select>

    </div>

    <div class="form-group">

        <label asp-for="LocationId" class="control-label"></label>

        <select asp-for="LocationId" class="form-control" asp-items="ViewBag.LocationId" disabled></select>

    </div>

    <div class="form-group">

        <label asp-for="State" class="control-label"></label>

        <select asp-for="State" class="form-control" asp-items="ViewBag.States" disabled></select>

    </div>

    <div class="form-group">

        <label asp-for="Name" class="control-label"></label>

        <input asp-for="Name" class="form-control" />

        <span asp-validation-for="Name" class="text-danger"></span>

    </div>

    <br />

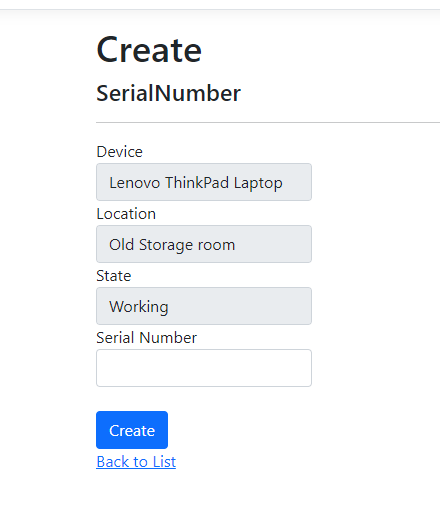
    <div class="form-group">

        <input type="submit" value="Create" class="btn btn-primary" />

    </div>

</form>

However this method does not post the disabled forms into the database. And alternative is thereadonly tag but it only works on input tags



## Active Directory

1. Install-Package Microsoft.Windows.Compatibility
2. Create a class in the Infrastructure Folder

using Microsoft.AspNetCore.Mvc.Rendering;

namespace Inventory.Infrastructure

{

    public class UserViewModel

    {

        public IEnumerable<SelectListItem>? MyADUsers{ get; set; }

        public int SelectedPersonId { get; set; }

    }

}

1. Create a controller or a method to be used in a controller

public class UsersController:Controller

{

    public IActionResult Index()

    {

        UserViewModel myADUsers = new UserViewModel

        {

            MyADUsers = GetADUsers()

        };

        return View(myADUsers);

    }

    public IEnumerable<SelectListItem> GetADUsers()

    {

        List<SelectListItem> \_users = new List<SelectListItem>();

        //Instantiate Active Directory Server Connection

        PrincipalContext adServer = new PrincipalContext(ContextType.Domain, null); //default domain

        var userPrincipal = new UserPrincipal(adServer);

        using (var search = new PrincipalSearcher(userPrincipal))

        {

            foreach(var p in search.FindAll())

            {

                if(p.DisplayName != null)

                {

                    \_users.Add(new SelectListItem { Text=p.DisplayName, Value=p.DisplayName})

                }

            }

            \_users=\_users.OrderBy(x => x.Value).ToList();

        }

        IEnumerable<SelectListItem> myADUsers = \_users;

        return myADUsers;

    }

}

1. Add View Page

@model Inventory.Infrastructure.UserViewModel

<h1>Active directory</h1>

<div class="row">

    <div class="col-md-4">

        <form asp-action="Create">

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

            <!--Active Directory Users Dropdown List-->

            <div class="form-group">

                <label asp-for="MyADUsers" class="control-label">Users</label>

                <select asp-for="MyADUsers" class="form-control" asp-items="@Model.MyADUsers"></select>

            </div>

            <div class="form-group">

                <label asp-for="MyADUsers" class="control-label col-md-2"></label>

                <div class="col-md-10">

                    @Html.DropDownList("My AD users", Model.MyADUsers, "-----Select User-----")

                    @Html.ValidationMessageFor(m=>m.MyADUsers)

                </div>

            </div>

        </form>

    </div>

</div>

### Adding Extra items into Active Directory Viewbag

1. Alter Controller

public IEnumerable<SelectListItem> GetADUsers()

{

    List<SelectListItem> \_users = new List<SelectListItem>();

    \_users.Add(new SelectListItem { Text = "Shipping Department", Value = "Shipping Deprtment" });

    \_users.Add(new SelectListItem { Text = "Leaf Accounts Department", Value = "Leaf Accounts Department" });

    //Instantiate Active Directory Server Connection

    PrincipalContext adServer = new PrincipalContext(ContextType.Domain, null); //default domain

    var userPrincipal = new UserPrincipal(adServer);

    using (var search = new PrincipalSearcher(userPrincipal))

    {

        foreach (var p in search.FindAll())

        {

            if (p.DisplayName != null)

            {

                \_users.Add(new SelectListItem { Text = p.DisplayName, Value = p.DisplayName });

            }

        }

        \_users = \_users.OrderBy(x => x.Value).ToList();

    }

    IEnumerable<SelectListItem> myADUsers = \_users;

        return myADUsers;

}

## Adding Image

1. Create Model which contains the File properties

public class Device

{

    public int Id { get; set; }

    [DisplayName("Brand")]

    public int BrandId { get; set; }

    [ForeignKey("BrandId")]

    public Brand? Brand { get; set; }

    [DisplayName("Model")]

    public int ModelId { get; set; }

    [ForeignKey("ModelId")]

    public Model? Model { get; set; }

    [DisplayName("Category")]

    public int CategoryId { get; set; }

    [ForeignKey("CategoryId")]

    public Category? Category { get; set; }

    public string? DeviceName { get; set; }

    //File Properties

    public string? Image { get; set; }

    [NotMapped]

    [FileExtension]

    public IFormFile? ImageUpload { get; set; }

}

1. In the Create and Edit pages

<div class="form-group">

    <label asp-for="Image" class="control-label">Product Image</label>

    <input asp-for="ImageUpload" class="form-control" />

    <br />

    <img src="" id="imgpreview" alt="" />

    <span asp-validation-for="ImageUpload" class="text-danger"></span>

</div>

1. Alter the form line

<form asp-action="Edit" enctype="multipart/form-data">

1. For preview, ensure line 5 in step 2 is present
2. In the site.js add this function

function readURL(input) {

    if (input.files && input.files[0]) {

        let reader = new FileReader();

        reader.onload = function (e) {

            $("img#imgpreview").attr("src", e.target.result).width(200).height(200);

        };

        reader.readAsDataURL(input.files[0]);

    }

};

1. In the create and edit page at the bottom

@section Scripts {

    @{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

    <script>

        $("#ImageUpload").change(function () {

            readURL(this);

        });

    </script>

}

### Create a custom validation for files

1. Create class in Infrastructure

public class FileExtensionAttribute:ValidationAttribute

{

    protected override ValidationResult? IsValid(object? value, ValidationContext validationContext)

    {

        //var \_context = (CmsShoppingCartContext)validationContext.GetService(typeof(CmsShoppingCartContext));

        var file = value as IFormFile;

        if(file != null)

        {

            var extension = Path.GetExtension(file.FileName);

            string[] extensions =

            {

                "jpg", "png"

            };

            bool result = extension.Any(x => extension.EndsWith(x));

            if (!result)

            {

                return new ValidationResult(GetErrorMessage());

            }

        }

        return ValidationResult.Success;

    }

    private string GetErrorMessage()

    {

        return "Allowed extensions are jpg and png.";

    }

}

1. Ensure that the data annotation is present in the model

[FileExtension]

public IFormFile? ImageUpload { get; set; }

### Posting to database

1. Alter the POST method

string imageName = "noimage.png";

if (device.ImageUpload != null)

{

    string uploadDir = Path.Combine(\_webHostEnvironment.WebRootPath, "media/products");

    imageName = Guid.NewGuid().ToString() + "\_" + product.ImageUpload.FileName;

    string filePath = Path.Combine(uploadDir, imageName);

    FileStream fs = new FileStream(filePath, FileMode.Create);

    await product.ImageUpload.CopyToAsync(fs);

    fs.Close();

}

device.Image = imageName;

\_context.Add(device);

await \_context.SaveChangesAsync();

1. Alter Controller Constructor

private readonly ApplicationDbContext \_context;

private readonly IWebHostEnvironment webHostEnvironment;

public DevicesController(ApplicationDbContext context, IWebHostEnvironment webHostEnvironment)

{

    \_context = context;

    this.webHostEnvironment = webHostEnvironment;

}

1. In the [www.root](http://www.root) folder, create a new folder called media then another one inside it called devices
2. Paste a picture that says no image, called noimage.png
3. Add image tag in the index and details pages

<img src="/media/devices/@item.Image" width="100px" />

1. Remove this from the edit and Create methods

[Bind("Id,BrandId,ModelId,CategoryId,DeviceName")]

## Validating a foreign key select list

1. Add validation in the model

[Range(1, int.MaxValue, ErrorMessage = "You must choose a brand")]

public int BrandId { get; set; }

## Devices by Category

1. Copy the index method and rename it

 // GET: Devices/Category

public async Task<IActionResult> DevicesByCategory(string categoryName)

{

    Category category = await \_context.Categories.Where(x => x.Name == categoryName).FirstOrDefaultAsync();

    if (category == null)

    {

        RedirectToAction("Index");

    }

    var applicationDbContext = \_context.Devices.Include(d => d.Brand).Include(d => d.Category).Include(d => d.Model).Where(x=>x.CategoryId == category.Id);

    ViewBag.CategoryName = categoryName;

    return View(await applicationDbContext.ToListAsync());

}

1. Create a custom route in program.cs

app.MapControllerRoute(

    "devices",

    "Devices/{categoryName}",

    defaults: new {controller="Devices", action = "DevicesByCategory"}

);

1. Copy Index page and rename to DevicesByCategory.cshtml

@model IEnumerable<Inventory.Models.Device>

    @{

        ViewData["Title"] = ViewBag.CategoryName;

    }

    <h1>ViewBag.CategoryName</h1>

    <p>

        <a asp-action="Create">Create New</a>

    </p>

    <table class="table">

        <thead>

            <tr>

                <th>

                    @Html.DisplayNameFor(model => model.Brand)

                </th>

                <th>

                    @Html.DisplayNameFor(model => model.Model)

                </th>

                <th>

                    @Html.DisplayNameFor(model => model.Category)

                </th>

                <th>

                    @Html.DisplayNameFor(model => model.DeviceName)

                </th>

                <th>

                    Inage

                </th>

                <th></th>

            </tr>

        </thead>

        <tbody>

    @foreach (var item in Model) {

            <tr>

                <td>

                    @Html.DisplayFor(modelItem => item.Brand.Name)

                </td>

                <td>

                    @Html.DisplayFor(modelItem => item.Model.Name)

                </td>

                <td>

                    @Html.DisplayFor(modelItem => item.Category.Name)

                </td>

                <td>

                    @Html.DisplayFor(modelItem => item.DeviceName)

                </td>

                <td>

                        <img src="/media/devices/@item.Image" width="100px" />

                </td>

                <td>

                    <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |

                    <a asp-action="Details" asp-route-id="@item.Id">Details</a> |

                    <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>

                </td>

            </tr>

    }

        </tbody>

    </table>

## Inline edit

### Select list

1. In the Controller add this method

// POST: SerialNumbers/Edit/5

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> UpdateRecord(int mainid, string usersel, SerialNumber serialNumber)

{

    var serialNum = await \_context.SerialNumbers.FindAsync(mainid);

    serialNum.User = usersel;

    \_context.Update(serialNum);

    await \_context.SaveChangesAsync();

    return RedirectToAction(nameof(Main));

}

1. In the view, make sure to give the select tag a class of userselect

<td>

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

        <input type="hidden" name="mainid" value="@item.Id"/>

        <div class="form-group">

            <label asp-for="@item.User" class="control-label"></label>

            <select asp-for="@item.User" class="form-control userselect" name="usersel" asp-items="ViewBag.Users">

                <option value=" " class="text-center">---Choose User---</option>

            </select>

        </div>

    </form>

</td>

@section scripts{

    <script type="text/javascript">

        $(function (){

            $(".userselect").change(function (){

                $(this).closest('form').submit();

            })

        })

    </script>

}

## Modal Pop Up

### Creating

1. Create class

namespace Inventory.Models

{

    public class Deallocations

    {

        public int Id { get; set; }

        public int SerialNumberId { get; set; }

        public SerialNumber SerialNumber { get; set; }

        public DateTime DeallocationDate { get; set; }

        public string User { get; set; }

    }

}

1. Add migrations and all other overheads
2. In the Controller

[HttpGet]

public IActionResult Create()

{

    Deallocations deallocations = new Deallocations();

    ViewData["SerialNumberId"] = new SelectList(\_context.SerialNumbers, "Id", "Name", deallocations.SerialNumberId);

    return PartialView("\_ConfirmPartialView", deallocations);

}

[HttpPost]

public IActionResult Create(Deallocations deallocations)

{

    ViewData["SerialNumberId"] = new SelectList(\_context.SerialNumbers, "Id", "Name", deallocations.SerialNumberId);

    ViewData["SerialNumberId"] = new SelectList(\_context.SerialNumbers, "Id", "Name", deallocations.SerialNumberId);

    \_context.Deallocations.Add(deallocations);

    \_context.SaveChanges();

    return PartialView("\_ConfirmPartialView", deallocations);

}

1. Create a partial View in the same folder \_ConfirmPartialView

@model Deallocations

<div class="modal fade" id="#addDeallocation" data-bs-backdrop="static" data-bs-keyboard="false">

    <div class="modal-dialog">

        <div class="modal-content">

            <div class="modal-header">

                <h5 class="modal-title" id="addDeallocationLabel">Deallocation</h5>

                <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close">

                    <span aria-hidden="true"hidden>&times;</span>

                </button>

            </div>

            <div class="modal-body">

                <form asp-action="Create">

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

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

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

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

                    <input type="hidden" asp-for="DeallocationDate" value="@DateTime.Today;"/>

                    <div class="form-group">

                        <label asp-for="SerialNumber.Device.DeviceName" class="control-label">Device </label>

                        <select asp-for="SerialNumber.Device.DeviceName" class="form-control" asp-items="ViewBag.Users">

                            <option value="0" class="text-center">---Choose User---</option>

                        </select>

                    </div>

                    <div class="form-group">

                        <label asp-for="SerialNumberId" class="control-label">SerialNumber</label>

                        <select asp-for="SerialNumberId" class="form-control" asp-items="ViewBag.SerialNumberId">

                            <option value="0" class="text-center">---Choose SerialNumber---</option>

                        </select>

                    </div>

                    <div class="form-group">

                        <label asp-for="User" class="control-label"></label>

                        <select asp-for="User" class="form-control" asp-items="ViewBag.Users" >

                            <option value="0" class="text-center">---Choose User---</option>

                        </select>

                    </div>

                    <br />

                    <div class="form-group">

                        <input type="submit" value="Save" class="btn btn-primary" />

                    </div>

                </form>

            </div>

            <div class="modal-footer">

                <button type="button" class="btn btn-primary" data-dismiss="modal">Close</button>

                <button type="button" class="btn btn-primary" data-save="modal">Save</button>

            </div>

        </div>

    </div>

</div>

1. In the site.js

$(function () {

    var PlaceHolderElement = $('#PlaceHolderHere');

    $('button[data-toggle="ajax-modal"]').click(function (event) {

        var url = $(this).data('url');

        $.get(url).done(function (data) {

            PlaceHolderElement.html(data);

            PlaceHolderElement.find('.modal').modal('show');

        })

    })

});

## Cascading Dropdown

1. In the Create View

<form asp-action="Create" enctype="multipart/form-data">

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

    <div class="form-group">

        <label asp-for="BrandId" class="control-label"></label>

        <select asp-for="BrandId" id="BrandList" class="form-control" asp-items="@ViewBag.BrandId");"></select>

    </div>

    <div class="form-group">

    <label asp-for="ModelId" class="control-label"></label>

    <select id="ModelList" name="ModelId" class="form-control" asp-items="ViewBag.ModelId">

        <option value="0">---Select Model---</option>

    </select>

</div>

    <div class="form-group">

        <label asp-for="CategoryId" class="control-label"></label>

        <select asp-for="CategoryId" id="CategoryList" class="form-control" asp-items="ViewBag.CategoryId">

            <option value="0">---Select Category---</option>

        </select>

    </div>

    <div class="form-group">

        <label asp-for="Image" class="control-label"></label>

        <input asp-for="ImageUpload" class="form-control" />

        <br />

        <img src="" id="imgpreview" alt="" />

        <span asp-validation-for="ImageUpload" class="text-danger"></span>

    </div>

    <div class="form-group">

        <input type="submit" value="Create" class="btn btn-primary" />

    </div>

</form>

1. At the bottom in the section scripts of the view

@section Scripts {

    @{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

    <script>

        //Cascading Dropdown

        $(function() {

            $("select#BrandList").change(function(){

                var cid = $(this).val();

                $("select#ModelList").empty();

                var url = '@Url.Content("~/")' + "Devices/LoadModel?cid=${cid}";

                var ddlsource = "#BrandList";

                $.getJSON(url,{id:$(ddlsource).val()}, function(data){

                    var items='';

                    $("select#ModelList").append($('<option></option>').val('').text('- Please select -'));

                    $.each(data, function (i, row){

                        $("select#ModelList").append(`<option value="${row.value}">${row.text}</option>`);

                    });

                });

            })

        });

    </script>

}

1. In the Controller add this method

[HttpGet]

public JsonResult LoadModel (int Id)

{

    var model = \_context.Models.Where(c => c.BrandId == Id).ToList();

    return Json(new SelectList(model, "Id","Name"));

}

1. On the POST method definition in the Controller, initialise a parameter for the filtered input

 public async Task<IActionResult> Create(Device device, int ModelId)

## Multi-level dropdown menus

1. In the layout View

<nav class="navbar navbar-expand-lg navbar-dark bg-primary">

<div class="container-fluid">

<a class="navbar-brand" href="#">Brand</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#main\_nav"  aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="main\_nav">

    <ul class="navbar-nav">

    <li class="nav-item active"> <a class="nav-link" href="#">Home </a> </li>

    <li class="nav-item"><a class="nav-link" href="#"> About </a></li>

    <li class="nav-item dropdown" id="myDropdown">

        <a class="nav-link dropdown-toggle" href="#" data-bs-toggle="dropdown">  Treeview menu  </a>

        <ul class="dropdown-menu">

        <li> <a class="dropdown-item" href="#"> Dropdown item 1 </a></li>

        <li> <a class="dropdown-item" href="#"> Dropdown item 2 &raquo; </a>

            <ul class="submenu dropdown-menu">

            <li><a class="dropdown-item" href="#">Submenu item 1</a></li>

            <li><a class="dropdown-item" href="#">Submenu item 2</a></li>

            <li><a class="dropdown-item" href="#">Submenu item 3 &raquo; </a>

                <ul class="submenu dropdown-menu">

                <li><a class="dropdown-item" href="#">Multi level 1</a></li>

                <li><a class="dropdown-item" href="#">Multi level 2</a></li>

                </ul>

            </li>

            <li><a class="dropdown-item" href="#">Submenu item 4</a></li>

            <li><a class="dropdown-item" href="#">Submenu item 5</a></li>

            </ul>

        </li>

        <li><a class="dropdown-item" href="#"> Dropdown item 3 </a></li>

        <li><a class="dropdown-item" href="#"> Dropdown item 4 </a></li>

        </ul>

    </li>

    </ul>

</div>

<!-- navbar-collapse.// -->

</div>

<!-- container-fluid.// -->

</nav>

1. In the script.cs file

document.addEventListener("DOMContentLoaded", function(){

// make it as accordion for smaller screens

if (window.innerWidth < 992) {

    // close all inner dropdowns when parent is closed

    document.querySelectorAll('.navbar .dropdown').forEach(function(everydropdown){

    everydropdown.addEventListener('hidden.bs.dropdown', function () {

        // after dropdown is hidden, then find all submenus

        this.querySelectorAll('.submenu').forEach(function(everysubmenu){

            // hide every submenu as well

            everysubmenu.style.display = 'none';

        });

    })

    });

    document.querySelectorAll('.dropdown-menu a').forEach(function(element){

    element.addEventListener('click', function (e) {

        let nextEl = this.nextElementSibling;

        if(nextEl && nextEl.classList.contains('submenu')) {

            // prevent opening link if link needs to open dropdown

            e.preventDefault();

            if(nextEl.style.display == 'block'){

            nextEl.style.display = 'none';

            } else {

            nextEl.style.display = 'block';

            }

        }

    });

    })

}

// end if innerWidth

});

// DOMContentLoaded  end

1. In the style.css file

/\* ============ desktop view ============ \*/

@media all and (min-width: 992px) {

    .dropdown-menu li{ position: relative;  }

    .nav-item .submenu{

        display: none;

        position: absolute;

        left:100%; top:-7px;

    }

    .nav-item .submenu-left{

        right:100%; left:auto;

    }

    .dropdown-menu > li:hover{ background-color: #f1f1f1 }

    .dropdown-menu > li:hover > .submenu{ display: block; }

}

/\* ============ desktop view .end// ============ \*/

/\* ============ small devices ============ \*/

@media (max-width: 991px) {

  .dropdown-menu .dropdown-menu{

      margin-left:0.7rem; margin-right:0.7rem; margin-bottom: .5rem;

  }

}

/\* ============ small devices .end// ============ \*/

## Multiplication of two textboxes on change

1. In Html

<!DOCTYPE html>

<html>

<head>

<meta charset=utf-8 />

<title>JS Bin</title>

</head>

<body>

<table width="80%" border="0">

  <tr>

    <th>Box 1</th>

    <th>Box 2</th>

    <th>Result</th>

  </tr>

  <tr>

    <td><input id="box1" type="text" oninput="calculate()" /></td>

    <td><input id="box2" type="text" oninput="calculate()" /></td>

    <td><input id="result" /></td>

  </tr>

  <tr>

    <td>&nbsp;</td>

    <td>&nbsp;</td>

    <td>&nbsp;</td>

  </tr>

</table>

</body>

</html>

1. In Javascript

function calculate() {

    var myBox1 = document.getElementById('box1').value;

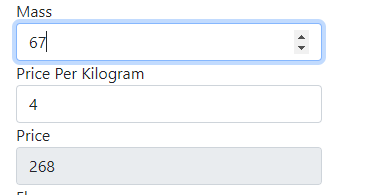
    var myBox2 = document.getElementById('box2').value;

    var result = document.getElementById('result');

    var myResult = myBox1 \* myBox2;

    result.value = myResult;

}



## Troubleshooting

1. Add-migration failed

PM> add-migration a

add-migration : The term 'add-migration' is not recognized as the name of a cmdlet, function, script file, or operable program. Check the spelling of the name, or if a path was included, verify that the path is correct and try again.

At line:1 char:1

+ add-migration a

+ ~~~~~~~~~~~~~

    + CategoryInfo          : ObjectNotFound: (add-migration:String) [], CommandNotFoundException

    + FullyQualifiedErrorId : CommandNotFoundException

Solution-Install efcore tools

1. Displaying DateTime variable as a number

servicePeriod = period.Value.Days;

Filtered dropdownList

[How to make Cascading dropdown in asp.net core mvc? - Stack Overflow](https://stackoverflow.com/questions/65662762/how-to-make-cascading-dropdown-in-asp-net-core-mvc)