Contents

[Adding Database 2](#_Toc124405949)

[Adding Seed Data to Database 3](#_Toc124405950)

[Creating Admin Area 4](#_Toc124405951)

[Creating Controller in Admin Areas 5](#_Toc124405952)

[Creating a select dropdown list 5](#_Toc124405953)

[Login Authentication/Authorization 6](#_Toc124405954)

[Identity 8](#_Toc124405955)

[Register New User 8](#_Toc124405956)

[Identity 2 8](#_Toc124405957)

[Adding Password validation in UserIdentity class 10](#_Toc124405958)

[Adding Login class 11](#_Toc124405959)

[Data Annotations 12](#_Toc124405960)

[Creating Foreign Key Implementation 14](#_Toc124405961)

[Routing 16](#_Toc124405962)

[Layout Pages 17](#_Toc124405963)

## 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>

## 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>

}

## 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("ClientId")]

        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>

## 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";

}