.NET CORE / XAMARIN FORMS / XAMARIN CLASSIC / MVVM CROSS

[**Create the Solution**](#_vkzdi5k13zc) **1**

[**Create the Database**](#_3znysh7) **3**

[**Modify DB**](#_2et92p0) **4**

[**Seed the DB with initial data**](#_tyjcwt) **5**

[**Implement the pattern repository**](#_3dy6vkm) **6**

[**Add User Identities**](#_1t3h5sf) **11**

[**Add API**](#_4d34og8) **15**

[**Implementing login and logout**](#_2s8eyo1) **16**

[**Registering new users**](#_17dp8vu) **19**

[**Modifying users**](#_3rdcrjn) **23**

[**Add Tokens Generation**](#_26in1rg) **28**

[**Add Font Awesome For Icons**](#_lnxbz9) **30**

[**Add Roles**](#_35nkun2) **31**

[**Orders functionality**](#_1ksv4uv) **33**

[**Add Modal Windows**](#_44sinio) **45**

[**Date Picker**](#_2jxsxqh) **48**

[**Cascade Drop Down List**](#_z337ya) **52**

[**Confirm Email Registration**](#_3j2qqm3) **77**

[**Password Recovery**](#_1y810tw) **81**

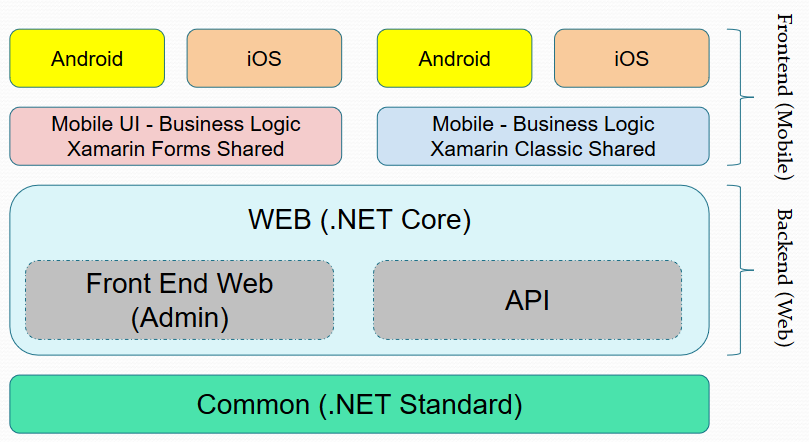
[**Installing MongoDB as a Service.**](#_cuwfbaxh285l) **86**

[**Creating Asp .net Core with MongoDB.**](#_soj4fqf6wb3w) **94**

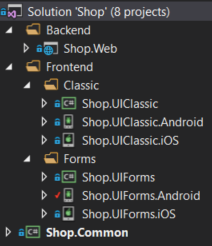
# 

# Create the Solution

Create the following solution:



In Visual Studio, you must build something similar to:



# Create the Database

1. Create the entities (in folder Common.Models):

using System;

using System.ComponentModel.DataAnnotations;

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

[DisplayFormat(DataFormatString = "{0:C2}", ApplyFormatInEditMode = false)]

public decimal Price { get; set; }

[Display(Name = "Image")]

public string ImageUrl { get; set; }

[Display(Name = "Last Purchase")]

public DateTime LastPurchase { get; set; }

[Display(Name = "Last Sale")]

public DateTime LastSale { get; set; }

[Display(Name = "Is Availabe?")]

public bool IsAvailabe { get; set; }

[DisplayFormat(DataFormatString = "{0:N2}", ApplyFormatInEditMode = false)]

public double Stock { get; set; }

}

1. Create the context class (in folder Data):

using Common.Models;

using Microsoft.EntityFrameworkCore;

public class DataContext : DbContext

{

public DbSet<Product> Products { get; set; }

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

{

}

}

1. Add the connection string to the configuration json file (see the SQL Server Object Explorer):

{

"Logging": {

"LogLevel": {

"Default": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\ProjectsV13;Database=Shop;Trusted\_Connection=True;MultipleActiveResultSets=true"

}

}

1. Add the database injection in startup class (before MVC services lines):

services.AddDbContext<DataContext>(cfg =>

{

cfg.UseSqlServer(this.Configuration.GetConnectionString("DefaultConnection"));

});

1. Run this commands by command line in the same folder that is the web project:

donet ef database update

dotnet ef migrations add InitialDb

donet ef database update

Or you can run this commands in package manager console:

PM> update-database

PM> add-migration InitialDb

PM> update-database

1. Add the products controller.
2. Add the products menu and test the DB connection.

<ul class="nav navbar-nav">

<li><a asp-area="" asp-controller="Home" asp-action="Index">Home</a></li>

<li><a asp-area="" asp-controller="Home" asp-action="About">About</a></li>

<li><a asp-area="" asp-controller="Home" asp-action="Contact">Contact</a></li>

<li><a asp-area="" asp-controller="Products" asp-action="Index">Products</a></li>

</ul>

# Modify DB

1. Modify the model product by:

using System;

using System.ComponentModel.DataAnnotations;

public class Product

{

public int Id { get; set; }

[MaxLength(50)]

[Required]

public string Name { get; set; }

[DisplayFormat(DataFormatString = "{0:C2}", ApplyFormatInEditMode = false)]

public decimal Price { get; set; }

[Display(Name = "Image")]

public string ImageUrl { get; set; }

[Display(Name = "Last Purchase")]

public DateTime? LastPurchase { get; set; }

[Display(Name = "Last Sale")]

public DateTime? LastSale { get; set; }

[Display(Name = "Is Availabe?")]

public bool IsAvailabe { get; set; }

[DisplayFormat(DataFormatString = "{0:N2}", ApplyFormatInEditMode = false)]

public double Stock { get; set; }

}

1. Run this commands:

dotnet ef migrations add ModifyProducts

donet ef database update

Or you can run this commands in package manager console:

PM> add-migration ModifyProducts

PM> update-database

1. Finally, add this method on context class to avoid the compilation warning.

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<Product>()

.Property(p => p.Price)

.HasColumnType("decimal(18,2)");

}

1. Test it.

# Seed the DB with initial data

1. Create the seed class, with your population data logic:

using System;

using System.Linq;

using System.Threading.Tasks;

using Common.Models;

public class SeedDb

{

private readonly DataContext context;

private Random random;

public SeedDb(DataContext context)

{

this.context = context;

this.random = new Random();

}

public async Task SeedAsync()

{

await this.context.Database.EnsureCreatedAsync();

if (!this.context.Products.Any())

{

this.AddProduct("First Product");

this.AddProduct("Second Product");

this.AddProduct("Third Product");

await this.context.SaveChangesAsync();

}

}

private void AddProduct(string name)

{

this.context.Products.Add(new Product

{

Name = name,

Price = this.random.Next(100),

IsAvailabe = true,

Stock = this.random.Next(100)

});

}

}

1. Modify the Program class by:

using Data;

using Microsoft.AspNetCore;

using Microsoft.AspNetCore.Hosting;

using Microsoft.Extensions.DependencyInjection;

public class Program

{

public static void Main(string[] args)

{

var host = CreateWebHostBuilder(args).Build();

RunSeeding(host);

host.Run();

}

private static void RunSeeding(IWebHost host)

{

var scopeFactory = host.Services.GetService<IServiceScopeFactory>();

using (var scope = scopeFactory.CreateScope())

{

var seeder = scope.ServiceProvider.GetService<SeedDb>();

seeder.SeedAsync().Wait();

}

}

public static IWebHostBuilder CreateWebHostBuilder(string[] args) =>

WebHost.CreateDefaultBuilder(args)

.UseStartup<Startup>();

}

1. Add the injection for the seeder in Startup class (before cookie policy options lines):

services.AddTransient<SeedDb>();

1. Test it.

# Implement the pattern repository

1. Create the repository class:

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Common.Models;

public class Repository

{

private readonly DataContext context;

public Repository(DataContext context)

{

this.context = context;

}

public IEnumerable<Product> GetProducts()

{

return this.context.Products.OrderBy(p => p.Name);

}

public Product GetProduct(int id)

{

return this.context.Products.Find(id);

}

public void AddProduct(Product product)

{

this.context.Products.Add(product);

}

public void UpdateProduct(Product product)

{

this.context.Update(product);

}

public void RemoveProduct(Product product)

{

this.context.Products.Remove(product);

}

public async Task<bool> SaveAllAsync()

{

return await this.context.SaveChangesAsync() > 0;

}

public bool ProductExists(int id)

{

return this.context.Products.Any(p => p.Id == id);

}

}

1. Extract the interface for the repository class:

using System.Collections.Generic;

using System.Threading.Tasks;

using Common.Models;

public interface IRepository

{

void AddProduct(Product product);

Product GetProduct(int id);

IEnumerable<Product> GetProducts();

bool ProductExists(int id);

void RemoveProduct(Product product);

Task<bool> SaveAllAsync();

void UpdateProduct(Product product);

}

1. Replace the controller to uses the repository and not uses the database context:

using Data;

using Data.Entities;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using System.Threading.Tasks;

public class ProductsController : Controller

{

private readonly IRepository repository;

public ProductsController(IRepository repository)

{

this.repository = repository;

}

public IActionResult Index()

{

return View(this.repository.GetProducts());

}

public IActionResult Details(int? id)

{

if (id == null)

{

return NotFound();

}

var product = this.repository.GetProduct(id.Value);

if (product == null)

{

return NotFound();

}

return View(product);

}

public IActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(Product product)

{

if (ModelState.IsValid)

{

this.repository.AddProduct(product);

await this.repository.SaveAllAsync();

return RedirectToAction(nameof(Index));

}

return View(product);

}

public IActionResult Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var product = this.repository.GetProduct(id.Value);

if (product == null)

{

return NotFound();

}

return View(product);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(Product product)

{

if (ModelState.IsValid)

{

try

{

this.repository.UpdateProduct(product);

await this.repository.SaveAllAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!this.repository.ProductExists(product.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(product);

}

public IActionResult Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var product = this.repository.GetProduct(id.Value);

if (product == null)

{

return NotFound();

}

return View(product);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var product = this.repository.GetProduct(id);

this.repository.RemoveProduct(product);

await this.repository.SaveAllAsync();

return RedirectToAction(nameof(Index));

}

}

1. Add the injection for the repository in Startup class (before cookie policy options lines):

services.AddScoped<IRepository, Repository>();

1. Test it.

# Add User Identities

1. Create your own users class inherit from IdentityUser class (in Common.Models):

using Microsoft.AspNetCore.Identity;

public class User : IdentityUser

{

public string FirstName { get; set; }

public string LastName { get; set; }

}

1. Modify the data context class:

using Entities;

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

public class DataContext : IdentityDbContext<User>

{

public DbSet<Product> Products { get; set; }

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

{

}

}

1. Make the relations with other models:

public User User { get; set; }

1. Drop the database and add the new migrations with those commands:

dotnet ef database drop

dotnet ef migrations add Users

dotnet ef database update

Or you can run this commands in package manager console:

PM> drop-database

PM> add-migration Users

PM> update-database

1. Modify the seeder to add some user:

using System;

using System.Linq;

using System.Threading.Tasks;

using Common.Models;

using Microsoft.AspNetCore.Identity;

public class SeedDb

{

private readonly DataContext context;

private readonly UserManager<User> userManager;

private Random random;

public SeedDb(DataContext context, UserManager<User> userManager)

{

this.context = context;

this.userManager = userManager;

this.random = new Random();

}

public async Task SeedAsync()

{

await this.context.Database.EnsureCreatedAsync();

var user = await this.userManager.FindByEmailAsync("jzuluaga55@gmail.com");

if (user == null)

{

user = new User

{

FirstName = "Juan",

LastName = "Zuluaga",

Email = "jzuluaga55@gmail.com",

UserName = "jzuluaga55@gmail.com"

};

var result = await this.userManager.CreateAsync(user, "123456");

if (result != IdentityResult.Success)

{

throw new InvalidOperationException("Could not create the user in seeder");

}

}

if (!this.context.Products.Any())

{

this.AddProduct("First Product", user);

this.AddProduct("Second Product", user);

this.AddProduct("Third Product", user);

await this.context.SaveChangesAsync();

}

}

private void AddProduct(string name, User user)

{

this.context.Products.Add(new Product

{

Name = name,

Price = this.random.Next(100),

IsAvailabe = true,

Stock = this.random.Next(100),

User = user

});

}

}

1. Modify the configuration to setup the new functionality:

public void ConfigureServices(IServiceCollection services)

{

services.AddIdentity<User, IdentityRole>(cfg =>

{

cfg.User.RequireUniqueEmail = true;

cfg.Password.RequireDigit = false;

cfg.Password.RequiredUniqueChars = 0;

cfg.Password.RequireLowercase = false;

cfg.Password.RequireNonAlphanumeric = false;

cfg.Password.RequireUppercase = false;

})

.AddEntityFrameworkStores<DataContext>();

services.AddDbContext<DataContext>(cfg =>

{

cfg.UseSqlServer(this.Configuration.GetConnectionString("DefaultConnection"));

});

services.AddTransient<SeedDb>();

services.AddScoped<IRepository, Repository>();

services.Configure<CookiePolicyOptions>(options =>

{

// This lambda determines whether user consent for non-essential cookies is needed for a given request.

options.CheckConsentNeeded = context => true;

options.MinimumSameSitePolicy = SameSiteMode.None;

});

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1);

}

public void Configure(IApplicationBuilder app, IHostingEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/Home/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseAuthentication();

app.UseCookiePolicy();

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

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

});

}

1. Test it.

# Add API

1. Create the API controller, this is an example (in Web.Controllers.API):

using Data;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Logging;

using System;

[Route("api/[Controller]")]

public class ProductsController : Controller

{

private readonly IRepository repository;

private readonly ILogger<ProductsController> logger;

public ProductsController(IRepository repository, ILogger<ProductsController> logger)

{

this.repository = repository;

this.logger = logger;

}

[HttpGet]

public IActionResult GetProducts()

{

try

{

return this.Ok(this.repository.GetProducts());

}

catch (Exception ex)

{

var message = $"Error: {ex}";

this.logger.LogError(message);

return this.BadRequest(message);

}

}

}

1. Test it.

# Implementing login and logout

1. Create the model for login (in Web.Models):

using System.ComponentModel.DataAnnotations;

public class LoginViewModel

{

[Required]

[EmailAddress]

public string Username { get; set; }

[Required]

public string Password { get; set; }

public bool RememberMe { get; set; }

}

1. Create the controller for login:

using System.Linq;

using System.Threading.Tasks;

using Common.Models;

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using Models;

public class AccountController : Controller

{

private readonly SignInManager<User> signInManager;

public AccountController(SignInManager<User> signInManager)

{

this.signInManager = signInManager;

}

public IActionResult Login()

{

if (this.User.Identity.IsAuthenticated)

{

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

}

return this.View();

}

[HttpPost]

public async Task<IActionResult> Login(LoginViewModel model)

{

if (this.ModelState.IsValid)

{

var result = await this.signInManager.PasswordSignInAsync(

model.Username,

model.Password,

model.RememberMe,

false);

if (result.Succeeded)

{

if (this.Request.Query.Keys.Contains("ReturnUrl"))

{

return this.Redirect(this.Request.Query["ReturnUrl"].First());

}

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

}

}

this.ModelState.AddModelError(string.Empty, "Failed to login.");

return this.View(model);

}

public async Task<IActionResult> Logout()

{

await this.signInManager.SignOutAsync();

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

}

}

1. Create the view for login:

@model ShopPrep.Web.Models.LoginViewModel

@{

ViewData["Title"] = "Login";

}

<h2>Login</h2>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div asp-validation-summary="ModelOnly"></div>

<div class="form-group">

<label asp-for="Username">Username</label>

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

<span asp-validation-for="Username" class="text-warning"></span>

</div>

<script src="~/lib/jquery-validation/dist/jquery.validate.js"></script>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

<div class="form-check">

<input asp-for="RememberMe" type="checkbox" class="form-check-input" />

<label asp-for="RememberMe" class="form-check-label">Remember Me?</label>

</div>

<span asp-validation-for="RememberMe" class="text-warning"></span>

</div>

<div class="form-group">

<input type="submit" value="Login" class="btn btn-success" />

<a asp-action="Register" class="btn btn-primary">Register New User</a>

</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. Add the annotation authorize to the other controllers:

[Authorize]

1. Add the options login and logout in the menu:

<ul class="nav navbar-nav navbar-right">

@if (this.User.Identity.IsAuthenticated)

{

<li><a asp-area="" asp-controller="Account" asp-action="ChangeUser">@this.User.Identity.Name</a></li>

<li><a asp-area="" asp-controller="Account" asp-action="Logout">Logout</a></li>

}

else

{

<li><a asp-area="" asp-controller="Account" asp-action="Login">Login</a></li>

}

</ul>

1. If the any user is logged in, don’t show the products option in menu:

@if (this.User.Identity.IsAuthenticated)

{

<li><a asp-area="" asp-controller="Products" asp-action="Index">Products</a></li>

}

1. Test it.

# Registering new users

1. Create the model for register new users (in Web.Models):

using System.ComponentModel.DataAnnotations;

public class RegisterNewUserViewModel

{

[Required]

[Display(Name = "First Name")]

public string FirstName { get; set; }

[Required]

[Display(Name = "Last Name")]

public string LastName { get; set; }

[Required]

[DataType(DataType.EmailAddress)]

public string Username { get; set; }

[Required]

public string Password { get; set; }

[Required]

[Compare("Password")]

public string Confirm { get; set; }

}

1. Create the actions in the controller:

public IActionResult Register()

{

return this.View();

}

[HttpPost]

public async Task<IActionResult> Register(RegisterNewUserViewModel model)

{

if (this.ModelState.IsValid)

{

var user = await this.userManager.FindByEmailAsync(model.Username);

if (user == null)

{

user = new User

{

FirstName = model.FirstName,

LastName = model.LastName,

Email = model.Username,

UserName = model.Username

};

var result = await this.userManager.CreateAsync(user, model.Password);

if (result != IdentityResult.Success)

{

this.ModelState.AddModelError(string.Empty, "The user couldn't be created.");

return this.View(model);

}

var result2 = await this.signInManager.PasswordSignInAsync(

model.Username,

model.Password,

true,

false);

if (result2.Succeeded)

{

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

}

this.ModelState.AddModelError(string.Empty, "The user couldn't be login.");

return this.View(model);

}

this.ModelState.AddModelError(string.Empty, "The username is already registered.");

}

return this.View(model);

}

1. Create the register view:

@model ShopPrep.Web.Models.RegisterNewUserViewModel

@{

ViewData["Title"] = "Register";

}

<h2>Register New User</h2>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div asp-validation-summary="ModelOnly"></div>

<div class="form-group">

<label asp-for="FirstName">First Name</label>

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

<span asp-validation-for="FirstName" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="LastName">Last Name</label>

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

<span asp-validation-for="LastName" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="Username">Username</label>

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

<span asp-validation-for="Username" class="text-warning"></span>

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

<label asp-for="Confirm">Confirm</label>

<input asp-for="Confirm" type="password" class="form-control" />

<span asp-validation-for="Confirm" class="text-warning"></span>

</div>

<div class="form-group">

<input type="submit" value="Register New User" class="btn btn-primary" />

</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. Test it.

# Modifying users

1. Create this new models (in Web.Models):

using System.ComponentModel.DataAnnotations;

public class ChangeUserViewModel

{

[Required]

[Display(Name = "First Name")]

public string FirstName { get; set; }

[Required]

[Display(Name = "Last Name")]

public string LastName { get; set; }

}

And:

using System.ComponentModel.DataAnnotations;

public class ChangePasswordViewModel

{

[Required]

[Display(Name = "Current password")]

public string OldPassword { get; set; }

[Required]

[Display(Name = "New password")]

public string NewPassword { get; set; }

[Required]

[Compare("NewPassword")]

public string Confirm { get; set; }

}

1. Create this actions in the account controller:

public async Task<IActionResult> ChangeUser()

{

var user = await this.userManager.FindByEmailAsync(this.User.Identity.Name);

var model = new ChangeUserViewModel();

if (user != null)

{

model.FirstName = user.FirstName;

model.LastName = user.LastName;

}

return this.View(model);

}

[HttpPost]

public async Task<IActionResult> ChangeUser(ChangeUserViewModel model)

{

if (this.ModelState.IsValid)

{

var user = await this.userManager.FindByEmailAsync(this.User.Identity.Name);

if (user != null)

{

user.FirstName = model.FirstName;

user.LastName = model.LastName;

var respose = await this.userManager.UpdateAsync(user);

if (respose.Succeeded)

{

this.ViewBag.UserMessage = "User updated!";

}

else

{

this.ModelState.AddModelError(string.Empty, respose.Errors.FirstOrDefault().Description);

}

}

else

{

this.ModelState.AddModelError(string.Empty, "User no found.");

}

}

return this.View(model);

}

1. Create this view:

@model ShopPrep.Web.Models.ChangeUserViewModel

@{

ViewData["Title"] = "Register";

}

<h2>Update User</h2>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div asp-validation-summary="ModelOnly"></div>

<div class="form-group">

<label asp-for="FirstName">First Name</label>

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

<span asp-validation-for="FirstName" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="LastName">Last Name</label>

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

<span asp-validation-for="LastName" class="text-warning"></span>

</div>

<div class="form-group">

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

<a asp-action="ChangePassword" class="btn btn-success">Change Password</a>

</div>

<div class="text-success">@ViewBag.UserMessage</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. And now this actions in the controller to password modification:

public IActionResult ChangePassword()

{

return this.View();

}

[HttpPost]

public async Task<IActionResult> ChangePassword(ChangePasswordViewModel model)

{

if (this.ModelState.IsValid)

{

var user = await this.userManager.FindByNameAsync(this.User.Identity.Name);

if (user != null)

{

var result = await this.userManager.ChangePasswordAsync(user, model.OldPassword, model.NewPassword);

if (result.Succeeded)

{

return this.RedirectToAction("ChangeUser");

}

else

{

this.ModelState.AddModelError(string.Empty, result.Errors.FirstOrDefault().Description);

}

}

else

{

this.ModelState.AddModelError(string.Empty, "User no found.");

}

}

return this.View(model);

}

1. Finally add this view:

@model ShopPrep.Web.Models.ChangePasswordViewModel

@{

ViewData["Title"] = "Register";

}

@section Scripts {

<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>

<script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"></script>

}

<h2>Change Password</h2>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div asp-validation-summary="ModelOnly"></div>

<div class="form-group">

<label asp-for="OldPassword">Current password</label>

<input asp-for="OldPassword" type="password" class="form-control" />

<span asp-validation-for="OldPassword" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="NewPassword">New password</label>

<input asp-for="NewPassword" type="password" class="form-control" />

<span asp-validation-for="NewPassword" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="Confirm">Confirm</label>

<input asp-for="Confirm" type="password" class="form-control" />

<span asp-validation-for="Confirm" class="text-warning"></span>

</div>

<div class="form-group">

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

<a asp-action="ChangeUser" class="btn btn-success">Back to user</a>

</div>

</form>

</div>

</div>

1. Test it.

# Add Tokens Generation

1. Add this values in json configuration file:

{

"Logging": {

"LogLevel": {

"Default": "Warning"

}

},

"AllowedHosts": "\*",

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\ProjectsV13;Database=Core3;Trusted\_Connection=True;MultipleActiveResultSets=true"

},

"Tokens": {

"Key": "asdfghjikbnvcgfdsrtfyhgcvgfxdgc",

"Issuer": "localhost",

"Audience": "users"

}

}

1. Modify the accounts controller constructor:

public AccountController(

SignInManager<User> signInManager,

UserManager<User> userManager,

IConfiguration configuration)

{

this.signInManager = signInManager;

this.userManager = userManager;

this.configuration = configuration;

}

1. Add the method to generate the token in the account controller:

[HttpPost]

public async Task<IActionResult> CreateToken([FromBody] LoginViewModel model)

{

if (this.ModelState.IsValid)

{

var user = await this.userManager.FindByNameAsync(model.Username);

if (user != null)

{

var result = await this.signInManager.CheckPasswordSignInAsync(

user,

model.Password,

false);

if (result.Succeeded)

{

var claims = new[]

{

new Claim(JwtRegisteredClaimNames.Sub, user.Email),

new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(this.configuration["Tokens:Key"]));

var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(

this.configuration["Tokens:Issuer"],

this.configuration["Tokens:Audience"],

claims,

expires: DateTime.UtcNow.AddDays(15),

signingCredentials: credentials);

var results = new

{

token = new JwtSecurityTokenHandler().WriteToken(token),

expiration = token.ValidTo

};

return this.Created(string.Empty, results);

}

}

}

return this.BadRequest();

}

1. Add the authorization annotation to API controllers:

[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

1. Add the new configuration for validate the tokens (before data context lines):

services.AddAuthentication()

.AddCookie()

.AddJwtBearer(cfg =>

{

cfg.TokenValidationParameters = new TokenValidationParameters

{

ValidIssuer = this.Configuration["Tokens:Issuer"],

ValidAudience = this.Configuration["Tokens:Audience"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(this.Configuration["Tokens:Key"]))

};

});

1. Test it.

# Add Font Awesome For Icons

1. Add a NPM configuration file and add the line that references Font Awesome library:

{

"version": "1.0.0",

"name": "asp.net",

"private": true,

"devDependencies": {

"font-awesome": "^4.7.0"

}

}

1. Copy the hidden folder “node\_modules” into “wwwroot”.
2. Reference the font awesome css in “\_Layout”:

<environment include="Development">

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />

<link href="~/node\_modules/font-awesome/css/font-awesome.min.css" rel="stylesheet" />

<link rel="stylesheet" href="~/css/site.css" />

</environment>

1. Add some funny icons, for example in create a product view:

<div class="form-group">

<button type="submit" class="btn btn-primary"><i class="fa fa-save"></i> Create</button>

<a asp-action="Index" class="btn btn-success"><i class="fa fa-chevron-left"></i> Back to List</a>

</div>

# Add Roles

1. Modify the constructor for the seed class to add a role manager:

public SeedDb(

DataContext context,

UserManager<User> userManager,

RoleManager<IdentityRole> roleManager)

{

this.context = context;

this.userManager = userManager;

this.roleManager = roleManager;

}

1. Modify the seed method to add some roles, for example:

public async Task SeedAsync()

{

await this.context.Database.EnsureCreatedAsync();

await this.CheckRole("Admin");

await this.CheckRole("Customer");

var user = await this.userManager.FindByEmailAsync("jzuluaga55@gmail.com");

if (user == null)

{

user = new User

{

FirstName = "Juan",

LastName = "Zuluaga",

Email = "jzuluaga55@gmail.com",

UserName = "jzuluaga55@gmail.com"

};

var result = await this.userManager.CreateAsync(user, "123456");

if (result != IdentityResult.Success)

{

throw new InvalidOperationException("Could not create the user in seeder");

}

await this.userManager.AddToRoleAsync(user, "Admin");

}

var isInRole = await this.userManager.IsInRoleAsync(user, "Admin");

if (!isInRole)

{

await this.userManager.AddToRoleAsync(user, "Admin");

}

if (!this.context.Products.Any())

{

this.AddProduct("First Product", user);

this.AddProduct("Second Product", user);

this.AddProduct("Third Product", user);

await this.context.SaveChangesAsync();

}

}

private async Task CheckRole(string roleName)

{

var roleExists = await this.roleManager.RoleExistsAsync(roleName);

if (!roleExists)

{

await this.roleManager.CreateAsync(new IdentityRole

{

Name = roleName

});

}

}

1. Now you can include the role in authorization annotation:

[Authorize(Roles = "Admin")]

1. And check roles in the views:

@if (this.User.Identity.IsAuthenticated && this.User.IsInRole("Admin"))

{

<li><a asp-area="" asp-controller="Products" asp-action="Index">Products</a></li>

}

1. And add some role when the user is created:

if (result2.Succeeded)

{

await this.userManager.AddToRoleAsync(user, "Customer");

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

}

1. Test it.

# Products improvements

1. Create the folder **Helpers** and inside it, add the interface **IUserHelper**:

using System.Threading.Tasks;

using Common.Models;

public interface IUserHelper

{

Task<User> GetUserByEmail(string email);

}

1. In the same folder add the class **UserHelper**:

using System.Threading.Tasks;

using Common.Models;

using Microsoft.AspNetCore.Identity;

public class UserHelper : IUserHelper

{

private readonly UserManager<User> userManager;

public UserHelper(UserManager<User> userManager)

{

this.userManager = userManager;

}

public async Task<User> GetUserByEmail(string email)

{

var user = await this.userManager.FindByEmailAsync(email);

return user;

}

}

1. Configure the injection:

services.AddTransient<SeedDb>();

services.AddScoped<IRepository, Repository>();

services.AddScoped<IUserHelper, UserHelper>();

1. Modify the method **GetProducts** in **Respository** class:

public Product GetProduct(int id)

{

return this.context.Products.Include(p => p.User).Where(p => p.Id == id).FirstOrDefault();

}

1. Inside **wwwroot** folder, add the folders **images/Products**.
2. Modify the **ProductsController**:

using System.IO;

using System.Threading.Tasks;

using Common.Models;

using Data;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Web.Helpers;

using Web.Models;

[Authorize(Roles = "Admin")]

public class ProductsController : Controller

{

private readonly IRepository repository;

private readonly IUserHelper userHelper;

public ProductsController(IRepository repository, IUserHelper userHelper)

{

this.repository = repository;

this.userHelper = userHelper;

}

public IActionResult Index()

{

return View(this.repository.GetProducts());

}

public IActionResult Details(int? id)

{

if (id == null)

{

return NotFound();

}

var product = this.repository.GetProduct(id.Value);

if (product == null)

{

return NotFound();

}

return View(product);

}

public IActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(ProductViewModel view)

{

if (ModelState.IsValid)

{

var path = string.Empty;

if (view.ImageFile != null && view.ImageFile.Length > 0)

{

path = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot\\images\\Products", view.ImageFile.FileName);

using (var stream = new FileStream(path, FileMode.Create))

{

await view.ImageFile.CopyToAsync(stream);

}

path = $"~/images/Products/{view.ImageFile.FileName}";

}

var product = await this.ToProduct(view, path);

this.repository.AddProduct(product);

await this.repository.SaveAllAsync();

return RedirectToAction(nameof(Index));

}

return View(view);

}

private async Task<Product> ToProduct(ProductViewModel view, string path)

{

return new Product

{

Id = view.Id,

ImageUrl = path,

IsAvailabe = view.IsAvailabe,

LastPurchase = view.LastPurchase,

LastSale = view.LastSale,

Name = view.Name,

Price = view.Price,

Stock = view.Stock,

User = await this.userHelper.GetUserByEmail(this.User.Identity.Name)

};

}

public IActionResult Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var product = this.repository.GetProduct(id.Value);

if (product == null)

{

return NotFound();

}

var view = this.ToProducViewModel(product);

return View(view);

}

private ProductViewModel ToProducViewModel(Product product)

{

return new ProductViewModel

{

Id = product.Id,

ImageUrl = product.ImageUrl,

IsAvailabe = product.IsAvailabe,

LastPurchase = product.LastPurchase,

LastSale = product.LastSale,

Name = product.Name,

Price = product.Price,

Stock = product.Stock,

User = product.User

};

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(ProductViewModel view)

{

if (ModelState.IsValid)

{

try

{

var path = view.ImageUrl;

if (view.ImageFile != null && view.ImageFile.Length > 0)

{

path = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot\\images\\Products", view.ImageFile.FileName);

using (var stream = new FileStream(path, FileMode.Create))

{

await view.ImageFile.CopyToAsync(stream);

}

path = $"~/images/Products/{view.ImageFile.FileName}";

}

var product = await this.ToProduct(view, path);

this.repository.UpdateProduct(product);

await this.repository.SaveAllAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!this.repository.ProductExists(view.Id))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction(nameof(Index));

}

return View(view);

}

public IActionResult Delete(int? id)

{

if (id == null)

{

return NotFound();

}

var product = this.repository.GetProduct(id.Value);

if (product == null)

{

return NotFound();

}

return View(product);

}

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public async Task<IActionResult> DeleteConfirmed(int id)

{

var product = this.repository.GetProduct(id);

this.repository.RemoveProduct(product);

await this.repository.SaveAllAsync();

return RedirectToAction(nameof(Index));

}

}

1. Modify the products Views:

**Create:**

@model ShopPrep.Web.Models.ProductViewModel

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<h4>Product</h4>

<hr />

<div class="row">

<div class="col-md-4">

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

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

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

<input asp-for="ImageFile" class="form-control" type="file"/>

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

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

<div class="checkbox">

<label>

<input asp-for="IsAvailabe" /> @Html.DisplayNameFor(model => model.IsAvailabe)

</label>

</div>

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

<button type="submit" class="btn btn-primary"><i class="fa fa-save"></i> Create</button>

<a asp-action="Index" class="btn btn-success"><i class="fa fa-chevron-left"></i> Back to List</a>

</div>

</form>

</div>

</div>

@section Scripts {

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

}

**Delete:**

@model ShopPrep.Common.Models.Product

@{

ViewData["Title"] = "Delete";

}

<h2>Delete</h2>

<h3>Are you sure you want to delete this?</h3>

<div>

<h4>Product</h4>

<hr />

<dl class="dl-horizontal">

<dt>

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

</dt>

<dd>

@Html.DisplayFor(model => model.Name)

</dd>

<dt>

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

</dt>

<dd>

@Html.DisplayFor(model => model.Price)

</dd>

<dt>

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

</dt>

<dd>

@if (!string.IsNullOrEmpty(Model.ImageUrl))

{

<img src="@Url.Content(Model.ImageUrl)" alt="Image" style="width:200px;height:300px;max-width: 100%; height: auto;" />

}

</dd>

<dt>

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

</dt>

<dd>

@Html.DisplayFor(model => model.LastPurchase)

</dd>

<dt>

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

</dt>

<dd>

@Html.DisplayFor(model => model.LastSale)

</dd>

<dt>

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

</dt>

<dd>

@Html.DisplayFor(model => model.IsAvailabe)

</dd>

<dt>

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

</dt>

<dd>

@Html.DisplayFor(model => model.Stock)

</dd>

</dl>

<form asp-action="Delete">

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

<input type="submit" value="Delete" class="btn btn-danger" />

<a asp-action="Index" class="btn btn-success"><i class="fa fa-chevron-left"></i> Back to List</a>

</form>

</div>

**Details**:

@model ShopPrep.Common.Models.Product

@{

ViewData["Title"] = "Details";

}

<h2>Details</h2>

<div>

<h4>Product</h4>

<hr />

<dl class="dl-horizontal">

<dt>

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

</dt>

<dd>

@Html.DisplayFor(model => model.Name)

</dd>

<dt>

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

</dt>

<dd>

@Html.DisplayFor(model => model.Price)

</dd>

<dt>

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

</dt>

<dd>

@if (!string.IsNullOrEmpty(Model.ImageUrl))

{

<img src="@Url.Content(Model.ImageUrl)" alt="Image" style="width:200px;height:300px;max-width: 100%; height: auto;" />

}

</dd>

<dt>

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

</dt>

<dd>

@Html.DisplayFor(model => model.LastPurchase)

</dd>

<dt>

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

</dt>

<dd>

@Html.DisplayFor(model => model.LastSale)

</dd>

<dt>

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

</dt>

<dd>

@Html.DisplayFor(model => model.IsAvailabe)

</dd>

<dt>

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

</dt>

<dd>

@Html.DisplayFor(model => model.Stock)

</dd>

</dl>

</div>

<div>

<a asp-action="Edit" asp-route-id="@Model.Id" class="btn btn-warning">Edit</a>

<a asp-action="Index" class="btn btn-success"><i class="fa fa-chevron-left"></i> Back to List</a>

</div>

**Edit**:

@model ShopPrep.Web.Models.ProductViewModel

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>Product</h4>

<hr />

<div class="row">

<div class="col-md-4">

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

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

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

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

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

<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>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

<input asp-for="ImageFile" class="form-control" type="file" />

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

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

<div class="checkbox">

<label>

<input asp-for="IsAvailabe" /> @Html.DisplayNameFor(model => model.IsAvailabe)

</label>

</div>

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

<a asp-action="Index" class="btn btn-success"><i class="fa fa-chevron-left"></i> Back to List</a>

</div>

</form>

</div>

<div class="col-md-4">

@if (!string.IsNullOrEmpty(Model.ImageUrl))

{

<img src="@Url.Content(Model.ImageUrl)" alt="Image" style="width:200px;height:300px;max-width: 100%; height: auto;" />

}

</div>

</div>

@section Scripts {

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

}

**Index**:

@model IEnumerable<ShopPrep.Common.Models.Product>

@{

ViewData["Title"] = "Index";

}

<h2>Index</h2>

<p>

<a asp-action="Create" class="btn btn-primary">Create New</a>

</p>

<table class="table">

<thead>

<tr>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model) {

<tr>

<td>

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

</td>

<td>

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

</td>

<td>

@if (!string.IsNullOrEmpty(item.ImageUrl))

{

<img src="@Url.Content(item.ImageUrl)" alt="Image" style="width:100px;height:150px;max-width: 100%; height: auto;" />

}

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

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

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

</td>

</tr>

}

</tbody>

</table>

1. Test it.

# Orders functionality

1. Add order detail temporarily model (in Common.Models):

using System.ComponentModel.DataAnnotations;

public class OrderDetailTemp

{

public int Id { get; set; }

public User User { get; set; }

public Product Product { get; set; }

[DisplayFormat(DataFormatString = "{0:C2}")]

public decimal Price { get; set; }

[DisplayFormat(DataFormatString = "{0:N2}")]

public double Quantity { get; set; }

[DisplayFormat(DataFormatString = "{0:C2}")]

public decimal Value { get { return this.Price \* (decimal)this.Quantity; } }

}

1. Add order detail model:

using System.ComponentModel.DataAnnotations;

public class OrderDetail

{

public int Id { get; set; }

public Product Product { get; set; }

[DisplayFormat(DataFormatString = "{0:C2}")]

public decimal Price { get; set; }

[DisplayFormat(DataFormatString = "{0:N2}")]

public double Quantity { get; set; }

[DisplayFormat(DataFormatString = "{0:C2}")]

public decimal Value { get { return this.Price \* (decimal)this.Quantity; } }

}

1. Add order model:

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

public class Order

{

public int Id { get; set; }

[Display(Name = "Order date")]

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd hh:mm tt}", ApplyFormatInEditMode = false)]

public DateTime OrderDate { get; set; }

[Display(Name = "Delivery date")]

[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd hh:mm tt}", ApplyFormatInEditMode = false)]

public DateTime DeliveryDate { get; set; }

public User User { get; set; }

public IEnumerable<OrderDetail> Items { get; set; }

[DisplayFormat(DataFormatString = "{0:N2}")]

public double Quantity { get { return this.Items == null ? 0 : this.Items.Sum(i => i.Quantity); } }

[DisplayFormat(DataFormatString = "{0:C2}")]

public decimal Value { get { return this.Items == null ? 0 : this.Items.Sum(i => i.Value); } }

}

1. Add the order and order detail temporarily to data context, it’s not necessary to add order detail, but I recommend to include it.

public DbSet<Product> Products { get; set; }

public DbSet<Order> Orders { get; set; }

public DbSet<OrderDetail> OrderDetails { get; set; }

public DbSet<OrderDetailTemp> OrderDetailTemps { get; set; }

1. Save all and run this commands to update the database:

dotnet ef migrations add OrderModels

dotnet ef database update

Or you can run this commands in package manager console:

PM> add-migration OrderModels

PM> update-database

1. Add the method to interface and implementation:

Task<IEnumerable<Order>> GetOrdersAsync(string userName);

And the implementation:

public async Task<IEnumerable<Order>> GetOrdersAsync(string userName)

{

var user = await this.userManager.FindByNameAsync(userName);

if (user == null)

{

return null;

}

var orders = this.context.Orders

.Include(o => o.Items)

.ThenInclude(i => i.Product)

.Where(o => o.User == user)

.OrderBy(o => o.OrderDate);

return orders;

}

1. Add an empty controller **OrdersController**:
2. Add the method to the orders controller:

using System.Threading.Tasks;

using Data;

using Microsoft.AspNetCore.Mvc;

public class OrdersController : Controller

{

private readonly IRepository repository;

public OrdersController(IRepository repository)

{

this.repository = repository;

}

public async Task<IActionResult> Index()

{

var model = await this.repository.GetOrdersAsync(this.User.Identity.Name);

return View(model);

}

}

1. Add the corresponding view:

@model IEnumerable<ShopPrep.Common.Models.Order>

@{

ViewData["Title"] = "Index";

}

<h2>Orders</h2>

<p>

<a asp-action="Create" class="btn btn-primary">Create New</a>

</p>

<table class="table">

<thead>

<tr>

<th>

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

</th>

<th>

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

</th>

<th>

# Lines

</th>

<th>

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

</th>

<th>

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

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

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

</td>

<td>

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

</td>

<td>

@Html.DisplayFor(modelItem => item.Items.Count())

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

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

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

</td>

</tr>

}

</tbody>

</table>

1. Add the new menu:

@if (this.User.Identity.IsAuthenticated)

{

@if (this.User.IsInRole("Admin"))

{

<li><a asp-area="" asp-controller="Products" asp-action="Index">Products</a></li>

}

<li><a asp-area="" asp-controller="Orders" asp-action="Index">Orders</a></li>

}

1. Add the method to get temporary orders for a user:

Task<IEnumerable<OrderDetailTemp>> GetDetailTempsAsync(string userName);

And the implementation:

public async Task<IEnumerable<OrderDetailTemp>> GetDetailTempsAsync(string userName)

{

var user = await this.userManager.FindByNameAsync(userName);

if (user == null)

{

return null;

}

var orderDetailTemps = this.context.OrderDetailTemps

.Include(o => o.Product)

.Where(o => o.User == user)

.OrderBy(o => o.Product.Name);

return orderDetailTemps;

}

1. Add the method create to the orders controller:

public async Task<IActionResult> Create()

{

var model = await this.repository.GetDetailTempsAsync(this.User.Identity.Name);

return this.View(model);

}

1. And their corresponding view:

@model IEnumerable<ShopPrep.Common.Models.OrderDetailTemp>

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<p>

<a asp-action="AddProduct" class="btn btn-success">Add Product</a>

<a asp-action="ConfirmOrder" class="btn btn-primary">Confirm Order</a>

</p>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Product.Name)

</th>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

<a asp-action="Increase" asp-route-id="@item.Id" class="btn btn-warning"><i class="fa fa-plus"></i></a>

<a asp-action="Decrease" asp-route-id="@item.Id" class="btn btn-info"><i class="fa fa-minus"></i></a>

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

</td>

</tr>

}

</tbody>

</table>

1. Create the model to add products to order temporary:

using Microsoft.AspNetCore.Mvc.Rendering;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

public class AddItemViewModel

{

[Display(Name = "Product")]

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

public int ProductId { get; set; }

[Range(0.0001, double.MaxValue, ErrorMessage = "The quantiy must be a positive number")]

public double Quantity { get; set; }

public IEnumerable<SelectListItem> Products { get; set; }

}

1. Add those methods to the interfaz:

IEnumerable<SelectListItem> GetComboProducts();

Task AddItemToOrderAsync(AddItemViewModel model, string userName);

Task ModifyOrderDetailTempQuantityAsync(int id, double quantity);

And to the implementation:

public IEnumerable<SelectListItem> GetComboProducts()

{

var list = this.context.Products.Select(p => new SelectListItem

{

Text = p.Name,

Value = p.Id.ToString()

}).ToList();

list.Insert(0, new SelectListItem

{

Text = "(Select a product...)",

Value = "0"

});

return list;

}

public async Task AddItemToOrderAsync(AddItemViewModel model, string userName)

{

var user = await this.userManager.FindByNameAsync(userName);

if (user == null)

{

return;

}

var product = await this.context.Products.FindAsync(model.ProductId);

if (product == null)

{

return;

}

var orderDetailTemp = await this.context.OrderDetailTemps

.Where(odt => odt.User == user && odt.Product == product)

.FirstOrDefaultAsync();

if (orderDetailTemp == null)

{

orderDetailTemp = new OrderDetailTemp

{

Price = product.Price,

Product = product,

Quantity = model.Quantity,

User = user,

};

this.context.OrderDetailTemps.Add(orderDetailTemp);

}

else

{

orderDetailTemp.Quantity += model.Quantity;

this.context.OrderDetailTemps.Update(orderDetailTemp);

}

await this.context.SaveChangesAsync();

}

public async Task ModifyOrderDetailTempQuantityAsync(int id, double quantity)

{

var orderDetailTemp = await this.context.OrderDetailTemps.FindAsync(id);

if (orderDetailTemp == null)

{

return;

}

orderDetailTemp.Quantity += quantity;

if (orderDetailTemp.Quantity > 0)

{

this.context.OrderDetailTemps.Update(orderDetailTemp);

await this.context.SaveChangesAsync();

}

}

1. Add the view model:

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using Microsoft.AspNetCore.Mvc.Rendering;

public class OrderDetailViewModel

{

[Display(Name = "Product")]

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

public int ProductId { get; set; }

public double Quantity { get; set; }

public IEnumerable<SelectListItem> Products { get; set; }

}

1. Add the methods to the orders controller:

public IActionResult AddProduct()

{

var model = new OrderDetailViewModel

{

Quantity = 1,

Products = this.GetProductsList()

};

return View(model);

}

private IEnumerable<SelectListItem> GetProductsList()

{

var products = this.repository.GetProducts().ToList();

products.Insert(0, new Product

{

Id = 0,

Name = "(Select a product...)"

});

return products.Select(p => new SelectListItem

{

Value = p.Id.ToString(),

Text = p.Name

}).ToList();

}

[HttpPost]

public async Task<IActionResult> AddProduct(AddItemViewModel model)

{

if (this.ModelState.IsValid)

{

await this.repository.AddItemToOrderAsync(model, this.User.Identity.Name);

return this.RedirectToAction("Create");

}

return this.View(model);

}

1. Add the view:

@model ShopPrep.Web.Models.OrderDetailViewModel

@{

ViewData["Title"] = "AddProduct";

}

<h2>Add Product</h2>

<h4>To Order</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="AddProduct">

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

<div class="form-group">

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

<select asp-for="ProductId" asp-items="Model.Products" class="form-control"></select>

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

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

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

</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. Add this method to interface and repository:

Task DeleteDetailTempAsync(int id);

And repository:

public async Task DeleteDetailTempAsync(int id)

{

var orderDetailTemp = await this.context.OrderDetailTemps.FindAsync(id);

if (orderDetailTemp == null)

{

return;

}

this.context.OrderDetailTemps.Remove(orderDetailTemp);

await this.context.SaveChangesAsync();

}

1. Now implement the delete item in orders controller:

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

{

if (id == null)

{

return NotFound();

}

await this.repository.DeleteDetailTempAsync(id.Value);

return this.RedirectToAction("Create");

}

1. Add the confirm order method in the interface and implementation:

Task<bool> ConfirmOrderAsync(string userName);

And in the implementation:

public async Task<bool> ConfirmOrderAsync(string userName)

{

var user = await this.userManager.FindByNameAsync(userName);

if (user == null)

{

return false;

}

var orderTmps = await this.context.OrderDetailTemps

.Include(o => o.Product)

.Where(o => o.User == user)

.ToListAsync();

if (orderTmps == null || orderTmps.Count == 0)

{

return false;

}

var details = orderTmps.Select(o => new OrderDetail

{

Price = o.Price,

Product = o.Product,

Quantity = o.Quantity

}).ToList();

var order = new Order

{

OrderDate = DateTime.UtcNow,

User = user,

Items = details,

};

this.context.Orders.Add(order);

this.context.OrderDetailTemps.RemoveRange(orderTmps);

await this.context.SaveChangesAsync();

return true;

}

1. Modify the order model:

public IEnumerable<OrderDetail> Items { get; set; }

[DisplayFormat(DataFormatString = "{0:N0}")]

public int Lines { get { return this.Items == null ? 0 : this.Items.Count(); } }

[DisplayFormat(DataFormatString = "{0:N2}")]

public double Quantity { get { return this.Items == null ? 0 : this.Items.Sum(i => i.Quantity); } }

1. And the index view in Orders:

@model IEnumerable<Core4.Data.Entities.Order>

@{

ViewData["Title"] = "Index";

}

<h2>Orders</h2>

<p>

<a asp-action="Create" class="btn btn-primary">Create New</a>

</p>

<table class="table">

<thead>

<tr>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

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

</td>

<td>

@if (item.DeliveryDate != DateTime.MinValue)

{

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

}

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

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

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

</td>

</tr>

}

</tbody>

</table>

1. Add the method to the controller:

public async Task<IActionResult> ConfirmOrder()

{

var response = await this.repository.ConfirmOrderAsync(this.User.Identity.Name);

if (response)

{

return this.RedirectToAction("Index");

}

return this.RedirectToAction("Create");

}

1. Test it.

# Add Modal Windows

1. To add a validation to confirm the order, add this lines at the end of crete view in orders:

@model IEnumerable<ShopPrep.Common.Models.OrderDetailTemp>

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<p>

<a asp-action="AddProduct" class="btn btn-success">Add Product</a>

<a asp-action="ConfirmOrder" class="btn btn-primary" id="btnConfirm">Confirm Order</a>

</p>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Product.Name)

</th>

<th>

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

</th>

<th>

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

</th>

<th>

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

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

<a asp-action="Increase" asp-route-id="@item.Id" class="btn btn-warning"><i class="fa fa-plus"></i></a>

<a asp-action="Decrease" asp-route-id="@item.Id" class="btn btn-info"><i class="fa fa-minus"></i></a>

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

</td>

</tr>

}

</tbody>

</table>

<div id="confirmDialog" class="modal fade">

<div class="modal-dialog modal-sm">

<div class="modal-content">

<div class="modal-header">

<button type="button" class="close" data-dismiss="modal"><i class="fa fa-window-close"></i></button>

<h4 class="modal-title">Confirm</h4>

</div>

<div class="modal-body">

<p>Do you want to confirm the order?</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-primary" id="btnYes">Yes</button>

<button type="button" class="btn btn-success" id="btnNo">No</button>

</div>

</div>

</div>

</div>

@section Scripts {

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

<script type="text/javascript">

$(document).ready(function () {

$("#btnConfirm").click(function () {

$("#confirmDialog").modal('show');

return false;

});

$("#btnNo").click(function () {

$("#confirmDialog").modal('hide');

return false;

});

$("#btnYes").click(function () {

window.location.href = '/Orders/ConfirmOrder';

});

});

</script>

}

1. Test it.
2. To add a validation to delete a product from the order, make this modifications in the view:

...

</td>

<td id="@item.Id">

<a asp-action="Increase" asp-route-id="@item.Id" class="btn btn-warning"><i class="fa fa-plus"></i></a>

<a asp-action="Decrease" asp-route-id="@item.Id" class="btn btn-info"><i class="fa fa-minus"></i></a>

<a asp-action="DeleteItem" asp-route-id="@item.Id" class="btn btn-danger" id="btnDeleteItem">Delete</a>

</td>

</tr>

}

</tbody>

</table>

<div id="confirmDialog" class="modal fade">

<div class="modal-dialog modal-sm">

<div class="modal-content">

<div class="modal-header">

<button type="button" class="close" data-dismiss="modal"><i class="fa fa-window-close"></i></button>

<h4 class="modal-title">Confirm</h4>

</div>

<div class="modal-body">

<p>Do you want to confirm the order?</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-primary" id="btnYesConfirm">Yes</button>

<button type="button" class="btn btn-success" id="btnNoConfirm">No</button>

</div>

</div>

</div>

</div>

<div id="deleteDialog" class="modal fade">

<div class="modal-dialog modal-sm">

<div class="modal-content">

<div class="modal-header">

<button type="button" class="close" data-dismiss="modal"><i class="fa fa-window-close"></i></button>

<h4 class="modal-title">Delete</h4>

</div>

<div class="modal-body">

<p>Do you want to delete the product from order?</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-danger" id="btnYesDelete">Delete</button>

<button type="button" class="btn btn-success" id="btnNoDelete">No</button>

</div>

</div>

</div>

</div>

@section Scripts {

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

<script type="text/javascript">

$(document).ready(function () {

var id = 0;

$("#btnConfirm").click(function () {

$("#confirmDialog").modal('show');

return false;

});

$("#btnNoConfirm").click(function () {

$("#confirmDialog").modal('hide');

return false;

});

$("#btnYesConfirm").click(function () {

window.location.href = '/Orders/ConfirmOrder';

});

$('a[id\*=btnDeleteItem]').click(function () {

debugger;

id = $(this).parent()[0].id;

$("#deleteDialog").modal('show');

return false;

});

$("#btnNoDelete").click(function () {

$("#deleteDialog").modal('hide');

return false;

});

$("#btnYesDelete").click(function () {

window.location.href = '/Orders/DeleteItem/' + id;

});

});

</script>

}

1. Test it.

# Date Picker

1. Add to de package json file this line:

{

"version": "1.0.0",

"name": "asp.net",

"private": true,

"devDependencies": {

"font-awesome": "^4.7.0",

"bootstrap-datepicker": "^1.8.0"

}

}

1. Save the file and copy the bootstrap date picker into folder root node modules.
2. Add this lines to \_layout:

<environment include="Development">

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />

<link href="~/node\_modules/font-awesome/css/font-awesome.min.css" rel="stylesheet" />

<link rel="stylesheet" href="~/css/site.css" />

<link href="~/node\_modules/bootstrap-datepicker/dist/css/bootstrap-datepicker.min.css" rel="stylesheet" />

</environment>

….

<environment include="Development">

<script src="~/lib/jquery/dist/jquery.js"></script>

<script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>

<script src="~/node\_modules/bootstrap-datepicker/dist/js/bootstrap-datepicker.min.js"></script>

<script src="~/js/site.js" asp-append-version="true"></script>

</environment>

1. Add the view model:

using System;

using System.ComponentModel.DataAnnotations;

public class DeliverViewModel

{

public int Id { get; set; }

[Display(Name = "Delivery date")]

[DisplayFormat(DataFormatString = "{0:MM/dd/yyyy}", ApplyFormatInEditMode = true)]

public DateTime DeliveryDate { get; set; }

}

1. Add the method to interfaz and repository

Task DeliverOrder(DeliverViewModel model);

And the repository:

public async Task DeliverOrder(DeliverViewModel model)

{

var order = await this.context.Orders.FindAsync(model.Id);

if (order == null)

{

return;

}

order.DeliveryDate = model.DeliveryDate;

this.context.Orders.Update(order);

await this.context.SaveChangesAsync();

}

1. Add the method to interfaz and repository

Task<Order> GetOrdersAsync(int id);

And the repository:

public async Task<Order> GetOrdersAsync(int id)

{

return await this.context.Orders.FindAsync(id);

}

1. Add this method to the orders controller:

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

{

if (id == null)

{

return NotFound();

}

var order = await this.repository.GetOrdersAsync(id.Value);

if (order == null)

{

return NotFound();

}

var model = new DeliverViewModel

{

Id = order.Id,

DeliveryDate = DateTime.Today

};

return View(model);

}

[HttpPost]

public async Task<IActionResult> Deliver(DeliverViewModel model)

{

if (this.ModelState.IsValid)

{

await this.repository.DeliverOrder(model);

return this.RedirectToAction("Index");

}

return this.View();

}

1. Add the view:

@model Core4.Models.DeliverViewModel

@{

ViewData["Title"] = "Deliver";

}

<h2>Deliver</h2>

<h4>Order</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Deliver">

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

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

<div class="form-group">

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

<div class="input-group date" data-provide="datepicker">

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

<span class="input-group-addon">

<span class="glyphicon glyphicon-calendar"></span>

</span>

</div>

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

</div>

<div class="form-group">

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

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

</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. Modify the order index view:

<td id="@item.Id">

<a asp-action="Deliver" asp-route-id="@item.Id" class="btn btn-info" id="btnDeliver">Deliver</a>

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

</td>

1. Test it.
2. Modify the Create and Edit products in views:

<div class="form-group">

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

<div class="input-group date" data-provide="datepicker">

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

<span class="input-group-addon">

<span class="glyphicon glyphicon-calendar"></span>

</span>

</div>

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

</div>

<div class="form-group">

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

<div class="input-group date" data-provide="datepicker">

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

<span class="input-group-addon">

<span class="glyphicon glyphicon-calendar"></span>

</span>

</div>

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

</div>

1. Test it.

# Cascade Drop Down List

1. First add the new entities:

using System.ComponentModel.DataAnnotations;

public class City

{

public int Id { get; set; }

[Required]

[Display(Name = "City")]

[MaxLength(50)]

public string Name { get; set; }

}

And:

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

public class Country

{

public int Id { get; set; }

[Required]

[Display(Name = "Country")]

[MaxLength(50)]

public string Name { get; set; }

public ICollection<City> Cities { get; set; }

[Display(Name = "# Cities")]

public int NumberCities { get { return this.Cities == null ? 0 : this.Cities.Count; } }

}

1. And modify the user entity, adding this properties:

[MaxLength(100)]

public string Address { get; set; }

public int CityId { get; set; }

public City City { get; set; }

1. Add this lines to the data context:

public DbSet<Country> Countries { get; set; }

public DbSet<City> Cities { get; set; }

1. Save all and run this commands to update the database, it’s important delete the database for ensure that all users have the new fields:

dotnet ef database drop

dotnet ef migrations add CountriesAndCities

dotnet ef database update

Or you can run this commands in package manager console:

PM> drop-database

PM> add-migration CountriesAndCities

PM> update-database

1. Modify the seeder class:

await this.CheckRole("Admin");

await this.CheckRole("Customer");

if (!this.context.Countries.Any())

{

var cities = new List<City>();

cities.Add(new City { Name = "Medellín" });

cities.Add(new City { Name = "Bogotá" });

cities.Add(new City { Name = "Calí" });

this.context.Countries.Add(new Country

{

Cities = cities,

Name = "Colombia"

});

await this.context.SaveChangesAsync();

}

var user = await this.userManager.FindByEmailAsync("jzuluaga55@gmail.com");

if (user == null)

{

user = new User

{

FirstName = "Juan",

LastName = "Zuluaga",

Email = "jzuluaga55@gmail.com",

UserName = "jzuluaga55@gmail.com",

Address = "Calle Luna Calle Sol",

PhoneNumber = "350 634 2747",

CityId = this.context.Countries.FirstOrDefault().Cities.FirstOrDefault().Id,

City = this.context.Countries.FirstOrDefault().Cities.FirstOrDefault()

};

var result = await this.userManager.CreateAsync(user, "123456");

1. Add the new view:

using System.ComponentModel.DataAnnotations;

public class CityViewModel

{

public int CountryId { get; set; }

public int CityId { get; set; }

[Required]

[Display(Name = "City")]

[MaxLength(50)]

public string Name { get; set; }

}

1. Add this methods to the repository, obviously interfaz and implementation:

public async Task<IEnumerable<Country>> GetCountriesAsync()

{

return await this.context.Countries

.Include(c => c.Cities)

.OrderBy(c => c.Name)

.ToListAsync();

}

public async Task<Country> GetCountryAsync(int id)

{

return await this.context.Countries

.Include(c => c.Cities)

.Where(c => c.Id == id)

.FirstOrDefaultAsync();

}

public async Task AddCountryAsync(Country country)

{

this.context.Countries.Add(country);

await this.context.SaveChangesAsync();

}

public async Task UpdateCountryAsync(Country country)

{

this.context.Countries.Update(country);

await this.context.SaveChangesAsync();

}

public async Task RemoveCountryAsync(Country country)

{

this.context.Countries.Remove(country);

await this.context.SaveChangesAsync();

}

public async Task<City> GetCityAsync(int id)

{

return await this.context.Cities.FindAsync(id);

}

public async Task AddCity(CityViewModel model)

{

var country = await this.GetCountryAsync(model.CountryId);

if (country == null)

{

return;

}

country.Cities.Add(new City { Name = model.Name });

this.context.Countries.Update(country);

await this.context.SaveChangesAsync();

}

public async Task<int> UpdateCity(City city)

{

var country = await this.context.Countries.Where(c => c.Cities.Any(ci => ci.Id == city.Id)).FirstOrDefaultAsync();

if (country == null)

{

return 0;

}

this.context.Cities.Update(city);

await this.context.SaveChangesAsync();

return country.Id;

}

public async Task<int> DeleteCityAsync(City city)

{

var country = await this.context.Countries.Where(c => c.Cities.Any(ci => ci.Id == city.Id)).FirstOrDefaultAsync();

if (country == null)

{

return 0;

}

this.context.Cities.Remove(city);

await this.context.SaveChangesAsync();

return country.Id;

}

1. Add the countries controller:

using System.Threading.Tasks;

using Common.Models;

using Data;

using Microsoft.AspNetCore.Mvc;

using Models;

public class CountriesController : Controller

{

private readonly IRepository repository;

public CountriesController(IRepository repository)

{

this.repository = repository;

}

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

{

if (id == null)

{

return NotFound();

}

var city = await this.repository.GetCityAsync(id.Value);

if (city == null)

{

return NotFound();

}

var countryId = await this.repository.DeleteCityAsync(city);

return this.RedirectToAction($"Details/{countryId}");

}

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

{

if (id == null)

{

return NotFound();

}

var city = await this.repository.GetCityAsync(id.Value);

if (city == null)

{

return NotFound();

}

return View(city);

}

[HttpPost]

public async Task<IActionResult> EditCity(City city)

{

if (this.ModelState.IsValid)

{

var countryId = await this.repository.UpdateCity(city);

if (countryId != 0)

{

return this.RedirectToAction($"Details/{countryId}");

}

}

return this.View(city);

}

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

{

if (id == null)

{

return NotFound();

}

var country = await this.repository.GetCountryAsync(id.Value);

if (country == null)

{

return NotFound();

}

var model = new CityViewModel { CountryId = country.Id };

return View(model);

}

[HttpPost]

public async Task<IActionResult> AddCity(CityViewModel model)

{

if (this.ModelState.IsValid)

{

await this.repository.AddCity(model);

return this.RedirectToAction($"Details/{model.CountryId}");

}

return this.View(model);

}

public async Task<IActionResult> Index()

{

return View(await this.repository.GetCountriesAsync());

}

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

{

if (id == null)

{

return NotFound();

}

var country = await this.repository.GetCountryAsync(id.Value);

if (country == null)

{

return NotFound();

}

return View(country);

}

public IActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(Country country)

{

if (ModelState.IsValid)

{

await this.repository.AddCountryAsync(country);

return RedirectToAction(nameof(Index));

}

return View(country);

}

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

{

if (id == null)

{

return NotFound();

}

var country = await this.repository.GetCountryAsync(id.Value);

if (country == null)

{

return NotFound();

}

return View(country);

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(Country country)

{

if (ModelState.IsValid)

{

await this.repository.UpdateCountryAsync(country);

return RedirectToAction(nameof(Index));

}

return View(country);

}

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

{

if (id == null)

{

return NotFound();

}

var country = await this.repository.GetCountryAsync(id.Value);

if (country == null)

{

return NotFound();

}

await this.repository.RemoveCountryAsync(country);

return RedirectToAction(nameof(Index));

}

}

1. Add the corresponding Views:

**Index:**

@model IEnumerable<ShopPrep.Common.Models.Country>

@{

ViewData["Title"] = "Index";

}

<h2>Countries</h2>

<p>

<a asp-action="Create" class="btn btn-primary">Create New</a>

</p>

<table class="table">

<thead>

<tr>

<th>

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

</th>

<th>

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

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model)

{

<tr>

<td>

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

</td>

<td>

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

</td>

<td id="@item.Id">

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

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

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

</td>

</tr>

}

</tbody>

</table>

<div id="deleteDialog" class="modal fade">

<div class="modal-dialog modal-sm">

<div class="modal-content">

<div class="modal-header">

<button type="button" class="close" data-dismiss="modal"><i class="fa fa-window-close"></i></button>

<h4 class="modal-title">Delete</h4>

</div>

<div class="modal-body">

<p>Do you want to delete the country?</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-danger" id="btnYesDelete">Delete</button>

<button type="button" class="btn btn-success" id="btnNoDelete">No</button>

</div>

</div>

</div>

</div>

@section Scripts {

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

<script type="text/javascript">

$(document).ready(function () {

var id = 0;

$('a[id\*=btnDelete]').click(function () {

debugger;

id = $(this).parent()[0].id;

$("#deleteDialog").modal('show');

return false;

});

$("#btnNoDelete").click(function () {

$("#deleteDialog").modal('hide');

return false;

});

$("#btnYesDelete").click(function () {

window.location.href = '/Countries/Delete/' + id;

});

});

</script>

}

**Create:**

@model ShopPrep.Common.Models.Country

@{

ViewData["Title"] = "Create";

}

<h2>Create</h2>

<h4>Country</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="Create">

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

<div class="form-group">

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

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

</div>

</form>

</div>

</div>

@section Scripts {

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

}

**Edit:**

@model ShopPrep.Common.Models.Country

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<h4>Country</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="Id" />

<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>

<div class="form-group">

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

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

</div>

</form>

</div>

</div>

@section Scripts {

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

}

**Details:**

@model ShopPrep.Common.Models.Country

@{

ViewData["Title"] = "Details";

}

<h2>Details</h2>

<div>

<h4>Country</h4>

<hr />

<dl class="dl-horizontal">

<dt>

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

</dt>

<dd>

@Html.DisplayFor(model => model.Name)

</dd>

</dl>

</div>

<div>

<a asp-action="Edit" asp-route-id="@Model.Id" class="btn btn-warning">Edit</a>

<a asp-action="AddCity" asp-route-id="@Model.Id" class="btn btn-info">Add City</a>

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

</div>

<h4>Cities</h4>

@if (Model.Cities == null || Model.Cities.Count == 0)

{

<h5>No cities added yet</h5>

}

else

{

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Cities.FirstOrDefault().Name)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Cities.OrderBy(c => c.Name))

{

<tr>

<td>

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

</td>

<td id="@item.Id">

<a asp-action="EditCity" asp-route-id="@item.Id" class="btn btn-warning">Edit</a>

<a asp-action="DeleteCity" asp-route-id="@item.Id" class="btn btn-danger" id="btnDelete">Delete</a>

</td>

</tr>

}

</tbody>

</table>

}

<div id="deleteDialog" class="modal fade">

<div class="modal-dialog modal-sm">

<div class="modal-content">

<div class="modal-header">

<button type="button" class="close" data-dismiss="modal"><i class="fa fa-window-close"></i></button>

<h4 class="modal-title">Delete</h4>

</div>

<div class="modal-body">

<p>Do you want to delete the city?</p>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-danger" id="btnYesDelete">Delete</button>

<button type="button" class="btn btn-success" id="btnNoDelete">No</button>

</div>

</div>

</div>

</div>

@section Scripts {

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

<script type="text/javascript">

$(document).ready(function () {

var id = 0;

$('a[id\*=btnDelete]').click(function () {

debugger;

id = $(this).parent()[0].id;

$("#deleteDialog").modal('show');

return false;

});

$("#btnNoDelete").click(function () {

$("#deleteDialog").modal('hide');

return false;

});

$("#btnYesDelete").click(function () {

window.location.href = '/Countries/DeleteCity/' + id;

});

});

</script>

}

**Add city:**

@model ShopPrep.Web.Models.CityViewModel

<h4>City</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="AddCity">

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

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

<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>

<div class="form-group">

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

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

</div>

</form>

</div>

</div>

@section Scripts {

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

}

**Edit city:**

@model ShopPrep.Common.Models.City

<h4>City</h4>

<hr />

<div class="row">

<div class="col-md-4">

<form asp-action="EditCity">

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

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

<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>

<div class="form-group">

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

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

</div>

</form>

</div>

</div>

@section Scripts {

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

}

1. Add to the new menu for countries:

@if (this.User.Identity.IsAuthenticated)

{

@if (this.User.IsInRole("Admin"))

{

<li><a asp-area="" asp-controller="Products" asp-action="Index">Products</a></li>

<li><a asp-area="" asp-controller="Countries" asp-action="Index">Countries</a></li>

}

<li><a asp-area="" asp-controller="Orders" asp-action="Index">Orders</a></li>

}

1. Test it and add some countries and cities.
2. Modify the **RegisterNewUserViewModel**:

[Required]

[Compare("Password")]

public string Confirm { get; set; }

[MaxLength(100)]

public string Address { get; set; }

[MaxLength(20)]

public string PhoneNumber { get; set; }

[Display(Name = "City")]

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

public int CityId { get; set; }

public IEnumerable<SelectListItem> Cities { get; set; }

[Display(Name = "Country")]

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

public int CountryId { get; set; }

public IEnumerable<SelectListItem> Countries { get; set; }

1. Add this methods to the repository (and interfaz too):

public IEnumerable<SelectListItem> GetComboCountries()

{

var list = this.context.Countries.Select(c => new SelectListItem

{

Text = c.Name,

Value = c.Id.ToString()

}).OrderBy(l => l.Text).ToList();

list.Insert(0, new SelectListItem

{

Text = "(Select a country...)",

Value = "0"

});

return list;

}

public IEnumerable<SelectListItem> GetComboCities(int conuntryId)

{

var country = this.context.Countries.Find(conuntryId);

var list = new List<SelectListItem>();

if (country != null)

{

list = country.Cities.Select(c => new SelectListItem

{

Text = c.Name,

Value = c.Id.ToString()

}).OrderBy(l => l.Text).ToList();

}

list.Insert(0, new SelectListItem

{

Text = "(Select a city...)",

Value = "0"

});

return list;

}

public async Task<Country> GetCountryAsync(City city)

{

return await this.context.Countries.Where(c => c.Cities.Any(ci => ci.Id == city.Id)).FirstOrDefaultAsync();

}

1. Change the register method in account controller:

public IActionResult Register()

{

var model = new RegisterNewUserViewModel

{

Countries = this.repository.GetComboCountries(),

Cities = this.repository.GetComboCities(0)

};

return this.View(model);

}

[HttpPost]

public async Task<IActionResult> Register(RegisterNewUserViewModel model)

{

if (this.ModelState.IsValid)

{

var user = await this.userManager.FindByEmailAsync(model.Username);

if (user == null)

{

var city = await this.repository.GetCityAsync(model.CityId);

user = new User

{

FirstName = model.FirstName,

LastName = model.LastName,

Email = model.Username,

UserName = model.Username,

Address = model.Address,

PhoneNumber = model.PhoneNumber,

CityId = model.CityId,

City = city

};

var result = await this.userManager.CreateAsync(user, model.Password);

if (result != IdentityResult.Success)

{

this.ModelState.AddModelError(string.Empty, "The user couldn't be created.");

return this.View(model);

}

var result2 = await this.signInManager.PasswordSignInAsync(

model.Username,

model.Password,

true,

false);

if (result2.Succeeded)

{

await this.userManager.AddToRoleAsync(user, "Customer");

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

}

this.ModelState.AddModelError(string.Empty, "The user couldn't be login.");

return this.View(model);

}

this.ModelState.AddModelError(string.Empty, "The username is already registered.");

}

return this.View(model);

}

1. Modify the register view with the new fields:

<div class="form-group">

<label asp-for="FirstName">First Name</label>

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

<span asp-validation-for="FirstName" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="LastName">Last Name</label>

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

<span asp-validation-for="LastName" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="Username">Username</label>

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

<span asp-validation-for="Username" class="text-warning"></span>

</div>

<div class="form-group">

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

<select asp-for="CountryId" asp-items="Model.Countries" class="form-control"></select>

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

</div>

<div class="form-group">

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

<select asp-for="CityId" asp-items="Model.Cities" class="form-control"></select>

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

</div>

<div class="form-group">

<label asp-for="Address">Address</label>

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

<span asp-validation-for="Address" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="PhoneNumber">Phone Number</label>

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

<span asp-validation-for="PhoneNumber" class="text-warning"></span>

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

<label asp-for="Confirm">Confirm</label>

<input asp-for="Confirm" type="password" class="form-control" />

<span asp-validation-for="Confirm" class="text-warning"></span>

</div>

1. Test the code until this point.
2. Now implement the cascade drop down list.
3. Add this method to account controller:

public async Task<JsonResult> GetCities(int countryId)

{

var country = await this.repository.GetCountryAsync(countryId);

return this.Json(country.Cities.OrderBy(c => c.Name));

}

1. And modify the register view:

@model Core4.Models.RegisterNewUserViewModel

@{

ViewData["Title"] = "Register";

}

<h2>Register New User</h2>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div asp-validation-summary="ModelOnly"></div>

<div class="form-group">

<label asp-for="FirstName">First Name</label>

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

<span asp-validation-for="FirstName" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="LastName">Last Name</label>

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

<span asp-validation-for="LastName" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="Username">Username</label>

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

<span asp-validation-for="Username" class="text-warning"></span>

</div>

<div class="form-group">

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

<select asp-for="CountryId" asp-items="Model.Countries" class="form-control"></select>

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

</div>

<div class="form-group">

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

<select asp-for="CityId" asp-items="Model.Cities" class="form-control"></select>

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

</div>

<div class="form-group">

<label asp-for="Address">Address</label>

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

<span asp-validation-for="Address" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="PhoneNumber">Phone Number</label>

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

<span asp-validation-for="PhoneNumber" class="text-warning"></span>

</div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

<label asp-for="Confirm">Confirm</label>

<input asp-for="Confirm" type="password" class="form-control" />

<span asp-validation-for="Confirm" class="text-warning"></span>

</div>

<div class="form-group">

<input type="submit" value="Register New User" class="btn btn-primary" />

</div>

</form>

</div>

</div>

@section Scripts {

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

<script type="text/javascript">

$(document).ready(function () {

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

$("#CityId").empty();

$.ajax({

type: 'POST',

url: '@Url.Action("GetCities")',

dataType: 'json',

data: { countryId: $("#CountryId").val() },

success: function (cities) {

debugger;

$("#CityId").append('<option value="0">(Select a city...)</option>');

$.each(cities, function (i, city) {

$("#CityId").append('<option value="'

+ city.id + '">'

+ city.name + '</option>');

});

},

error: function (ex) {

alert('Failed to retrieve cities.' + ex);

}

});

return false;

})

});

</script>

}

1. Test it.
2. Now we’ll continue with the user modification. Please modify the model **ChangeUserViewModel**:

[Required]

[Display(Name = "Last Name")]

public string LastName { get; set; }

[MaxLength(100)]

public string Address { get; set; }

[MaxLength(20)]

public string PhoneNumber { get; set; }

[Display(Name = "City")]

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

public int CityId { get; set; }

public IEnumerable<SelectListItem> Cities { get; set; }

[Display(Name = "Country")]

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

public int CountryId { get; set; }

public IEnumerable<SelectListItem> Countries { get; set; }

1. Modify the change user method in account controller:

public async Task<IActionResult> ChangeUser()

{

var user = await this.userManager.FindByEmailAsync(this.User.Identity.Name);

var model = new ChangeUserViewModel();

if (user != null)

{

model.FirstName = user.FirstName;

model.LastName = user.LastName;

model.Address = user.Address;

model.PhoneNumber = user.PhoneNumber;

var city = await this.repository.GetCityAsync(user.CityId);

if (city != null)

{

var country = await this.repository.GetCountryAsync(city);

if (country != null)

{

model.CountryId = country.Id;

model.Cities = this.repository.GetComboCities(country.Id);

model.Countries = this.repository.GetComboCountries();

model.CityId = user.CityId;

}

}

}

model.Cities = this.repository.GetComboCities(model.CountryId);

model.Countries = this.repository.GetComboCountries();

return this.View(model);

}

[HttpPost]

public async Task<IActionResult> ChangeUser(ChangeUserViewModel model)

{

if (this.ModelState.IsValid)

{

var user = await this.userManager.FindByEmailAsync(this.User.Identity.Name);

if (user != null)

{

var city = await this.repository.GetCityAsync(model.CityId);

user.FirstName = model.FirstName;

user.LastName = model.LastName;

user.Address = model.Address;

user.PhoneNumber = model.PhoneNumber;

user.CityId = model.CityId;

user.City = city;

var respose = await this.userManager.UpdateAsync(user);

if (respose.Succeeded)

{

this.ViewBag.UserMessage = "User updated!";

}

else

{

this.ModelState.AddModelError(string.Empty, respose.Errors.FirstOrDefault().Description);

}

}

else

{

this.ModelState.AddModelError(string.Empty, "User no found.");

}

}

return this.View(model);

}

1. Modify the view:

@model ShopPrep.Web.Models.ChangeUserViewModel

@{

ViewData["Title"] = "Register";

}

<h2>Update User</h2>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div asp-validation-summary="ModelOnly"></div>

<div class="form-group">

<label asp-for="FirstName">First Name</label>

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

<span asp-validation-for="FirstName" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="LastName">Last Name</label>

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

<span asp-validation-for="LastName" class="text-warning"></span>

</div>

<div class="form-group">

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

<select asp-for="CountryId" asp-items="Model.Countries" class="form-control"></select>

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

</div>

<div class="form-group">

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

<select asp-for="CityId" asp-items="Model.Cities" class="form-control"></select>

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

</div>

<div class="form-group">

<label asp-for="Address">Address</label>

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

<span asp-validation-for="Address" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="PhoneNumber">Phone Number</label>

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

<span asp-validation-for="PhoneNumber" class="text-warning"></span>

</div>

<div class="form-group">

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

<a asp-action="ChangePassword" class="btn btn-success">Change Password</a>

</div>

<div class="text-success">@ViewBag.UserMessage</div>

</form>

</div>

</div>

@section Scripts {

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

<script type="text/javascript">

$(document).ready(function () {

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

$("#CityId").empty();

$.ajax({

type: 'POST',

url: '@Url.Action("GetCities")',

dataType: 'json',

data: { countryId: $("#CountryId").val() },

success: function (cities) {

debugger;

$("#CityId").append('<option value="0">(Select a city...)</option>');

$.each(cities, function (i, city) {

$("#CityId").append('<option value="'

+ city.id + '">'

+ city.name + '</option>');

});

},

error: function (ex) {

alert('Failed to retrieve cities.' + ex);

}

});

return false;

})

});

</script>

}

1. Test it.

# Confirm Email Registration

1. First, change the setup project:

services.AddIdentity<User, IdentityRole>(cfg =>

{

cfg.Tokens.AuthenticatorTokenProvider = TokenOptions.DefaultAuthenticatorProvider;

cfg.SignIn.RequireConfirmedEmail = true;

cfg.User.RequireUniqueEmail = true;

cfg.Password.RequireDigit = false;

cfg.Password.RequiredUniqueChars = 0;

cfg.Password.RequireLowercase = false;

cfg.Password.RequireNonAlphanumeric = false;

cfg.Password.RequireUppercase = false;

})

.AddDefaultTokenProviders()

.AddEntityFrameworkStores<DataContext>();

1. Check if your email account is enabled to send email in: <https://myaccount.google.com/lesssecureapps>
2. Add this parameters to the configuration file:

"Mail": {

"From": "youremail@gmail.com",

"Smtp": "smtp.gmail.com",

"Port": 587,

"Password": "yourpassword"

}

1. Add the nuget “**Mailkit**”.
2. In **Helpers** folder add the interface **IMailHelper**:

public interface IMailHelper

{

void SendMail(string to, string subject, string body);

}

1. In **Helpers** folder add the implementation **MailHelper**:

using MailKit.Net.Smtp;

using Microsoft.Extensions.Configuration;

using MimeKit;

public class MailHelper : IMailHelper

{

private readonly IConfiguration configuration;

public MailHelper(IConfiguration configuration)

{

this.configuration = configuration;

}

public void SendMail(string to, string subject, string body)

{

var from = this.configuration["Mail:From"];

var smtp = this.configuration["Mail:Smtp"];

var port = this.configuration["Mail:Port"];

var password = this.configuration["Mail:Password"];

var message = new MimeMessage();

message.From.Add(new MailboxAddress(from));

message.To.Add(new MailboxAddress(to));

message.Subject = subject;

var bodyBuilder = new BodyBuilder();

bodyBuilder.HtmlBody = body;

message.Body = bodyBuilder.ToMessageBody();

using (var client = new SmtpClient())

{

client.Connect(smtp, int.Parse(port), false);

client.Authenticate(from, password);

client.Send(message);

client.Disconnect(true);

}

}

}

1. Configure the injection for the new interface:

services.AddScoped<IUserHelper, UserHelper>();

services.AddScoped<IMailHelper, MailHelper>();

1. Modify the register post method:

[HttpPost]

public async Task<IActionResult> Register(RegisterNewUserViewModel model)

{

if (this.ModelState.IsValid)

{

var user = await this.userManager.FindByEmailAsync(model.Username);

if (user == null)

{

var city = await this.repository.GetCityAsync(model.CityId);

user = new User

{

FirstName = model.FirstName,

LastName = model.LastName,

Email = model.Username,

UserName = model.Username,

Address = model.Address,

PhoneNumber = model.PhoneNumber,

CityId = model.CityId,

City = city

};

var result = await this.userManager.CreateAsync(user, model.Password);

if (result != IdentityResult.Success)

{

this.ModelState.AddModelError(string.Empty, "The user couldn't be created.");

return this.View(model);

}

var myToken = await this.userManager.GenerateEmailConfirmationTokenAsync(user);

var tokenLink = this.Url.Action("ConfirmEmail", "Account", new

{

userid = user.Id,

token = myToken

}, protocol: HttpContext.Request.Scheme);

var mailSender = new MailHelper(configuration);

mailSender.SendMail(model.Username, "Email confirmation", $"<h1>Email Confirmation</h1>" +

$"To allow the user, " +

$"plase click in this link:</br></br><a href = \"{tokenLink}\">Confirm Email</a>");

this.ViewBag.Message = "The instructions to allow your user has been sent to email.";

return this.View(model);

}

this.ModelState.AddModelError(string.Empty, "The username is already registered.");

}

return this.View(model);

}

1. Modify the register view:

...

<div class="form-group">

<input type="submit" value="Register New User" class="btn btn-primary" />

</div>

</form>

</div>

</div>

<div class="text-success">

<p>

@ViewBag.Message

</p>

</div>

@section Scripts {

...

1. Create the method confirm email in account controller:

public async Task<IActionResult> ConfirmEmail(string userId, string token)

{

if (string.IsNullOrEmpty(userId) || string.IsNullOrEmpty(token))

{

return this.NotFound();

}

var user = await this.userManager.FindByIdAsync(userId);

if (user == null)

{

return this.NotFound();

}

var result = await this.userManager.ConfirmEmailAsync(user, token);

if (!result.Succeeded)

{

return this.NotFound();

}

return View();

}

1. Create the view:

@{

ViewData["Title"] = "Confirm email";

}

<h2>@ViewData["Title"]</h2>

<div>

<p>

Thank you for confirming your email.

</p>

</div>

1. Drop the database (PM> drop-database) to ensure that all the users have a confirmed email.
2. Modify the seed class:

var result = await this.userManager.CreateAsync(user, "123456");

if (result != IdentityResult.Success)

{

throw new InvalidOperationException("Could not create the user in seeder");

}

await this.userManager.AddToRoleAsync(user, "Admin");

var token = await this.userManager.GenerateEmailConfirmationTokenAsync(user);

await this.userManager.ConfirmEmailAsync(user, token);

}

var isInRole = await this.userManager.IsInRoleAsync(user, "Admin");

1. Test it.

# Password Recovery

1. Modify the login view:

<div class="form-group">

<input type="submit" value="Login" class="btn btn-success" />

<a asp-action="Register" class="btn btn-primary">Register New User</a>

<a asp-action="RecoverPassword" class="btn btn-link">Forgot your password?</a>

</div>

1. Add the model:

using System.ComponentModel.DataAnnotations;

public class RecoverPasswordViewModel

{

[Required]

[EmailAddress]

public string Email { get; set; }

}

1. Add the model:

using System.ComponentModel.DataAnnotations;

public class ResetPasswordViewModel

{

[Required]

public string UserName { get; set; }

[Required]

[DataType(DataType.Password)]

public string Password { get; set; }

[Required]

[DataType(DataType.Password)]

[Compare("Password")]

public string ConfirmPassword { get; set; }

[Required]

public string Token { get; set; }

}

1. Add this methods to account controller:

public IActionResult RecoverPassword()

{

return this.View();

}

[HttpPost]

public async Task<IActionResult> RecoverPassword(RecoverPasswordViewModel model)

{

if (this.ModelState.IsValid)

{

var user = await this.userManager.FindByEmailAsync(model.Email);

if (user == null)

{

ModelState.AddModelError(string.Empty, "The email doesn't correspont to a registered user.");

return this.View(model);

}

var myToken = await this.userManager.GeneratePasswordResetTokenAsync(user);

var link = this.Url.Action("ResetPassword", "Account", new { token = myToken }, protocol: HttpContext.Request.Scheme);

var mailSender = new MailHelper(configuration);

mailSender.SendMail(model.Email, "Password Reset", $"<h1>Recover Password</h1>" +

$"To reset the password click in this link:</br></br>" +

$"<a href = \"{link}\">Reset Password</a>");

this.ViewBag.Message = "The instructions to recover your password has been sent to email.";

return this.View();

}

return this.View(model);

}

public IActionResult ResetPassword(string token)

{

return View();

}

[HttpPost]

public async Task<IActionResult> ResetPassword(ResetPasswordViewModel model)

{

var user = await this.userManager.FindByNameAsync(model.UserName);

if (user != null)

{

var result = await this.userManager.ResetPasswordAsync(user, model.Token, model.Password);

if (result.Succeeded)

{

this.ViewBag.Message = "Password reset successful.";

return this.View();

}

this.ViewBag.Message = "Error while resetting the password.";

return View(model);

}

this.ViewBag.Message = "User not found.";

return View(model);

}

1. Add the view:

@model ShopPrep.Web.Models.RecoverPasswordViewModel

@{

ViewData["Title"] = "Recover Password";

}

<h2>Recover Password</h2>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div asp-validation-summary="ModelOnly"></div>

<div class="form-group">

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

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

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

</div>

<div class="form-group">

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

<a asp-action="Login" class="btn btn-success">Back to login</a>

</div>

</form>

<div class="text-success">

<p>

@ViewBag.Message

</p>

</div>

</div>

</div>

@section Scripts {

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

}

1. Add the view:

@model ShopPrep.Web.Models.ResetPasswordViewModel

@{

ViewData["Title"] = "Reset Password";

}

<h1>Reset Your Password</h1>

<div class="row">

<div class="col-md-4 offset-md-4">

<form method="post">

<div asp-validation-summary="All"></div>

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

<div class="form-group">

<label asp-for="UserName">Email</label>

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

<span asp-validation-for="UserName" class="text-warning"></span>

</div>

<div class="form-group">

<label asp-for="Password">New password</label>

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

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

</div>

<div class="form-group">

<label asp-for="ConfirmPassword">Confirm</label>

<input asp-for="ConfirmPassword" type="password" class="form-control" />

<span asp-validation-for="ConfirmPassword" class="text-warning"></span>

</div>

<div class="form-group">

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

</div>

</form>

<div class="text-success">

<p>

@ViewBag.Message

</p>

</div>

</div>

</div>

@section Scripts {

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

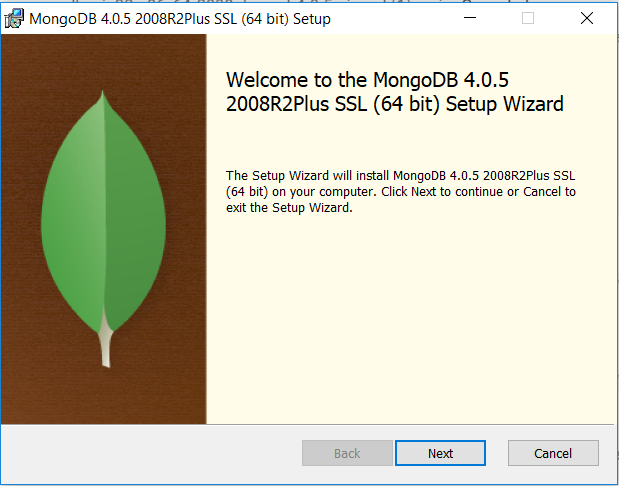
}

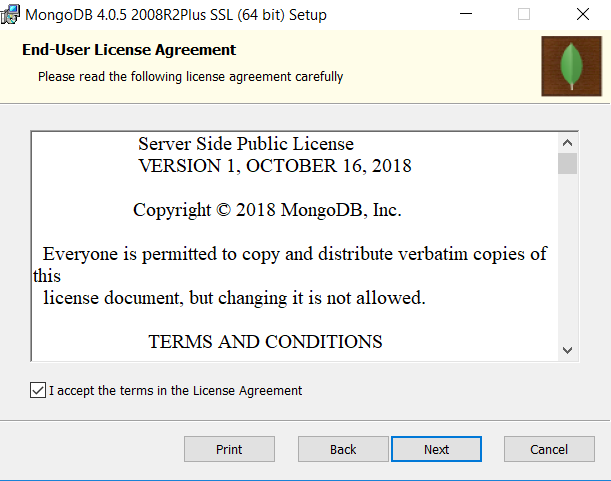
1. Test it.

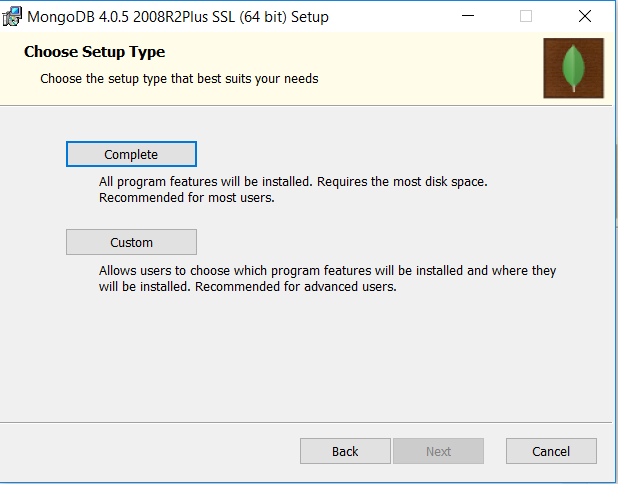
# Installing MongoDB as a Service.

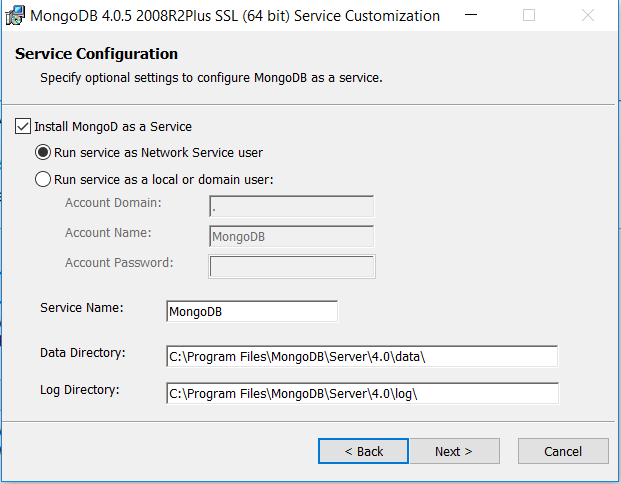
Go to the following web page: <https://www.mongodb.com/download-center/community> and download the MongoDB installer.

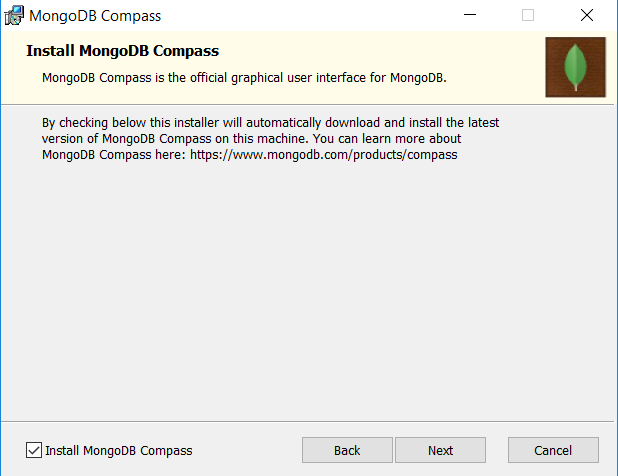
Open up the installer and continue as we will show you here below:

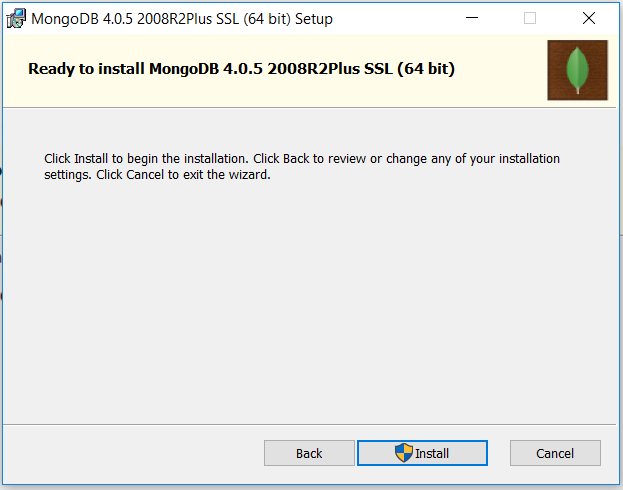






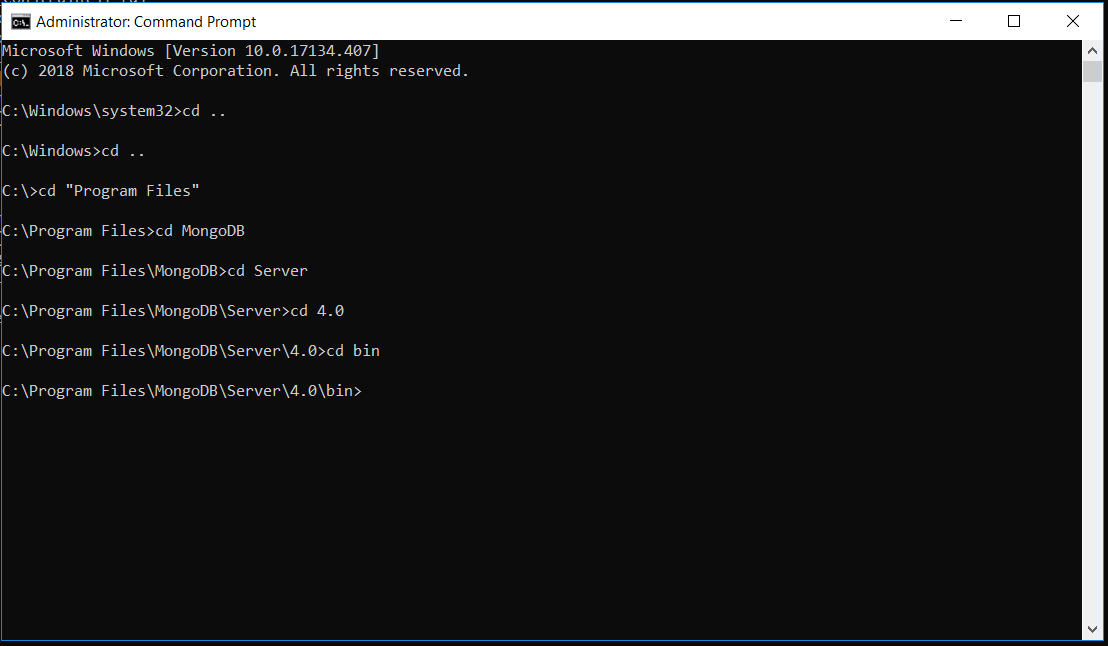






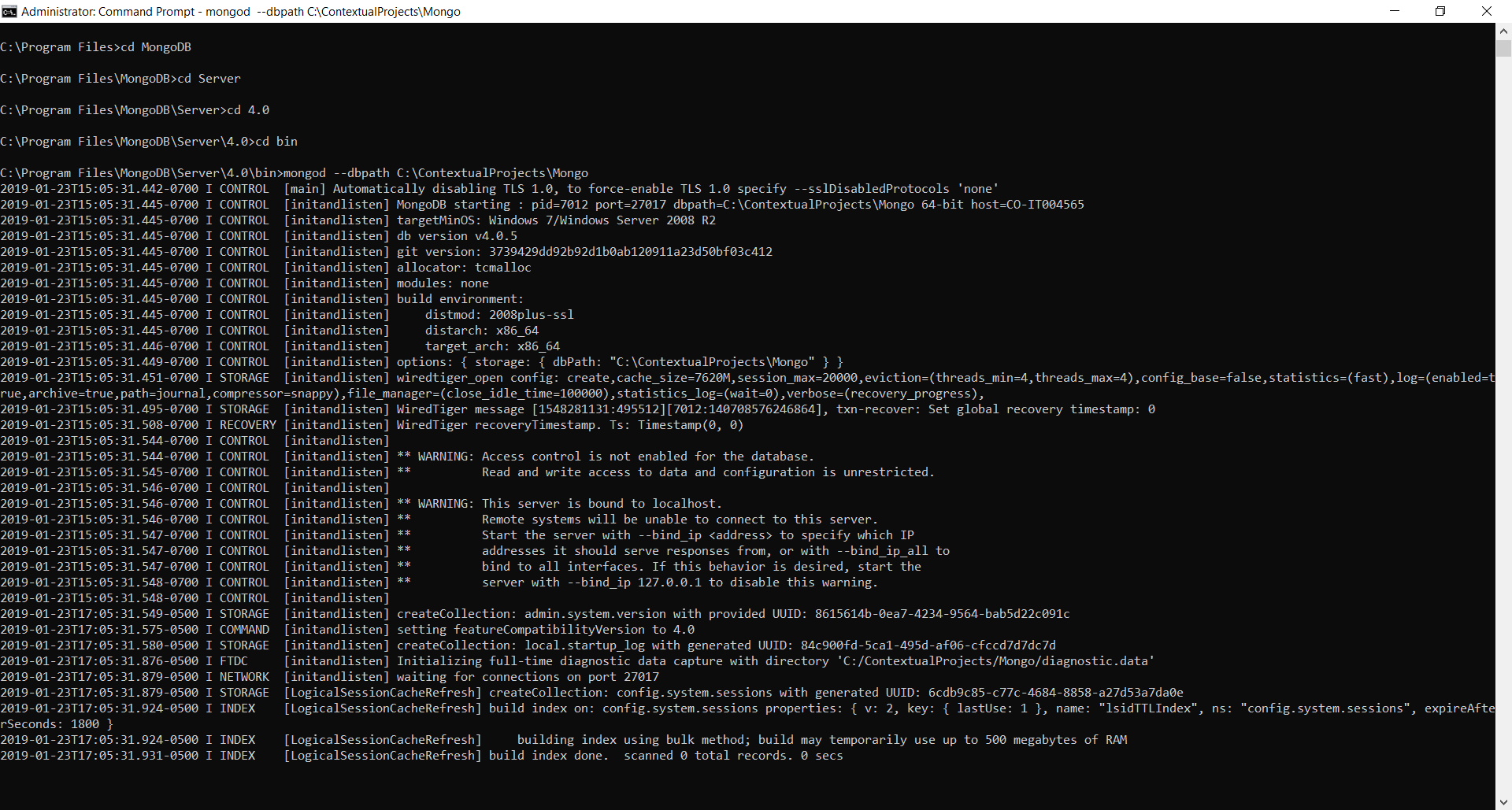
when the installer has finished clic on Finish button.

Open up the command prompt then typing the following path: C:\Program Files\MongoDB\Server\4.0\bin, as you can see in the below pic



then you have to type the following command: mongod --dbpath C:\ContextualProjects\Mongo but before, you must create the folders

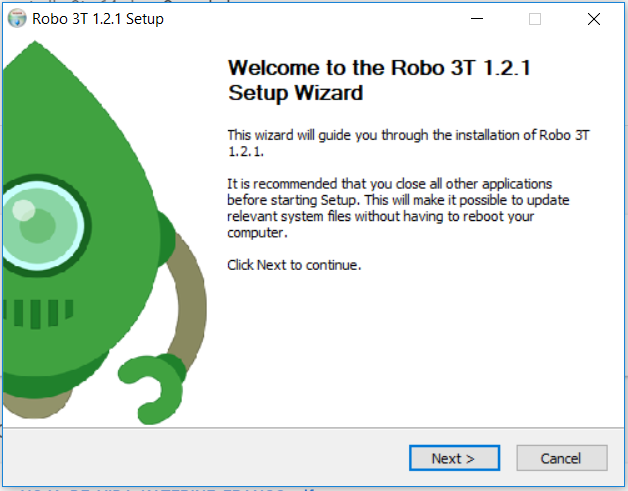
ContextualProjects and Mongo.

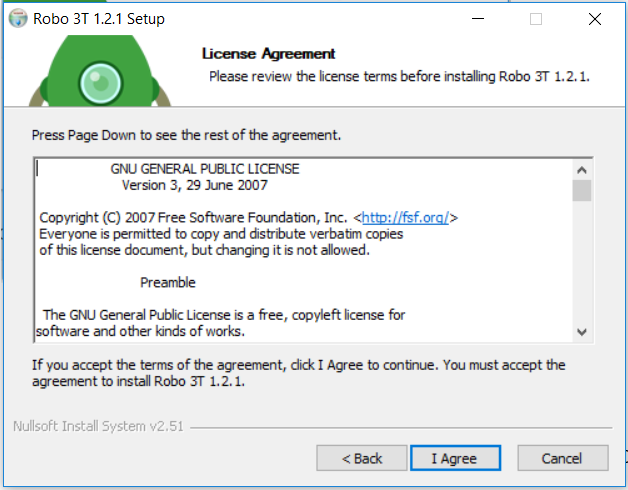


Right now, you can see waiting for connections on port 27017, this is the default port in MongoDB.

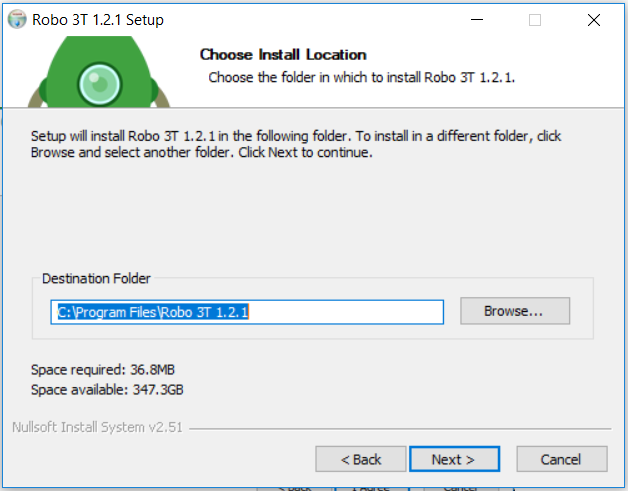
Install Robo 3T as MongoDB Client.

Go to the following website: <https://robomongo.org/download> and download Robo 3T, and run the Robo 3T installer.

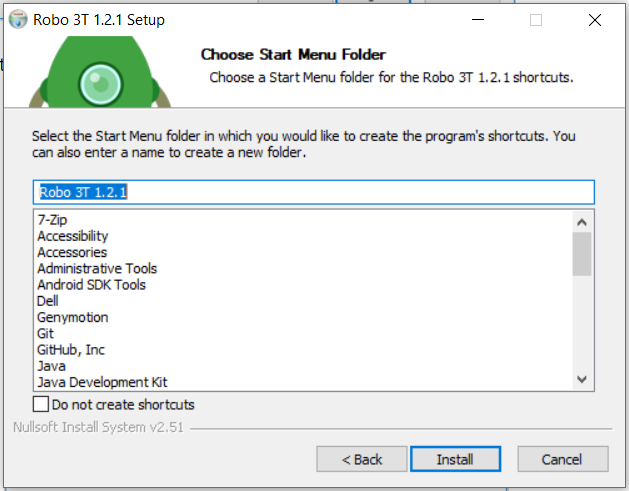




You can leave the default path as you can see in the below pic

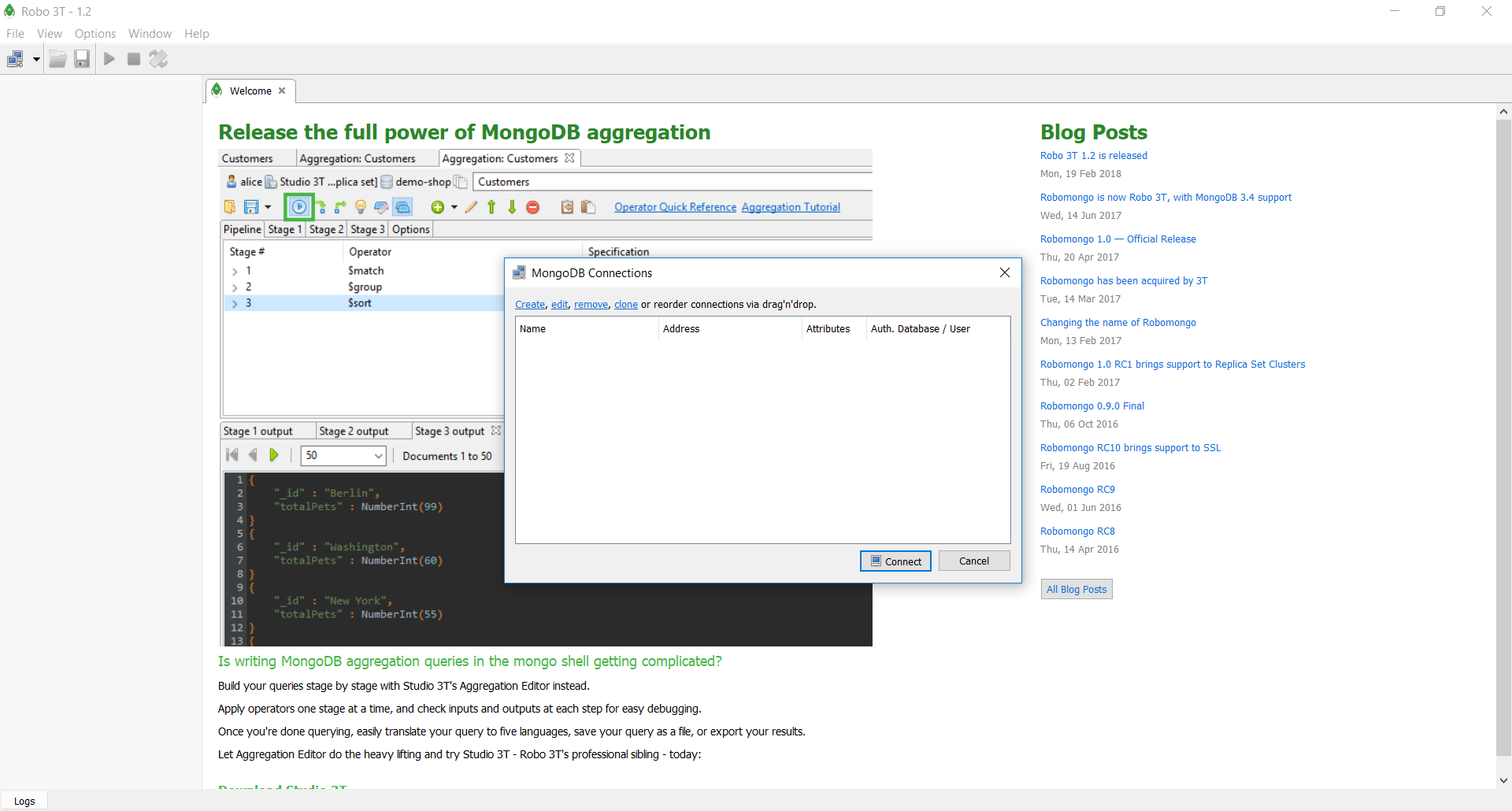


Click on install



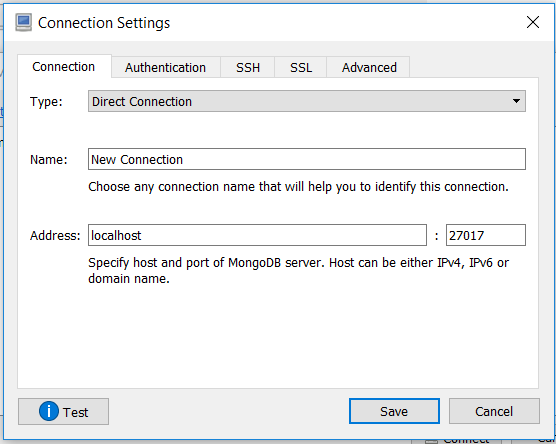
Finally click on finish

You can see the Robo 3T client open up.



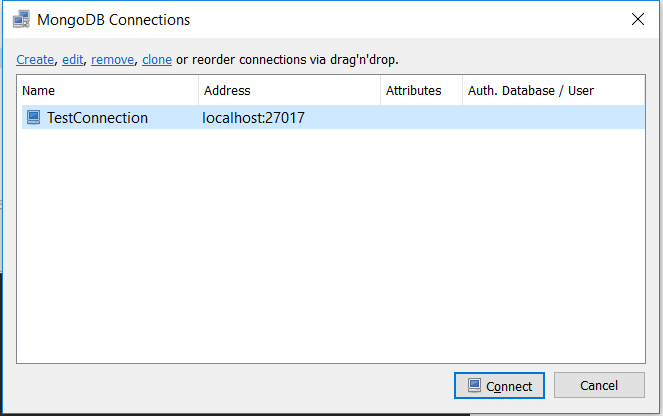
On MongoDB Connections popup we are not going to see a record.

Now you should click on create



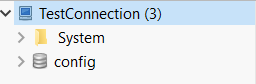
In the Connection Settings fields on Name you must add a new name for example: TestConnection on Address by default, then Click on save.

Now we are going to see this:



and click on Connect.

Now we are going to see all databases better than console view.



Note: In order to connect Robo 3T client with MongoDB service, you always have to execute the following command mongod --dbpath C:\ContextualProjects\Mongo at the root of mongo installation and then run Robo 3T Client.

# Creating Asp .net Core with MongoDB.

We are going to create a new .net Core Project then we are going to add MongoDB.Driver package.

On the root of the project we are going to create two folders, the first one will be called Data and inside of the Data folder will be called Entities.