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

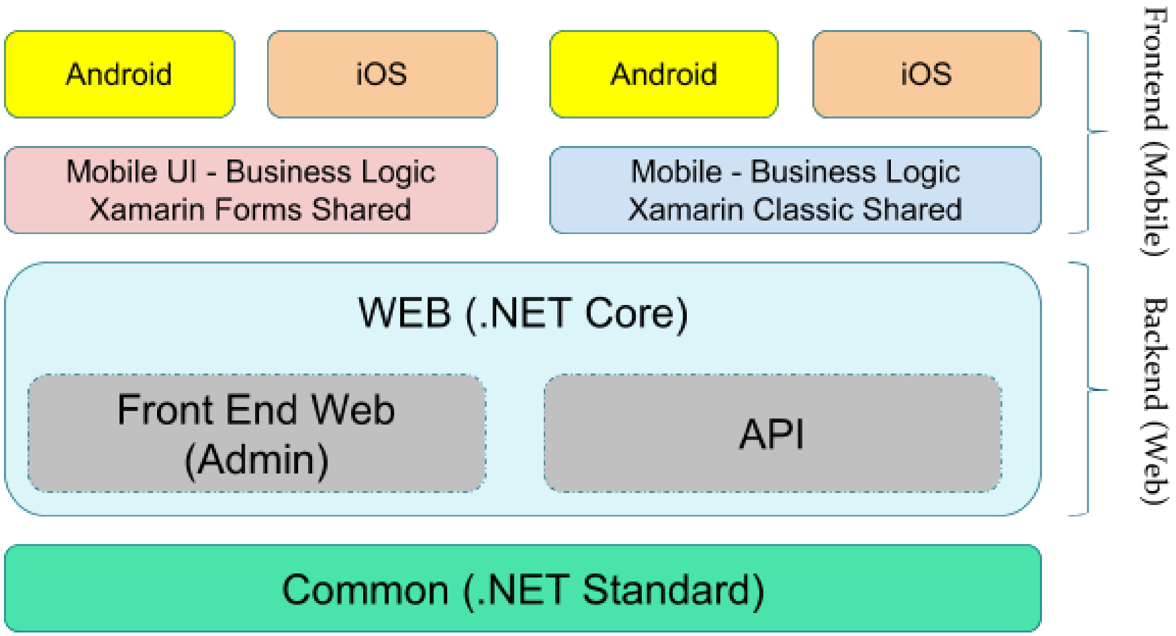
By Zulu
Medellín 2019

Index

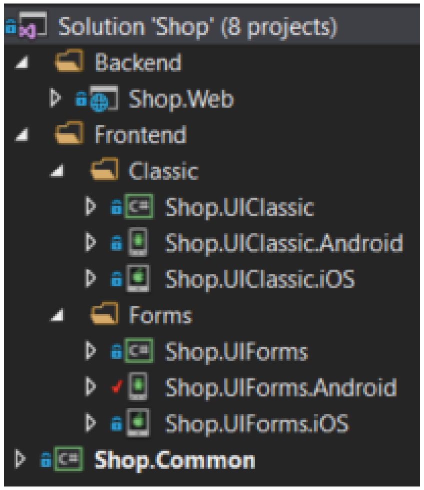
Create the Solution

Note: all the code are in: https://github.com/Zulu55/Shop

Create the following solution:



In Visual Studio, you must build something similar to:



Create the Database

Note: in this project we'll work with entity framework code first, if you want to work with EF database first, I recommend this article: https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/existing-db

1. Create the entities (in folder Web.Data.Entities):

```
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; }
}
```

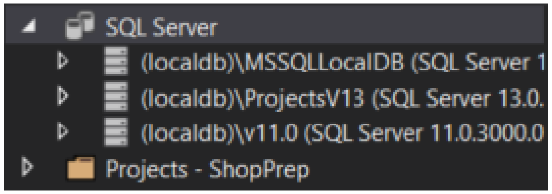
2. Create the context class (in folder Data):

```
using Common.Models;
using Microsoft.EntityFrameworkCore;

public class DataContext : DbContext
{
        public DbSet<Product> Products { get; set; }

        public DataContext(DbContextOptions<DataContext> options) : base(options)
        {
        }
}
```

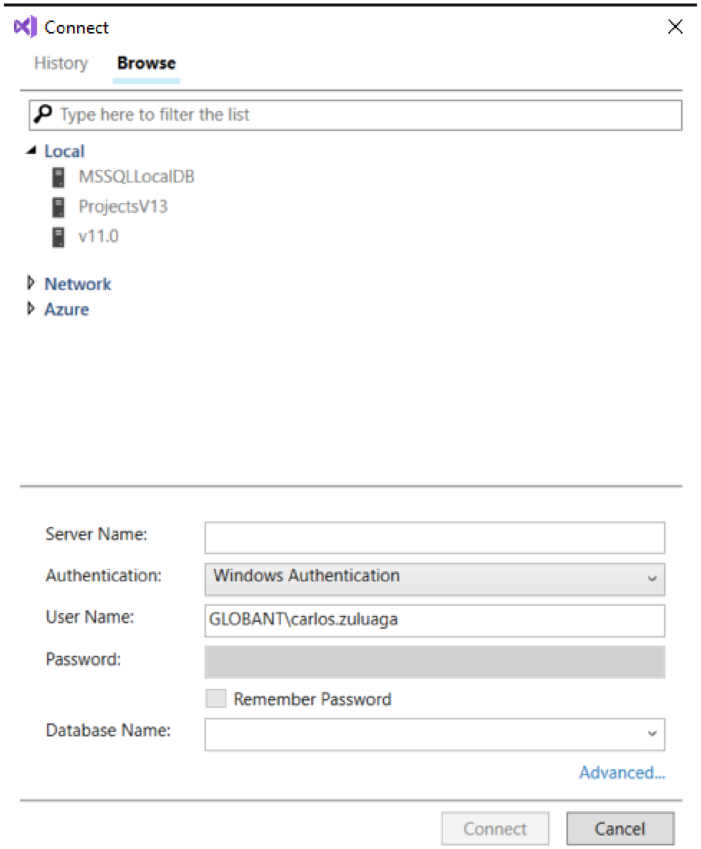
3. Add the connection string to the configuration json file (see the SQL Server Object Explorer):

```
{
  "Logging": {
          "LogLevel": {
          "Default": "Warning"
          }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
          "DefaultConnection":
"Server=(localdb)\\ProjectsV13;Database=Shop;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

Note: You must be sure of the servers names in your installation, you can check it out, by clicking in SQL Server Object Explorer:


SQL Server
(localdb)\MSSQLLocalDB (SQL Server 1
(localdb)\ProjectsV13 (SQL Server 13.0.
(localdb)\v11.0 (SQL Server 11.0.3000.0
Projects - ShopPrep

In this case, there are three available servers: **(localdb)\MSSQLLocalDB**, **(localdb)\ProjectsV13** and **(localdb)\v11.0**. Or you can explore your server by clicking on “Add SQL Server” icon:



Server Name:

Authentication: Windows Authentication

User Name: GLOBANT\carlos.zuluaga

Password:

□ Remember Password

Database Name:

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

```
services.AddDbContext<DataContext>(cfg =>
{
   cfg.UseSqlServer(this.Configuration.GetConnectionString("DefaultConnection"));
});
```

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

```
dotnet ef database update
dotnet ef migrations add InitialDb
dotnet ef database update
```

Or you can run this commands in package manager console:

```
PM> update-database
PM> add-migration InitialDb
PM> update-database
```

6. Add the products controller.

7. 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, ErrorMessage = "The field {0} only can contain a maximum {1} characters")]
        [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; }
}
```

2. Run this commands:

```
dotnet ef migrations add ModifyProducts
dotnet ef database update
```

Or you can run this commands in package manager console:

```
PM> add-migration ModifyProducts
PM> update-database
```

3. 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)
    });
    }
}
```

2. 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>();
}
```

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

```
services.AddTransient<SeedDb>();
```

4. 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);
    }
}
```

2. 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);
}
```

3. 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));
    }
}
```

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

```
services.AddScoped<IRepository, Repository>();
```

5. 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; }
}
```

2. 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)
   {
   }
}
```

3. Make the relations with other models:

```
public User User { get; set; }
```

4. 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 those commands in package manager console:

```
PM> drop-database
PM> add-migration Users
PM> update-database
```

5. 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
    });
    }
}
```

6. 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?}");
          });
}
```

7. Test it.

Implement A Generic Repository & Some Fixes

(Tanks to Fabian Camargo https://www.youtube.com/user/fabiancv90)

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

```
using System.Threading.Tasks;
using Data.Entities;
using Microsoft.AspNetCore.Identity;

public interface IUserHelper
{
          Task<User> GetUserByEmailAsync(string email);

          Task<IdentityResult> AddUserAsync(User user, string password);
}
```

2. In the same folder add the implementation (**UserHelper**):

```
using System.Threading.Tasks;
using Data.Entities;
using Microsoft.AspNetCore.Identity;

public class UserHelper : IUserHelper
{
        private readonly UserManager<User> userManager;

        public UserHelper(UserManager<User> userManager)
        {
        this.userManager = userManager;
        }

        public async Task<IdentityResult> AddUserAsync(User user, string password)
        {
        return await this.userManager.CreateAsync(user, password);
        }

        public async Task<User> GetUserByEmailAsync(string email)
        {
        var user = await this.userManager.FindByEmailAsync(email);
        return user;
        }
}
```

3. In **Web.Data.Entities** add the interface **IEntity**:

```
public interface IEntity
{
        int Id { get; set; }
}
```

4. Modify the **Products** entity:

```
public class Product : IEntity
```

5. In **Data** add the interfaz **IGenericRepository**:

```
using System.Linq;
using System.Threading.Tasks;

public interface IGenericRepository<T> where T : class
{
	IQueryable<T> GetAll();

	Task<T> GetByIdAsync(int id);

	Task CreateAsync(T entity);

	Task UpdateAsync(T entity);

	Task DeleteAsync(T entity);

	Task<bool> ExistAsync(int id);
}
```

6. In the same folder add the implementation (**GenericRepository**):

```
using System.Linq;
using System.Threading.Tasks;
using Entities;
using Microsoft.EntityFrameworkCore;
```

```
public class GenericRepository<T> : IGenericRepository<T> where T : class, IEntity
{
    private readonly DataContext context;

    public GenericRepository(DataContext context)
    {
    this.context = context;
    }

    public IQueryable<T> GetAll()
    {
    return this.context.Set<T>().AsNoTracking();
    }

    public async Task<T> GetByIdAsync(int id)
    {
    return await this.context.Set<T>()
    .AsNoTracking()
    .FirstOrDefaultAsync(e => e.Id == id);
    }

    public async Task CreateAsync(T entity)
    {
    await this.context.Set<T>().AddAsync(entity);
    await SaveAllAsync();
    }

    public async Task UpdateAsync(T entity)
    {
    this.context.Set<T>().Update(entity);
    await SaveAllAsync();
```

```
        }

        public async Task DeleteAsync(T entity)
        {
        this.context.Set<T>().Remove(entity);
        await SaveAllAsync();
        }

        public async Task<bool> ExistAsync(int id)
        {
        return await this.context.Set<T>().AnyAsync(e => e.Id == id);

        }

        public async Task<bool> SaveAllAsync()
        {
        return await this.context.SaveChangesAsync() > 0;
        }
}
```

7. Add the **IProductRepository**:

```
using Entities;

public interface IProductRepository : IGenericRepository<Product>
{
}
```

8. Add the **ProductRepository**:

```
using Entities;
```

```
public class ProductRepository : GenericRepository<Product>, IProductRepository
{
	public ProductRepository(DataContext context) : base(context)
	{
	}
}
```

9. Delete the previous repository (**Repository** and **IRepository**).

10. Modify the **Startup**:

```
services.AddScoped<IProductRepository, ProductRepository>();
```

11. Modify the **ProductsController**:

```
using System.Threading.Tasks;
using Data;
using Data.Entities;
using Helpers;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

public class ProductsController : Controller
{
	private readonly IProductRepository productRepository;

	private readonly IUserHelper userHelper;

	public ProductsController(IProductRepository productRepository, IUserHelper userHelper)
	{
```

```
this.productRepository = productRepository;
this.userHelper = userHelper;
}

// GET: Products
public IActionResult Index()
{
return View(this.productRepository.GetAll());
}

// GET: Products/Details/5
public async Task<IActionResult> Details(int? id)
{
if (id == null)
{
return NotFound();
}

var product = await this.productRepository.GetByIdAsync(id.Value);
if (product == null)
{
return NotFound();
}

return View(product);
}

// GET: Products/Create
public IActionResult Create()
{
return View();
```

```
}

// POST: Products/Create
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(Product product)
{
if (ModelState.IsValid)
{
// TODO: Pending to change to: this.User.Identity.Name
product.User = await this.userHelper.GetUserByEmailAsync("jzuluaga55@gmail.com");
await this.productRepository.CreateAsync(product);
return RedirectToAction(nameof(Index));
}

return View(product);
}

// GET: Products/Edit/5
public async Task<IActionResult> Edit(int? id)
{
if (id == null)
{
return NotFound();
}

var product = await this.productRepository.GetByIdAsync(id.Value);
if (product == null)
{
return NotFound();
}
```

```
return View(product);
}

// POST: Products/Edit/5
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(Product product)
{
if (ModelState.IsValid)
{
try
{
        // TODO: Pending to change to: this.User.Identity.Name
        product.User = await this.userHelper.GetUserByEmailAsync("jzuluaga55@gmail.com");
        await this.productRepository.UpdateAsync(product);
}
catch (DbUpdateConcurrencyException)
{
        if (!await this.productRepository.ExistAsync(product.Id))
        {
        return NotFound();
        }
        else
        {
        throw;
        }
}
return RedirectToAction(nameof(Index));
}
```

```
    return View(product);
    }

    // GET: Products/Delete/5
    public async Task<IActionResult> Delete(int? id)
    {
    if (id == null)
    {
    return NotFound();
    }

    var product = await this.productRepository.GetByIdAsync(id.Value);
    if (product == null)
    {
    return NotFound();
    }

    return View(product);
    }

    // POST: Products/Delete/5
    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
    var product = await this.productRepository.GetByIdAsync(id);
    await this.productRepository.DeleteAsync(product);
    return RedirectToAction(nameof(Index));
    }
}
```

12. Modify the **SeedDb**:

```
using System;
using System.Linq;
using System.Threading.Tasks;
using Entities;
using Microsoft.AspNetCore.Identity;
using Shop.Web.Helpers;

public class SeedDb
{
    private readonly DataContext context;
    private readonly IUserHelper userHelper;
    private Random random;

    public SeedDb(DataContext context, IUserHelper userHelper)
    {
    this.context = context;
    this.userHelper = userHelper;
    this.random = new Random();
    }

    public async Task SeedAsync()
    {
    await this.context.Database.EnsureCreatedAsync();

    // Add user
    var user = await this.userHelper.GetUserByEmail("jzuluaga55@gmail.com");
    if (user == null)
    {
    user = new User
```

```
{
        FirstName = "Juan",
        LastName = "Zuluaga",
        Email = "jzuluaga55@gmail.com",
        UserName = "jzuluaga55@gmail.com",
        PhoneNumber = "3506342747"
};

var result = await this.userHelper.AddUser(user, "123456");
if (result != IdentityResult.Success)
{
        throw new InvalidOperationException("Could not create the user in seeder");
}
}

// Add products
if (!this.context.Products.Any())
{
this.AddProduct("iPhone X", user);
this.AddProduct("Magic Mouse", user);
this.AddProduct("iWatch Series 4", user);
await this.context.SaveChangesAsync();
}
}

private void AddProduct(string name, User user)
{
this.context.Products.Add(new Product
{
Name = name,
Price = this.random.Next(1000),
            IsAvailabe = true,
            Stock = this.random.Next(100),
            User = user
            });
            }
}
```

13. Test it.

14. Now to take advance the this implementation, we’ll create another entity that we’ll use nearly. Add the entity **Country**:

```
using System.ComponentModel.DataAnnotations;

public class Country : IEntity
{
            public int Id { get; set; }

            [MaxLength(50, ErrorMessage = "The field {0} only can contain {1} characters length.")]
            [Required]
            [Display(Name = "Country")]
            public string Name { get; set; }
}
```

15. Add the interface for countries:

```
using Entities;

public interface ICountryRepository : IGenericRepository<Country>
{
}
```

16. And add the implementation:

```
using Entities;

public class CountryRepository : GenericRepository<Country>, ICountryRepository
{
	public CountryRepository(DataContext context) : base(context)
	{
	}
}
```

17. Add the injection in **StartUp**:

```
services.AddScoped<ICountryRepository, CountryRepository>();
```

18. Add the property in the **DataContext**.

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

19. Save all and run those commands to update the database:

```
dotnet ef migrations add Countries
dotnet ef database update
```

Or you can run this commands in package manager console:

```
PM> add-migration Countries
PM> update-database
```

20. Run the App and test it.

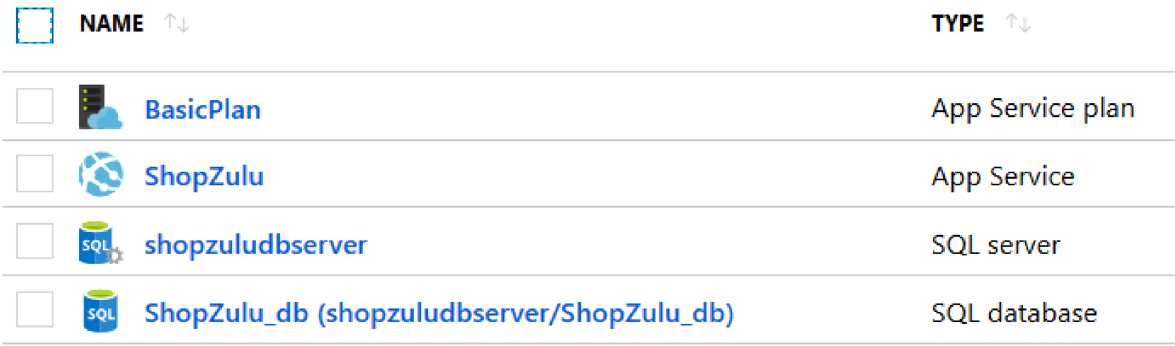
Add API

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

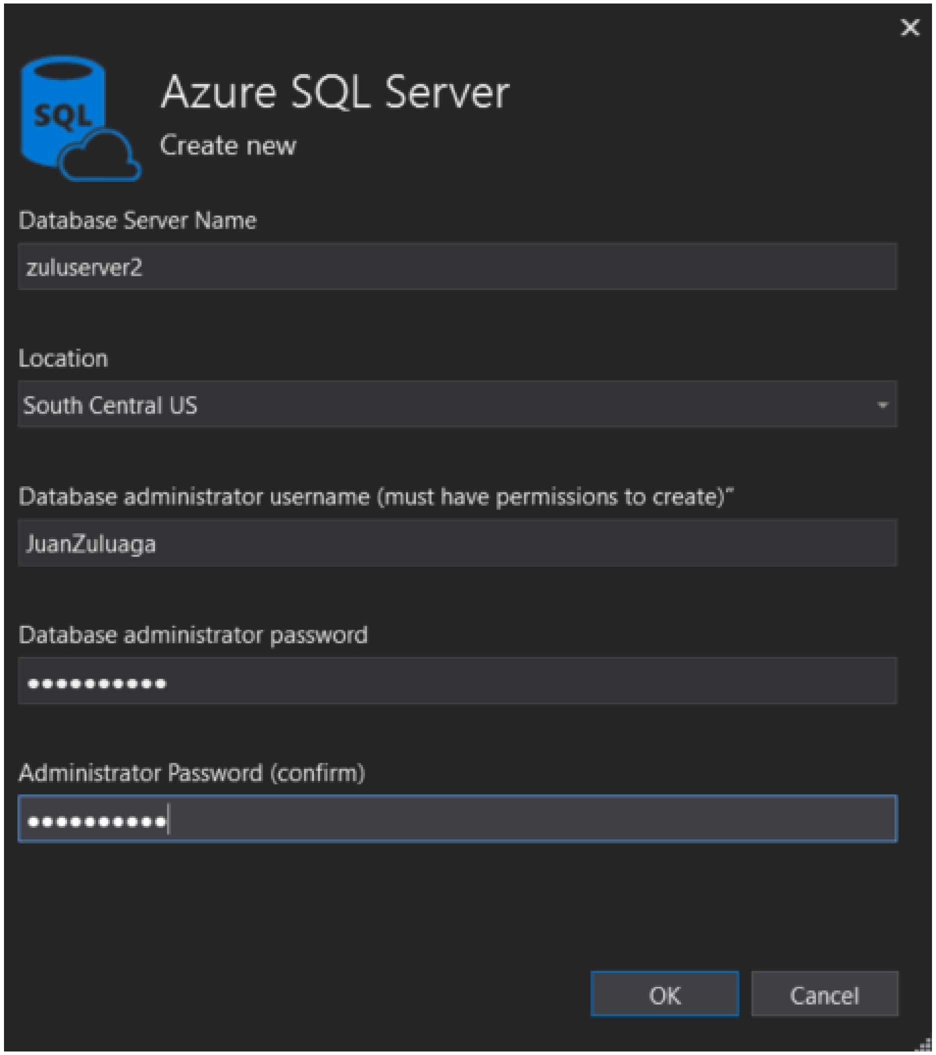
```
using Data;
using Microsoft.AspNetCore.Mvc;

[Route("api/[Controller]")]
public class ProductsController : Controller
{
        private readonly IProductRepository productRepository;

        public ProductsController(IProductRepository productRepository)
        {
        this.productRepository = productRepository;
        }

        [HttpGet]
        public IActionResult GetProducts()
        {
        return this.Ok(this.productRepository.GetAll());
        }
}
```

2. Test it.

3. Publish the App in Azure.



□	NAME ↑↓	TYPE ↑↓
□	BasicPlan	App Service plan
□	ShopZulu	App Service
□	shopzuludbserver	SQL server
□	ShopZulu_db (shopzuludbserver/ShopZulu_db)	SQL database



Azure SQL Server

Create new

Database Server Name

zuluserver2

Location

South Central US

Database administrator username (must have permissions to create)"

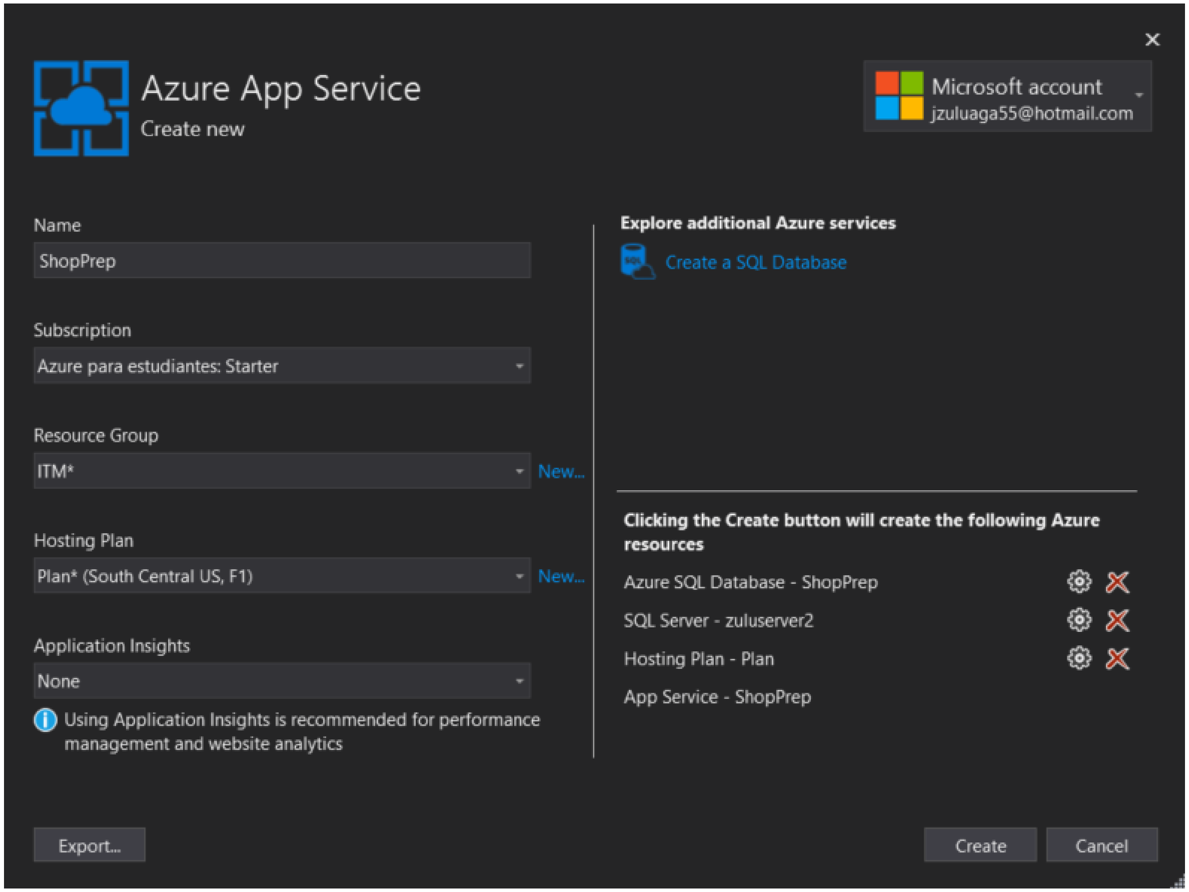
JuanZuluaga

Database administrator password

●●●●●●●●●●●

Administrator Password (confirm)

●●●●●●●●●●●



Name: ShopPrep

Subscription: Azure para estudiantes: Starter

Resource Group: ITM*

Hosting Plan: Plan* (South Central US, F1)

Application Insights: None

Using Application Insights is recommended for performance management and website analytics

Explore additional Azure services

Create a SQL Database

Clicking the Create button will create the following Azure resources

Azure SQL Database - ShopPrep

SQL Server - zuluserver2

Hosting Plan - Plan

App Service - ShopPrep

Adding Images

1. In Web the folder **Models** and the class **MainViewModel**.

```
using System.ComponentModel.DataAnnotations;
using Data.Entities;
using Microsoft.AspNetCore.Http;

public class ProductViewModel : Product
{
        [Display(Name = "Image")]
        public IFormFile ImageFile { get; set; }
}
```

2. Modify the **Create** products view:

```
@model Shop.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">
…
```

3. Add the folder **Products** into **wwwroot/images**.

4. Modify the method **Create** POST and the class **ProductsController**:

```
[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}";
    }

    // TODO: Pending to change to: this.User.Identity.Name
    view.User = await this.userHelper.GetUserByEmail("jzuluaga55@gmail.com");
    var product = this.ToProduct(view, path);
    await this.productRepository.CreateAsync(product);
    return RedirectToAction(nameof(Index));
    }

    return View(view);
}

private 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 = view.User
```

```
        };
}
```

5. Modify the products index view:

```
<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>
```

6. Test it what we do until the moment.

7. Now modify the GET and POST Edit in **ProductsController**.

```
// GET: Products/Edit/5
public async Task<IActionResult> Edit(int? id)
{
        if (id == null)
        {
        return NotFound();
        }

        var product = await this.productRepository.GetByIdAsync(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
    };
}

// POST: Products/Edit/5
[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}";
    }

    // TODO: Pending to change to: this.User.Identity.Name
    view.User = await this.userHelper.GetUserByEmail("jzuluaga55@gmail.com");
    var product = this.ToProduct(view, path);
    await this.productRepository.UpdateAsync(product);
    }
    catch (DbUpdateConcurrencyException)
    {
    if (!await this.productRepository.ExistAsync(view.Id))
    {
        return NotFound();
    }
    else
    {
        throw;
    }
    }
    return RedirectToAction(nameof(Index));
    }
    return View(view);
}
```

8. Modify the edit product view model:

```
@model Shop.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" />

        <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-primary" />
                <a asp-action="Index" class="btn btn-success">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");}
}
```

9. Test it.

10. Modify the details product view model:

```
<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>
```

11. Modify the delete product view model:

```
<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>
```

12. Test it.

.

13. Finally add this property to Product entity:

```
public string ImageFullPath
{
        get
        {
        if (string.IsNullOrEmpty(this.ImageUrl))
        {
        return null;
        }

        return $"https://shopzulu.azurewebsites.net{this.ImageUrl.Substring(1)}";
        }
}
```

14. Ant test the API and publish the Changes in Azure.

Adding Other Methods To Generic Repository

1. Modify the **IProductRepository**.

```
using Entities;
using System.Linq;

public interface IProductRepository : IGenericRepository<Product>
{
        IQueryable GetAllWithUsers();
}
```

2. Modify the **ProductRepository**.

```
using System.Linq;
using Entities;
using Microsoft.EntityFrameworkCore;

public class ProductRepository : GenericRepository<Product>, IProductRepository
{
        private readonly DataContext context;

        public ProductRepository(DataContext context) : base(context)
        {
        this.context = context;
        }

        public IQueryable GetAllWithUsers()
        {
```

```
        return this.context.Products.Include(p => p.User).OrderBy(p => p.Name);
        }
}
```

3. Modify the product API Controller.

```
public IQueryable GetAllWithUsers()
{
        return this.context.Products.Include(p => p.User).OrderBy(p => p.Name);
}
```

4. Test it.

Starting with Xamarin Forms

1. Create the folder **ViewModels** and inside it add the class **ProductViewModel**.

```
public class MainViewModel
{
}
```

2. Create the folder **Infrastructure** and inside it add the class **InstanceLocator**.

```
public class InstanceLocator
{
        public MainViewModel Main { get; set; }

        public InstanceLocator()
        {
        this.Main = new MainViewModel();
        }
}
```

3. Modify the **App.xaml** to add an application dictionary:

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:infra="clr-namespace:ShopPrep.UIForms.Infrastructure"
        x:Class="ShopPrep.UIForms.App">
        <Application.Resources>
        <ResourceDictionary>
        <!-- Locator -->
        <infra:InstanceLocator x:Key="Locator"/>
        </ResourceDictionary>
        </Application.Resources>
</Application>
```

4. Add the folder **Views** and inside it, create the **LoginPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="Shop.UIForms.Views.LoginPage"
        BindingContext="{Binding Main, Source={StaticResource Locator}}"
        Title="Login">
        <ContentPage.Content>
        <ScrollView
        BindingContext="{Binding Login}">
        <StackLayout
                Padding="5">
                <Label
                Text="Email">
                </Label>
                <Entry
                Keyboard="Email"
                Placeholder="Enter your email..."
                Text="{Binding Email}">
                </Entry>
                <Label
                Text="Password">
                </Label>
                <Entry
                IsPassword="True"
                Placeholder="Enter your password..."
                Text="{Binding Password}">
                </Entry>
                <Button
                Command="{Binding LoginCommand}"
                Text="Login">
                </Button>
        </StackLayout>
        </ScrollView>
        </ContentPage.Content>
</ContentPage>
```

5. Add the NuGet **MvvmLigthLibsStd10**. (Plaase search as: **Mvvm Ligth Libs Std**)

6. In ViewModels add the class **LoginViewModel**:

```
using System.Windows.Input;
using GalaSoft.MvvmLight.Command;
using Xamarin.Forms;

public class LoginViewModel
{
    public string Email { get; set; }

    public string Password { get; set; }

    public ICommand LoginCommand => new RelayCommand(this.Login);

    private async void Login()
    {
    if (string.IsNullOrEmpty(this.Email))
    {
    await Application.Current.MainPage.DisplayAlert("Error", "You must enter an email", "Accept");
    return;
    }

    if (string.IsNullOrEmpty(this.Password))
    {
    await Application.Current.MainPage.DisplayAlert("Error", "You must enter a password", "Accept");
    return;
    }

    if (!this.Email.Equals("jzuluaga55@gmail.com") || !this.Password.Equals("123456"))
    {
    await Application.Current.MainPage.DisplayAlert("Error", "Incorrect user or password", "Accept");
    return;
    }

    await Application.Current.MainPage.DisplayAlert("Ok", "Fuck yeah!!!", "Accept");
```

```
        }
}
```

7. Modify the **MainViewModel**:

```
public class MainViewModel
{
        public LoginViewModel Login { get; set; }

        public MainViewModel()
        {
        this.Login = new LoginViewModel();
        }
}
```

8. Modify the **App.xaml.cs**:

```
using Views;
using Xamarin.Forms;

public partial class App : Application
{
        public App()
        {
        InitializeComponent();

        this.MainPage = new NavigationPage(new LoginPage());
        }

        protected override void OnStart()
        {
```

```
        // Handle when your app starts
        }

        protected override void OnSleep()
        {
        // Handle when your app sleeps
        }

        protected override void OnResume()
        {
        // Handle when your app resumes
        }
}
```

9. Test it.

Fix Bug to Don’t Replace Images

1. Modify the MVC **ProductsController** in Create and Edit:

```
if (view.ImageFile != null && view.ImageFile.Length > 0)
{
        var guid = Guid.NewGuid().ToString();
        var file = $"{guid}.jpg";

        path = Path.Combine(
        Directory.GetCurrentDirectory(),
        "wwwroot\\images\\Products",
        file);
```

```
    using (var stream = new FileStream(path, FileMode.Create))
    {
    await view.ImageFile.CopyToAsync(stream);
    }

    path = $"~/images/Products/{file}";
}
```

2. Test it.

Consuming RestFull

1. Add the NuGet **Newtonsoft.Json** to project **Commond**.

2. Add the folder **Models** and inside it those classes (I recommend use the http://json2csharp.com/ page):

```
using System;
using Newtonsoft.Json;

public class User
{
    [JsonProperty("firstName")]
    public string FirstName { get; set; }

    [JsonProperty("lastName")]
    public string LastName { get; set; }

    [JsonProperty("id")]
    public Guid Id { get; set; }
```

```
[JsonProperty("userName")]
public string UserName { get; set; }

[JsonProperty("normalizedUserName")]
public string NormalizedUserName { get; set; }

[JsonProperty("email")]
public string Email { get; set; }

[JsonProperty("normalizedEmail")]
public string NormalizedEmail { get; set; }

[JsonProperty("emailConfirmed")]
public bool EmailConfirmed { get; set; }

[JsonProperty("passwordHash")]
public string PasswordHash { get; set; }

[JsonProperty("securityStamp")]
public string SecurityStamp { get; set; }

[JsonProperty("concurrencyStamp")]
public Guid ConcurrencyStamp { get; set; }

[JsonProperty("phoneNumber")]
public string PhoneNumber { get; set; }

[JsonProperty("phoneNumberConfirmed")]
public bool PhoneNumberConfirmed { get; set; }

[JsonProperty("twoFactorEnabled")]
```

```
    public bool TwoFactorEnabled { get; set; }

    [JsonProperty("lockoutEnd")]
    public object LockoutEnd { get; set; }

    [JsonProperty("lockoutEnabled")]
    public bool LockoutEnabled { get; set; }

    [JsonProperty("accessFailedCount")]
    public long AccessFailedCount { get; set; }
}
```

And:

```
using Newtonsoft.Json;
using System;

public class Product
{
    [JsonProperty("id")]
    public int Id { get; set; }

    [JsonProperty("name")]
    public string Name { get; set; }

    [JsonProperty("price")]
    public decimal Price { get; set; }

    [JsonProperty("imageUrl")]
    public string ImageUrl { get; set; }
    [JsonProperty("lastPurchase")]
    public DateTime LastPurchase { get; set; }

    [JsonProperty("lastSale")]
    public DateTime LastSale { get; set; }

    [JsonProperty("isAvailabe")]
    public bool IsAvailabe { get; set; }

    [JsonProperty("stock")]
    public double Stock { get; set; }

    [JsonProperty("user")]
    public User User { get; set; }

    [JsonProperty("imageFullPath")]
    public Uri ImageFullPath { get; set; }
}
```

3. Add the **Response** model.

```
public class Response
{
    public bool IsSuccess { get; set; }

    public string Message { get; set; }

    public object Result { get; set; }
}
```

4. In Common project add the folder **Services** and inside it add the class **ApiService**.

```
using System;
using System.Collections.Generic;
using System.Net.Http;
using Models;
using Newtonsoft.Json;
using System.Threading.Tasks;

public class ApiService
{
    public async Task<Response> GetListAsync<T>(string urlBase, string servicePrefix, string controller)
    {
    try
    {
    var client = new HttpClient
    {
        BaseAddress = new Uri(urlBase)
    };
    var url = $"{servicePrefix}{controller}";
    var response = await client.GetAsync(url);
    var result = await response.Content.ReadAsStringAsync();

    if (!response.IsSuccessStatusCode)
    {
        return new Response
        {
        IsSuccess = false,
        Message = result,
        };
    }
```

```
        var list = JsonConvert.DeserializeObject<List<T>>(result);
        return new Response
        {
                IsSuccess = true,
                Result = list
        };
        }
        catch (Exception ex)
        {
        return new Response
        {
                IsSuccess = false,
                Message = ex.Message
        };
        }
        }
}
```

5. Add the **ProductsPage**.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="Shop.UIForms.Views.ProductsPage"
        BindingContext="{Binding Main, Source={StaticResource Locator}}"
        Title="Products">
        <ContentPage.Content>
        <StackLayout
        BindingContext="{Binding Products}"
        Padding="5">
        <ListView
```

```
HasUnevenRows="True"
ItemsSource="{Binding Products}">
<ListView.ItemTemplate>
<DataTemplate>
<ViewCell>
        <Grid>
        <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <Image
        Grid.Column="0"
        Source="{Binding ImageFullPath}"
        WidthRequest="100">
        </Image>
        <StackLayout
        Grid.Column="1"
        VerticalOptions="Center">
        <Label
                FontAttributes="Bold"
                FontSize="Medium"
                Text="{Binding Name}"
                TextColor="Black">
        </Label>
        <Label
                Text="{Binding Price, StringFormat='{0:C2}'}"
                TextColor="Black">
        </Label>
        </StackLayout>
        </Grid>
</ViewCell>
```

```
        </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
    </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

6. Add the **BaseViewModel**:

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Runtime.CompilerServices;

public class BaseViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected void OnPropertyChanged([CallerMemberName] string propertyName = null)
    {
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }

    protected void SetValue<T>(ref T backingField, T value, [CallerMemberName] string propertyName = null)
    {
    if (EqualityComparer<T>.Default.Equals(backingField, value))
    {
    return;
    }

    backingField = value;
    OnPropertyChanged(propertyName);
```

```
          }
}
```

7. Add the **ProductsViewModel**:

```
using System.Collections.Generic;
using System.Collections.ObjectModel;
using Common.Models;
using Common.Services;
using Xamarin.Forms;

public class ProductsViewModel : BaseViewModel
{
          private ApiService apiService;
          private ObservableCollection<Product> products;

          public ObservableCollection<Product> Products
          {
          get { return this.products; }
          set { this.SetValue(ref this.products, value); }
          }

          public ProductsViewModel()
          {
          this.apiService = new ApiService();
          this.LoadProducts();
          }

          private async void LoadProducts()
          {
          var response = await this.apiService.GetListAsync<Product>(
          "https://shopzulu.azurewebsites.net",
          "/api",
          "/Products");
          if (!response.IsSuccess)
          {
          await Application.Current.MainPage.DisplayAlert(
                      "Error",
                      response.Message,
                      "Accept");
          return;
          }

          var products = (List<Product>)response.Result;
          this.Products = new ObservableCollection<Product>(products);
          }
}
```

8. Modify the **MainViewModel**.

```
public class MainViewModel
{
            private static MainViewModel instance;

            public LoginViewModel Login { get; set; }

            public ProductsViewModel Products { get; set; }

            public MainViewModel()
            {
            instance = this;
            this.Login = new LoginViewModel();
```

```
    }

    public static MainViewModel GetInstance()
    {
    if (instance == null)
    {
    return new MainViewModel();
    }

    return instance;
    }
}
```

9. Modify the **LoginViewModel**.

```
if (!this.Email.Equals("jzuluaga55@gmail.com") || !this.Password.Equals("123456"))
{
    await Application.Current.MainPage.DisplayAlert("Error", "Incorrect user or password", "Accept");
    return;
}

MainViewModel.GetInstance().Products = new ProductsViewModel();
await Application.Current.MainPage.Navigation.PushAsync(new ProductsPage());
```

10. Now add an activity indicator and refresh to the list view. Modify the **ProductsPage**:

```
<ListView
    IsPullToRefreshEnabled="True"
    IsRefreshing="{Binding IsRefreshing}"
    HasUnevenRows="True"
    ItemsSource="{Binding Products}"
    RefreshCommand="{Binding RefreshCommand}">
```

11. Modify the **ProductViewModel**:

```
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Windows.Input;
using Common.Models;
using Common.Services;
using GalaSoft.MvvmLight.Command;
using Xamarin.Forms;

public class ProductsViewModel : BaseViewModel
{
    private readonly ApiService apiService;
    private ObservableCollection<Product> products;
    private bool isRefreshing;

    public ObservableCollection<Product> Products
    {
    get => this.products;
    set => this.SetValue(ref this.products, value);
    }

    public bool IsRefreshing
    {
    get => this.isRefreshing;
    set => this.SetValue(ref this.isRefreshing, value);
    }

    public ICommand RefreshCommand => new RelayCommand(this.LoadProducts);
    public ProductsViewModel()
    {
    this.apiService = new ApiService();
    this.LoadProducts();
    }

    private async void LoadProducts()
    {
    this.IsRefreshing = true;
    var response = await this.apiService.GetListAsync<Product>(
    "https://shopzulu.azurewebsites.net",
    "/api",
    "/Products");
    if (!response.IsSuccess)
    {
    await Application.Current.MainPage.DisplayAlert(
            "Error",
            response.Message,
            "Accept");
    this.IsRefreshing = false;
    return;
    }

    var products = (List<Product>)response.Result;
    this.Products = new ObservableCollection<Product>(products);
    this.IsRefreshing = false;
    }
}
```

12. Test it.

To Disable Cascade Delete Rule & Avoid Warnings in Update Database

(Tanks to Starling Germosen (https://www.youtube.com/user/sgrysoft?feature=em-comments))

1. Add this method to **DataContext**:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
        modelBuilder.Entity<Product>()
        .Property(p => p.Price)
        .HasColumnType("decimal(18,2)");

        var cascadeFKs = modelBuilder.Model
        .GetEntityTypes()
        .SelectMany(t => t.GetForeignKeys())
        .Where(fk => !fk.IsOwnership && fk.DeleteBehavior == DeleteBehavior.Cascade);
        foreach (var fk in cascadeFKs)
        {
        fk.DeleteBehavior = DeleteBehavior.Restrict;
        }

        base.OnModelCreating(modelBuilder);
}
```

Implementing login and logout in Web

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; }
}
```

2. Add those methods to interface and implementation:

```
Task<SignInResult> LoginAsync(LoginViewModel model);

Task LogoutAsync();
```

Implementation:

```
public UserHelper(UserManager<User> userManager, SignInManager<User> signInManager)
{
            this.userManager = userManager;
            this.signInManager = signInManager;
}



public async Task<SignInResult> LoginAsync(LoginViewModel model)
{
            return await this.signInManager.PasswordSignInAsync(
            model.Username,
            model.Password,
            model.RememberMe,
            false);
}

public async Task LogoutAsync()
{
            await this.signInManager.SignOutAsync();
}
```

3. Create the controller for login:

```
using System.Linq;
using System.Threading.Tasks;
using Helpers;
using Microsoft.AspNetCore.Mvc;
using Models;

public class AccountController : Controller
{
        private readonly IUserHelper userHelper;

        public AccountController(IUserHelper userHelper)
        {
        this.userHelper = userHelper;
        }

        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.userHelper.LoginAsync(model);
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.userHelper.LogoutAsync();
        return this.RedirectToAction("Index", "Home");
        }
}
```

4. Create the view for login:

```
@model Shop.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");}
}
```

5. Add the annotation authorize to the other controllers:

```
[Authorize]
```

6. 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>
```

7. 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>
}
```

8. 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; }
}
```

2. 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.userHelper.GetUserByEmailAsync(model.Username);
        if (user == null)
        {
        user = new User
        {
                FirstName = model.FirstName,
                LastName = model.LastName,
                Email = model.Username,
    UserName = model.Username
};

var result = await this.userHelper.AddUserAsync(user, model.Password);
if (result != IdentityResult.Success)
{
    this.ModelState.AddModelError(string.Empty, "The user couldn't be created.");
    return this.View(model);
}


var loginViewModel = new LoginViewModel
{
    Password = model.Password,
    RememberMe = false,
    Username = model.Username
};

var result2 = await this.userHelper.LoginAsync(loginViewModel);

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);
}
```

3. Create the register view:

```
@model Shop.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");}
}
```

4. Test it.

Modifying users

1. Create those 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; }
}
```

2. Add this methods to **IUserHelper**:

```
Task<IdentityResult> UpdateUserAsync(User user);

Task<IdentityResult> ChangePasswordAsync(User user, string oldPassword, string newPassword);
```

And the implementation:

```
public async Task<IdentityResult> UpdateUserAsync(User user)
{
        return await this.userManager.UpdateAsync(user);
}

public async Task<IdentityResult> ChangePasswordAsync(User user, string oldPassword, string newPassword)
{
        return await this.userManager.ChangePasswordAsync(user, oldPassword, newPassword);
}
```

3. Create this actions in the account controller:

```
public async Task<IActionResult> ChangeUser()
{
        var user = await this.userHelper.GetUserByEmailAsync(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.userHelper.GetUserByEmailAsync(this.User.Identity.Name);
    if (user != null)
    {
    user.FirstName = model.FirstName;
    user.LastName = model.LastName;
    var respose = await this.userHelper.UpdateUserAsync(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);
}
```

4. Create this view:

```
@model Shop.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");}
}
```

5. 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.userHelper.GetUserByEmailAsync(this.User.Identity.Name);
    if (user != null)
    {
    var result = await this.userHelper.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);
}
```

6. Finally add this view:

```
@model Shop.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>
```

7. Test it.

Add Tokens Generation

1. Add those 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"
  }
}
```

2. Add this method to **IUserHelper**:

```
Task<SignInResult> ValidatePasswordAsync(User user, string password);
```

And the implementation:

```
public async Task<SignInResult> ValidatePasswordAsync(User user, string password)
{
        return  await this.signInManager.CheckPasswordSignInAsync(
        user,
        password,
        false);
}
```

3. Modify the accounts controller constructor:

```
public AccountController(
   SignInManager<User> signInManager,
   UserManager<User> userManager,
   IConfiguration configuration)
{
   this.signInManager = signInManager;
   this.userManager = userManager;
   this.configuration = configuration;
}
```

4. 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.userHelper.GetUserByEmailAsync(model.Username);
        if (user != null)
{
var result = await this.userHelper.ValidatePasswordAsync(
    user,
    model.Password);

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();
}
```

5. Add the authorization annotation to API Products controllers:

```
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
```

6. 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"]))
        };
    });
```

7. 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"
  }
}
```

2. Copy the hidden folder “node_modules” into “wwwroot”.

3. 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>
```

4. 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. Add those methods to **IUserHelper**:

```
Task CheckRoleAsync(string roleName);
Task AddUserToRoleAsync(User user, string roleName);

Task<bool> IsUserInRoleAsync(User user, string roleName);
```

And the implementation:

```
public UserHelper(
    UserManager<User> userManager,
    SignInManager<User> signInManager,
    RoleManager<IdentityRole> roleManager)
{
    this.userManager = userManager;
    this.signInManager = signInManager;
    this.roleManager = roleManager;
}

public async Task CheckRoleAsync(string roleName)
{
    var roleExists = await this.roleManager.RoleExistsAsync(roleName);
    if (!roleExists)
    {
    await this.roleManager.CreateAsync(new IdentityRole
    {
    Name = roleName
    });
    }
}

public async Task AddUserToRoleAsync(User user, string roleName)
{
    await this.userManager.AddToRoleAsync(user, roleName);
}

public async Task<bool> IsUserInRoleAsync(User user, string roleName)
{
    return await this.userManager.IsInRoleAsync(user, roleName);
}
```

2. Modify the seeder class:

```
public async Task SeedAsync()
{
    await this.context.Database.EnsureCreatedAsync();

    await this.userHelper.CheckRoleAsync("Admin");
    await this.userHelper.CheckRoleAsync("Customer");

    // Add user
    var user = await this.userHelper.GetUserByEmailAsync("jzuluaga55@gmail.com");
    if (user == null)
    {
    user = new User
    {
    FirstName = "Juan",
    LastName = "Zuluaga",
    Email = "jzuluaga55@gmail.com",
    UserName = "jzuluaga55@gmail.com",
    PhoneNumber = "3506342747"
    };

    var result = await this.userHelper.AddUserAsync(user, "123456");
    if (result != IdentityResult.Success)
    {
    throw new InvalidOperationException("Could not create the user in seeder");
    }

    await this.userHelper.AddUserToRoleAsync(user, "Admin");
    }

    var isInRole = await this.userHelper.IsUserInRoleAsync(user, "Admin");
    if (!isInRole)
    {
    await this.userHelper.AddUserToRoleAsync(user, "Admin");
    }

    // Add products
    if (!this.context.Products.Any())
    {
    this.AddProduct("iPhone X", user);
    this.AddProduct("Magic Mouse", user);
    this.AddProduct("iWatch Series 4", user);
    await this.context.SaveChangesAsync();
    }
}
```

3. Now you can include the role in authorization annotation in methods Create, Edit and Delete in Products MVC controller:

```
[Authorize(Roles = "Admin")]
```

4. Test it.

Redirect Pages

(Thanks to Gonzalo Jaimes)

Not Authorized

1. Create **NotAuthorized** method on **AccountController**:

```
public IActionResult NotAuthorized()
{
    return this.View();
}
```

2. Create correspondent view with this lines:

```
@{
    ViewData["Title"] = "NotAuthorized";
}

<h2>YOU ARE NOT AUTHORIZED TO PERFORM THIS ACTION!</h2>
```

3. Modify **Startup.cs** to configure the Application Cookie Options (after cookies lines)

```
services.ConfigureApplicationCookie(options =>
{
        options.LoginPath = "/Account/NotAuthorized";
        options.AccessDeniedPath = "/Account/NotAuthorized";
});
```

4. Test it!

Handle Not Found Errors Gracefully

1. Create **NotFoundViewResult** Class (Inside **Helpers** Folder). This way we can customize the page depending on the controller action.

```
using Microsoft.AspNetCore.Mvc;
using System.Net;

public class NotFoundViewResult : ViewResult
{
        public NotFoundViewResult(string viewName)
        {
        ViewName = viewName;
        StatusCode = (int)HttpStatusCode.NotFound;
        }
}
```

2. In the controller Action call the **NotFoundViewResult** method when you`ll expect a not found event

```
// GET: Products/Details/5
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return new NotFoundViewResult("ProductNotFound");
    }

    var product = await this.productRepository.GetByIdAsync(id.Value);
    if (product == null)
    {
        return new NotFoundViewResult("ProductNotFound");
    }
```

```
        return View(product);
    }
```

3. Create the **ProductNotFound** action or any other custom view depending on what you want.

```
public IActionResult ProductNotFound()
{
        return this.View();
}
```

4. Add the view.

```
@{
   ViewData["Title"] = "ProductNotFound";
}

<h2>Product Not Found</h2>
```

5. Test it!.

Manage Not Found Pages

When a page is not found, for instance, trying to execute an non-existing controller action, we need to handle the 404 not found error. StatusCodePagesWithReExecute is a clever piece of Middleware that handles non-success status codes *where the response has not already started*. This means that when we are handling the error inside a controller action it will not be handled by this middleware which is what we want.

1. We add it to the pipeline inside **Startup.cs** with a wildcard as a parameter.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
    app.UseDeveloperExceptionPage();
    }
    else
    {
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
    }

    app.UseStatusCodePagesWithReExecute("/error/{0}");
    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseAuthentication();
    app.UseCookiePolicy();

    app.UseMvc(routes =>
    {
    routes.MapRoute(
        name: "default",
        template: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

2. Inside the Home Controller create the following action.

```
[Route("error/404")]
public IActionResult Error404()
{
    return View();
}
```

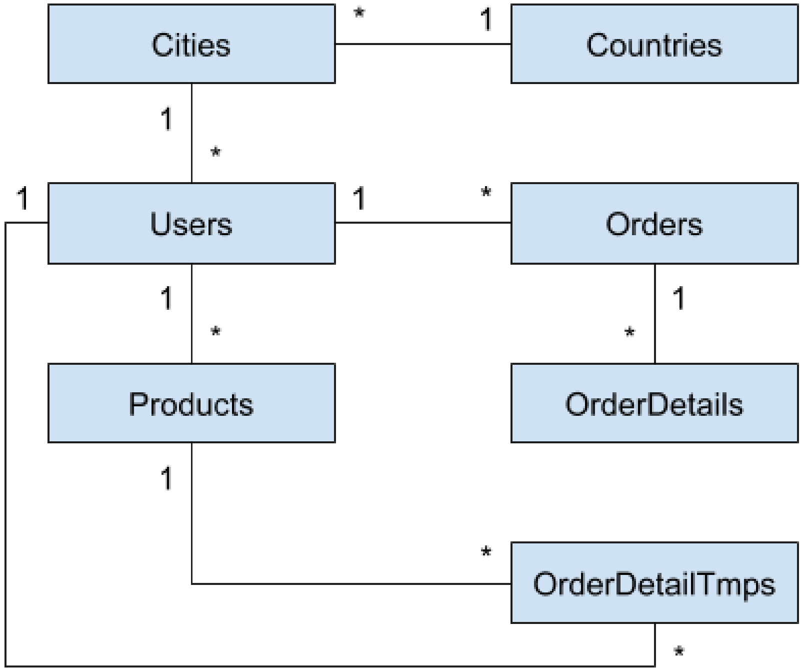
3. Create the correspondent view.

```
@{
    ViewData["Title"] = "Error404";
}

<br />
<br />
<img src="~/images/gopher_head-min.png" />
<h2>Sorry, page not found</h2>
```

4. Test it!.

Orders Functionality

We need to build this:



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

```
using System.ComponentModel.DataAnnotations;

public class OrderDetailTemp : IEntity
{
        public int Id { get; set; }
        [Required]
        public User User { get; set; }

        [Required]
        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; } }
}
```

2. Add order detail model:

```
using System.ComponentModel.DataAnnotations;

public class OrderDetail : IEntity
{
    public int Id { get; set; }

    [Required]
    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; } }
}
```

3. Add order model:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;

public class Order : IEntity
{
        public int Id { get; set; }

        [Required]
        [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; }

        [Required]
        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); } }
}
```

4. 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; }
```

5. 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
```

6. Add the new repository **IOrderRepository**:

```
using System.Linq;
using System.Threading.Tasks;
using Entities;

public interface IOrderRepository : IGenericRepository<Order>
{
        Task<IQueryable<Order>> GetOrdersAsync(string userName);
}
```

7. Add the implementation **OrderRepository**:

```
using System.Linq;
using System.Threading.Tasks;
using Entities;
using Helpers;
using Microsoft.EntityFrameworkCore;

public class OrderRepository : GenericRepository<Order>, IOrderRepository
{
        private readonly DataContext context;
        private readonly IUserHelper userHelper;

        public OrderRepository(DataContext context, IUserHelper userHelper) : base(context)
        {
        this.context = context;
        this.userHelper = userHelper;
        }

        public async Task<IQueryable<Order>> GetOrdersAsync(string userName)
        {
        var user = await this.userHelper.GetUserByEmailAsync(userName);
        if (user == null)
```

```
    {
    return null;
    }

    if (await this.userHelper.IsUserInRoleAsync(user, "Admin"))
    {
    return this.context.Orders
        .Include(o => o.Items)
        .ThenInclude(i => i.Product)
        .OrderByDescending(o => o.OrderDate);
    }

    return this.context.Orders
    .Include(o => o.Items)
    .ThenInclude(i => i.Product)
    .Where(o => o.User == user)
    .OrderByDescending(o => o.OrderDate);
    }
}
```

8. Add the injection for the new repository:

```
services.AddScoped<ICountryRepository, CountryRepository>();
services.AddScoped<IOrderRepository, OrderRepository>();
services.AddScoped<IUserHelper, UserHelper>();
```

9. Add an empty controller **OrdersController**:

```
using System.Threading.Tasks;
using Data;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

[Authorize]
public class OrdersController : Controller
{
    private readonly IOrderRepository orderRepository;

    public OrdersController(IOrderRepository orderRepository)
    {
    this.orderRepository = orderRepository;
    }

    public async Task<IActionResult> Index()
    {
    var model = await orderRepository.GetOrdersAsync(this.User.Identity.Name);
    return View(model);
    }
}
```

10. Add this property to users entity:

```
[Display(Name = "Full Name")]
public string FullName { get { return $"{this.FirstName} {this.LastName}"; }  }
```

11. Add the corresponding view:

```
@model IEnumerable<Shop.Web.Data.Entities.Order>

@{
    ViewData["Title"] = "Index";
}
```

```
@model IEnumerable<Shop.Web.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>
	@if (this.User.IsInRole("Admin"))
	{
		<th>
		@Html.DisplayNameFor(model => model.User.FullName)
		</th>
	}
	<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>
@if (this.User.IsInRole("Admin"))
{
<th>
        @Html.DisplayFor(modelItem => item.User.FullName)
</th>
}
<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>
```

12. Add the new menu:

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

13. Test it what we do until this step.

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

```
Task<IQueryable<OrderDetailTemp>> GetDetailTempsAsync(string userName);
```

And the implementation:

```
public async Task<IQueryable<OrderDetailTemp>> GetDetailTempsAsync(string userName)
{
        var user = await this.userHelper.GetUserByEmailAsync(userName);
        if (user == null)
        {
        return null;
        }
        return this.context.OrderDetailTemps
        .Include(o => o.Product)
        .Where(o => o.User == user)
        .OrderBy(o => o.Product.Name);
}
```

15. Add the method create to the orders controller:

```
public async Task<IActionResult> Create()
{
        var model = await this.orderRepository.GetDetailTempsAsync(this.User.Identity.Name);
        return this.View(model);
}
```

16. And their corresponding view:

```
@model IEnumerable<Shop.Web.Data.Entities.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>
```

17. Test it.

18. 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; }
}
```

19. Add this method to **IProductRepository**:

```
IEnumerable<SelectListItem> GetComboProducts();
```

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;
}
```

20. Add this method to **IOrderRepository**:

```
Task AddItemToOrderAsync(AddItemViewModel model, string userName);

Task ModifyOrderDetailTempQuantityAsync(int id, double quantity);
```

And to the implementation:

```
public async Task AddItemToOrderAsync(AddItemViewModel model, string userName)
{
        var user = await this.userHelper.GetUserByEmailAsync(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();
        }
}
```

21. Add those methods to the **OrdersController**:

```
public IActionResult AddProduct()
{
        var model = new AddItemViewModel
        {
        Quantity = 1,
        Products = this.productRepository.GetComboProducts()
        };

        return View(model);
}


[HttpPost]
public async Task<IActionResult> AddProduct(AddItemViewModel model)
{
        if (this.ModelState.IsValid)
        {
        await this.orderRepository.AddItemToOrderAsync(model, this.User.Identity.Name);
        return this.RedirectToAction("Create");
        }

        return this.View(model);
}
```

22. Add the view:

```
@model Shop.Web.Models.AddItemViewModel

@{
        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");}
}
```

23. Test it.

24. Add this method to interface **IOrderRepository**:

```
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();
}
```

25. Now implement those methods in the controller:

```
public async Task<IActionResult> DeleteItem(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    await this.orderRepository.DeleteDetailTempAsync(id.Value);
    return this.RedirectToAction("Create");
}

public async Task<IActionResult> Increase(int? id)
```

```
{
        if (id == null)
        {
        return NotFound();
        }

        await this.orderRepository.ModifyOrderDetailTempQuantityAsync(id.Value, 1);
        return this.RedirectToAction("Create");
}

public async Task<IActionResult> Decrease(int? id)
{
        if (id == null)
        {
        return NotFound();
        }

        await this.orderRepository.ModifyOrderDetailTempQuantityAsync(id.Value, -1);
        return this.RedirectToAction("Create");
}
```

26. Test it.

27. Add the confirm order method in the interface and implementation in **IOrderRepository**:

```
Task<bool> ConfirmOrderAsync(string userName);
```

And in the implementation:

```
public async Task<bool> ConfirmOrderAsync(string userName)
{
```

```
var user = await this.userHelper.GetUserByEmailAsync(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;
}
```

28. 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); } }
```

29. Modify 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>
              @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>
```

30. Add the method to the controller:

```
public async Task<IActionResult> ConfirmOrder()
{
        var response = await this.orderRepository.ConfirmOrderAsync(this.User.Identity.Name);
        if (response)
        {
        return this.RedirectToAction("Index");
        }

        return this.RedirectToAction("Create");
}
```

31. Add this property to entity **Order**.

```
[Display(Name = "Order date")]
[DisplayFormat(DataFormatString = "{0:yyyy/MM/dd hh:mm tt}", ApplyFormatInEditMode = false)]
public DateTime? OrderDateLocal
{
        get
        {
        if (this.OrderDate == null)
        {
        return null;
        }

        return this.OrderDate.ToLocalTime();
        }
}
```

Change the index view to show this new property (and do the same for other data fields).

```
<th>
        @Html.DisplayNameFor(model => model.OrderDateLocal)
</th>
```

And:

```
<td>
        @Html.DisplayFor(modelItem => item.OrderDateLocal)
</td>
```

32. Fix the bug in **OrderRepository** in method **GetOrdersAsync** to get the user in the query.

```
if (await this.userHelper.IsUserInRoleAsync(user, "Admin"))
{
        return this.context.Orders
        .Include(o => o.User)
        .Include(o => o.Items)
        .ThenInclude(i => i.Product)
        .OrderByDescending(o => o.OrderDate);
}
```

33. Test it.

Add Modal Windows

1. To add a validation to confirm the order, add those 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>
}
```

2. Test it.

3. 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>
}
```

4. 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"
  }
}
```

2. Save the file and copy the bootstrap date picker into folder root node modules.

3. Add those 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>
```

4. 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; }
}
```

5. Add those methods to interface **IOrderRepository**:

```
Task DeliverOrder(DeliverViewModel model);

Task<Order> GetOrdersAsync(int id);
```

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();
}

public async Task<Order> GetOrdersAsync(int id)
{
        return await this.context.Orders.FindAsync(id);
}
```

6. Add this method to the orders controller:

```
public async Task<IActionResult> Deliver(int? id)
{
        if (id == null)
        {
        return NotFound();
        }

        var order = await this.orderRepository.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.orderRepository.DeliverOrder(model);
        return this.RedirectToAction("Index");
        }
        return this.View();
}
```

7. Add the view:

```
@model Shop.Web.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");}
}
```

8. 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>
```

9. Test it.

10. 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>
```

11. Test it.

Cascade Drop Down List

1. First add the new entities:

```
using System.ComponentModel.DataAnnotations;

public class City : IEntity
{
        public int Id { get; set; }

        [Required]
```

```
    [Display(Name = "City")]
    [MaxLength(50, ErrorMessage = "The field {0} only can contain {1} characters length.")]
    public string Name { get; set; }
}
```

And modify the previous country entity :

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

public class Country : IEntity
{
    public int Id { get; set; }

    [Required]
    [Display(Name = "Country")]
    [MaxLength(50, ErrorMessage = "The field {0} only can contain {1} characters length.")]
    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; } }
}
```

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

```
[MaxLength(100, ErrorMessage = "The field {0} only can contain {1} characters length.")]
public string Address { get; set; }

public int CityId { get; set; }
public City City { get; set; }
```

3. Add this lines to the data context:

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

4. 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
```

5. 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",
        PhoneNumber = "350 634 2747",
        Address = "Calle Luna Calle Sol",
        CityId = this.context.Countries.FirstOrDefault().Cities.FirstOrDefault().Id,
        City = this.context.Countries.FirstOrDefault().Cities.FirstOrDefault()
    };

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

6. Add the new view model:

```
using System.ComponentModel.DataAnnotations;
```

```
public class CityViewModel
{
	public int CountryId { get; set; }

	public int CityId { get; set; }

	[Required]
	[Display(Name = "City")]
	[MaxLength(50, ErrorMessage = "The field {0} only can contain {1} characters length.")]
	public string Name { get; set; }
}
```

7. Add this methods to the repository **ICountryRepository**:

```
using System.Linq;
using System.Threading.Tasks;
using Entities;
using Models;

public interface ICountryRepository : IGenericRepository<Country>
{
	IQueryable GetCountriesWithCities();

	Task<Country> GetCountryWithCitiesAsync(int id);

	Task<City> GetCityAsync(int id);

	Task AddCityAsync(CityViewModel model);

	Task<int> UpdateCityAsync(City city);
```

```
    Task<int> DeleteCityAsync(City city);
}
```

And the implementation:

```
using System.Linq;
using System.Threading.Tasks;
using Entities;
using Microsoft.EntityFrameworkCore;
using Models;

public class CountryRepository : GenericRepository<Country>, ICountryRepository
{
    private readonly DataContext context;

    public CountryRepository(DataContext context) : base(context)
    {
    this.context = context;
    }

    public async Task AddCityAsync(CityViewModel model)
    {
    var country = await this.GetCountryWithCitiesAsync(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> 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;
    }

    public IQueryable GetCountriesWithCities()
    {
    return this.context.Countries
    .Include(c => c.Cities)
    .OrderBy(c => c.Name);
    }

    public async Task<Country> GetCountryWithCitiesAsync(int id)
    {
    return await this.context.Countries
    .Include(c => c.Cities)
    .Where(c => c.Id == id)
    .FirstOrDefaultAsync();
    }

    public async Task<int> UpdateCityAsync(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<City> GetCityAsync(int id)
    {
    return await this.context.Cities.FindAsync(id);
    }
}
```

8. Add the countries controller:

```
using System.Threading.Tasks;
using Data;
using Data.Entities;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Models;

[Authorize(Roles = "Admin")]
public class CountriesController : Controller
{
    private readonly ICountryRepository countryRepository;
```

```
public CountriesController(ICountryRepository countryRepository)
{
this.countryRepository = countryRepository;
}

public async Task<IActionResult> DeleteCity(int? id)
{
if (id == null)
{
return NotFound();
}

var city = await this.countryRepository.GetCityAsync(id.Value);
if (city == null)
{
return NotFound();
}

var countryId = await this.countryRepository.DeleteCityAsync(city);
return this.RedirectToAction($"Details/{countryId}");
}

public async Task<IActionResult> EditCity(int? id)
{
if (id == null)
{
return NotFound();
}

var city = await this.countryRepository.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.countryRepository.UpdateCityAsync(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.countryRepository.GetByIdAsync(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.countryRepository.AddCityAsync(model);
return this.RedirectToAction($"Details/{model.CountryId}");
}

return this.View(model);
}

public IActionResult Index()
{
return View(this.countryRepository.GetCountriesWithCities());
}

public async Task<IActionResult> Details(int? id)
{
if (id == null)
{
return NotFound();
}

var country = await this.countryRepository.GetCountryWithCitiesAsync(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.countryRepository.CreateAsync(country);
return RedirectToAction(nameof(Index));
}

return View(country);
}

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

```
if (id == null)
{
return NotFound();
}

var country = await this.countryRepository.GetByIdAsync(id.Value);
if (country == null)
{
return NotFound();
}
return View(country);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(Country country)
{
if (ModelState.IsValid)
{
await this.countryRepository.UpdateAsync(country);
return RedirectToAction(nameof(Index));
}

return View(country);
}

public async Task<IActionResult> Delete(int? id)
{
if (id == null)
{
return NotFound();
```

```
            }

            var country = await this.countryRepository.GetByIdAsync(id.Value);
            if (country == null)
            {
            return NotFound();
            }

            await this.countryRepository.DeleteAsync(country);
            return RedirectToAction(nameof(Index));
            }
}
```

9. Add the corresponding Views:

Index:

```
@model IEnumerable<Shop.Web.Data.Entities.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 Shop.Web.Data.Entities.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 Shop.Web.Data.Entities.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 Shop.Web.Data.Entities.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 Shop.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 Shop.Web.Data.Entities.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");}
}
```

10. Add to the new menu for countries:

```
<li><a asp-area="" asp-controller="Products" asp-action="Index">Products</a></li>
@if (this.User.Identity.IsAuthenticated)
{
        <li><a asp-area="" asp-controller="Orders" asp-action="Index">Orders</a></li>
        @if (this.User.IsInRole("Admin"))
        {
        <li><a asp-area="" asp-controller="Countries" asp-action="Index">Countries</a></li>
        }
}
```

11. Test it and add some countries and cities.

12. Modify the **RegisterNewUserViewModel**:

```
[Required]
[Compare("Password")]
public string Confirm { get; set; }

[MaxLength(100, ErrorMessage = "The field {0} only can contain {1} characters length.")]
public string Address { get; set; }

[MaxLength(20, ErrorMessage = "The field {0} only can contain {1} characters length.")]
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; }
```

13. Add this methods to the repository **ICountryRepository**:

```
IEnumerable<SelectListItem> GetComboCountries();

IEnumerable<SelectListItem> GetComboCities(int conuntryId);

Task<Country> GetCountryAsync(City city);
```

And the implementation:

```
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();
}
```

14. Change the register method in account controller (first inject **ICountryRepository**):

```
public IActionResult Register()
{
            var model = new RegisterNewUserViewModel
            {
            Countries = this.countryRepository.GetComboCountries(),
            Cities = this.countryRepository.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.countryRepository.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);
}
```

15. 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>
```

16. Test the code until this point.

17. Now implement the cascade drop down list.

18. Add this method to account controller:

```
public async Task<JsonResult> GetCitiesAsync(int countryId)
{
        var country = await this.countryRepository.GetCountryWithCitiesAsync(countryId);
        return this.Json(country.Cities.OrderBy(c => c.Name));
}
```

19. And modify the register view:

```
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
    <script type="text/javascript">
        $(document).ready(function () {
            $("#CountryId").change(function () {
                $("#CityId").empty();
                $.ajax({
                    type: 'POST',
                    url: '@Url.Action("GetCitiesAsync")',
                    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>
}
```

20. Test it.

21. 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, ErrorMessage = "The field {0} only can contain {1} characters length.")]
public string Address { get; set; }

[MaxLength(20, ErrorMessage = "The field {0} only can contain {1} characters length.")]
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; }
```

22. Modify the change user method in account controller:

```
public async Task<IActionResult> ChangeUser()
{
    var user = await this.userHelper.GetUserByEmailAsync(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.countryRepository.GetCityAsync(user.CityId);
    if (city != null)
    {
    var country = await this.countryRepository.GetCountryAsync(city);
    if (country != null)
    {
        model.CountryId = country.Id;
        model.Cities = this.countryRepository.GetComboCities(country.Id);
        model.Countries = this.countryRepository.GetComboCountries();
        model.CityId = user.CityId;
    }
    }
    }

    model.Cities = this.countryRepository.GetComboCities(model.CountryId);
    model.Countries = this.countryRepository.GetComboCountries();
    return this.View(model);
}

[HttpPost]
public async Task<IActionResult> ChangeUser(ChangeUserViewModel model)
{
    if (this.ModelState.IsValid)
    {
    var user = await this.userHelper.GetUserByEmailAsync(this.User.Identity.Name);
    if (user != null)
    {
    var city = await this.countryRepository.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.userHelper.UpdateUserAsync(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);
}
```

23. Modify the view:

```
@model Shop.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>
}
```

24. 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>();
```

2. Check if your email account is enabled to send email in: https://myaccount.google.com/lesssecureapps

3. Add this parameters to the configuration file:

```
"Mail": {
  "From": "youremail@gmail.com",
  "Smtp": "smtp.gmail.com",
  "Port": 587,
  "Password": "yourpassword"
}
```

4. Add the nuget “**Mailkit**”.

5. In **Helpers** folder add the interface **IMailHelper**:

```
public interface IMailHelper
{
        void SendMail(string to, string subject, string body);
}
```

6. 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);
    }
    }
}
```

7. Configure the injection for the new interface:

```
services.AddScoped<IMailHelper, MailHelper>();
```

8. Add those methods to **IUserHelper**:

```
Task<string> GenerateEmailConfirmationTokenAsync(User user);

Task<IdentityResult> ConfirmEmailAsync(User user, string token);

Task<User> GetUserByIdAsync(string userId);
```

And the implementation:

```
public async Task<IdentityResult> ConfirmEmailAsync(User user, string token)
{
    return await this.userManager.ConfirmEmailAsync(user, token);
}

public async Task<string> GenerateEmailConfirmationTokenAsync(User user)
{
    return await this.userManager.GenerateEmailConfirmationTokenAsync(user);
}

public async Task<User> GetUserByIdAsync(string userId)
{
    return await this.userManager.FindByIdAsync(userId);
}
```

9. Modify the register post method (first inject the **IMailHelper** in **AccountController**):

```
[HttpPost]
public async Task<IActionResult> Register(RegisterNewUserViewModel model)
{
    if (this.ModelState.IsValid)
    {
    var user = await this.userHelper.GetUserByEmailAsync(model.Username);
```

```
if (user == null)
{
var city = await this.countryRepository.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.userHelper.AddUserAsync(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.userHelper.GenerateEmailConfirmationTokenAsync(user);
var tokenLink = this.Url.Action("ConfirmEmail", "Account", new
{
        userid = user.Id,
        token = myToken
}, protocol: HttpContext.Request.Scheme);

this.mailHelper.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);
}
```

10. 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 {
...
```

11. 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.userHelper.GetUserByIdAsync(userId);
        if (user == null)
        {
        return this.NotFound();
        }

        var result = await this.userHelper.ConfirmEmailAsync(user, token);
        if (!result.Succeeded)
        {
        return this.NotFound();
        }

        return View();
}
```

12. Create the view:

```
@{
   ViewData["Title"] = "Confirm email";
}

<h2>@ViewData["Title"]</h2>
```

```
<div>
    <p>
        Thank you for confirming your email. Now you can login into system.
    </p>
</div>
```

13. Drop the database (PM> drop-database) to ensure that all the users have a confirmed email.

14. Modify the seed class:

```
await this.userHelper.AddUserToRoleAsync(user, "Admin");
var token = await this.userHelper.GenerateEmailConfirmationTokenAsync(user);
await this.userHelper.ConfirmEmailAsync(user, token);
```

15. 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>
```

2. Add the model:

```
using System.ComponentModel.DataAnnotations;
```

```
public class RecoverPasswordViewModel
{
    [Required]
    [EmailAddress]
    public string Email { get; set; }
}
```

3. 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; }
}
```

4. Add those methods to **IUserHelper**:

```
Task<string> GeneratePasswordResetTokenAsync(User user);

Task<IdentityResult> ResetPasswordAsync(User user, string token, string password);
```

And the implementation:

```
public async Task<string> GeneratePasswordResetTokenAsync(User user)
{
    return await this.userManager.GeneratePasswordResetTokenAsync(user);
}

public async Task<IdentityResult> ResetPasswordAsync(User user, string token, string password)
{
    return await this.userManager.ResetPasswordAsync(user, token, password);
}
```

5. 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.userHelper.GetUserByEmailAsync(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.userHelper.GeneratePasswordResetTokenAsync(user);
    var link = this.Url.Action(
    "ResetPassword",
    "Account",
    new { token = myToken }, protocol: HttpContext.Request.Scheme);
    this.mailHelper.SendMail(model.Email, "Shop Password Reset", $"<h1>Shop Password Reset</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.userHelper.GetUserByEmailAsync(model.UserName);
    if (user != null)
    {
    var result = await this.userHelper.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);
}
```

6. Add the view:

```
@model Shop.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");}
}
```

7. Add the view:

```
@model Shop.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");}
}
```

8. Test it.

9. Finally, delete all records in Azure DB and re-publish the solution.

Fix AwesomeFont and DatePicker on Published Site

(Thanks to Gonzalo Jaimes)

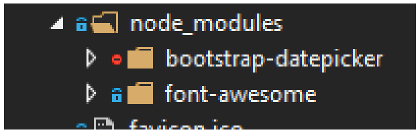
1. Modify the **_Layout**:

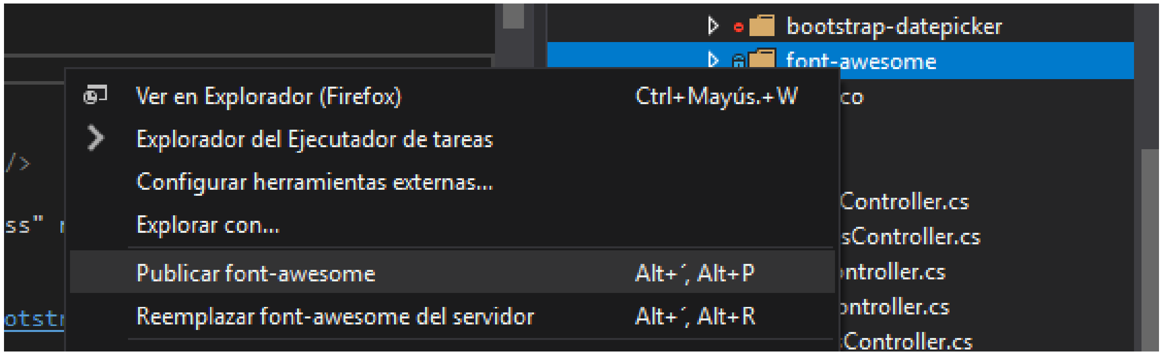
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Shop.Web </title>

    <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 exclude="Development">
        <link rel="stylesheet" href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/css/bootstrap.min.css"
              asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"
              asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-value="absolute" />
        <link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />
        <link href="~/node_modules/bootstrap-datepicker/dist/css/bootstrap-datepicker.min.css" rel="stylesheet" />
        <link href="~/node_modules/font-awesome/css/font-awesome.min.css" rel="stylesheet" />
    </environment>

</head>
<body>
    <nav class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a asp-area="" asp-controller="Home" asp-action="Index" class="navbar-brand">Shop.Web</a>
            </div>
            <div class="navbar-collapse collapse">
                <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>
                    @if (this.User.Identity.IsAuthenticated)
                    {
                        <li><a asp-area="" asp-controller="Products" asp-action="Index">Products</a></li>
                        @if (this.User.IsInRole("Admin"))
                        {
                            <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>
                    }

                </ul>
                <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>

            </div>
        </div>
    </nav>

    <partial name="_CookieConsentPartial" />

    <div class="container body-content">
        @RenderBody()
```

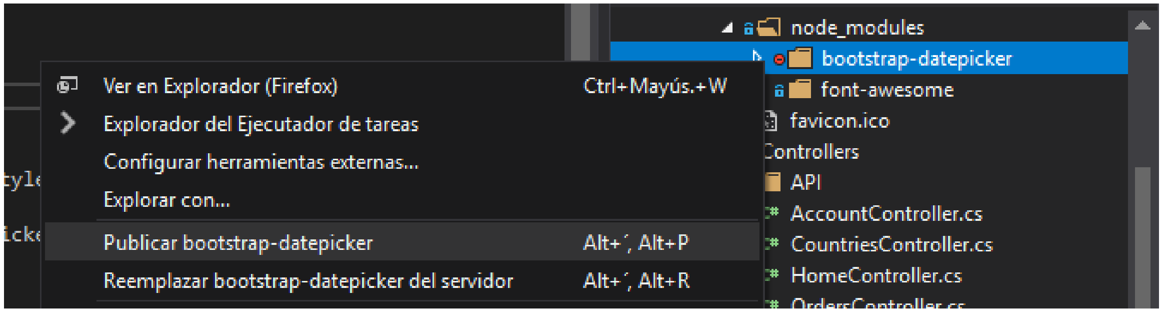
```
        <hr />
        <footer>
            <p>&copy; 2019 - Shop.Web</p>
        </footer>
    </div>

    <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>
    <environment exclude="Development">
        <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.3.1.min.js"
                asp-fallback-src="~/lib/jquery/dist/jquery.min.js"
                asp-fallback-test="window.jQuery"
                crossorigin="anonymous"
                integrity="sha384-tsQFqpEReu7ZLhBV2VZlAu7zcOV+rXbYlF2cqB8txI/8aZajjp4Bqd+V6D5IgvKT">
        </script>
        <script src="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.7/bootstrap.min.js"
                asp-fallback-src="~/lib/bootstrap/dist/js/bootstrap.min.js"
                asp-fallback-test="window.jQuery && window.jQuery.fn && window.jQuery.fn.modal"
                crossorigin="anonymous"
                integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa">
        </script>
        <script src="~/node_modules/bootstrap-datepicker/dist/js/bootstrap-datepicker.min.js"></script>
        <script src="~/js/site.min.js" asp-append-version="true"></script>
    </environment>

    @RenderSection("Scripts", required: false)
</body>
```

```
</html>
```

2. Publish the folders **FontAwesome** and **Datepicker** by separate:




bootstrap-datepicker
font-awesome
Ver en Explorador (Firefox) Ctrl+Mayús.+W
Explorador del Ejecutador de tareas
Configurar herramientas externas...
Explorar con...
Publicar font-awesome Alt+´, Alt+P
Reemplazar font-awesome del servidor Alt+´, Alt+R
Controller.cs
sController.cs
ntroller.cs
ontroller.cs
Controller.cs


node_modules
bootstrap-datepicker
font-awesome
favicon.ico
Controllers
API
AccountController.cs
CountriesController.cs
HomeController.cs
OrdersController.cs
Ver en Explorador (Firefox) Ctrl+Mayús.+W
Explorador del Ejecutador de tareas
Configurar herramientas externas...
Explorar con...
Publicar bootstrap-datepicker Alt+´, Alt+P
Reemplazar bootstrap-datepicker del servidor Alt+´, Alt+R

3. Test It!.

Improve the Seeder

1. Add the products images:

2. Modify the seeder:

```
using System;
using System.Linq;
using System.Threading.Tasks;
using Entities;
using Helpers;
using Microsoft.AspNetCore.Identity;

public class SeedDb
{
    private readonly DataContext context;
    private readonly IUserHelper userHelper;
    private readonly Random random;

    public SeedDb(DataContext context, IUserHelper userHelper)
    {
    this.context = context;
    this.userHelper = userHelper;
    this.random = new Random();
    }

    public async Task SeedAsync()
    {
    await this.context.Database.EnsureCreatedAsync();
    await this.CheckRoles();

    if (!this.context.Countries.Any())
    {
    await this.AddCountriesAndCitiesAsync();
    }

    await this.CheckUserAsync("brad@gmail.com", "Brad", "Pit", "Customer");
    await this.CheckUserAsync("angelina@gmail.com", "Angelina", "Jolie", "Customer");
    var user = await this.CheckUserAsync("jzuluaga55@gmail.com", "Juan", "Zuluaga", "Admin");

    // Add products
    if (!this.context.Products.Any())
    {
    this.AddProduct("AirPods", 159, user);
    this.AddProduct("Blackmagic eGPU Pro", 1199, user);
    this.AddProduct("iPad Pro", 799, user);
    this.AddProduct("iMac", 1398, user);
    this.AddProduct("iPhone X", 749, user);
    this.AddProduct("iWatch Series 4", 399, user);
    this.AddProduct("Mac Book Air", 789, user);
    this.AddProduct("Mac Book Pro", 1299, user);
    this.AddProduct("Mac Mini", 708, user);
    this.AddProduct("Mac Pro", 2300, user);
    this.AddProduct("Magic Mouse", 47, user);
    this.AddProduct("Magic Trackpad 2", 140, user);
    this.AddProduct("USB C Multiport", 145, user);
    this.AddProduct("Wireless Charging Pad", 67.67M, user);
    await this.context.SaveChangesAsync();
    }
    }
```

```
private async Task<User> CheckUserAsync(string userName, string firstName, string lastName, string role)
{
// Add user
var user = await this.userHelper.GetUserByEmailAsync(userName);
if (user == null)
{
user = await this.AddUser(userName, firstName, lastName, role);
}

var isInRole = await this.userHelper.IsUserInRoleAsync(user, role);
if (!isInRole)
{
await this.userHelper.AddUserToRoleAsync(user, role);
}

return user;
}

private async Task<User> AddUser(string userName, string firstName, string lastName, string role)
{
var user = new User
{
FirstName = firstName,
LastName = lastName,
Email = userName,
UserName = userName,
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.userHelper.AddUserAsync(user, "123456");
    if (result != IdentityResult.Success)
    {
    throw new InvalidOperationException("Could not create the user in seeder");
    }

    await this.userHelper.AddUserToRoleAsync(user, role);
    var token = await this.userHelper.GenerateEmailConfirmationTokenAsync(user);
    await this.userHelper.ConfirmEmailAsync(user, token);
    return user;
    }

    private async Task AddCountriesAndCitiesAsync()
    {
    this.AddCountry("Colombia", new string[] { "Medellín", "Bogota", "Calí", "Barranquilla", "Bucaramanga", "Cartagena", "Pereira"
});
    this.AddCountry("Argentina", new string[] { "Córdoba", "Buenos Aires", "Rosario", "Tandil", "Salta", "Mendoza" });
    this.AddCountry("Estados Unidos", new string[] { "New York", "Los Ángeles", "Chicago", "Washington", "San Francisco",
"Miami", "Boston" });
    this.AddCountry("Ecuador", new string[] { "Quito", "Guayaquil", "Ambato", "Manta", "Loja", "Santo" });
    this.AddCountry("Peru", new string[] { "Lima", "Arequipa", "Cusco", "Trujillo", "Chiclayo", "Iquitos" });
    this.AddCountry("Chile", new string[] { "Santiago", "Valdivia", "Concepcion", "Puerto Montt", "Temucos", "La Sirena" });
    this.AddCountry("Uruguay", new string[] { "Montevideo", "Punta del Este", "Colonia del Sacramento", "Las Piedras" });
    this.AddCountry("Bolivia", new string[] { "La Paz", "Sucre", "Potosi", "Cochabamba" });
    this.AddCountry("Venezuela", new string[] { "Caracas", "Valencia", "Maracaibo", "Ciudad Bolivar", "Maracay", "Barquisimeto"
});
    this.AddCountry("Paraguay", new string[] { "Asunción", "Ciudad del Este", "Encarnación", "San  Lorenzo", "Luque", "Areguá"
});
```

```
    this.AddCountry("Brasil", new string[] { "Rio de Janeiro", "São Paulo", "Salvador", "Porto Alegre", "Curitiba", "Recife", "Belo Horizonte", "Fortaleza" });
    this.AddCountry("Panamá", new string[] { "Chitré", "Santiago", "La Arena", "Agua Dulce", "Monagrillo", "Ciudad de Panamá", "Colón", "Los Santos" });
    this.AddCountry("México", new string[] { "Guadalajara", "Ciudad de México", "Monterrey", "Ciudad Obregón", "Hermosillo", "La Paz", "Culiacán", "Los Mochis" });
    await this.context.SaveChangesAsync();
    }

    private void AddCountry(string country, string[] cities)
    {
    var theCities = cities.Select(c => new City { Name = c }).ToList();
    this.context.Countries.Add(new Country
    {
    Cities = theCities,
    Name = country
    });
    }

    private async Task CheckRoles()
    {
    await this.userHelper.CheckRoleAsync("Admin");
    await this.userHelper.CheckRoleAsync("Customer");
    }

    private void AddProduct(string name, decimal price, User user)
    {
    this.context.Products.Add(new Product
    {
    Name = name,
    Price = price,
            IsAvailabe = true,
            Stock = this.random.Next(100),
            User = user,
            ImageUrl = $"~/images/Products/{name}.png"
            });
            }
}
```

3. Drop the database and test it.

Making A Little Easier The Modal Dialogs

(Thanks to William Andres Hurtado)

1. Modify the view **Create** in **Orders**:

```
@model IEnumerable<Shop.Web.Data.Entities.OrderDetailTemp>

@{
        ViewData["Title"] = "Create";
}

<h2>Create</h2>

<p>
        <a asp-action="AddProduct" class="btn btn-success">Add Product</a>
        <button type="button" class="btn btn-primary" data-toggle="modal" data-target="#confirmDialog">Confirm Order</button>
</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>
        <button data-id="@item.Id" class="btn btn-danger deleteItem" data-toggle="modal"
data-target="#deleteDialog">Delete</button>
        </td>
    </tr>
    }
    </tbody>
</table>

<!-- Confirm Order-->
<div class="modal fade" id="confirmDialog" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel" aria-hidden="true">
    <div class="modal-dialog modal-sm" role="document">
    <div class="modal-content">
    <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel">Confirm Order</h5>
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
        <span aria-hidden="true">&times;</span>
        </button>
    </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-secondary" data-dismiss="modal">Close</button>
        <button type="button" class="btn btn-success" id="btnYes">Save changes</button>
    </div>
    </div>
    </div>
```

```
</div>

<!--Delete Item-->
<div class="modal fade" id="deleteDialog" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel" aria-hidden="true">
    <div class="modal-dialog" role="document">
    <div class="modal-content">
    <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel">Delete Item</h5>
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
        <span aria-hidden="true">&times;</span>
        </button>
    </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-primary" data-dismiss="modal">Close</button>
        <button type="button" class="btn btn-danger" id="btnYesDelete">Delete</button>
    </div>
    </div>
    </div>
</div>

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

    <script type="text/javascript">
    $(document).ready(function () {
    // Confirm Order
    $("#btnYes").click(function () {
        window.location.href = '/Orders/ConfirmOrder';
    });

    // Delete item
    var item_to_delete;

    $('.deleteItem').click((e) => {
        item_to_delete = e.currentTarget.dataset.id;
    });

    $("#btnYesDelete").click(function () {
        window.location.href = '/Orders/DeleteItem/' + item_to_delete;
    });
    });
    </script>
}
```

2. Test it.

Improve Index View

(Thanks to Jimmy Davila)

1. Modify the index view by:

```
@model IEnumerable<Shop.Web.Data.Entities.Product>

@{
        ViewData["Title"] = "Index";
}
```

```
<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />
<br />

<p>
	<a asp-action="Create" class="btn btn-primary"><i class="glyphicon glyphicon-plus"></i> Create New</a>
</p>

<div class="row">
	<div class="col-md-12">
	<div class="panel panel-default">
	<div class="panel-heading">
		<h3 class="panel-title">Products</h3>
	</div>
	<div class="panel-body">
		<table class="table table-hover table-responsive table-striped" id="ProductsTable">
		<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:75px;height:75px;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" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon
glyphicon-pencil"></i> </a>
                        <a asp-action="Details" class="btn btn-default" asp-route-id="@item.Id"><i class="glyphicon glyphicon-list">
</i> </a>
                        <a asp-action="Delete" class="btn btn-danger" asp-route-id="@item.Id"><i class="glyphicon
glyphicon-trash"></i> </a>
                        </td>
                        </tr>
                }
                </tbody>
                </table>
        </div>
        </div>
        </div>
</div>

@section Scripts {
        <script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>
        <script>
        $(document).ready(function () {
```

```
        $('#ProductsTable').DataTable();
        });
        </script>
}
```

2. Test it.

User Management

1. Modify the **User** entity (in **Web.Data.Entities**):

```
using Microsoft.AspNetCore.Identity;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

public class User : IdentityUser
{
        [Display(Name = "First Name")]
        public string FirstName { get; set; }

        [Display(Name = "Last Name")]
        public string LastName { get; set; }

        [MaxLength(100, ErrorMessage = "The field {0} only can contain {1} characters length.")]
        public string Address { get; set; }

        public int CityId { get; set; }

        public City City { get; set; }
```

```
    [Display(Name = "Phone Number")]
    public override string PhoneNumber { get => base.PhoneNumber; set => base.PhoneNumber = value; }

    [Display(Name = "Full Name")]
    public string FullName { get { return $"{this.FirstName} {this.LastName}"; }  }

    [Display(Name = "Email Confirmed")]
    public override bool EmailConfirmed { get => base.EmailConfirmed; set => base.EmailConfirmed = value; }

    [NotMapped]
    [Display(Name = "Is Admin?")]
    public bool IsAdmin { get; set; }
}
```

2. Add this methods to **IUserHelper** (in **Web.Helpers**):

```
Task<List<User>> GetAllUsersAsync();

Task RemoveUserFromRoleAsync(User user, string roleName);

Task DeleteUserAsync(User user);
```

And the implementation:

```
public async Task<List<User>> GetAllUsersAsync()
{
    return await this.userManager.Users
    .Include(u => u.City)
    .OrderBy(u => u.FirstName)
    .ThenBy(u => u.LastName)
```

```
        .ToListAsync();
}

public async Task RemoveUserFromRoleAsync(User user, string roleName)
{
        await this.userManager.RemoveFromRoleAsync(user, roleName);
}

public async Task DeleteUserAsync(User user)
{
        await this.userManager.DeleteAsync(user);
}
```

3. Add those methods to **AccountController** (in **Web.Controllers**):

```
public async Task<IActionResult> Index()
{
        var users = await this.userHelper.GetAllUsersAsync();
        foreach (var user in users)
        {
        var myUser = await this.userHelper.GetUserByIdAsync(user.Id);
        if (myUser != null)
        {
        user.IsAdmin = await this.userHelper.IsUserInRoleAsync(myUser, "Admin");
        }
        }

        return this.View(users);
}

public async Task<IActionResult> AdminOff(string id)
```

```
{
        if (string.IsNullOrEmpty(id))
        {
        return NotFound();
        }

        var user = await this.userHelper.GetUserByIdAsync(id);
        if (user == null)
        {
        return NotFound();
        }

        await this.userHelper.RemoveUserFromRoleAsync(user, "Admin");
        return this.RedirectToAction(nameof(Index));
}

public async Task<IActionResult> AdminOn(string id)
{
        if (string.IsNullOrEmpty(id))
        {
        return NotFound();
        }

        var user = await this.userHelper.GetUserByIdAsync(id);
        if (user == null)
        {
        return NotFound();
        }

        await this.userHelper.AddUserToRoleAsync(user, "Admin");
        return this.RedirectToAction(nameof(Index));
}

public async Task<IActionResult> DeleteUser(string id)
{
        if (string.IsNullOrEmpty(id))
        {
        return NotFound();
        }

        var user = await this.userHelper.GetUserByIdAsync(id);
        if (user == null)
        {
        return NotFound();
        }

        await this.userHelper.DeleteUserAsync(user);
        return this.RedirectToAction(nameof(Index));
}
```

4. Finally add the **Index** view:

```
@model IEnumerable<Shop.Web.Data.Entities.User>

@{
        ViewData["Title"] = "Index";
}

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />
<br />

<div class="row">
```

```
<div class="col-md-12">
<div class="panel panel-default">
<div class="panel-heading">
        <h3 class="panel-title">Users</h3>
</div>
<div class="panel-body">
        <table class="table table-hover table-responsive table-striped" id="UsersTable">
        <thead>
        <tr>
                <th>
                @Html.DisplayNameFor(model => model.FirstName)
                </th>
                <th>
                @Html.DisplayNameFor(model => model.LastName)
                </th>
                <th>
                @Html.DisplayNameFor(model => model.Email)
                </th>
                <th>
                @Html.DisplayNameFor(model => model.Address)
                </th>
                <th>
                @Html.DisplayNameFor(model => model.PhoneNumber)
                </th>
                <th>
                @Html.DisplayNameFor(model => model.City.Name)
                </th>
                <th>
                @Html.DisplayNameFor(model => model.IsAdmin)
                </th>
                <th>
                @Html.DisplayNameFor(model => model.EmailConfirmed)
                </th>
                <th></th>
        </tr>
        </thead>
        <tbody>
        @foreach (var item in Model)
        {
        <tr>
                <td>
                @Html.DisplayFor(modelItem => item.FirstName)
                </td>
                <td>
                @Html.DisplayFor(modelItem => item.LastName)
                </td>
                <td>
                @Html.DisplayFor(modelItem => item.Email)
                </td>
                <td>
                @Html.DisplayFor(modelItem => item.Address)
                </td>
                <td>
                @Html.DisplayFor(modelItem => item.PhoneNumber)
                </td>
                <td>
                @Html.DisplayFor(modelItem => item.City.Name)
                </td>
                <td>
                @Html.DisplayFor(modelItem => item.IsAdmin)
                </td>
                <td>
```

```
            @Html.DisplayFor(modelItem => item.EmailConfirmed)
            </td>
            <td>
            <button data-id="@item.Id" class="btn btn-danger deleteItem" data-toggle="modal"
data-target="#deleteDialog">Delete</button>
            @if (item.IsAdmin)
            {
            <a asp-action="AdminOff" asp-route-id="@item.Id" class="btn btn-warning">Admin Off</a>
            }
            else
            {
            <a asp-action="AdminOn" asp-route-id="@item.Id" class="btn btn-primary">Admin On</a>
            }
            </td>
        </tr>
        }
        </tbody>
        </table>
    </div>
    </div>
    </div>
</div>

<div class="modal fade" id="deleteDialog" tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel" aria-hidden="true">
    <div class="modal-dialog" role="document">
    <div class="modal-content">
    <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel">Delete Item</h5>
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
        <span aria-hidden="true">&times;</span>
        </button>
    </div>
    <div class="modal-body">
        <p>Do you want to delete the user?</p>
    </div>
    <div class="modal-footer">
        <button type="button" class="btn btn-primary" data-dismiss="modal">Close</button>
        <button type="button" class="btn btn-danger" id="btnYesDelete">Delete</button>
    </div>
    </div>
    </div>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
    <script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>
    <script type="text/javascript">
    $(document).ready(function () {
    $('#UsersTable').DataTable();

    // Delete item
    var item_to_delete;

    $('.deleteItem').click((e) => {
        item_to_delete = e.currentTarget.dataset.id;
    });

    $("#btnYesDelete").click(function () {
        window.location.href = '/Account/DeleteUser/' + item_to_delete;
    });
    });
    </script>
}
```

5. Add the new entry in menu.

```
@if (this.User.Identity.IsAuthenticated)
{
        <li><a asp-area="" asp-controller="Products" asp-action="Index">Products</a></li>
        @if (this.User.IsInRole("Admin"))
        {
        <li><a asp-area="" asp-controller="Countries" asp-action="Index">Countries</a></li>
        <li><a asp-area="" asp-controller="Account" asp-action="Index">Users</a></li>
        }
        <li><a asp-area="" asp-controller="Orders" asp-action="Index">Orders</a></li>
}
```

6. Test it.

Login in Xamarin Forms

1. Add the class **TokenRequest** (in **Common.Models**):

```
public class TokenRequest
{
        public string Username { get; set; }

        public string Password { get; set; }
}
```

7. Add the class **TokenResponse** (in **Common.Models**):

```
using System;
using Newtonsoft.Json;

public class TokenResponse
{
        [JsonProperty("token")]
        public string Token { get; set; }

        [JsonProperty("expiration")]
        public DateTime Expiration { get; set; }
}
```

1. In the **ApiService** add the methods **GetTokenAsync** and overload the method **GetListAsync**:

```
public async Task<Response> GetListAsync<T>(
        string urlBase,
        string servicePrefix,
        string controller,
        string tokenType,
        string accessToken)
{
        try
        {
        var client = new HttpClient
        {
        BaseAddress = new Uri(urlBase),
        };

        client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);
```

```
    var url = $"{servicePrefix}{controller}";
    var response = await client.GetAsync(url);
    var result = await response.Content.ReadAsStringAsync();

    if (!response.IsSuccessStatusCode)
    {
    return new Response
    {
        IsSuccess = false,
        Message = result,
    };
    }

    var list = JsonConvert.DeserializeObject<List<T>>(result);
    return new Response
    {
    IsSuccess = true,
    Result = list
    };
    }
    catch (Exception ex)
    {
    return new Response
    {
    IsSuccess = false,
    Message = ex.Message
    };
    }
}

public async Task<Response> GetTokenAsync(
    string urlBase,
    string servicePrefix,
    string controller,
    TokenRequest request)
{
    try
    {
    var requestString = JsonConvert.SerializeObject(request);
    var content = new StringContent(requestString, Encoding.UTF8, "application/json");
    var client = new HttpClient
    {
    BaseAddress = new Uri(urlBase)
    };

    var url = $"{servicePrefix}{controller}";
    var response = await client.PostAsync(url, content);
    var result = await response.Content.ReadAsStringAsync();

    if (!response.IsSuccessStatusCode)
    {
    return new Response
    {
        IsSuccess = false,
        Message = result,
    };
    }

    var token = JsonConvert.DeserializeObject<TokenResponse>(result);
    return new Response
    {
    IsSuccess = true,
    Result = token
    };
    }
    catch (Exception ex)
    {
    return new Response
    {
    IsSuccess = false,
    Message = ex.Message
    };
    }
}
```

2. Modify the **LoginPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="Shop.UIForms.Views.LoginPage"
    BindingContext="{Binding Main, Source={StaticResource Locator}}"
    Title="Login">
    <ContentPage.Content>
    <ScrollView
    BindingContext="{Binding Login}">
    <StackLayout
        Padding="5">
        <Label
        Text="Email">
        </Label>
        <Entry
        Keyboard="Email"
        Placeholder="Enter your email..."
        Text="{Binding Email}">
        </Entry>
        <Label
        Text="Password">
        </Label>
        <Entry
        IsPassword="True"
        Placeholder="Enter your password..."
        Text="{Binding Password}">
        </Entry>
        <ActivityIndicator
        IsRunning="{Binding IsRunning}">
        </ActivityIndicator>
        <Button
        Command="{Binding LoginCommand}"
        IsEnabled="{Binding IsEnabled}"
        Text="Login">
        </Button>
    </StackLayout>
    </ScrollView>
    </ContentPage.Content>
</ContentPage>
```

3. Add the property **Token** to **MainViewModel**:

```
public TokenResponse Token { get; set; }
```

4. Add the new value into resource dictionary:

```
<!-- Locator -->
```

```
<infra:InstanceLocator x:Key="Locator"/>

<!-- Parameters -->
<x:String x:Key="UrlAPI">https://shopzulu.azurewebsites.net</x:String>
```

5. Modify the **LoginViewModel**:

```
using Common.Services;
using GalaSoft.MvvmLight.Command;
using Shop.Common.Models;
using System.Windows.Input;
using Views;
using Xamarin.Forms;

public class LoginViewModel : BaseViewModel
{
        private bool isRunning;
        private bool isEnabled;
        private readonly ApiService apiService;

        public bool IsRunning
        {
        get => this.isRunning;
        set => this.SetValue(ref this.isRunning, value);
        }

        public bool IsEnabled
        {
        get => this.isEnabled;
        set => this.SetValue(ref this.isEnabled, value);
```

```
}

public string Email { get; set; }

public string Password { get; set; }

public ICommand LoginCommand => new RelayCommand(this.Login);

public LoginViewModel()
{
this.apiService = new ApiService();
this.IsEnabled = true;
this.Email = "jzuluaga55@gmail.com";
this.Password = "123456";
}

private async void Login()
{
if (string.IsNullOrEmpty(this.Email))
{
await Application.Current.MainPage.DisplayAlert("Error", "You must enter an email.", "Accept");
return;
}

if (string.IsNullOrEmpty(this.Password))
{
await Application.Current.MainPage.DisplayAlert("Error", "You must enter a password.", "Accept");
return;
}

this.IsRunning = true;
```

```
    this.IsEnabled = false;

    var request = new TokenRequest
    {
    Password = this.Password,
    Username = this.Email
    };

    var url = Application.Current.Resources["UrlAPI"].ToString();
    var response = await this.apiService.GetTokenAsync(
    url,
    "/Account",
    "/CreateToken",
    request);

    this.IsRunning = false;
    this.IsEnabled = true;

    if (!response.IsSuccess)
    {
    await Application.Current.MainPage.DisplayAlert("Error", "Email or password incorrect.", "Accept");
    return;
    }

    var token = (TokenResponse)response.Result;
    var mainViewModel = MainViewModel.GetInstance();
    mainViewModel.Token = token;
    mainViewModel.Products = new ProductsViewModel();
    await Application.Current.MainPage.Navigation.PushAsync(new ProductsPage());
    }
}
```

6. Test it.

7. Finally, modify the method **LoadProducts** in **ProductsViewModel**):

```
private async void LoadProducts()
{
    this.IsRefreshing = true;

    var url = Application.Current.Resources["UrlAPI"].ToString();
    var response = await this.apiService.GetListAsync<Product>(
    url,
    "/api",
    "/Products",
    "bearer",
    MainViewModel.GetInstance().Token.Token);



    if (!response.IsSuccess)
    {
    await Application.Current.MainPage.DisplayAlert(
    "Error",
    response.Message,
    "Accept");
    this.IsRefreshing = false;
    return;
    }

    var products = (List<Product>)response.Result;
    this.Products = new ObservableCollection<Product>(products);
```

```
        this.IsRefreshing = false;
}
```

8. Test it.

Master Detail in Xamarin Forms

1. Add a new page call **MenuPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="Shop.UIForms.Views.MenuPage"
        BackgroundColor="BlueViolet"
        BindingContext="{Binding Main, Source={StaticResource Locator}}"
        Title="Menu">
        <ContentPage.Content>
        <StackLayout>
        <Label
                TextColor="White"
                Text="Menu"
                VerticalOptions="CenterAndExpand"
                HorizontalOptions="CenterAndExpand" />
        </StackLayout>
        </ContentPage.Content>
</ContentPage>
```

2. Add a new page call **MasterPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
```

```
<MasterDetailPage xmlns="http://xamarin.com/schemas/2014/forms"
                  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                  xmlns:pages="clr-namespace:Shop.UIForms.Views"
                  x:Class="Shop.UIForms.Views.MasterPage">
        <MasterDetailPage.Master>
        <pages:MenuPage/>
        </MasterDetailPage.Master>
        <MasterDetailPage.Detail>
        <NavigationPage x:Name="Navigator">
        <x:Arguments>
                  <pages:ProductsPage/>
        </x:Arguments>
        </NavigationPage>
        </MasterDetailPage.Detail>
</MasterDetailPage>
```

3. Modify the code behind **MasterPage.xaml.cs**:

```
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

[XamlCompilation(XamlCompilationOptions.Compile)]
public partial class MasterPage : MasterDetailPage
{
        public MasterPage()
        {
        InitializeComponent();
        }

        protected override void OnAppearing()
        {
```

```
        base.OnAppearing();
        App.Navigator = this.Navigator;
        }
}
```

4. Add the **Navigator** property.

5. Modify the **LoginViewModel**:

```
var token = (TokenResponse)response.Result;
var mainViewModel = MainViewModel.GetInstance();
mainViewModel.Token = token;
mainViewModel.Products = new ProductsViewModel();
Application.Current.MainPage = new MasterPage();
```

6. Test it, that we do until the moment.

7. Add icons for: **About**, **Setup**, **Exit** and an image for the solution. I recommend Android Asset Studio. And add those icons in their corresponding folder by the platform.

8. Add the **Menu** mode (in **Common.Models**):

```
public class Menu
{
        public string Icon { get; set; }

        public string Title { get; set; }

        public string PageName { get; set; }
}
```

9. Add a new page call **AboutPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="Shop.UIForms.Views.AboutPage"
        Title="About">
        <ContentPage.Content>
        <StackLayout>
        <Label Text="About"
                VerticalOptions="CenterAndExpand"
                HorizontalOptions="CenterAndExpand" />
        </StackLayout>
        </ContentPage.Content>
</ContentPage>
```

10. Add a new page call **SetupPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="Shop.UIForms.Views.SetupPage"
        Title="Setup">
        <ContentPage.Content>
        <StackLayout>
        <Label Text="Setup"
                VerticalOptions="CenterAndExpand"
                HorizontalOptions="CenterAndExpand" />
        </StackLayout>
        </ContentPage.Content>
</ContentPage>
```

11. Modify the **MenuPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="Shop.UIForms.Views.MenuPage"
    BackgroundColor="Gainsboro"
    BindingContext="{Binding Main, Source={StaticResource Locator}}"
    Title="Menu">
    <ContentPage.Content>
    <StackLayout
    Padding="10">
    <Image
        HeightRequest="150"
        Source="shop">
    </Image>
    <ListView
        ItemsSource="{Binding Menus}"
        HasUnevenRows="True"
        SeparatorVisibility="None">
        <ListView.ItemTemplate>
        <DataTemplate>
        <ViewCell>
            <Grid>
            <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"></ColumnDefinition>
            <ColumnDefinition Width="*"></ColumnDefinition>
            </Grid.ColumnDefinitions>
            <Image
            Grid.Column="0"
            HeightRequest="50"
            Source="{Binding Icon}"
            WidthRequest="50">
            </Image>
            <Label
            Grid.Column="1"
            FontAttributes="Bold"
            VerticalOptions="Center"
            TextColor="White"
            Text="{Binding Title}">
            </Label>
            </Grid>
        </ViewCell>
        </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
    </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

12. Modify the **LoginPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="Shop.UIForms.Views.LoginPage"
    BindingContext="{Binding Main, Source={StaticResource Locator}}"
    Title="Login">
    <ContentPage.Content>
    <ScrollView
    BindingContext="{Binding Login}">
```

```
<StackLayout
    Padding="10">
    <Image
    HeightRequest="150"
    Source="shop">
    </Image>
    <Label
    Text="Email">
    </Label>
    <Entry
    Keyboard="Email"
    Placeholder="Enter your email..."
    Text="{Binding Email}">
    </Entry>
    <Label
    Text="Password">
    </Label>
    <Entry
    IsPassword="True"
    Placeholder="Enter your password..."
    Text="{Binding Password}">
    </Entry>
    <ActivityIndicator
    IsRunning="{Binding IsRunning}"
    VerticalOptions="CenterAndExpand">
    </ActivityIndicator>
    <Button
    BackgroundColor="Navy"
    BorderRadius="23"
    Command="{Binding LoginCommand}"
    HeightRequest="46"
```

```
        IsEnabled="{Binding IsEnabled}"
        Text="Login"
        TextColor="White">
        </Button>
    </StackLayout>
    </ScrollView>
    </ContentPage.Content>
</ContentPage>
```

13. Test it:

14. Modify the **MenuPage**:

```
<Grid>
    <Grid.GestureRecognizers>
    <TapGestureRecognizer Command="{Binding SelectMenuCommand}"/>
    </Grid.GestureRecognizers>
    <Grid.ColumnDefinitions>
```

15. Add the **MenuItemViewModel**:

```
using System.Windows.Input;
using GalaSoft.MvvmLight.Command;
using Views;
using Xamarin.Forms;

public class MenuItemViewModel : Common.Models.Menu
{
    public ICommand SelectMenuCommand => new RelayCommand(this.SelectMenu);

    private async void SelectMenu()
    {
    var mainViewModel = MainViewModel.GetInstance();

    switch (this.PageName)
    {
    case "AboutPage":
        await App.Navigator.PushAsync(new AboutPage());
        break;
    case "SetupPage":
        await App.Navigator.PushAsync(new SetupPage());
        break;
    default:
        MainViewModel.GetInstance().Login = new LoginViewModel();
        Application.Current.MainPage = new NavigationPage(new LoginPage());
        break;
    }
    }
}
```

16. Modify the **MainViewModel**:

```
…
public ObservableCollection<MenuItemViewModel> Menus { get; set; }
…
public MainViewModel()
{
    instance = this;
    this.LoadMenus();
}
...
private void LoadMenus()
{
    var menus = new List<Menu>
    {
    new Menu
    {
    Icon = "ic_info",
    PageName = "AboutPage",
    Title = "About"
    },

    new Menu
    {
    Icon = "ic_phonelink_setup",
    PageName = "SetupPage",
    Title = "Setup"
    },

    new Menu
    {
    Icon = "ic_exit_to_app",
    PageName = "LoginPage",
    Title = "Close session"
    }
    };

    this.Menus = new ObservableCollection<MenuItemViewModel>(menus.Select(m => new MenuItemViewModel
    {
    Icon = m.Icon,
    PageName = m.PageName,
    Title = m.Title
    }).ToList());
}
```

17. Test it, that we do until the moment.

18. Modify the **MasterPage.xaml.cs**:

```
protected override void OnAppearing()
{
        base.OnAppearing();
        App.Navigator = this.Navigator;
        App.Master = this;
}
```

19. Add the property in **App**:

```
public static MasterPage Master { get; internal set; }
```

20. Finally add this line in **SelectMenu** in **MenuItemViewModel**.

```
App.Master.IsPresented = false;
```

21. Test it.

Completing the products API

1. Add reference to **Common** project in **Web** project.

2. Fix the **Product** model in **Common.Models**:

```
[JsonProperty("lastPurchase")]
public DateTime? LastPurchase { get; set; }
```

```
[JsonProperty("lastSale")]
public DateTime? LastSale { get; set; }
```

3. Modify those methods in **IGenericRepository**.

```
Task<T> CreateAsync(T entity);

Task<T> UpdateAsync(T entity);
```

And implementation:

```
public async Task<T> CreateAsync(T entity)
{
	await this.context.Set<T>().AddAsync(entity);
	await SaveAllAsync();
	return entity;
}

public async Task<T> UpdateAsync(T entity)
{
	this.context.Set<T>().Update(entity);
	await SaveAllAsync();
	return entity;
}
```

4. Add those methods to **Products** API Controller.

```
[HttpPost]
public async Task<IActionResult> PostProduct([FromBody] Common.Models.Product product)
{
        if (!ModelState.IsValid)
        {
        return this.BadRequest(ModelState);
        }

        var user = await this.userHelper.GetUserByEmailAsync(product.User.Email);
        if (user == null)
        {
        return this.BadRequest("Invalid user");
        }

        //TODO: Upload images
        var entityProduct = new Product
        {
        IsAvailabe = product.IsAvailabe,
        LastPurchase = product.LastPurchase,
        LastSale = product.LastSale,
        Name = product.Name,
        Price = product.Price,
        Stock = product.Stock,
        User = user
        };

        var newProduct = await this.productRepository.CreateAsync(entityProduct);
        return Ok(newProduct);
}

[HttpPut("{id}")]
public async Task<IActionResult> PutProduct([FromRoute] int id, [FromBody] Common.Models.Product product)
{
        if (!ModelState.IsValid)
        {
        return this.BadRequest(ModelState);
        }

        if (id != product.Id)
        {
        return BadRequest();
        }

        var oldProduct = await this.productRepository.GetByIdAsync(id);
        if (oldProduct == null)
        {
        return this.BadRequest("Product Id don't exists.");
        }

        //TODO: Upload images
        oldProduct.IsAvailabe = product.IsAvailabe;
        oldProduct.LastPurchase = product.LastPurchase;
        oldProduct.LastSale = product.LastSale;
        oldProduct.Name = product.Name;
        oldProduct.Price = product.Price;
        oldProduct.Stock = product.Stock;

        var updatedProduct =  await this.productRepository.UpdateAsync(oldProduct);
        return Ok(updatedProduct);
}

[HttpDelete("{id}")]
public async Task<IActionResult> DeleteProduct([FromRoute] int id)
{
        if (!ModelState.IsValid)
        {
        return this.BadRequest(ModelState);
        }

        var product = await this.productRepository.GetByIdAsync(id);
        if (product == null)
        {
        return this.NotFound();
        }

        await this.productRepository.DeleteAsync(product);
        return Ok(product);
}
```

5. Test it in PostMan.

6. Publish the changes in Azure.

Completing the CRUD in Xamarin Forms

1. Add the method **PostAsync** to **ApiService**:

```
public async Task<Response> PostAsync<T>(
        string urlBase,
        string servicePrefix,
        string controller,
        T model,
        string tokenType,
        string accessToken)
{
```

```
try
{
var request = JsonConvert.SerializeObject(model);
var content = new StringContent(request, Encoding.UTF8, "application/json");
var client = new HttpClient
{
BaseAddress = new Uri(urlBase)
};

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);
var url = $"{servicePrefix}{controller}";
var response = await client.PostAsync(url, content);
var answer = await response.Content.ReadAsStringAsync();
if (!response.IsSuccessStatusCode)
{
return new Response
{
        IsSuccess = false,
        Message = answer,
};
}

var obj = JsonConvert.DeserializeObject<T>(answer);
return new Response
{
IsSuccess = true,
Result = obj,
};
}
catch (Exception ex)
{
        return new Response
        {
        IsSuccess = false,
        Message = ex.Message,
        };
        }
}
```

2. Modify the **ProductsPage** to add the icon in the title bar:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="Shop.UIForms.Views.ProductsPage"
        BindingContext="{Binding Main, Source={StaticResource Locator}}"
        Title="Products">
        <ContentPage.ToolbarItems>
        <ToolbarItem Icon="ic_action_add_circle" Command="{Binding AddProductCommand}"/>
        </ContentPage.ToolbarItems>
        <ContentPage.Content>
...
```

3. Add the **AddProductPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="Shop.UIForms.Views.AddProductPage"
        BindingContext="{Binding Main, Source={StaticResource Locator}}"
        Title="Add Product">
        <ContentPage.Content>
```

```
<ScrollView
BindingContext="{Binding AddProduct}">
<StackLayout
        Padding="10">
        <Image
        HeightRequest="150"
        Source="{Binding Image}">
        </Image>
        <Label
        FontSize="Micro"
        HorizontalOptions="Center"
        Text="Tap the image to change it...">
        </Label>
        <Grid>
        <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <Label
        Grid.Column="0"
        Grid.Row="0"
        Text="Name"
        VerticalOptions="Center">
        </Label>
        <Entry
        Grid.Column="1"
        Grid.Row="0"
        Placeholder="Product name..."
        Text="{Binding Name}">
        </Entry>
        <Label
```

```
          Grid.Column="0"
          Grid.Row="1"
          Text="Price"
          VerticalOptions="Center">
          </Label>
          <Entry
          Grid.Column="1"
          Grid.Row="1"
          Keyboard="Numeric"
          Placeholder="Product price..."
          Text="{Binding Price}">
          </Entry>
          </Grid>
          <ActivityIndicator
          IsRunning="{Binding IsRunning}"
          VerticalOptions="CenterAndExpand">
          </ActivityIndicator>
          <Button
          BackgroundColor="Navy"
          BorderRadius="23"
          Command="{Binding SaveCommand}"
          HeightRequest="46"
          IsEnabled="{Binding IsEnabled}"
          Text="Save"
          TextColor="White">
          </Button>
     </StackLayout>
     </ScrollView>
     </ContentPage.Content>
</ContentPage>
```

4. Modify the **MainViewModel** to add the logged user and password:

```
public string UserEmail { get; set; }

public string UserPassword { get; set; }
```

5. Modify the **LoginViewModel** to storage the user email when this is logged in:

```
mainViewModel.Products = new ProductsViewModel();
mainViewModel.UserEmail = this.Email;
mainViewModel.UserPassword = this.Password;
Application.Current.MainPage = new MasterPage();
```

6. Add the **AddProductViewModel**:

```
using System.Windows.Input;
using Common.Models;
using Common.Services;
using GalaSoft.MvvmLight.Command;
using Xamarin.Forms;

public class AddProductViewModel : BaseViewModel
{
        private bool isRunning;
        private bool isEnabled;
        private readonly ApiService apiService;

        public string Image { get; set; }

        public bool IsRunning
```

```
{
get => this.isRunning;
set => this.SetValue(ref this.isRunning, value);
}

public bool IsEnabled
{
get => this.isEnabled;
set => this.SetValue(ref this.isEnabled, value);
}

public string Name { get; set; }

public string Price { get; set; }

public ICommand SaveCommand => new RelayCommand(this.Save);

public AddProductViewModel()
{
this.apiService = new ApiService();
this.Image = "noImage";
this.IsEnabled = true;
}

private async void Save()
{
if (string.IsNullOrEmpty(this.Name))
{
await Application.Current.MainPage.DisplayAlert("Error", "You must enter a product name.", "Accept");
return;
}
```

```
if (string.IsNullOrEmpty(this.Price))
{
await Application.Current.MainPage.DisplayAlert("Error", "You must enter a product price.", "Accept");
return;
}

var price = decimal.Parse(this.Price);
if (price <= 0)
{
await Application.Current.MainPage.DisplayAlert("Error", "The price must be a number greather than zero.", "Accept");
return;
}

this.IsRunning = true;
this.IsEnabled = false;

//TODO: Add image
var product = new Product
{
IsAvailabe = true,
Name = this.Name,
Price = price,
User = new User { UserName = MainViewModel.GetInstance().UserEmail }
};

var url = Application.Current.Resources["UrlAPI"].ToString();
var response = await this.apiService.PostAsync(
url,
"/api",
"/Products",
    product,
    "bearer",
    MainViewModel.GetInstance().Token.Token);

    if (!response.IsSuccess)
    {
    await Application.Current.MainPage.DisplayAlert("Error", response.Message, "Accept");
    return;
    }

    var newProduct = (Product)response.Result;
    MainViewModel.GetInstance().Products.Products.Add(newProduct);

    this.IsRunning = false;
    this.IsEnabled = true;
    await App.Navigator.PopAsync();
    }
}
```

7. Modify the **MainViewModel** adding the property **AddProduct**, **AddProductCommand** and **GoAddProduct** method:

```
public AddProductViewModel AddProduct { get; set; }

public ICommand AddProductCommand => new RelayCommand(this.GoAddProduct);
…
private async void GoAddProduct()
{
    this.AddProduct = new AddProductViewModel();
    await App.Navigator.PushAsync(new AddProductPage());
}
```

8. Add an image for “no image”.

9. Modify the property **ImageFullPath** in model **Product** (in **Common.Models**):

```
[JsonProperty("imageFullPath")]
public string ImageFullPath { get; set; }
```

10. Modify this line at the end of **ProductsViewModel** :

```
var myProducts = (List<Product>)response.Result;
this.Products = new ObservableCollection<Product>(myProducts.OrderBy(p => p.Name));
```

11. Test it.

12. Now we implement the update and delete operations. Add this methods to the API controller:

```
public async Task<Response> PutAsync<T>(
        string urlBase,
        string servicePrefix,
        string controller,
        int id,
        T model,
        string tokenType,
        string accessToken)
{
        try
        {
        var request = JsonConvert.SerializeObject(model);
        var content = new StringContent(request, Encoding.UTF8, "application/json");
        var client = new HttpClient
        {
```

```
BaseAddress = new Uri(urlBase)
};

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);
var url = $"{servicePrefix}{controller}/{id}";
var response = await client.PutAsync(url, content);
var answer = await response.Content.ReadAsStringAsync();
if (!response.IsSuccessStatusCode)
{
return new Response
{
        IsSuccess = false,
        Message = answer,
};
}

var obj = JsonConvert.DeserializeObject<T>(answer);
return new Response
{
IsSuccess = true,
Result = obj,
};
}
catch (Exception ex)
{
return new Response
{
IsSuccess = false,
Message = ex.Message,
};
}
```

```
}

public async Task<Response> DeleteAsync(
        string urlBase,
        string servicePrefix,
        string controller,
        int id,
        string tokenType,
        string accessToken)
{
        try
        {
        var client = new HttpClient
        {
        BaseAddress = new Uri(urlBase)
        };

        client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);
        var url = $"{servicePrefix}{controller}/{id}";
        var response = await client.DeleteAsync(url);
        var answer = await response.Content.ReadAsStringAsync();
        if (!response.IsSuccessStatusCode)
        {
        return new Response
        {
                IsSuccess = false,
                Message = answer,
        };
        }

        return new Response
        {
        IsSuccess = true
        };
        }
        catch (Exception ex)
        {
        return new Response
        {
        IsSuccess = false,
        Message = ex.Message,
        };
        }
}
```

13. Add the **EditProductPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="Shop.UIForms.Views.EditProductPage"
        BindingContext="{Binding Main, Source={StaticResource Locator}}"
        Title="Edit Product">
        <ContentPage.Content>
        <ScrollView
        BindingContext="{Binding EditProduct}">
        <StackLayout
                Padding="10">
                <Image
                HeightRequest="150"
                Source="{Binding Product.ImageFullPath}">
                </Image>
```

```
<Label
FontSize="Micro"
HorizontalOptions="Center"
Text="Tap the image to change it...">
</Label>
<Grid>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto"/>
<ColumnDefinition Width="*"/>
</Grid.ColumnDefinitions>
<Label
Grid.Column="0"
Grid.Row="0"
Text="Name"
VerticalOptions="Center">
</Label>
<Entry
Grid.Column="1"
Grid.Row="0"
Placeholder="Product name..."
Text="{Binding Product.Name}">
</Entry>
<Label
Grid.Column="0"
Grid.Row="1"
Text="Price"
VerticalOptions="Center">
</Label>
<Entry
Grid.Column="1"
Grid.Row="1"
```

```
Keyboard="Numeric"
Placeholder="Product price..."
Text="{Binding Product.Price}">
</Entry>
</Grid>
<ActivityIndicator
IsRunning="{Binding IsRunning}"
VerticalOptions="CenterAndExpand">
</ActivityIndicator>
<StackLayout
Orientation="Horizontal">
<Button
BackgroundColor="Navy"
BorderRadius="23"
Command="{Binding SaveCommand}"
HeightRequest="46"
HorizontalOptions="FillAndExpand"
IsEnabled="{Binding IsEnabled}"
Text="Save"
TextColor="White">
</Button>
<Button
BackgroundColor="Red"
BorderRadius="23"
Command="{Binding DeleteCommand}"
HeightRequest="46"
HorizontalOptions="FillAndExpand"
IsEnabled="{Binding IsEnabled}"
Text="Delete"
TextColor="White">
</Button>
```

```
                    </StackLayout>
            </StackLayout>
            </ScrollView>
            </ContentPage.Content>
</ContentPage>
```

14. Add the **ProductItemViewModel**:

```
using System.Windows.Input;
using Common.Models;
using GalaSoft.MvvmLight.Command;
using Views;

public class ProductItemViewModel : Product
{
            public ICommand SelectProductCommand => new RelayCommand(this.SelectProduct);

            private async void SelectProduct()
            {
            MainViewModel.GetInstance().EditProduct = new EditProductViewModel(this);
            await App.Navigator.PushAsync(new EditProductPage());
            }
}
```

15. Modify the **ProductsViewModel** to change the observable collection:

```
…
private List<Product> myProducts;
private ObservableCollection<ProductItemViewModel> products;
…
public ObservableCollection<ProductItemViewModel> Products
```

```
{
        get => this.products;
        set => this.SetValue(ref this.products, value);
}
…
        if (!response.IsSuccess)
        {
        await Application.Current.MainPage.DisplayAlert(
        "Error",
        response.Message,
        "Accept");
        return;
        }

        this.myProducts = (List<Product>)response.Result;
        this.RefresProductsList();
}

public void AddProductToList(Product product)
{
        this.myProducts.Add(product);
        this.RefresProductsList();
}

public void UpdateProductInList(Product product)
{
        var previousProduct = this.myProducts.Where(p => p.Id == product.Id).FirstOrDefault();
        if (previousProduct != null)
        {
        this.myProducts.Remove(previousProduct);
        }
        this.myProducts.Add(product);
        this.RefresProductsList();
}

public void DeleteProductInList(int productId)
{
        var previousProduct = this.myProducts.Where(p => p.Id == productId).FirstOrDefault();
        if (previousProduct != null)
        {
        this.myProducts.Remove(previousProduct);
        }

        this.RefresProductsList();
}

private void RefresProductsList()
{
        this.Products = new ObservableCollection<ProductItemViewModel>(myProducts.Select(p => new ProductItemViewModel
        {
        Id = p.Id,
        ImageUrl = p.ImageUrl,
        ImageFullPath = p.ImageFullPath,
        IsAvailabe = p.IsAvailabe,
        LastPurchase = p.LastPurchase,
        LastSale = p.LastSale,
        Name = p.Name,
        Price = p.Price,
        Stock = p.Stock,
        User = p.User
        })
        .OrderBy(p => p.Name)
        .ToList());
}
```

16. Add modify the grid definition in **ProductsPage**:

```
<Grid>
        <Grid.GestureRecognizers>
        <TapGestureRecognizer Command="{Binding SelectProductCommand}"/>
        </Grid.GestureRecognizers>
        <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="Auto"/>
        </Grid.ColumnDefinitions>
        <Image
        Grid.Column="0"
        Source="{Binding ImageFullPath}"
        WidthRequest="100">
        </Image>
        <StackLayout
        Grid.Column="1"
        VerticalOptions="Center">
        <Label
        FontAttributes="Bold"
        FontSize="Medium"
        Text="{Binding Name}"
        TextColor="Black">
        </Label>
        <Label
        Text="{Binding Price, StringFormat='Price: {0:C2}'}"
        TextColor="Navy">
        </Label>
        <Label
        Text="{Binding Stock, StringFormat='Stock: {0:N2}'}"
        TextColor="Black">
        </Label>
        </StackLayout>
        <Image
        Grid.Column="2"
        Source="ic_action_chevron_right">
        </Image>
</Grid>
```

17. Test it.

18. Add the **EditProductViewModel**:

```
using System.Linq;
using System.Windows.Input;
using Common.Models;
using Common.Services;
using GalaSoft.MvvmLight.Command;
using Xamarin.Forms;

public class EditProductViewModel : BaseViewModel
{
        private bool isRunning;
        private bool isEnabled;
        private readonly ApiService apiService;

        public Product Product { get; set; }
        public bool IsRunning
        {
        get => this.isRunning;
        set => this.SetValue(ref this.isRunning, value);
        }

        public bool IsEnabled
        {
        get => this.isEnabled;
        set => this.SetValue(ref this.isEnabled, value);
        }

        public ICommand SaveCommand => new RelayCommand(this.Save);

        public ICommand DeleteCommand => new RelayCommand(this.Delete);

        public EditProductViewModel(Product product)
        {
        this.Product = product;
        this.apiService = new ApiService();
        this.IsEnabled = true;
        }

        private async void Save()
        {
        if (string.IsNullOrEmpty(this.Product.Name))
        {
        await Application.Current.MainPage.DisplayAlert("Error", "You must enter a product name.", "Accept");
        return;
        }
```

```
if (this.Product.Price <= 0)
{
await Application.Current.MainPage.DisplayAlert("Error", "The price must be a number greather than zero.", "Accept");
return;
}

this.IsRunning = true;
this.IsEnabled = false;

var url = Application.Current.Resources["UrlAPI"].ToString();
var response = await this.apiService.PutAsync(
url,
"/api",
"/Products",
this.Product.Id,
this.Product,
"bearer",
MainViewModel.GetInstance().Token.Token);

this.IsRunning = false;
this.IsEnabled = true;

if (!response.IsSuccess)
{
await Application.Current.MainPage.DisplayAlert("Error", response.Message, "Accept");
return;
}

var modifiedProduct = (Product)response.Result;
MainViewModel.GetInstance().Products.UpdateProductInList(modifiedProduct);
```

```
    await App.Navigator.PopAsync();
    }

private async void Delete()
{
    var confirm = await Application.Current.MainPage.DisplayAlert("Confirm", "Are you sure to delete the product?", "Yes", "No");
    if (!confirm)
    {
    return;
    }

    this.IsRunning = true;
    this.IsEnabled = false;

    var url = Application.Current.Resources["UrlAPI"].ToString();
    var response = await this.apiService.DeleteAsync(
    url,
    "/api",
    "/Products",
    this.Product.Id,
    "bearer",
    MainViewModel.GetInstance().Token.Token);

    this.IsRunning = false;
    this.IsEnabled = true;

    if (!response.IsSuccess)
    {
    await Application.Current.MainPage.DisplayAlert("Error", response.Message, "Accept");
    return;
    }
```

```
        MainViewModel.GetInstance().Products.DeleteProductInList(this.Product.Id);
        await App.Navigator.PopAsync();
}
}
```

19. Add the property in the **MainViewModel**:

```
public EditProductViewModel EditProduct { get; set; }
```

20. Test the update and delete operations.

Implementing Settings in Xamarin Forms

1. Add the NuGet **Xam.Plugins.Settings**, in **Common** project.

2. Add the folder **Helpers** (in Common.Helpers), and inside it, add the class **Settings**:

```
using Plugin.Settings;
using Plugin.Settings.Abstractions;

public static class Settings
{
        private const string token = "token";
        private const string userEmail = "userEmail";
        private const string userPassword = "userPassword";
        private const string isRemember = "isRemember";
        private static readonly string stringDefault = string.Empty;
        private static readonly bool boolDefault = false;
```

```
    private static ISettings AppSettings => CrossSettings.Current;

    public static string Token
    {
    get => AppSettings.GetValueOrDefault(token, stringDefault);
    set => AppSettings.AddOrUpdateValue(token, value);
    }

    public static string UserEmail
    {
    get => AppSettings.GetValueOrDefault(userEmail, stringDefault);
    set => AppSettings.AddOrUpdateValue(userEmail, value);
    }

    public static string UserPassword
    {
    get => AppSettings.GetValueOrDefault(userPassword, stringDefault);
    set => AppSettings.AddOrUpdateValue(userPassword, value);
    }

    public static bool IsRemember
    {
    get => AppSettings.GetValueOrDefault(isRemember, boolDefault);
    set => AppSettings.AddOrUpdateValue(isRemember, value);
    }
}
```

3. Modify the **LoginPage**:

```
<Entry
    IsPassword="True"
```

```
          Placeholder="Enter your password..."
          Text="{Binding Password}">
</Entry>
<StackLayout
          HorizontalOptions="Center"
          Orientation="Horizontal">
          <Label
          Text="Rememberme in this device"
          VerticalOptions="Center">
          </Label>
          <Switch
          IsToggled="{Binding IsRemember}">
          </Switch>
</StackLayout>
<ActivityIndicator
          IsRunning="{Binding IsRunning}"
          VerticalOptions="CenterAndExpand">
</ActivityIndicator>
```

4. Modify the **LoginViewModel**.

```
public bool IsRemember { get; set; }
…
public LoginViewModel()
{
          this.apiService = new ApiService();
          this.IsEnabled = true;
          this.IsRemember = true;
}
…
var token = (TokenResponse)response.Result;
var mainViewModel = MainViewModel.GetInstance();
mainViewModel.Token = token;
mainViewModel.UserEmail = this.Email;
mainViewModel.Products = new ProductsViewModel();

Settings.IsRemember = this.IsToggled;
Settings.UserEmail = this.Email;
Settings.UserPassword = this.Password;
Settings.Token = JsonConvert.SerializeObject(token);

Application.Current.MainPage = new MasterPage();
```

5. Modify the **App.xaml.cs**.

```
public App()
{
        InitializeComponent();

        if (Settings.IsRemember)
        {
        var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);
        if (token.Expiration > DateTime.Now)
        {
        var mainViewModel = MainViewModel.GetInstance();
        mainViewModel.Token = token;
        mainViewModel.UserEmail = Settings.UserEmail;
        mainViewModel.UserPassword = Settings.UserPassword;
        mainViewModel.Products = new ProductsViewModel();
        this.MainPage = new MasterPage();
        return;
        }
```

```
    }

    MainViewModel.GetInstance().Login = new LoginViewModel();
    this.MainPage = new NavigationPage(new LoginPage());
}
```

6. Modify the **MenuItemViewModel**:

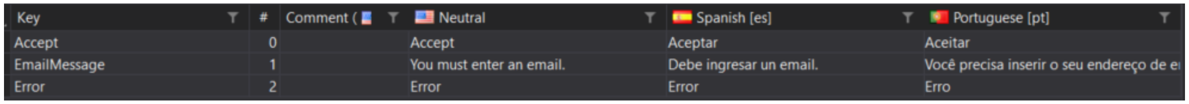
```
default:
    Settings.IsRemember = false;
    Settings.Token = string.Empty;
    Settings.UserEmail = string.Empty;
    Settings.UserPassword = string.Empty;

    MainViewModel.GetInstance().Login = new LoginViewModel();
    Application.Current.MainPage = new NavigationPage(new LoginPage());
    break;
```

7. Test it.

Multi Language in Xamarin Forms

1. If you don’t have the ResX Manager Tool, install from: https://marketplace.visualstudio.com/items?itemName=TomEnglert.ResXManager

2. In shared forms project add the folder **Resources** and inside it, add the resource call **Resource**, add some literals and translate with the ResX Manager tool. The default resource language must be Public, the others in no code generation.



Key	#	Comment (	Neutral	Spanish [es]	Portuguese [pt]
Accept	0		Accept	Aceptar	Aceitar
EmailMessage	1		You must enter an email.	Debe ingresar un email.	Você precisa inserir o seu endereço de e
Error	2		Error	Error	Erro

3. In shared forms project add the folder **Interfaces**, inside it, add the interface **ILocalize**.

```
using System.Globalization;

public interface ILocalize
{
        CultureInfo GetCurrentCultureInfo();

        void SetLocale(CultureInfo ci);
}
```

4. In the folder **Helpers** add the class **PlatformCulture**.

```
using System;

public class PlatformCulture
{
        public string PlatformString { get; private set; }

        public string LanguageCode { get; private set; }

        public string LocaleCode { get; private set; }

        public PlatformCulture(string platformCultureString)
        {
        if (string.IsNullOrEmpty(platformCultureString))
        {
```

```
        throw new ArgumentException("Expected culture identifier", "platformCultureString"); // in C# 6 use
nameof(platformCultureString)
        }

        PlatformString = platformCultureString.Replace("_", "-"); // .NET expects dash, not underscore
        var dashIndex = PlatformString.IndexOf("-", StringComparison.Ordinal);
        if (dashIndex > 0)
        {
        var parts = PlatformString.Split('-');
        LanguageCode = parts[0];
        LocaleCode = parts[1];
        }
        else
        {
        LanguageCode = PlatformString;
        LocaleCode = "";
        }
        }

        public override string ToString()
        {
        return PlatformString;
        }
}
```

5. In the same folder add the class **Languages** with the literals.

```
using Interfaces;
using Resources;
using Xamarin.Forms;
```

```
public static class Languages
{
        static Languages()
        {
        var ci = DependencyService.Get<ILocalize>().GetCurrentCultureInfo();
        Resource.Culture = ci;
        DependencyService.Get<ILocalize>().SetLocale(ci);
        }

        public static string Accept => Resource.Accept;

        public static string Error => Resource.Error;

        public static string EmailMessage => Resource.EmailMessage;
}
```

6. Implement the interface in **iOS** in the folder **Implementations**.

```
[assembly: Xamarin.Forms.Dependency(typeof(Shop.UIForms.iOS.Implementations.Localize))]
namespace Shop.UIForms.iOS.Implementations
{
        using System.Globalization;
        using System.Threading;
        using Foundation;
        using Helpers;
        using Interfaces;

        public class Localize : ILocalize
        {
        public CultureInfo GetCurrentCultureInfo()
        {
```

```
var netLanguage = "en";
if (NSLocale.PreferredLanguages.Length > 0)
{
	var pref = NSLocale.PreferredLanguages[0];
	netLanguage = iOSToDotnetLanguage(pref);
}
// this gets called a lot - try/catch can be expensive so consider caching or something
System.Globalization.CultureInfo ci = null;
try
{
	ci = new System.Globalization.CultureInfo(netLanguage);
}
catch (CultureNotFoundException e1)
{
	// iOS locale not valid .NET culture (eg. "en-ES" : English in Spain)
	// fallback to first characters, in this case "en"
	try
	{
	var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));
	ci = new System.Globalization.CultureInfo(fallback);
	}
	catch (CultureNotFoundException e2)
	{
	// iOS language not valid .NET culture, falling back to English
	ci = new System.Globalization.CultureInfo("en");
	}
}
return ci;
}

public void SetLocale(CultureInfo ci)
```

```
{
Thread.CurrentThread.CurrentCulture = ci;
Thread.CurrentThread.CurrentUICulture = ci;
}

string iOSToDotnetLanguage(string iOSLanguage)
{
var netLanguage = iOSLanguage;
//certain languages need to be converted to CultureInfo equivalent
switch (iOSLanguage)
{
        case "ms-MY":   // "Malaysian (Malaysia)" not supported .NET culture
        case "ms-SG":   // "Malaysian (Singapore)" not supported .NET culture
        netLanguage = "ms"; // closest supported
        break;
        case "gsw-CH":  // "Schwiizertüütsch (Swiss German)" not supported .NET culture
        netLanguage = "de-CH"; // closest supported
        break;
        // add more application-specific cases here (if required)
        // ONLY use cultures that have been tested and known to work
}

return netLanguage;
}

string ToDotnetFallbackLanguage(PlatformCulture platCulture)
{
var netLanguage = platCulture.LanguageCode; // use the first part of the identifier (two chars, usually);
switch (platCulture.LanguageCode)
{
        case "pt":
                netLanguage = "pt-PT"; // fallback to Portuguese (Portugal)
                break;
                case "gsw":
                netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app
                break;
                // add more application-specific cases here (if required)
                // ONLY use cultures that have been tested and known to work
        }

        return netLanguage;
        }
        }
}
```

7. Add this lintes into the **info.plist**.

```
<key>CFBundleLocalizations</key>
<array>
        <string>es</string>
        <string>pt</string>
</array>
<key>CFBundleDevelopmentRegion</key>
<string>en</string>
```

8. Implement the interface in **Android** in the folder **Implementations**.

```
[assembly: Xamarin.Forms.Dependency(typeof(Shop.UIForms.Droid.Implementations.Localize))]
namespace Shop.UIForms.Droid.Implementations
{
        using Helpers;
        using Interfaces;
        using System.Globalization;
        using System.Threading;

        public class Localize : ILocalize
        {
        public CultureInfo GetCurrentCultureInfo()
        {
        var netLanguage = "en";
        var androidLocale = Java.Util.Locale.Default;
        netLanguage = AndroidToDotnetLanguage(androidLocale.ToString().Replace("_", "-"));
        // this gets called a lot - try/catch can be expensive so consider caching or something
        System.Globalization.CultureInfo ci = null;
        try
        {
                ci = new System.Globalization.CultureInfo(netLanguage);
        }
        catch (CultureNotFoundException)
        {
                // iOS locale not valid .NET culture (eg. "en-ES" : English in Spain)
                // fallback to first characters, in this case "en"
                try
                {
                var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));
                ci = new System.Globalization.CultureInfo(fallback);
                }
                catch (CultureNotFoundException)
                {
                // iOS language not valid .NET culture, falling back to English
                ci = new System.Globalization.CultureInfo("en");
                }
        }
        return ci;
        }

        public void SetLocale(CultureInfo ci)
        {
        Thread.CurrentThread.CurrentCulture = ci;
        Thread.CurrentThread.CurrentUICulture = ci;
        }

        private string AndroidToDotnetLanguage(string androidLanguage)
        {
        var netLanguage = androidLanguage;
        //certain languages need to be converted to CultureInfo equivalent
        switch (androidLanguage)
        {
                case "ms-BN":   // "Malaysian (Brunei)" not supported .NET culture
                case "ms-MY":   // "Malaysian (Malaysia)" not supported .NET culture
                case "ms-SG":   // "Malaysian (Singapore)" not supported .NET culture
                netLanguage = "ms"; // closest supported
                break;
                case "in-ID":  // "Indonesian (Indonesia)" has different code in  .NET
                netLanguage = "id-ID"; // correct code for .NET
                break;
                case "gsw-CH":  // "Schwiizertüütsch (Swiss German)" not supported .NET culture
                netLanguage = "de-CH"; // closest supported
                break;
                // add more application-specific cases here (if required)
                // ONLY use cultures that have been tested and known to work
        }
        return netLanguage;
        }
```

```
    private string ToDotnetFallbackLanguage(PlatformCulture platCulture)
    {
    var netLanguage = platCulture.LanguageCode; // use the first part of the identifier (two chars, usually);
    switch (platCulture.LanguageCode)
    {
        case "gsw":
        netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app
        break;
        // add more application-specific cases here (if required)
        // ONLY use cultures that have been tested and known to work
    }
    return netLanguage;
    }
    }
}
```

9. Modify the **LoginViewModel**:

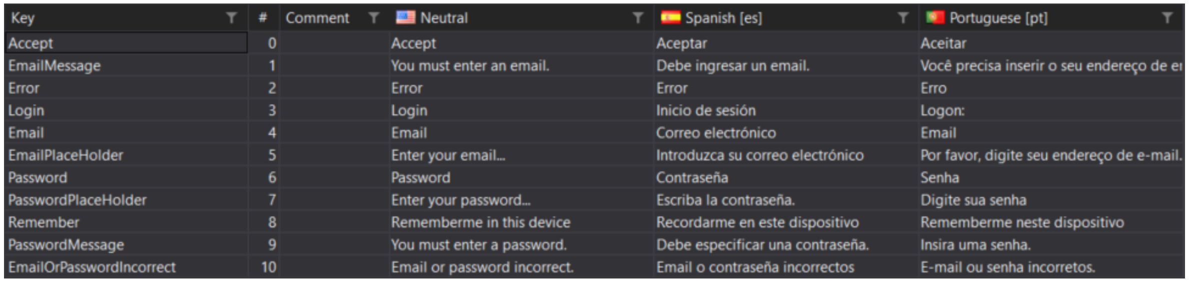
```
if (string.IsNullOrEmpty(this.Email))
{
    await Application.Current.MainPage.DisplayAlert(Languages.Error, Languages.EmailMessage, Languages.Accept);
    return;
}
```

10. Test it.

11. Now to translate literals directly in the XAML add the class **TranslateExtension** in folder **Helpers**:

```
using Interfaces;
using System;
using System.Globalization;
using System.Reflection;
using System.Resources;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

[ContentProperty("Text")]
public class TranslateExtension : IMarkupExtension
{
        private readonly CultureInfo ci;
        private const string ResourceId = "Shop.UIForms.Resources.Resource";
        private static readonly Lazy<ResourceManager> ResMgr =
        new Lazy<ResourceManager>(() => new ResourceManager(
        ResourceId,
        typeof(TranslateExtension).GetTypeInfo().Assembly));

        public TranslateExtension()
        {
        ci = DependencyService.Get<ILocalize>().GetCurrentCultureInfo();
        }

        public string Text { get; set; }

        public object ProvideValue(IServiceProvider serviceProvider)
        {
        if (Text == null)
        {
        return "";
        }

        var translation = ResMgr.Value.GetString(Text, ci);
        if (translation == null)
        {
#if DEBUG
        throw new ArgumentException(
                string.Format(
                "Key '{0}' was not found in resources '{1}' for culture '{2}'.",
                Text, ResourceId, ci.Name), "Text");
#else
        translation = Text; // returns the key, which GETS DISPLAYED TO THE USER
#endif
        }

        return translation;
        }
}
```

12. Complete the literals:



Key	#	Comment	Neutral	Spanish [es]	Portuguese [pt]
Accept	0		Accept	Aceptar	Aceitar
EmailMessage	1		You must enter an email.	Debe ingresar un email.	Você precisa inserir o seu endereço de e
Error	2		Error	Error	Erro
Login	3		Login	Inicio de sesión	Logon:
Email	4		Email	Correo electrónico	Email
EmailPlaceHolder	5		Enter your email…	Introduzca su correo electrónico	Por favor, digite seu endereço de e-mail.
Password	6		Password	Contraseña	Senha
PasswordPlaceHolder	7		Enter your password…	Escriba la contraseña.	Digite sua senha
Remember	8		Rememberme in this device	Recordarme en este dispositivo	Rememberme neste dispositivo
PasswordMessage	9		You must enter a password.	Debe especificar una contraseña.	Insira uma senha.
EmailOrPasswordIncorrect	10		Email or password incorrect.	Email o contraseña incorrectos	E-mail ou senha incorretos.

13. And add the properties in **Languages** class:

```
using Interfaces;
using Resources;
using Xamarin.Forms;

public static class Languages
{
        static Languages()
        {
        var ci = DependencyService.Get<ILocalize>().GetCurrentCultureInfo();
        Resource.Culture = ci;
        DependencyService.Get<ILocalize>().SetLocale(ci);
        }

        public static string Accept => Resource.Accept;

        public static string Error => Resource.Error;

        public static string EmailMessage => Resource.EmailMessage;

        public static string Login => Resource.Login;

        public static string EmailMEmailessage => Resource.Email;

        public static string EmailPlaceHolder => Resource.EmailPlaceHolder;

        public static string Password => Resource.Password;

        public static string PasswordPlaceHolder => Resource.PasswordPlaceHolder;

        public static string PasswordMessage => Resource.PasswordMessage;

        public static string Remember => Resource.Remember;
        public static string EmailOrPasswordIncorrect => Resource.EmailOrPasswordIncorrect;
}
```

14. Modify the **LoginVewModel** to complete the translations.

```
if (string.IsNullOrEmpty(this.Email))
{
        await Application.Current.MainPage.DisplayAlert(
        Languages.Error,
        Languages.EmailMessage,
        Languages.Accept);
        return;
}

if (string.IsNullOrEmpty(this.Password))
{
        await Application.Current.MainPage.DisplayAlert(
        Languages.Error,
        Languages.PasswordMessage,
        Languages.Accept);
        return;
        return;
}

var request = new TokenRequest
{
        Password = this.Password,
        Username = this.Email
};
```

```
this.IsRunning = true;
this.IsEnabled = false;

var response = await this.apiService.GetTokenAsync(
        "https://shopzulu.azurewebsites.net",
        "/Account",
        "/CreateToken",
        request);

if (!response.IsSuccess)
{
        await Application.Current.MainPage.DisplayAlert(
        Languages.Error,
        Languages.EmailOrPasswordIncorrect,
        Languages.Accept);
        return;
        return;
}
```

15. Modify the **LoginPage** for the translations in XAML.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:i18n="clr-namespace:Shop.UIForms.Helpers"
        x:Class="Shop.UIForms.Views.LoginPage"
        BindingContext="{Binding Main, Source={StaticResource Locator}}"
        Title="{i18n:Translate Login}">
        <ContentPage.Content>
        <ScrollView
        BindingContext="{Binding Login}">
```

```
<StackLayout
        Padding="10">
        <Image
        HeightRequest="150"
        Source="shop.png">
        </Image>
        <Label
        Text="{i18n:Translate Email}">
        </Label>
        <Entry
        Keyboard="{i18n:Translate Email}"
        Placeholder="{i18n:Translate EmailPlaceHolder}"
        Text="{Binding Email}">
        </Entry>
        <Label
        Text="{i18n:Translate Password}">
        </Label>
        <Entry
        IsPassword="True"
        Placeholder="{i18n:Translate PasswordPlaceHolder}"
        Text="{Binding Password}">
        </Entry>
        <StackLayout
        HorizontalOptions="Center"
        Orientation="Horizontal">
        <Label
        Text="{i18n:Translate Remember}"
        VerticalOptions="Center">
        </Label>
        <Switch
        IsToggled="{Binding IsToggled}">
        </Switch>
        </StackLayout>
        <ActivityIndicator
        IsRunning="{Binding IsRunning}"
        VerticalOptions="CenterAndExpand">
        </ActivityIndicator>
        <Button
        BackgroundColor="Navy"
        BorderRadius="23"
        Command="{Binding LoginCommand}"
        HeightRequest="46"
        IsEnabled="{Binding IsEnabled}"
        Text="{i18n:Translate Login}"
        TextColor="White">
        </Button>
    </StackLayout>
    </ScrollView>
    </ContentPage.Content>
</ContentPage>
```

16. Test it.

Acceding To Camera and Gallery in Xamarin Forms

1. Change the **AddProductPage**:

```
<Image
    HeightRequest="150"
    Source="{Binding ImageSource}">
    <Image.GestureRecognizers>
```

```
        <TapGestureRecognizer Command="{Binding ChangeImageCommand}"/>
        </Image.GestureRecognizers>
</Image>
```

2. Add the attribute and property in **AddProductViewModel**:

```
private ImageSource imageSource;

public ImageSource ImageSource
{
        get => this.imageSource;
        set => this.SetValue(ref this.imageSource, value);
}
```

And delete the **Image** property.

3. And initialize in the constructor:

```
this.ImageSource = "noImage";
```

4. Add the NuGet **Xam.Plugin.Media** in all Xamarin Forms projects:

5. Modify the **MainActivity**:

```
protected override void OnCreate(Bundle savedInstanceState)
{
        TabLayoutResource = Resource.Layout.Tabbar;
        ToolbarResource = Resource.Layout.Toolbar;

        base.OnCreate(savedInstanceState);
        CrossCurrentActivity.Current.Init(this, savedInstanceState);
        global::Xamarin.Forms.Forms.Init(this, savedInstanceState);
        LoadApplication(new App());
}

public override void OnRequestPermissionsResult(
        int requestCode,
        string[] permissions,
        [GeneratedEnum] Permission[] grantResults)
{
        PermissionsImplementation.Current.OnRequestPermissionsResult(
        requestCode,
        permissions,
        grantResults);
}
```

6. Modify the **AndroidManifest**:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0"
package="com.companyname.ShopPrep.UIForms">
  <uses-sdk android:minSdkVersion="21" android:targetSdkVersion="27" />
  <uses-permission android:name="android.permission.INTERNET" />
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
  <uses-permission android:name="android.permission.CAMERA" />
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
  <application android:label="ShopPrep.UIForms.Android">
        <provider android:name="android.support.v4.content.FileProvider"
        android:authorities="${applicationId}.fileprovider"
        android:exported="false"
        android:grantUriPermissions="true">
        <meta-data android:name="android.support.FILE_PROVIDER_PATHS"
                android:resource="@xml/file_paths"></meta-data>
        </provider>
  </application>
</manifest>
```

7. Add the folder **xml** inside **Resources** and inside it, add the **file_paths.xml**:

```
<?xml version="1.0" encoding="utf-8" ?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
  <external-files-path name="my_images" path="Pictures" />
  <external-files-path name="my_movies" path="Movies" />
</paths>
```

8. Modify the **info.plist**:

```
<key>NSCameraUsageDescription</key>
<string>This app needs access to the camera to take photos.</string>
<key>NSPhotoLibraryUsageDescription</key>
<string>This app needs access to photos.</string>
<key>NSMicrophoneUsageDescription</key>
<string>This app needs access to microphone.</string>
<key>NSPhotoLibraryAddUsageDescription</key>
<string>This app needs access to the photo gallery.</string>
```

9. Add the attribute in **AddProductViewModel**:

```
private MediaFile file;
```

10. Add the command in **AddProductViewModel**:

```
public ICommand ChangeImageCommand => new RelayCommand(this.ChangeImage);
```

11. Add the method in **AddProductViewModel**:

```
private async void ChangeImage()
{
        await CrossMedia.Current.Initialize();

        var source = await Application.Current.MainPage.DisplayActionSheet(
        "Where do you take the picture?",
        "Cancel",
        null,
        "From Gallery",
        "From Camera");

        if (source == "Cancel")
        {
        this.file = null;
        return;
        }

        if (source == "From Camera")
        {
        this.file = await CrossMedia.Current.TakePhotoAsync(
        new StoreCameraMediaOptions
        {
                Directory = "Sample",
                Name = "test.jpg",
                PhotoSize = PhotoSize.Small,
        }
        );
        }
        else
        {
        this.file = await CrossMedia.Current.PickPhotoAsync();
        }

        if (this.file != null)
        {
        this.ImageSource = ImageSource.FromStream(() =>
        {
        var stream = file.GetStream();
        return stream;
        });
        }
}
```

12. Test it.

Sending the Image to Backend

1. In **Web.Helpers** add the class **FilesHelper**:

```
using System.IO;

public class FilesHelper
{
        public static bool UploadPhoto(MemoryStream stream, string folder, string name)
        {
        try
        {
        stream.Position = 0;
        var path = Path.Combine(Directory.GetCurrentDirectory(), folder , name);
        File.WriteAllBytes(path, stream.ToArray());
        }
        catch
        {
        return false;
        }

        return true;
        }
}
```

2. Add the property **ImageArray** in **Product** (in Common.Models):

```
public byte[] ImageArray { get; set; }
```

3. Modify the method **PostProduct** in products API controller:

```
[HttpPost]
public async Task<IActionResult> PostProduct([FromBody] Common.Models.Product product)
{
        if (!ModelState.IsValid)
        {
        return this.BadRequest(ModelState);
        }

        var user = await this.userHelper.GetUserByEmail(product.UserName);
        if (user == null)
        {
```

```
return this.BadRequest("Invalid user");
}

var imageUrl = string.Empty;
if (product.ImageArray != null && product.ImageArray.Length > 0)
{
var stream = new MemoryStream(product.ImageArray);
var guid = Guid.NewGuid().ToString();
var file = $"{guid}.jpg";
var folder = "wwwroot\\images\\Products";
var fullPath = $"~/images/Products/{file}";
var response = FilesHelper.UploadPhoto(stream, folder, file);

if (response)
{
imageUrl = fullPath;
}
}

var entityProduct = new Product
{
IsAvailabe = product.IsAvailabe,
LastPurchase = product.LastPurchase,
LastSale = product.LastSale,
Name = product.Name,
Price = product.Price,
Stock = product.Stock,
User = user,
ImageUrl = imageUrl
};
```

```
    var newProduct = await this.repository.AddProductAsync(entityProduct);
    return Ok(newProduct);
}
```

4. Now in Xamarin Shared project, add the class **FilesHelper** in **Common.Helpers**:

```
using System.IO;

public class FilesHelper
{
    public static byte[] ReadFully(Stream input)
    {
    using (MemoryStream ms = new MemoryStream())
    {
    input.CopyTo(ms);
    return ms.ToArray();
    }
    }
}
```

5. Modify the **AddProductViewModel**:

```
this.IsRunning = true;
this.IsEnabled = false;

byte[] imageArray = null;
if (this.file != null)
{
    imageArray = FilesHelper.ReadFully(this.file.GetStream());
}
```

```
var product = new Product
{
        IsAvailabe = true,
        Name = this.Name,
        Price = price,
        UserEmail = MainViewModel.GetInstance().UserEmail,
        ImageArray = imageArray
};
```

6. Test it locally.

7. Publish the changes on Azure and make a complete test.

Register Users From App in Xamarin Forms

1. Create the **CountriesController** in **Web.Controllers.API**:

```
using Data;
using Microsoft.AspNetCore.Mvc;

[Route("api/[Controller]")]
public class CountriesController : Controller
{
        private readonly ICountryRepository countryRepository;

        public CountriesController(ICountryRepository countryRepository)
        {
        this.countryRepository = countryRepository;
        }
```

```
          [HttpGet]
          public IActionResult GetCountries()
          {
          return Ok(this.countryRepository.GetCountriesWithCities());
          }

}
```

2. Test it locally and then publish in Azure.

3. Now add the models in **Common.Models** (user Json2scharp):

```
using Newtonsoft.Json;

public class City
{
          [JsonProperty("id")]
          public int Id { get; set; }

          [JsonProperty("name")]
          public string Name { get; set; }
}
```

And:

```
using System.Collections.Generic;
using Newtonsoft.Json;

public class Country
{
          [JsonProperty("id")]
```

```
    public int Id { get; set; }

    [JsonProperty("name")]
    public string Name { get; set; }

    [JsonProperty("cities")]
    public List<City> Cities { get; set; }

    [JsonProperty("numberCities")]
    public int NumberCities { get; set; }
}
```

4. Add the **RegisterPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="Shop.UIForms.Views.RegisterPage"
    BindingContext="{Binding Main, Source={StaticResource Locator}}"
    Title="Register New User">
    <ContentPage.Content>
    <ScrollView
    BindingContext="{Binding Register}">
    <StackLayout
        Padding="8">
        <Grid>
        <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="2*"/>
        </Grid.ColumnDefinitions>
        <Label
```

```
Grid.Column="0"
Grid.Row="0"
Text="First name"
VerticalOptions="Center">
</Label>
<Entry
Grid.Column="1"
Grid.Row="0"
Placeholder="Enter your first name..."
Text="{Binding FirstName}">
</Entry>
<Label
Grid.Column="0"
Grid.Row="1"
Text="Last name"
VerticalOptions="Center">
</Label>
<Entry
Grid.Column="1"
Grid.Row="1"
Placeholder="Enter your last name..."
Text="{Binding LastName}">
</Entry>
<Label
Grid.Column="0"
Grid.Row="2"
Text="Email"
VerticalOptions="Center">
</Label>
<Entry
Grid.Column="1"
Grid.Row="2"
Keyboard="Email"
Placeholder="Enter your email..."
Text="{Binding Email}">
</Entry>
<Label
Grid.Column="0"
Grid.Row="3"
Text="Country"
VerticalOptions="Center">
</Label>
<Picker
Grid.Column="1"
Grid.Row="3"
ItemDisplayBinding="{Binding Name}"
ItemsSource="{Binding Countries}"
SelectedItem="{Binding Country}"
Title="Select a country...">
</Picker>
<Label
Grid.Column="0"
Grid.Row="4"
Text="City"
VerticalOptions="Center">
</Label>
<Picker
Grid.Column="1"
Grid.Row="4"
ItemDisplayBinding="{Binding Name}"
ItemsSource="{Binding Cities}"
SelectedItem="{Binding City}"
```

```
Title="Select a city...">
</Picker>
<Label
Grid.Column="0"
Grid.Row="5"
Text="Address"
VerticalOptions="Center">
</Label>
<Entry
Grid.Column="1"
Grid.Row="5"
Keyboard="Email"
Placeholder="Enter your address..."
Text="{Binding Address}">
</Entry>
<Label
Grid.Column="0"
Grid.Row="6"
Text="Pohone"
VerticalOptions="Center">
</Label>
<Entry
Grid.Column="1"
Grid.Row="6"
Keyboard="Telephone"
Placeholder="Enter your phone number..."
Text="{Binding Phone}">
</Entry>
<Label
Grid.Column="0"
Grid.Row="7"
```

```
Text="Password"
VerticalOptions="Center">
</Label>
<Entry
Grid.Column="1"
Grid.Row="7"
IsPassword="True"
Placeholder="Enter your password..."
Text="{Binding Password}">
</Entry>
<Label
Grid.Column="0"
Grid.Row="8"
Text="Password confirm"
VerticalOptions="Center">
</Label>
<Entry
Grid.Column="1"
Grid.Row="8"
IsPassword="True"
Placeholder="Enter your password confirm..."
Text="{Binding Confirm}">
</Entry>
</Grid>
<ActivityIndicator
IsRunning="{Binding IsRunning}"
VerticalOptions="CenterAndExpand">
</ActivityIndicator>
<Button
BackgroundColor="Navy"
BorderRadius="23"
```

```
                Command="{Binding RegisterCommand}"
                HeightRequest="46"
                HorizontalOptions="FillAndExpand"
                IsEnabled="{Binding IsEnabled}"
                Text="Register New User"
                TextColor="White">
                </Button>
        </StackLayout>
        </ScrollView>
        </ContentPage.Content>
</ContentPage>
```

5. Add the **RegisterViewModel**:

```
using System.Collections.ObjectModel;
using System.Windows.Input;
using Common.Models;
using GalaSoft.MvvmLight.Command;

public class RegisterViewModel : BaseViewModel
{
        private bool isRunning;
        private bool isEnabled;
        private ObservableCollection<Country> countries;
        private Country country;
        private ObservableCollection<City> cities;
        private City city;

        public string FirstName { get; set; }

        public string LastName { get; set; }
        public string Email { get; set; }

        public string Address { get; set; }

        public string Phone { get; set; }

        public string Password { get; set; }

        public string Confirm { get; set; }

        public Country Country
        {
        get => this.country;
        set => this.SetValue(ref this.country, value);
        }

        public City City
        {
        get => this.city;
        set => this.SetValue(ref this.city, value);
        }

        public ObservableCollection<Country> Countries
        {
        get => this.countries;
        set => this.SetValue(ref this.countries, value);
        }

        public ObservableCollection<City> Cities
        {
        get => this.cities;
        set => this.SetValue(ref this.cities, value);
        }

        public bool IsRunning
        {
        get => this.isRunning;
        set => this.SetValue(ref this.isRunning, value);
        }

        public bool IsEnabled
        {
        get => this.isEnabled;
        set => this.SetValue(ref this.isEnabled, value);
        }

        public ICommand RegisterCommand => new RelayCommand(this.Register);

        public RegisterViewModel()
        {
        this.IsEnabled = true;
        }

        private async void Register()
        {
        }
    }
```

6. Add the property **Register** in **MainViewModel**:

```
public RegisterViewModel Register { get; set; }
```

7. Modify the **LoginPage**:

```
<StackLayout
        Orientation="Horizontal">
        <Button
        BackgroundColor="Navy"
        BorderRadius="23"
        Command="{Binding LoginCommand}"
        HeightRequest="46"
        HorizontalOptions="FillAndExpand"
        IsEnabled="{Binding IsEnabled}"
        Text="{i18n:Translate Login}"
        TextColor="White">
        </Button>
        <Button
        BackgroundColor="Purple"
        BorderRadius="23"
        Command="{Binding RegisterCommand}"
        HeightRequest="46"
        HorizontalOptions="FillAndExpand"
        IsEnabled="{Binding IsEnabled}"
        Text="{i18n:Translate RegisterNewUser}"
        TextColor="White">
        </Button>
</StackLayout>
```

8. Modify the **LoginViewModel**:

```
public ICommand RegisterCommand => new RelayCommand(this.Register);
…
```

```
private async void Register()
{
            MainViewModel.GetInstance().Register = new RegisterViewModel();
            await Application.Current.MainPage.Navigation.PushAsync(new RegisterPage());
}
```

9. Test it what we do until this moment.

10. Now modify the RegisterViewModel to load countries and cities.

```
...
private readonly ApiService apiService;
…
public Country Country
{
            get => this.country;
            set
            {
            this.SetValue(ref this.country, value);
            this.Cities = new ObservableCollection<City>(this.Country.Cities.OrderBy(c => c.Name));
            }
}
…
public RegisterViewModel()
{
            this.apiService = new ApiService();
            this.IsEnabled = true;
            this.LoadCountries();
}
…
private async void LoadCountries()
```

```
{
    this.IsRunning = true;
    this.IsEnabled = false;

    var url = Application.Current.Resources["UrlAPI"].ToString();
    var response = await this.apiService.GetListAsync<Country>(
    url,
    "/api",
    "/Countries");

    this.IsRunning = false;
    this.IsEnabled = true;

    if (!response.IsSuccess)
    {
    await Application.Current.MainPage.DisplayAlert(
    "Error",
    response.Message,
    "Accept");
    return;
    }

    var myCountries = (List<Country>)response.Result;
    this.Countries = new ObservableCollection<Country>(myCountries);
}
```

11. Test it.

12. Add the **RegexHelper** into **Common.Helpers**.

```
using System;
using System.Net.Mail;

public static class RegexHelper
{
        public static bool IsValidEmail(string emailaddress)
        {
        try
        {
        var mail = new MailAddress(emailaddress);

        return true;
        }
        catch (FormatException)
        {
        return false;
        }
        }
}
```

13. Add the **NewUserRequest** into **Common.Models**.

```
using System.ComponentModel.DataAnnotations;

public class NewUserRequest
{
        [Required]
        public string FirstName { get; set; }

        [Required]
        public string LastName { get; set; }
```

```
    [Required]
    public string Email { get; set; }

    [Required]
    public string Address { get; set; }

    [Required]
    public string Phone { get; set; }

    [Required]
    public string Password { get; set; }

    [Required]
    public int CityId { get; set; }
}
```

14. Add the **AccountController** into **Web.Controllers.API**.

```
using System.Linq;
using System.Threading.Tasks;
using Common.Models;
using Data;
using Helpers;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

[Route("api/[Controller]")]
public class AccountController : Controller
{
    private readonly IUserHelper userHelper;
    private readonly ICountryRepository countryRepository;
private readonly IMailHelper mailHelper;

public AccountController(
IUserHelper userHelper,
ICountryRepository countryRepository,
IMailHelper mailHelper)
{
this.userHelper = userHelper;
this.countryRepository = countryRepository;
this.mailHelper = mailHelper;
}

[HttpPost]
public async Task<IActionResult> PostUser([FromBody] NewUserRequest request)
{
if (!ModelState.IsValid)
{
return this.BadRequest(new Response
{
	IsSuccess = false,
	Message = "Bad request"
});
}

var user = await this.userHelper.GetUserByEmailAsync(request.Email);
if (user != null)
{
return this.BadRequest(new Response
{
	IsSuccess = false,
	Message = "This email is already registered."
});
}

var city = await this.countryRepository.GetCityAsync(request.CityId);
if (city == null)
{
return this.BadRequest(new Response
{
        IsSuccess = false,
        Message = "City don't exists."
});
}

user = new Data.Entities.User
{
FirstName = request.FirstName,
LastName = request.LastName,
Email = request.Email,
UserName = request.Email,
Address = request.Address,
PhoneNumber = request.Phone,
CityId = request.CityId,
City = city
};

var result = await this.userHelper.AddUserAsync(user, request.Password);
if (result != IdentityResult.Success)
{
return this.BadRequest(result.Errors.FirstOrDefault().Description);
}
```

```
    var myToken = await this.userHelper.GenerateEmailConfirmationTokenAsync(user);
    var tokenLink = this.Url.Action("ConfirmEmail", "Account", new
    {
    userid = user.Id,
    token = myToken
    }, protocol: HttpContext.Request.Scheme);

    this.mailHelper.SendMail(request.Email, "Email confirmation", $"<h1>Email Confirmation</h1>" +
    $"To allow the user, " +
    $"plase click in this link:</br></br><a href = \"{tokenLink}\">Confirm Email</a>");

    return Ok(new Response
    {
    IsSuccess = true,
    Message = "A Confirmation email was sent. Plese confirm your account and log into the App."
    });
    }
}
```

15. Publish in Azure.

16. Add the method to **ApiService**:

```
public async Task<Response> RegisterUserAsync(
    string urlBase,
    string servicePrefix,
    string controller,
    NewUserRequest newUserRequest)
{
    try
    {
    var request = JsonConvert.SerializeObject(newUserRequest);
    var content = new StringContent(request, Encoding.UTF8, "application/json");
    var client = new HttpClient
    {
    BaseAddress = new Uri(urlBase)
    };

    var url = $"{servicePrefix}{controller}";
    var response = await client.PostAsync(url, content);
    var answer = await response.Content.ReadAsStringAsync();
    var obj = JsonConvert.DeserializeObject<Response>(answer);
    return obj;
    }
    catch (Exception ex)
    {
    return new Response
    {
    IsSuccess = false,
    Message = ex.Message,
    };
    }
}
```

17. Modify the **Register** in the **RegisterViewModel** class:

```
private async void Register()
{
        if (string.IsNullOrEmpty(this.FirstName))
        {
        await Application.Current.MainPage.DisplayAlert(
        "Error",
        "You must enter the first name.",
        "Accept");
        return;
        }

        if (string.IsNullOrEmpty(this.LastName))
        {
        await Application.Current.MainPage.DisplayAlert(
        "Error",
        "You must enter the last name.",
        "Accept");
        return;
        }

        if (string.IsNullOrEmpty(this.Email))
        {
        await Application.Current.MainPage.DisplayAlert(
        "Error",
        "You must enter an email.",
        "Accept");
        return;
        }

        if (!RegexHelper.IsValidEmail(this.Email))
        {
        await Application.Current.MainPage.DisplayAlert(
        "Error",
        "You must enter a valid email.",
        "Accept");
        return;
        }
```

```
if (this.Country == null)
{
await Application.Current.MainPage.DisplayAlert(
"Error",
"You must select a country.",
"Accept");
return;
}

if (this.City == null)
{
await Application.Current.MainPage.DisplayAlert(
"Error",
"You must select a city.",
"Accept");
return;
}

if (string.IsNullOrEmpty(this.Address))
{
await Application.Current.MainPage.DisplayAlert(
"Error",
"You must enter an address.",
"Accept");
return;
}

if (string.IsNullOrEmpty(this.Phone))
{
await Application.Current.MainPage.DisplayAlert(
"Error",
"You must enter a phone number.",
"Accept");
return;
}

if (string.IsNullOrEmpty(this.Password))
{
await Application.Current.MainPage.DisplayAlert(
"Error",
"You must enter a password.",
"Accept");
return;
}

if (this.Password.Length < 6)
{
await Application.Current.MainPage.DisplayAlert(
"Error",
"You password must be at mimimun 6 characters.",
"Accept");
return;
}

if (string.IsNullOrEmpty(this.Confirm))
{
await Application.Current.MainPage.DisplayAlert(
"Error",
"You must enter a password confirm.",
"Accept");
return;
```

```
}

if (!this.Password.Equals(this.Confirm))
{
await Application.Current.MainPage.DisplayAlert(
"Error",
"The password and the confirm do not match.",
"Accept");
return;
}

this.IsRunning = true;
this.IsEnabled = false;

var request = new NewUserRequest
{
Address = this.Address,
CityId = this.City.Id,
Email = this.Email,
FirstName = this.FirstName,
LastName = this.LastName,
Password = this.Password,
Phone = this.Phone
};

var url = Application.Current.Resources["UrlAPI"].ToString();
var response = await this.apiService.RegisterUserAsync(
url,
"/api",
"/Account",
request);
```

```
    this.IsRunning = false;
    this.IsEnabled = true;

    if (!response.IsSuccess)
    {
    await Application.Current.MainPage.DisplayAlert(
    "Error",
    response.Message,
    "Accept");
    return;
    }

    await Application.Current.MainPage.DisplayAlert(
    "Ok",
    response.Message,
    "Accept");
    await Application.Current.MainPage.Navigation.PopAsync();
}
```

18. Test it.

Recover Password From App in Xamarin Forms

1. Add the **RememberPasswordPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="Shop.UIForms.Views.RememberPasswordPage"
```

```
BindingContext="{Binding Main, Source={StaticResource Locator}}"
Title="Recover password">
<ContentPage.Content>
<ScrollView
BindingContext="{Binding RememberPassword}">
<StackLayout
        Padding="8">
        <Label
        Text="Email">
        </Label>
        <Entry
        Keyboard="Email"
        Placeholder="Enter email to recover the password..."
        Text="{Binding Email}">
        </Entry>
        <ActivityIndicator
        IsRunning="{Binding IsRunning}"
        VerticalOptions="CenterAndExpand">
        </ActivityIndicator>
        <Button
        BackgroundColor="Navy"
        BorderRadius="23"
        Command="{Binding RecoverCommand}"
        HeightRequest="46"
        HorizontalOptions="FillAndExpand"
        IsEnabled="{Binding IsEnabled}"
        Text="Recover Password"
        TextColor="White">
        </Button>
</StackLayout>
</ScrollView>
```

```
    </ContentPage.Content>
</ContentPage>
```

2. Add the **RememberPasswordViewModel**:

```
using System.Windows.Input;
using Common.Helpers;
using Common.Services;
using GalaSoft.MvvmLight.Command;
using Xamarin.Forms;

public class RememberPasswordViewModel : BaseViewModel
{
    private bool isRunning;
    private bool isEnabled;
    private readonly ApiService apiService;

    public bool IsRunning
    {
    get => this.isRunning;
    set => this.SetValue(ref this.isRunning, value);
    }

    public bool IsEnabled
    {
    get => this.isEnabled;
    set => this.SetValue(ref this.isEnabled, value);
    }

    public string Email { get; set; }
    public ICommand RecoverCommand => new RelayCommand(this.Recover);

    public RememberPasswordViewModel()
    {
    this.apiService = new ApiService();
    this.IsEnabled = true;
    }

    private async void Recover()
    {
    if (string.IsNullOrEmpty(this.Email))
    {
    await Application.Current.MainPage.DisplayAlert(
        "Error",
        "You must enter an email.",
        "Accept");
    return;
    }

    if (!RegexHelper.IsValidEmail(this.Email))
    {
    await Application.Current.MainPage.DisplayAlert(
        "Error",
        "You must enter a valid email.",
        "Accept");
    return;
    }
    }
}
```

3. Add the property **RememberPasswordViewModel** in **MainViewModel**:

```
public RememberPasswordViewModel RememberPassword { get; set; }
```

4. Modify the **LoginPage**:

```
<Label
	HorizontalOptions="Center"
	Text="Forgot my password"
	TextColor="Navy">
	<Label.GestureRecognizers>
	<TapGestureRecognizer Command="{Binding RememberPasswordCommand }"/>
	</Label.GestureRecognizers>
</Label>
```

5. Modify the **LoginViewModel**:

```
public ICommand RememberPasswordCommand => new RelayCommand(this.RememberPassword);
…
private async void RememberPassword()
{
	MainViewModel.GetInstance().RememberPassword = new RememberPasswordViewModel();
	await Application.Current.MainPage.Navigation.PushAsync(new RememberPasswordPage());
}
```

6. Test what we do until this moment.

7. Add the **RecoverPasswordRequest** in **Common.Models**.

```
using System.ComponentModel.DataAnnotations;

public class RecoverPasswordRequest
{
    [Required]
    public string Email { get; set; }
}
```

8. Add the **RecoverPassword** method in **AccountController** in **Web.Controllers.API**:

```
[HttpPost]
[Route("RecoverPassword")]
public async Task<IActionResult> RecoverPassword([FromBody] RecoverPasswordRequest request)
{
    if (!ModelState.IsValid)
    {
    return this.BadRequest(new Response
    {
    IsSuccess = false,
    Message = "Bad request"
    });
    }

    var user = await this.userHelper.GetUserByEmailAsync(request.Email);
    if (user == null)
    {
    return this.BadRequest(new Response
    {
    IsSuccess = false,
    Message = "This email is not assigned to any user."
    });
    }

    var myToken = await this.userHelper.GeneratePasswordResetTokenAsync(user);
    var link = this.Url.Action("ResetPassword", "Account", new { token = myToken }, protocol: HttpContext.Request.Scheme);
```

```
        this.mailHelper.SendMail(request.Email, "Password Reset", $"<h1>Recover Password</h1>" +
        $"To reset the password click in this link:</br></br>" +
        $"<a href = \"{link}\">Reset Password</a>");

        return Ok(new Response
        {
        IsSuccess = true,
        Message = "An email with instructions to change the password was sent."
        });
}
```

9. Publish on Azure.

10. add method **RecoverPasswordAsync** to **ApiService**:

```
public async Task<Response> RecoverPasswordAsync(
        string urlBase,
        string servicePrefix,
        string controller,
        RecoverPasswordRequest recoverPasswordRequest)
{
        try
        {
        var request = JsonConvert.SerializeObject(recoverPasswordRequest);
        var content = new StringContent(request, Encoding.UTF8, "application/json");
        var client = new HttpClient
        {
        BaseAddress = new Uri(urlBase)
        };

        var url = $"{servicePrefix}{controller}";
        var response = await client.PostAsync(url, content);
        var answer = await response.Content.ReadAsStringAsync();
        var obj = JsonConvert.DeserializeObject<Response>(answer);
        return obj;
        }
        catch (Exception ex)
        {
        return new Response
        {
        IsSuccess = false,
        Message = ex.Message,
        };
        }
}
```

11. Modify the **Recover** method on **RememberPasswordViewModel**:

```
private async void Recover()
{
        if (string.IsNullOrEmpty(this.Email))
        {
        await Application.Current.MainPage.DisplayAlert(
        "Error",
        "You must enter an email.",
        "Accept");
        return;
        }

        if (!RegexHelper.IsValidEmail(this.Email))
        {
        await Application.Current.MainPage.DisplayAlert(
        "Error",
        "You must enter a valid email.",
        "Accept");
        return;
        }

        this.IsRunning = true;
        this.IsEnabled = false;

        var request = new RecoverPasswordRequest
        {
        Email = this.Email
        };

        var url = Application.Current.Resources["UrlAPI"].ToString();
        var response = await this.apiService.RecoverPasswordAsync(
        url,
        "/api",
        "/Account/RecoverPassword",
        request);

        this.IsRunning = false;
        this.IsEnabled = true;

        if (!response.IsSuccess)
        {
        await Application.Current.MainPage.DisplayAlert(
        "Error",
        response.Message,
        "Accept");
        return;
```

```
    }

    await Application.Current.MainPage.DisplayAlert(
    "Ok",
    response.Message,
    "Accept");
    await Application.Current.MainPage.Navigation.PopAsync();
}
```

12. Test it.

Modify User From App in Xamarin Forms

1. Add the method **GetUserByEmail** in **AccountController** in **Web.Controllers.API**:

```
[HttpPost]
[Route("GetUserByEmail")]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
public async Task<IActionResult> GetUserByEmail([FromBody] RecoverPasswordRequest request)
{
    if (!ModelState.IsValid)
    {
    return this.BadRequest(new Response
    {
    IsSuccess = false,
    Message = "Bad request"
    });
    }

    var user = await this.userHelper.GetUserByEmailAsync(request.Email);
    if (user == null)
    {
    return this.BadRequest(new Response
    {
    IsSuccess = false,
    Message = "User don't exists."
    });
    }

    return Ok(user);
}
```

2. Publish on Azure.

3. Add the method **GetUserByEmail ApiService**:

```
public async Task<Response> GetUserByEmailAsync(
    string urlBase,
    string servicePrefix,
    string controller,
    string email,
    string tokenType,
    string accessToken)
{
    try
    {
    var request = JsonConvert.SerializeObject(new RecoverPasswordRequest { Email = email });
    var content = new StringContent(request, Encoding.UTF8, "application/json");
    var client = new HttpClient
    {
    BaseAddress = new Uri(urlBase)
    };

    client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);
    var url = $"{servicePrefix}{controller}";
    var response = await client.PostAsync(url, content);
    var answer = await response.Content.ReadAsStringAsync();
    if (!response.IsSuccessStatusCode)
    {
    return new Response
    {
        IsSuccess = false,
        Message = answer,
    };
    }

    var user = JsonConvert.DeserializeObject<User>(answer);
    return new Response
    {
    IsSuccess = true,
    Result = user,
    };
    }
    catch (Exception ex)
    {
    return new Response
    {
    IsSuccess = false,
    Message = ex.Message,
    };
    }
}
```

4. Modify the **User** model in **Common.Models**:

```
[JsonProperty("cityId")]
public int CityId { get; set; }

[JsonProperty("address")]
public string Address { get; set; }

public string FullName { get { return $"{this.FirstName} {this.LastName}"; }  }
```

5. Add the property **User** in **Settings** in **Common.Helpers**:

```
private const string user = "user";
…
public static string User
{
        get => AppSettings.GetValueOrDefault(user, stringDefault);
        set => AppSettings.AddOrUpdateValue(user, value);
}
```

6. Add the property **User** in **MainViewModel**:

```
public User User { get; set; }
```

7. Modify the **LoginViewModel**:

```
var token = (TokenResponse)response.Result;

var response2 = await this.apiService.GetUserByEmailAsync(
        url,
        "/api",
        "/Account/GetUserByEmail",
        this.Email,
        "bearer",
        token.Token);

var user = (User)response2.Result;

var mainViewModel = MainViewModel.GetInstance();
mainViewModel.User = user;
mainViewModel.Token = token;
mainViewModel.Products = new ProductsViewModel();
mainViewModel.UserEmail = this.Email;
mainViewModel.UserPassword = this.Password;

Settings.IsRemember = this.IsRemember;
Settings.UserEmail = this.Email;
Settings.UserPassword = this.Password;
Settings.Token = JsonConvert.SerializeObject(token);
Settings.User = JsonConvert.SerializeObject(user);

Application.Current.MainPage = new MasterPage();
```

8. Modify the constructor in **App**:

```
public App()
{
        InitializeComponent();

        if (Settings.IsRemember)
        {
```

```
    var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);
    var user = JsonConvert.DeserializeObject<User>(Settings.User);

    if (token.Expiration > DateTime.Now)
    {
    var mainViewModel = MainViewModel.GetInstance();
    mainViewModel.Token = token;
    mainViewModel.User = user;
    mainViewModel.UserEmail = Settings.UserEmail;
    mainViewModel.UserPassword = Settings.UserPassword;
    mainViewModel.Products = new ProductsViewModel();
    this.MainPage = new MasterPage();
    return;
    }
    }

    MainViewModel.GetInstance().Login = new LoginViewModel();
    this.MainPage = new NavigationPage(new LoginPage());
}
```

9. Add a new icon to modify user in menu.

10. Modify the method **LoadMenus** in **MainViewModel**:

```
private void LoadMenus()
{
    var menus = new List<Menu>
    {
    new Menu
    {
```

```
    Icon = "ic_info",
    PageName = "AboutPage",
    Title = "About"
    },
    new Menu
    {
    Icon = "ic_person",
    PageName = "ProfilePage",
    Title = "Modify User"
    },
    new Menu
    {
    Icon = "ic_phonelink_setup",
    PageName = "SetupPage",
    Title = "Setup"
    },
    new Menu
    {
    Icon = "ic_exit_to_app",
    PageName = "LoginPage",
    Title = "Close session"
    }
    };

    this.Menus = new ObservableCollection<MenuItemViewModel>(menus.Select(m => new MenuItemViewModel
    {
    Icon = m.Icon,
    PageName = m.PageName,
    Title = m.Title
    }).ToList());
}
```

11. Modify the **MenuPage**:

```
<Image
        HeightRequest="150"
        Source="shop.png">
</Image>
<Label
        FontSize="Large"
        Text="{Binding User.FullName, StringFormat='Welcome: {0}'}"
        TextColor="White">
</Label>
<ListView
```

12. Test it.

13. Add the **ProfilePage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="Shop.UIForms.Views.ProfilePage"
        BindingContext="{Binding Main, Source={StaticResource Locator}}"
        Title="Modify User">
        <ContentPage.Content>
        <ScrollView
        BindingContext="{Binding Profile}">
        <StackLayout
                Padding="8">
                <Grid>
                <Grid.ColumnDefinitions>
```

```
<ColumnDefinition Width="*"/>
<ColumnDefinition Width="2*"/>
</Grid.ColumnDefinitions>
<Label
Grid.Column="0"
Grid.Row="0"
Text="First name"
VerticalOptions="Center">
</Label>
<Entry
Grid.Column="1"
Grid.Row="0"
Placeholder="Enter your first name..."
Text="{Binding User.FirstName}">
</Entry>
<Label
Grid.Column="0"
Grid.Row="1"
Text="Last name"
VerticalOptions="Center">
</Label>
<Entry
Grid.Column="1"
Grid.Row="1"
Placeholder="Enter your last name..."
Text="{Binding User.LastName}">
</Entry>
<Label
Grid.Column="0"
Grid.Row="2"
Text="Country"
```

```
VerticalOptions="Center">
</Label>
<Picker
Grid.Column="1"
Grid.Row="2"
ItemDisplayBinding="{Binding Name}"
ItemsSource="{Binding Countries}"
SelectedItem="{Binding Country}"
Title="Select a country...">
</Picker>
<Label
Grid.Column="0"
Grid.Row="3"
Text="City"
VerticalOptions="Center">
</Label>
<Picker
Grid.Column="1"
Grid.Row="3"
ItemDisplayBinding="{Binding Name}"
ItemsSource="{Binding Cities}"
SelectedItem="{Binding City}"
Title="Select a city...">
</Picker>
<Label
Grid.Column="0"
Grid.Row="4"
Text="Address"
VerticalOptions="Center">
</Label>
<Entry
```

```
Grid.Column="1"
Grid.Row="4"
Placeholder="Enter your address..."
Text="{Binding User.Address}">
</Entry>
<Label
Grid.Column="0"
Grid.Row="5"
Text="Phone"
VerticalOptions="Center">
</Label>
<Entry
Grid.Column="1"
Grid.Row="5"
Keyboard="Telephone"
Placeholder="Enter your phone number..."
Text="{Binding User.PhoneNumber}">
</Entry>
</Grid>
<ActivityIndicator
IsRunning="{Binding IsRunning}"
VerticalOptions="CenterAndExpand">
</ActivityIndicator>
<StackLayout
Orientation="Horizontal">
<Button
BackgroundColor="Navy"
BorderRadius="23"
Command="{Binding SaveCommand}"
HeightRequest="46"
HorizontalOptions="FillAndExpand"
```

```
        IsEnabled="{Binding IsEnabled}"
        Text="Save"
        TextColor="White">
        </Button>
        <Button
        BackgroundColor="Purple"
        BorderRadius="23"
        Command="{Binding ModifyPasswordCommand}"
        HeightRequest="46"
        HorizontalOptions="FillAndExpand"
        IsEnabled="{Binding IsEnabled}"
        Text="Modify Password"
        TextColor="White">
        </Button>
        </StackLayout>
    </StackLayout>
    </ScrollView>
    </ContentPage.Content>
</ContentPage>
```

14. Add the **ProfileViewModel**:

```
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using Common.Models;
using Common.Services;
using Xamarin.Forms;

public class ProfileViewModel : BaseViewModel
{
```

```
private readonly ApiService apiService;
private bool isRunning;
private bool isEnabled;
private ObservableCollection<Country> countries;
private Country country;
private ObservableCollection<City> cities;
private City city;
private User user;
private List<Country> myCountries;

public Country Country
{
get => this.country;
set
{
this.SetValue(ref this.country, value);
this.Cities = new ObservableCollection<City>(this.Country.Cities.OrderBy(c => c.Name));
}
}

public City City
{
get => this.city;
set => this.SetValue(ref this.city, value);
}

public User User
{
get => this.user;
set => this.SetValue(ref this.user, value);
}
```

```
public ObservableCollection<Country> Countries
{
get => this.countries;
set => this.SetValue(ref this.countries, value);
}

public ObservableCollection<City> Cities
{
get => this.cities;
set => this.SetValue(ref this.cities, value);
}

public bool IsRunning
{
get => this.isRunning;
set => this.SetValue(ref this.isRunning, value);
}

public bool IsEnabled
{
get => this.isEnabled;
set => this.SetValue(ref this.isEnabled, value);
}

public ProfileViewModel()
{
this.apiService = new ApiService();
this.User = MainViewModel.GetInstance().User;
this.IsEnabled = true;
this.LoadCountries();
}

private async void LoadCountries()
{
this.IsRunning = true;
this.IsEnabled = false;

var url = Application.Current.Resources["UrlAPI"].ToString();
var response = await this.apiService.GetListAsync<Country>(
url,
"/api",
"/Countries");

this.IsRunning = false;
this.IsEnabled = true;

if (!response.IsSuccess)
{
await Application.Current.MainPage.DisplayAlert(
        "Error",
        response.Message,
        "Accept");
return;
}

this.myCountries = (List<Country>)response.Result;
this.Countries = new ObservableCollection<Country>(myCountries);
this.SetCountryAndCity();
}

private void SetCountryAndCity()
```

```
        {
        foreach (var country in this.myCountries)
        {
        var city = country.Cities.Where(c => c.Id == this.User.CityId).FirstOrDefault();
        if (city != null)
        {
                this.Country = country;
                this.City = city;
                return;
        }
        }
        }
}
```

15. Add the property to **MainViewModel**:

```
public ProfileViewModel Profile { get; set; }
```

16. Modify the **SelectMenu** in **MenuItemViewModel**:

```
private async void SelectMenu()
{
        var mainViewModel = MainViewModel.GetInstance();
        App.Master.IsPresented = false;

        switch (this.PageName)
        {
        case "AboutPage":
        await App.Navigator.PushAsync(new AboutPage());
        break;
        case "SetupPage":
```

```
    await App.Navigator.PushAsync(new SetupPage());
    break;
    case "ProfilePage":
    mainViewModel.Profile = new ProfileViewModel();
    await App.Navigator.PushAsync(new ProfilePage());
    break;
    default:
    Settings.User = string.Empty;
    Settings.IsRemember = false;
    Settings.Token = string.Empty;
    Settings.UserEmail = string.Empty;
    Settings.UserPassword = string.Empty;

    MainViewModel.GetInstance().Login = new LoginViewModel();
    Application.Current.MainPage = new NavigationPage(new LoginPage());
    break;
    }
}
```

17. Test it.

18. Add the method **PutUser** in **AccountController** in **Web.Controllers.API**:

```
[HttpPut]
public async Task<IActionResult> PutUser([FromBody] User user)
{
    if (!ModelState.IsValid)
    {
    return this.BadRequest(ModelState);
    }
```

```
        var userEntity = await this.userHelper.GetUserByEmailAsync(user.Email);
        if (userEntity == null)
        {
        return this.BadRequest("User not found.");
        }

        var city = await this.countryRepository.GetCityAsync(user.CityId);
        if (city != null)
        {
        userEntity.City = city;
        }

        userEntity.FirstName = user.FirstName;
        userEntity.LastName = user.LastName;
        userEntity.CityId = user.CityId;
        userEntity.Address = user.Address;
        userEntity.PhoneNumber = user.PhoneNumber;

        var respose = await this.userHelper.UpdateUserAsync(userEntity);
        if (!respose.Succeeded)
        {
        return this.BadRequest(respose.Errors.FirstOrDefault().Description);
        }

        var updatedUser = await this.userHelper.GetUserByEmailAsync(user.Email);
        return Ok(updatedUser);
}
```

19. Publish on Azure.

20. Overload the method **PutAsync** in **ApiService**:

```
public async Task<Response> PutAsync<T>(
    string urlBase,
    string servicePrefix,
    string controller,
    T model,
    string tokenType,
    string accessToken)
{
    try
    {
    var request = JsonConvert.SerializeObject(model);
    var content = new StringContent(request, Encoding.UTF8, "application/json");
    var client = new HttpClient
    {
    BaseAddress = new Uri(urlBase)
    };

    client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);
    var url = $"{servicePrefix}{controller}";
    var response = await client.PutAsync(url, content);
    var answer = await response.Content.ReadAsStringAsync();
    if (!response.IsSuccessStatusCode)
    {
    return new Response
    {
        IsSuccess = false,
        Message = answer,
    };
    }
```

```
            var obj = JsonConvert.DeserializeObject<T>(answer);
            return new Response
            {
            IsSuccess = true,
            Result = obj,
            };
            }
            catch (Exception ex)
            {
            return new Response
            {
            IsSuccess = false,
            Message = ex.Message,
            };
            }
}
```

21. Modify the **ProfileViewModel**:

```
public ICommand SaveCommand => new RelayCommand(this.Save);
…
private async void Save()
{
            if (string.IsNullOrEmpty(this.User.FirstName))
            {
            await Application.Current.MainPage.DisplayAlert(
            "Error",
            "You must enter the first name.",
            "Accept");
            return;
            }
```

```
if (string.IsNullOrEmpty(this.User.LastName))
{
await Application.Current.MainPage.DisplayAlert(
"Error",
"You must enter the last name.",
"Accept");
return;
}

if (this.Country == null)
{
await Application.Current.MainPage.DisplayAlert(
"Error",
"You must select a country.",
"Accept");
return;
}

if (this.City == null)
{
await Application.Current.MainPage.DisplayAlert(
"Error",
"You must select a city.",
"Accept");
return;
}

if (string.IsNullOrEmpty(this.User.Address))
{
await Application.Current.MainPage.DisplayAlert(
"Error",
"You must enter an address.",
"Accept");
return;
}

if (string.IsNullOrEmpty(this.User.PhoneNumber))
{
await Application.Current.MainPage.DisplayAlert(
"Error",
"You must enter a phone number.",
"Accept");
return;
}

this.IsRunning = true;
this.IsEnabled = false;

var url = Application.Current.Resources["UrlAPI"].ToString();
var response = await this.apiService.PutAsync(
url,
"/api",
"/Account",
this.User,
"bearer",
MainViewModel.GetInstance().Token.Token);

this.IsRunning = false;
this.IsEnabled = true;

if (!response.IsSuccess)
```

```
    {
    await Application.Current.MainPage.DisplayAlert(
    "Error",
    response.Message,
    "Accept");
    return;
    }

    MainViewModel.GetInstance().User = this.User;
    Settings.User = JsonConvert.SerializeObject(this.User);

    await Application.Current.MainPage.DisplayAlert(
    "Ok",
    "User updated!",
    "Accept");
    await App.Navigator.PopAsync();
}
```

22. Modify the **MainViewModel**:

```
public class MainViewModel :BaseViewModel
{
    private static MainViewModel instance;
    private User user;

    public User User
    {
    get => this.user;
    set => this.SetValue(ref this.user, value);
    }
```

23. Test it.

Modify Password From App in Xamarin Forms

1. Add the **ChangePasswordRequest** in **Common.Models**:

```
using System.ComponentModel.DataAnnotations;

public class ChangePasswordRequest
{
        [Required]
        public string OldPassword { get; set; }

        [Required]
        public string NewPassword { get; set; }

        [Required]
        public string Email { get; set; }
}
```

2. Add the method **ChangePassword** in **AccountController** in **Web.Controllers.API**:

```
[HttpPost]
[Route("ChangePassword")]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
public async Task<IActionResult> ChangePassword([FromBody] ChangePasswordRequest request)
{
        if (!ModelState.IsValid)
        {
        return this.BadRequest(new Response
        {
        IsSuccess = false,
        Message = "Bad request"
        });
        }

        var user = await this.userHelper.GetUserByEmailAsync(request.Email);
        if (user == null)
        {
        return this.BadRequest(new Response
        {
        IsSuccess = false,
        Message = "This email is not assigned to any user."
        });
        }

        var result = await this.userHelper.ChangePasswordAsync(user, request.OldPassword, request.NewPassword);
        if (!result.Succeeded)
        {
        return this.BadRequest(new Response
        {
        IsSuccess = false,
        Message = result.Errors.FirstOrDefault().Description
        });
        }

        return this.Ok(new Response
        {
        IsSuccess = true,
        Message = "The password was changed succesfully!"
        });
```

```
}
```

3. Publish on Azure.

4. Add the method **ChangePasswordAsync** in **ApiService**:

```
public async Task<Response> ChangePasswordAsync(
        string urlBase,
        string servicePrefix,
        string controller,
        ChangePasswordRequest changePasswordRequest,
        string tokenType,
        string accessToken)
{
        try
        {
        var request = JsonConvert.SerializeObject(changePasswordRequest);
        var content = new StringContent(request, Encoding.UTF8, "application/json");
        var client = new HttpClient
        {
        BaseAddress = new Uri(urlBase)
        };

        client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue(tokenType, accessToken);
        var url = $"{servicePrefix}{controller}";
        var response = await client.PostAsync(url, content);
        var answer = await response.Content.ReadAsStringAsync();
        var obj = JsonConvert.DeserializeObject<Response>(answer);
        return obj;
        }
        catch (Exception ex)
        {
        return new Response
        {
        IsSuccess = false,
        Message = ex.Message,
        };
        }
}
```

5. Add the **ChangePasswordPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="Shop.UIForms.Views.ChangePasswordPage"
        BindingContext="{Binding Main, Source={StaticResource Locator}}"
        Title="Change Password">
        <ContentPage.Content>
        <ScrollView
        BindingContext="{Binding ChangePassword}">
        <StackLayout
                Padding="8">
                <Grid>
                <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*"/>
                <ColumnDefinition Width="2*"/>
                </Grid.ColumnDefinitions>
                <Label
                Grid.Column="0"
                Grid.Row="0"
                Text="Current password"
```

```
VerticalOptions="Center">
</Label>
<Entry
Grid.Column="1"
Grid.Row="0"
IsPassword="True"
Placeholder="Enter your current password..."
Text="{Binding CurrentPassword}">
</Entry>
<Label
Grid.Column="0"
Grid.Row="1"
Text="New password"
VerticalOptions="Center">
</Label>
<Entry
Grid.Column="1"
Grid.Row="1"
IsPassword="True"
Placeholder="Enter the new password..."
Text="{Binding NewPassword}">
</Entry>
<Label
Grid.Column="0"
Grid.Row="2"
Text="Confirm new password"
VerticalOptions="Center">
</Label>
<Entry
Grid.Column="1"
Grid.Row="2"
```

```
        IsPassword="True"
        Placeholder="Renter the new password..."
        Text="{Binding PasswordConfirm}">
        </Entry>
        </Grid>
        <ActivityIndicator
        IsRunning="{Binding IsRunning}"
        VerticalOptions="CenterAndExpand">
        </ActivityIndicator>
        <Button
        BackgroundColor="Navy"
        BorderRadius="23"
        Command="{Binding ChangePasswordCommand}"
        HeightRequest="46"
        HorizontalOptions="FillAndExpand"
        IsEnabled="{Binding IsEnabled}"
        Text="Change Password"
        TextColor="White">
        </Button>
    </StackLayout>
    </ScrollView>
    </ContentPage.Content>
</ContentPage>
```

6. Add the **ChangePasswordViewModel**:

```
using System.Windows.Input;
using Common.Models;
using Common.Services;
using GalaSoft.MvvmLight.Command;
using Shop.Common.Helpers;
using Xamarin.Forms;

public class ChangePasswordViewModel : BaseViewModel
{
        private readonly ApiService apiService;
        private bool isRunning;
        private bool isEnabled;

        public string CurrentPassword { get; set; }

        public string NewPassword { get; set; }

        public string PasswordConfirm { get; set; }

        public bool IsRunning
        {
        get => this.isRunning;
        set => this.SetValue(ref this.isRunning, value);
        }

        public bool IsEnabled
        {
        get => this.isEnabled;
        set => this.SetValue(ref this.isEnabled, value);
        }

        public ICommand ChangePasswordCommand => new RelayCommand(this.ChangePassword);

        public ChangePasswordViewModel()
        {
        this.apiService = new ApiService();
        this.IsEnabled = true;
        }

        private async void ChangePassword()
        {
        if (string.IsNullOrEmpty(this.CurrentPassword))
        {
        await Application.Current.MainPage.DisplayAlert(
                "Error",
                "You must enter the current password.",
                "Accept");
        return;
        }

        if (!MainViewModel.GetInstance().UserPassword.Equals(this.CurrentPassword))
        {
        await Application.Current.MainPage.DisplayAlert(
                "Error",
                "The current password is incorrect.",
                "Accept");
        return;
        }

        if (string.IsNullOrEmpty(this.NewPassword))
        {
        await Application.Current.MainPage.DisplayAlert(
                "Error",
                "You must enter the new password.",
                "Accept");
        return;
        }
```

```
if (this.NewPassword.Length < 6)
{
await Application.Current.MainPage.DisplayAlert(
        "Error",
        "The password must have at least 6 characters length.",
        "Accept");
return;
}

if (string.IsNullOrEmpty(this.PasswordConfirm))
{
await Application.Current.MainPage.DisplayAlert(
        "Error",
        "You must enter the password confirm.",
        "Accept");
return;
}

if (!this.NewPassword.Equals(this.PasswordConfirm))
{
await Application.Current.MainPage.DisplayAlert(
        "Error",
        "The password and confirm does not match.",
        "Accept");
return;
}

this.IsRunning = true;
this.IsEnabled = false;
```

```
var request = new ChangePasswordRequest
{
Email = MainViewModel.GetInstance().UserEmail,
NewPassword = this.NewPassword,
OldPassword = this.CurrentPassword
};

var url = Application.Current.Resources["UrlAPI"].ToString();
var response = await this.apiService.ChangePasswordAsync(
url,
"/api",
"/Account/ChangePassword",
request,
"bearer",
MainViewModel.GetInstance().Token.Token);

this.IsRunning = false;
this.IsEnabled = true;

if (!response.IsSuccess)
{
await Application.Current.MainPage.DisplayAlert(
        "Error",
        response.Message,
        "Accept");
return;
}

MainViewModel.GetInstance().UserPassword = this.NewPassword;
Settings.UserPassword = this.NewPassword;
```

```
            await Application.Current.MainPage.DisplayAlert(
            "Ok",
            response.Message,
            "Accept");

            await App.Navigator.PopAsync();
            }
}
```

7. Add the property in **MainViewModel**:

```
public ChangePasswordViewModel ChangePassword { get; set; }
```

8. Modify the **ProfileViewModel**:

```
public ICommand ModifyPasswordCommand => new RelayCommand(this.ModifyPassword);
…
private async void ModifyPassword()
{
            MainViewModel.GetInstance().ChangePassword = new ChangePasswordViewModel();
            await App.Navigator.PushAsync(new ChangePasswordPage());
}
```

9. Test it.

Add Icon & Splash to Xamarin Forms For Android

1. Add a new image with the Splash in drawable, the dimensions are: 480 x 800 pixels. In the sample: **splash.png**.

2. Add this lines to **styles.xml**.

```
  </style>
  <style name="Theme.Splash" parent="android:Theme">
        <item name="android:windowBackground">@drawable/splash</item>
        <item name="android:windowNoTitle">true</item>
  </style>
</resources>
```

3. In Xamarin Android root project, add the **SplashActivity**.

```
using Android.App;
using Android.OS;

[Activity(
        Theme = "@style/Theme.Splash",
        MainLauncher = true,
        NoHistory = true)]
public class SplashActivity : Activity
{
        protected override void OnCreate(Bundle bundle)
        {
        base.OnCreate(bundle);
        System.Threading.Thread.Sleep(1800);
        this.StartActivity(typeof(MainActivity));
        }
}
```

4. Modify the **MainActivity** to change **MainLauncher** property to **false**.

```
[Activity(
        Label = "Shop",
        Icon = "@mipmap/icon",
        Theme = "@style/MainTheme",
        MainLauncher = false,
        ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
```

5. Test it.

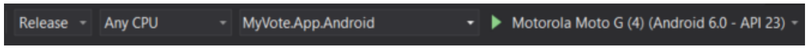
6. Now add the icon launcher. Go to https://romannurik.github.io/AndroidAssetStudio/ and personalizate your own icon launcher. And add the image to Android and iOS projects.

7. And define the application name in Android Properties.

8. Test it.

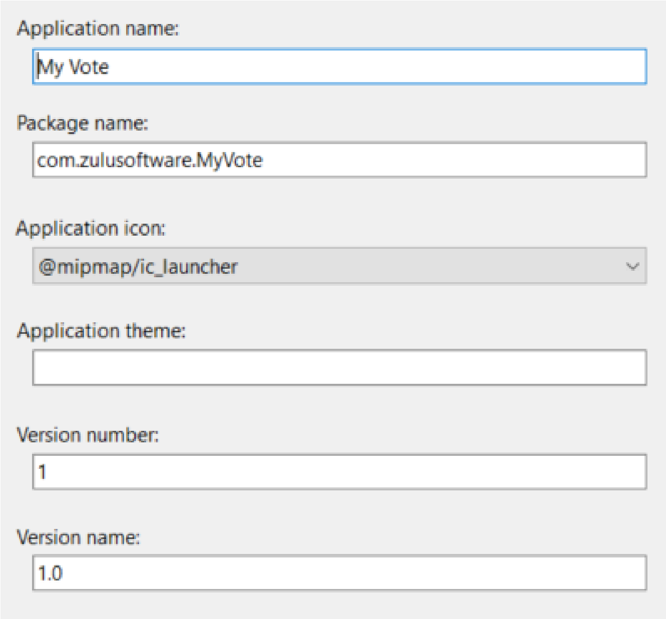
Publish on Google Play

1. Set the project on Release mode:



Release | Any CPU | MyVote.App.Android | ▶ Motorola Moto G (4) (Android 6.0 - API 23)

2. Ensure the you App have an Icon and this other properties:



Application name:
My Vote

Package name:
com.zulusoftware.MyVote

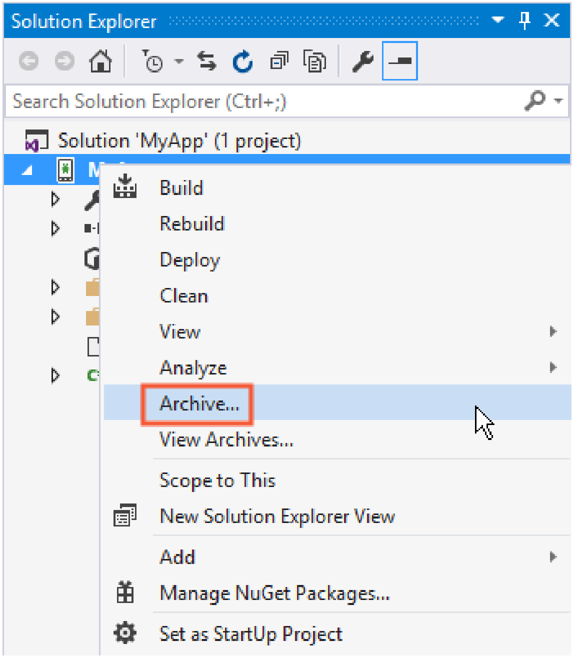
Application icon:
@mipmap/ic_launcher

Application theme:

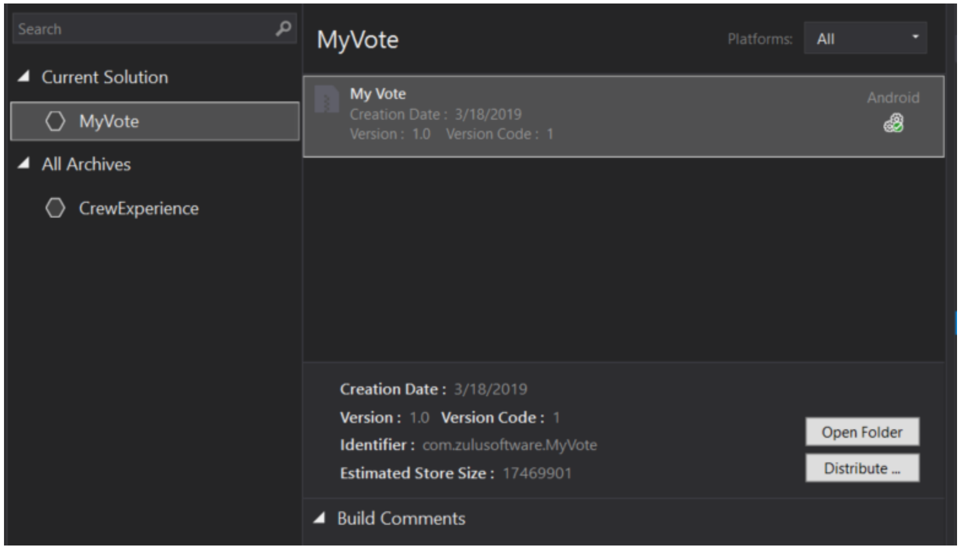
Version number:
1

Version name:
1.0

3. Generate the APK for Publish:



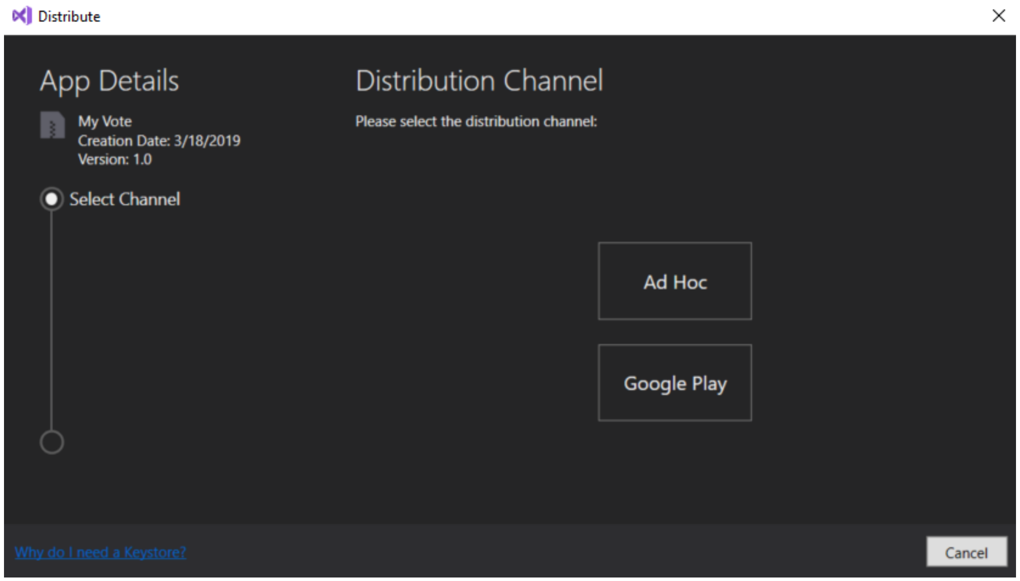
4. Then click on “Distribute”:



My Vote
Creation Date : 3/18/2019
Version : 1.0 Version Code : 1

Creation Date : 3/18/2019
Version : 1.0 Version Code : 1
Identifier : com.zulusoftware.MyVote
Estimated Store Size : 17469901

5. Then in a “Ah Hoc” button:



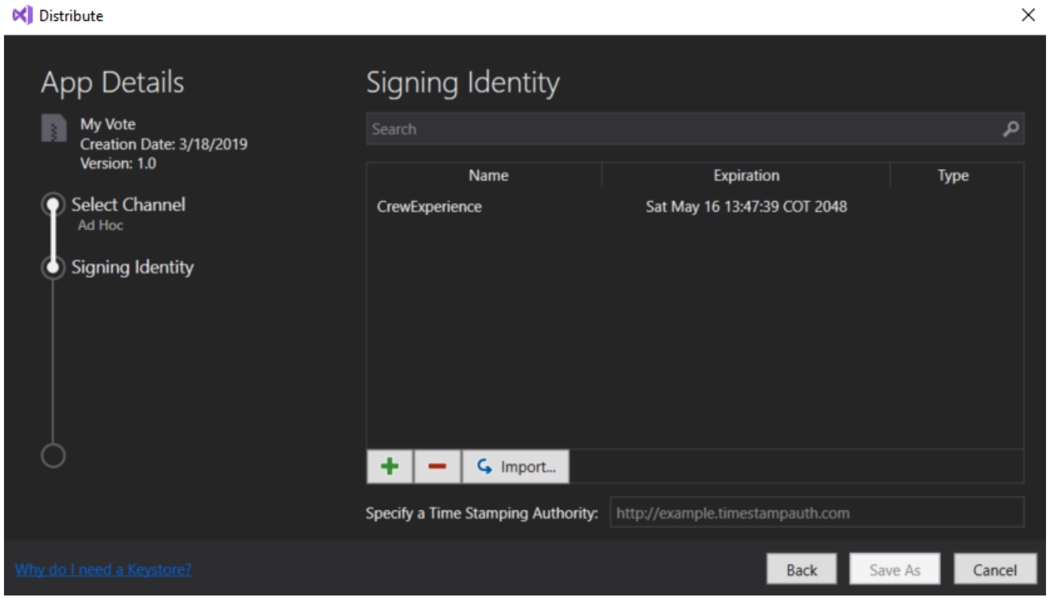
App Details

My Vote
Creation Date: 3/18/2019
Version: 1.0

Distribution Channel

Please select the distribution channel:

6. The first time, you need to create the “Sign” for the app, you shouldn’t forget the key :



My Vote
Creation Date: 3/18/2019
Version: 1.0

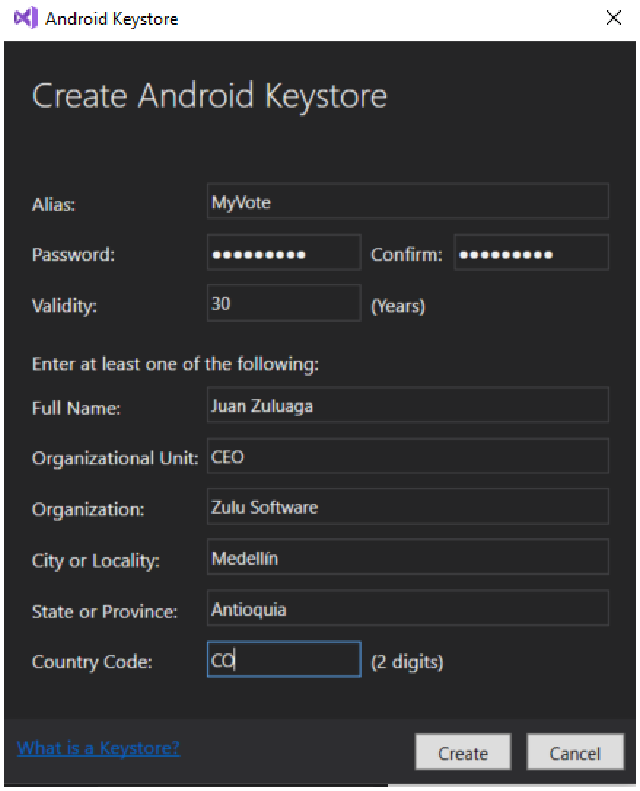
Select Channel
Ad Hoc

Signing Identity

Name	Expiration	Type
CrewExperience	Sat May 16 13:47:39 COT 2048	

Specify a Time Stamping Authority: http://example.timestampauth.com

7. Set the project on Release mode:



Create Android Keystore

Alias: MyVote

Password: ●●●●●●●●● Confirm: ●●●●●●●●●

Validity: 30 (Years)

Enter at least one of the following:

Full Name: Juan Zuluaga

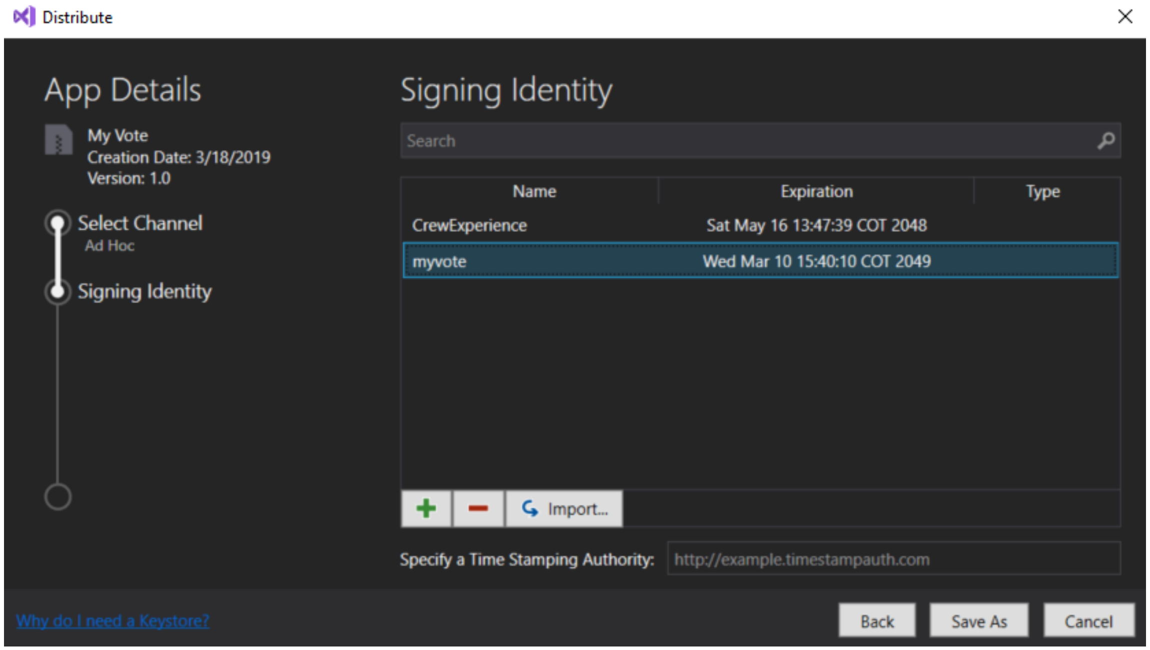
Organizational Unit: CEO

Organization: Zulu Software

City or Locality: Medellín

State or Province: Antioquia

Country Code: CO (2 digits)

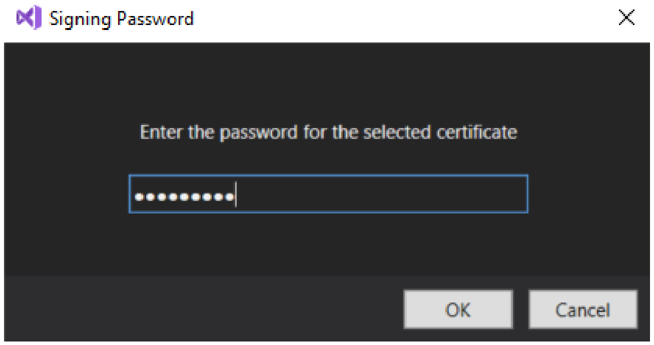


My Vote
Creation Date: 3/18/2019
Version: 1.0

Name	Expiration	Type
CrewExperience	Sat May 16 13:47:39 COT 2048	
myvote	Wed Mar 10 15:40:10 COT 2049	

Specify a Time Stamping Authority: http://example.timestampauth.com

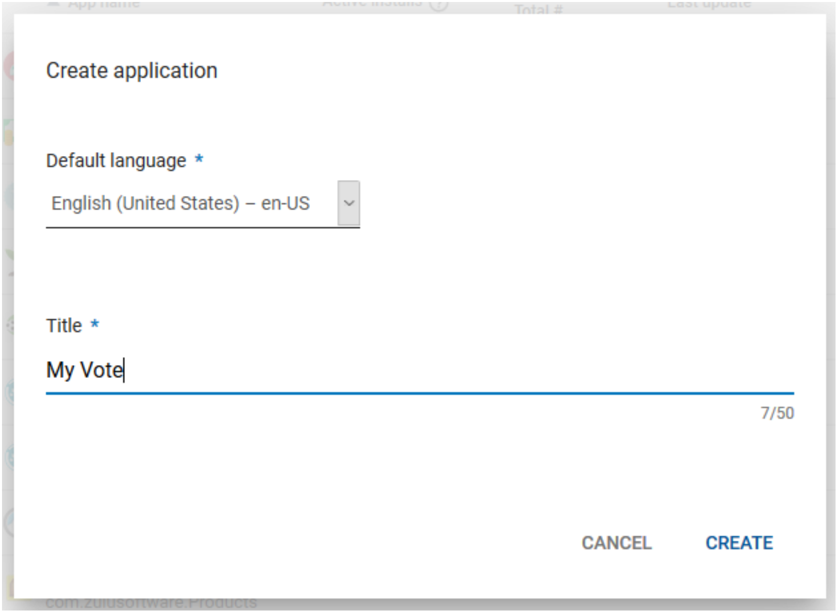
8. Click on “Save As”, select a location and then put the password provide before:



Enter the password for the selected certificate

••••••••••

9. You need the generated APK for the following steps. Enter to: https://developer.android.com/distribute/console?hl=es and click on “CREATE APPLICATION”:



Create application

Default language *

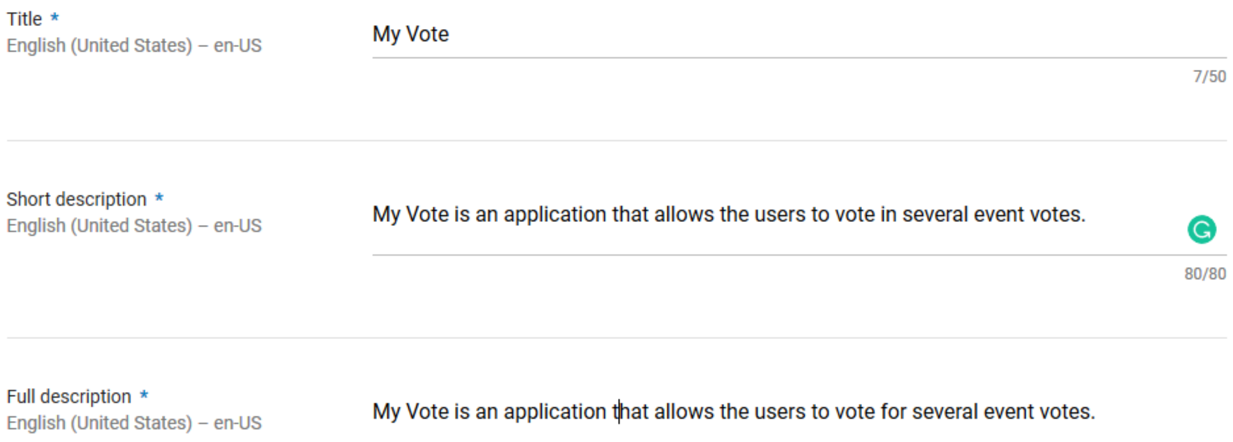
English (United States) – en-US

Title *

My Vote

7/50

10. Fill the form:



Title *
English (United States) – en-US

My Vote

7/50

Short description *
English (United States) – en-US

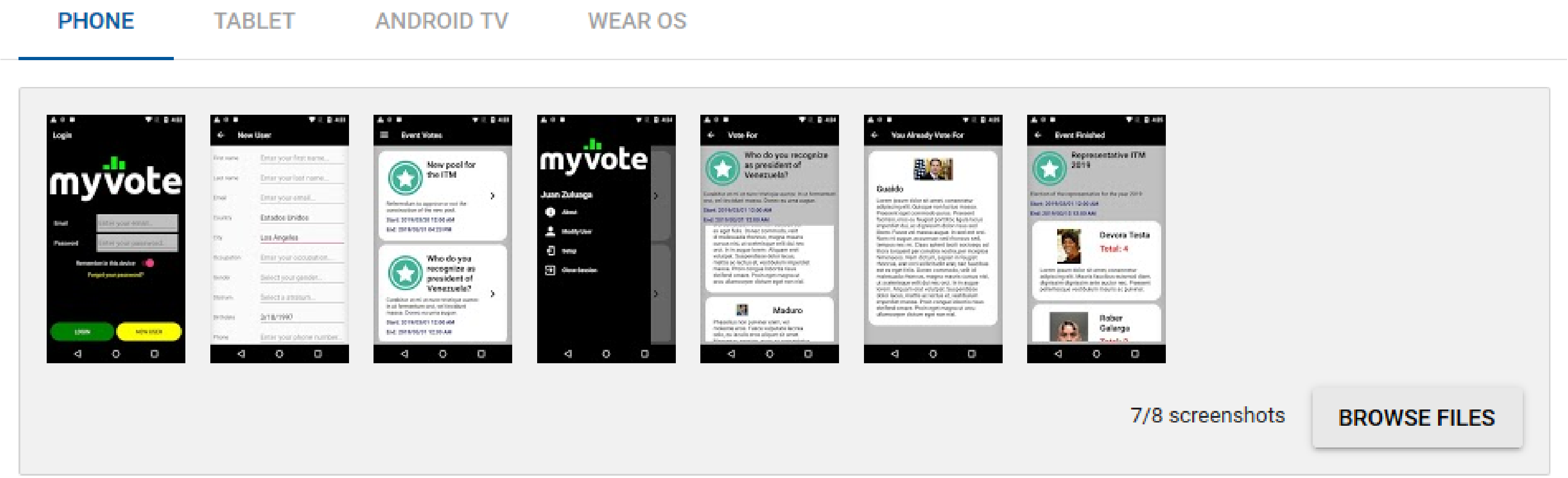
My Vote is an application that allows the users to vote in several event votes.

80/80

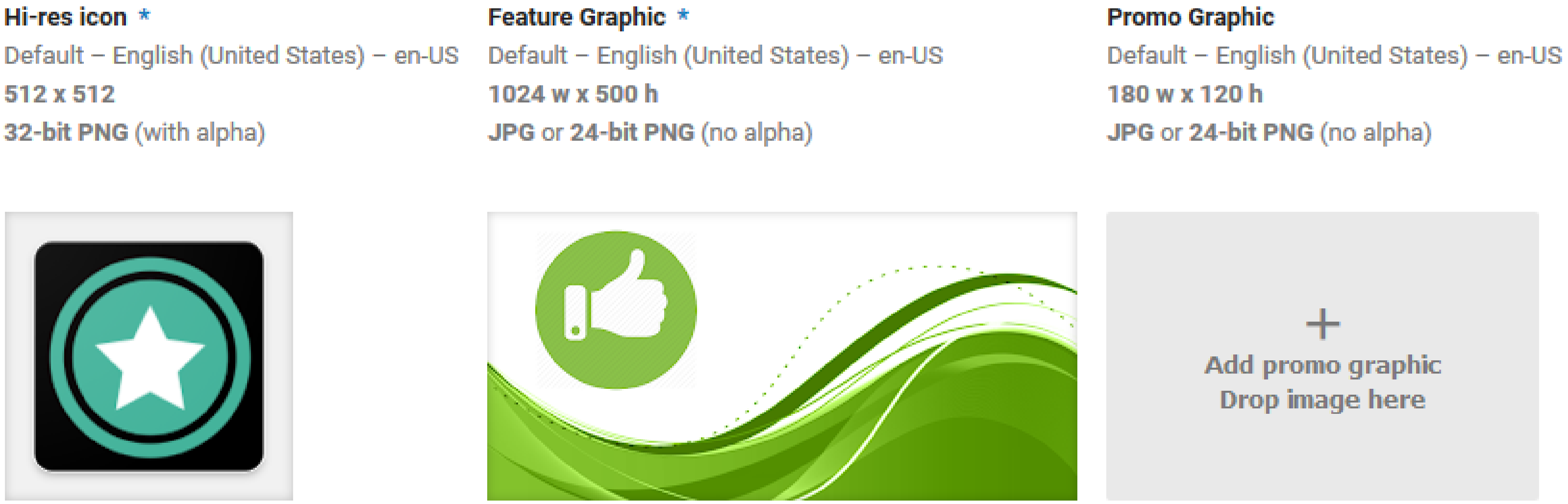
Full description *
English (United States) – en-US

My Vote is an application that allows the users to vote for several event votes.

11. Take application screenshots and put in this page:



12. You need the application icon in 512 x 512 pixels and image in 1024 x 500. Add in this page:

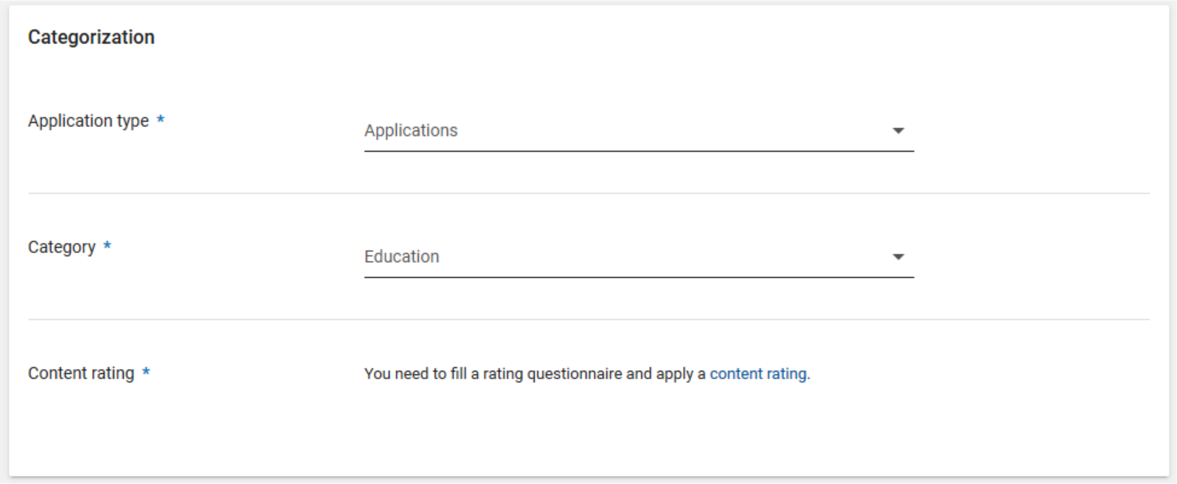


Hi-res icon *
Default – English (United States) – en-US
512 x 512
32-bit PNG (with alpha)

Feature Graphic *
Default – English (United States) – en-US
1024 w x 500 h
JPG or **24-bit PNG** (no alpha)

Promo Graphic
Default – English (United States) – en-US
180 w x 120 h
JPG or **24-bit PNG** (no alpha)

13. Fill this sections:, save as a draft and answer the content rating questionnaire:

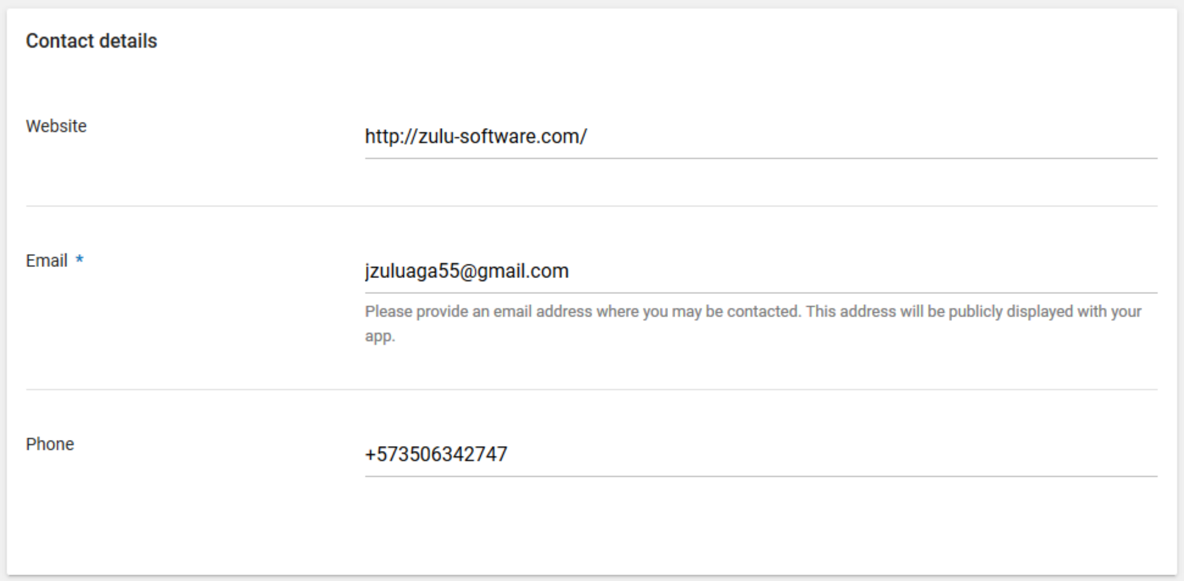


Categorization

Application type * Applications ▾

Category * Education ▾

Content rating * You need to fill a rating questionnaire and apply a content rating.



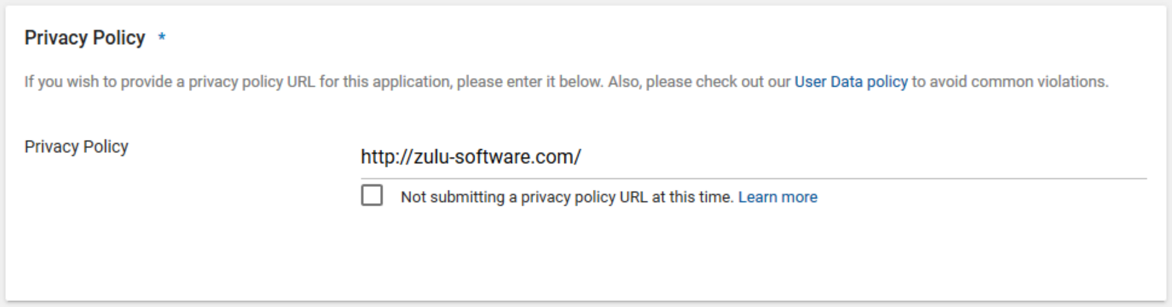
Contact details

Website http://zulu-software.com/

Email * jzuluaga55@gmail.com

Please provide an email address where you may be contacted. This address will be publicly displayed with your app.

Phone +573506342747



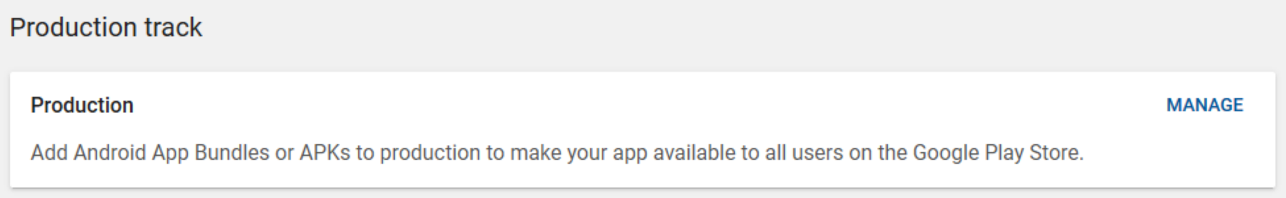
Privacy Policy *

If you wish to provide a privacy policy URL for this application, please enter it below. Also, please check out our User Data policy to avoid common violations.

Privacy Policy http://zulu-software.com/

□ Not submitting a privacy policy URL at this time. Learn more

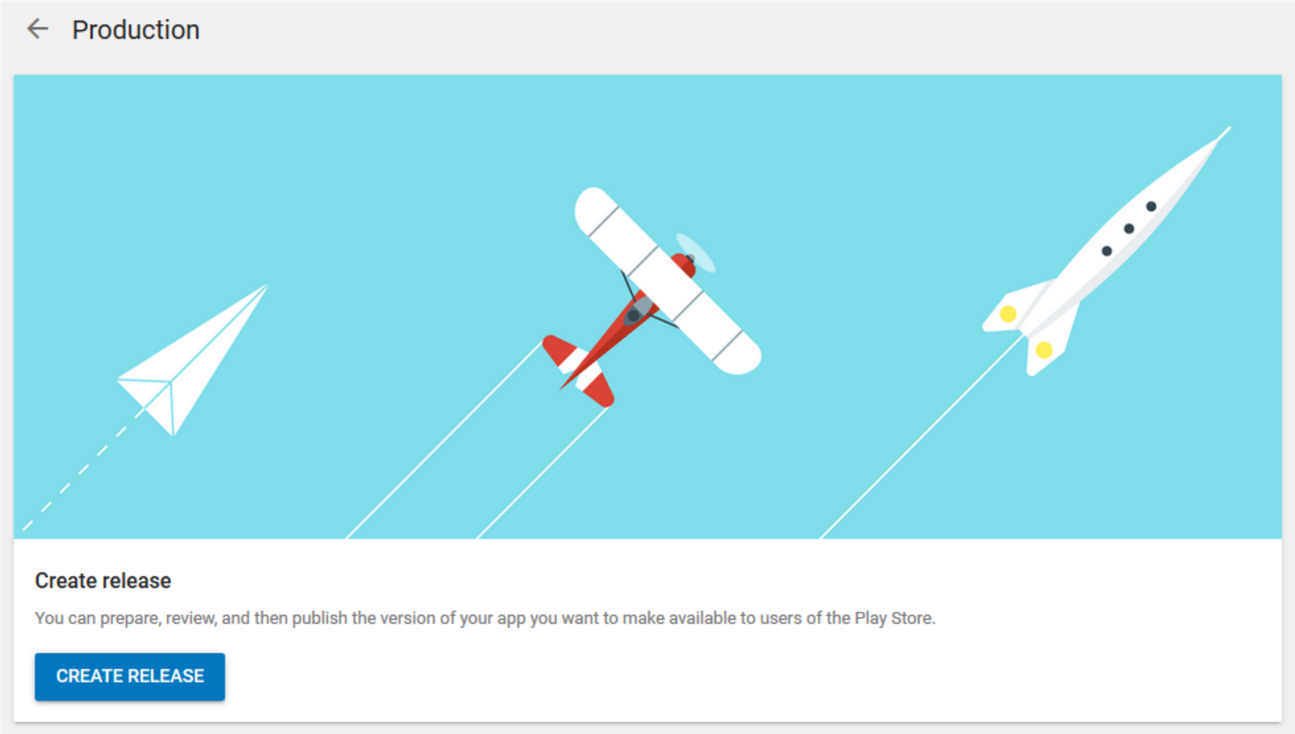
14. Go to App Releases and create a new one in production.



Production track

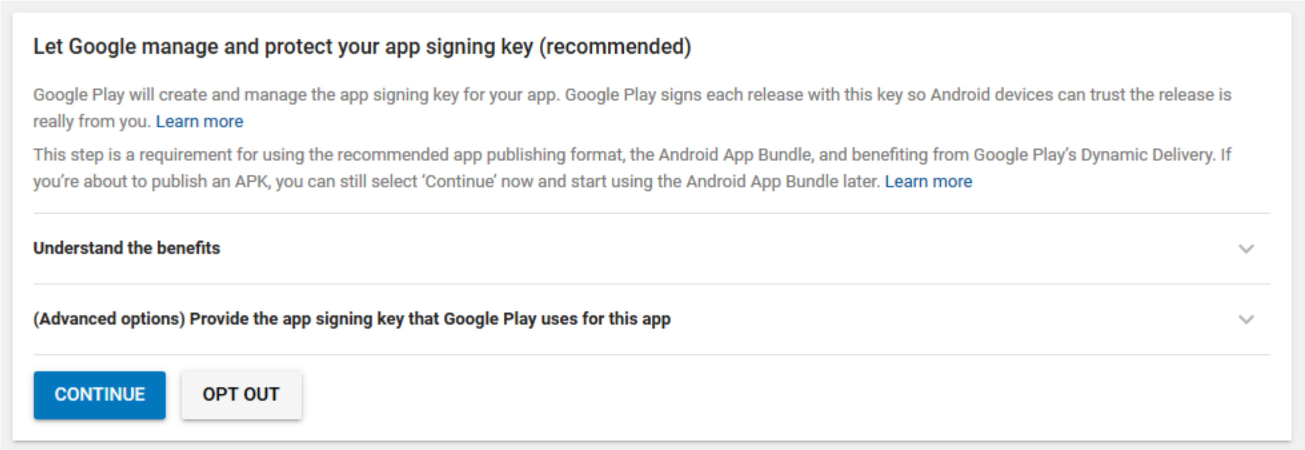
Production

Add Android App Bundles or APKs to production to make your app available to all users on the Google Play Store.



Create release

You can prepare, review, and then publish the version of your app you want to make available to users of the Play Store.



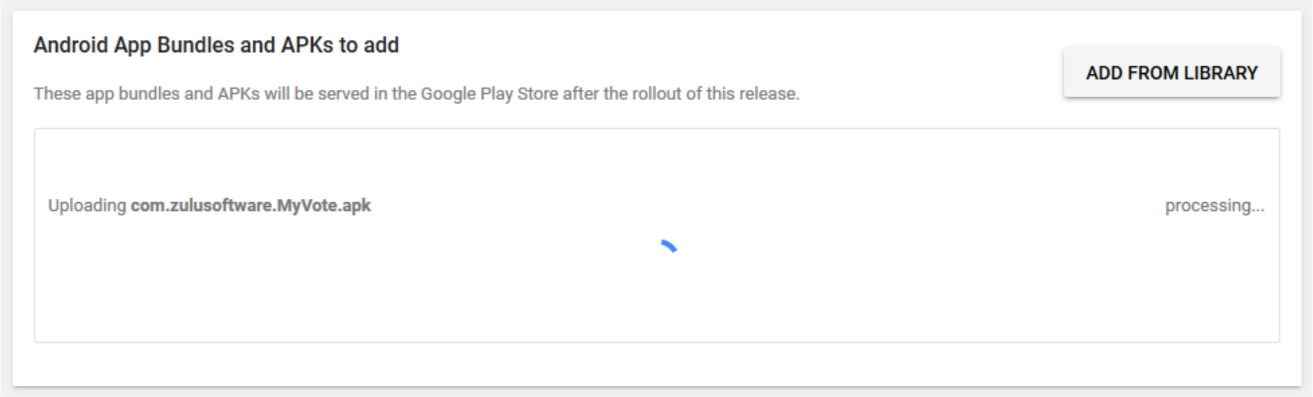
Let Google manage and protect your app signing key (recommended)

Google Play will create and manage the app signing key for your app. Google Play signs each release with this key so Android devices can trust the release is really from you. Learn more

This step is a requirement for using the recommended app publishing format, the Android App Bundle, and benefiting from Google Play's Dynamic Delivery. If you're about to publish an APK, you can still select 'Continue' now and start using the Android App Bundle later. Learn more

Understand the benefits
(Advanced options) Provide the app signing key that Google Play uses for this app
CONTINUE
OPT OUT

15. Upload your APK:



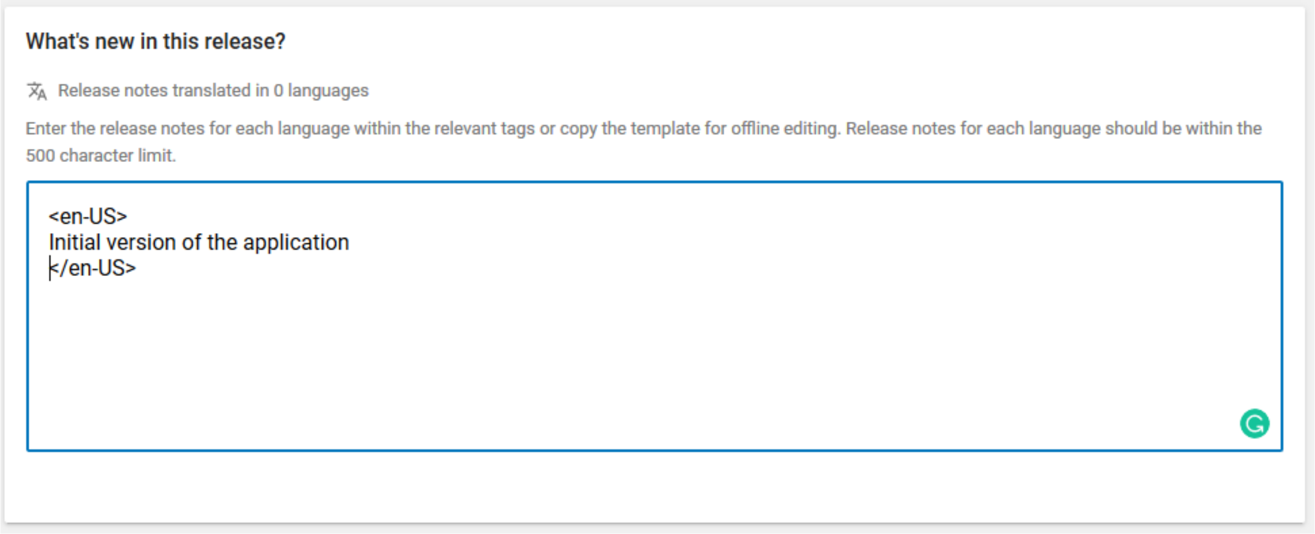
Android App Bundles and APKs to add

These app bundles and APKs will be served in the Google Play Store after the rollout of this release.

Uploading **com.zulusoftware.MyVote.apk**

processing...

16. Fill this and save:



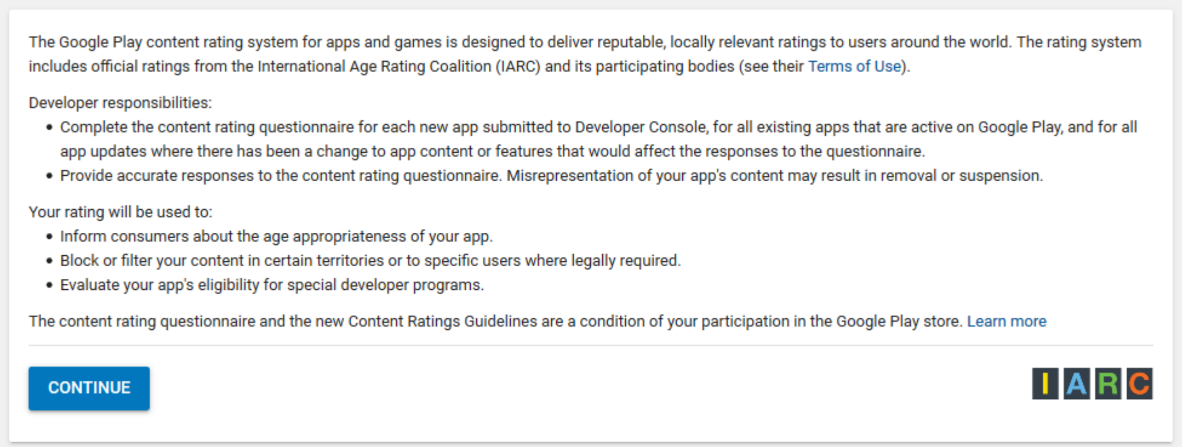
What's new in this release?

Release notes translated in 0 languages

Enter the release notes for each language within the relevant tags or copy the template for offline editing. Release notes for each language should be within the 500 character limit.

```
<en-US>
Initial version of the application
</en-US>
```

17. Go to content rating:



The Google Play content rating system for apps and games is designed to deliver reputable, locally relevant ratings to users around the world. The rating system includes official ratings from the International Age Rating Coalition (IARC) and its participating bodies (see their Terms of Use).

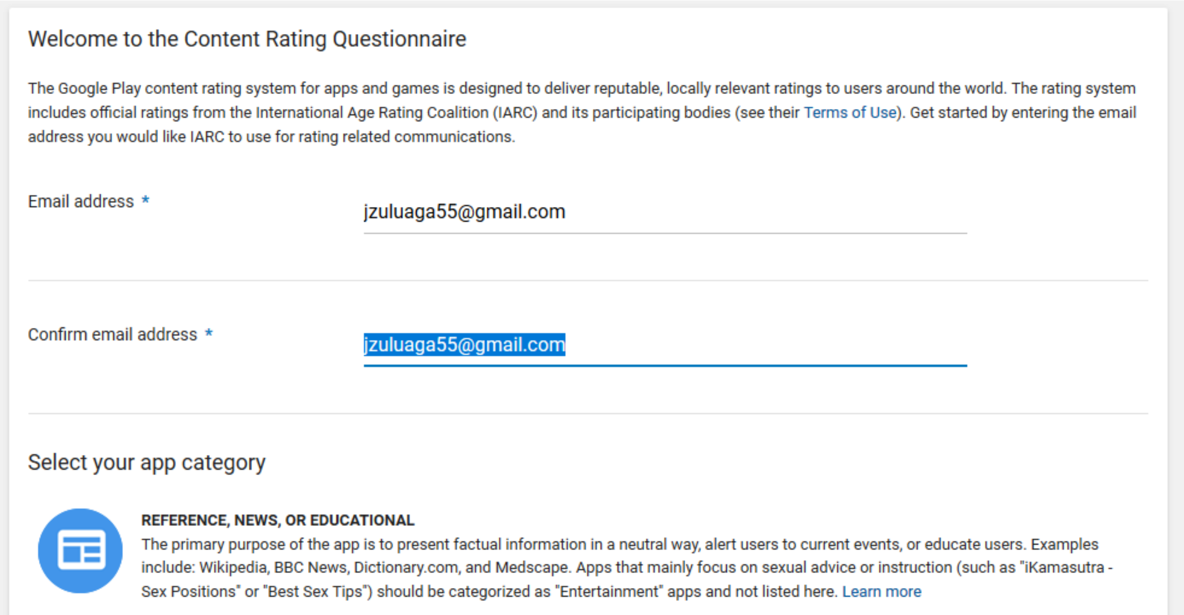
Developer responsibilities:

- Complete the content rating questionnaire for each new app submitted to Developer Console, for all existing apps that are active on Google Play, and for all app updates where there has been a change to app content or features that would affect the responses to the questionnaire.
- Provide accurate responses to the content rating questionnaire. Misrepresentation of your app's content may result in removal or suspension.

Your rating will be used to:

- Inform consumers about the age appropriateness of your app.
- Block or filter your content in certain territories or to specific users where legally required.
- Evaluate your app's eligibility for special developer programs.

The content rating questionnaire and the new Content Ratings Guidelines are a condition of your participation in the Google Play store. Learn more



Welcome to the Content Rating Questionnaire

The Google Play content rating system for apps and games is designed to deliver reputable, locally relevant ratings to users around the world. The rating system includes official ratings from the International Age Rating Coalition (IARC) and its participating bodies (see their Terms of Use). Get started by entering the email address you would like IARC to use for rating related communications.

Email address * jzuluaga55@gmail.com

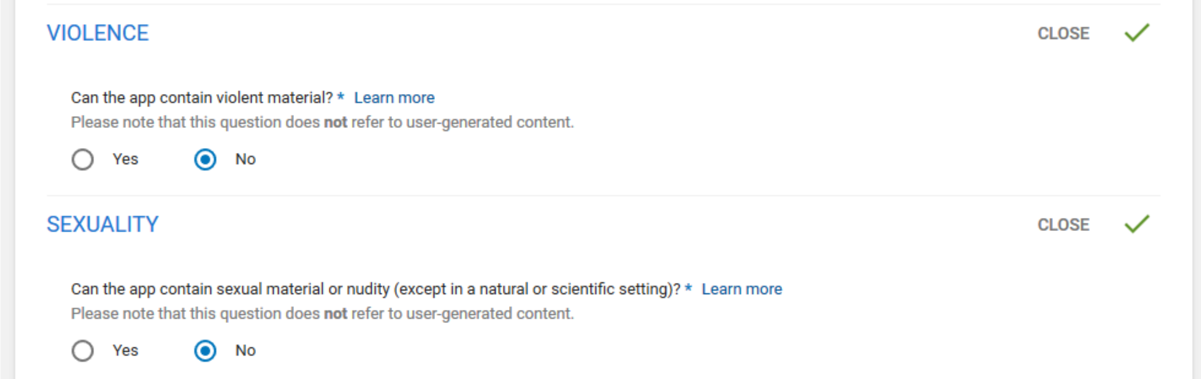
Confirm email address * jzuluaga55@gmail.com

Select your app category

REFERENCE, NEWS, OR EDUCATIONAL

The primary purpose of the app is to present factual information in a neutral way, alert users to current events, or educate users. Examples include: Wikipedia, BBC News, Dictionary.com, and Medscape. Apps that mainly focus on sexual advice or instruction (such as "iKamasutra - Sex Positions" or "Best Sex Tips") should be categorized as "Entertainment" apps and not listed here. Learn more

18. Answer all the questions and click on “SAVE QUESTIONNAIRE”:



VIOLENCE

Can the app contain violent material? * Learn more

Please note that this question does **not** refer to user-generated content.

○ Yes ● No

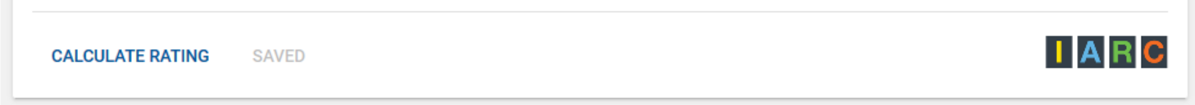
SEXUALITY

Can the app contain sexual material or nudity (except in a natural or scientific setting)? * Learn more

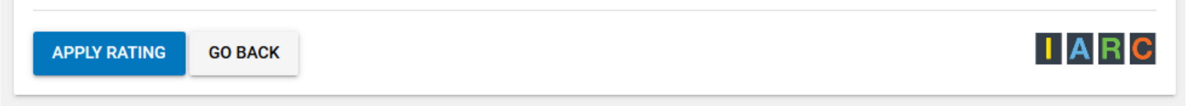
Please note that this question does **not** refer to user-generated content.

○ Yes ● No

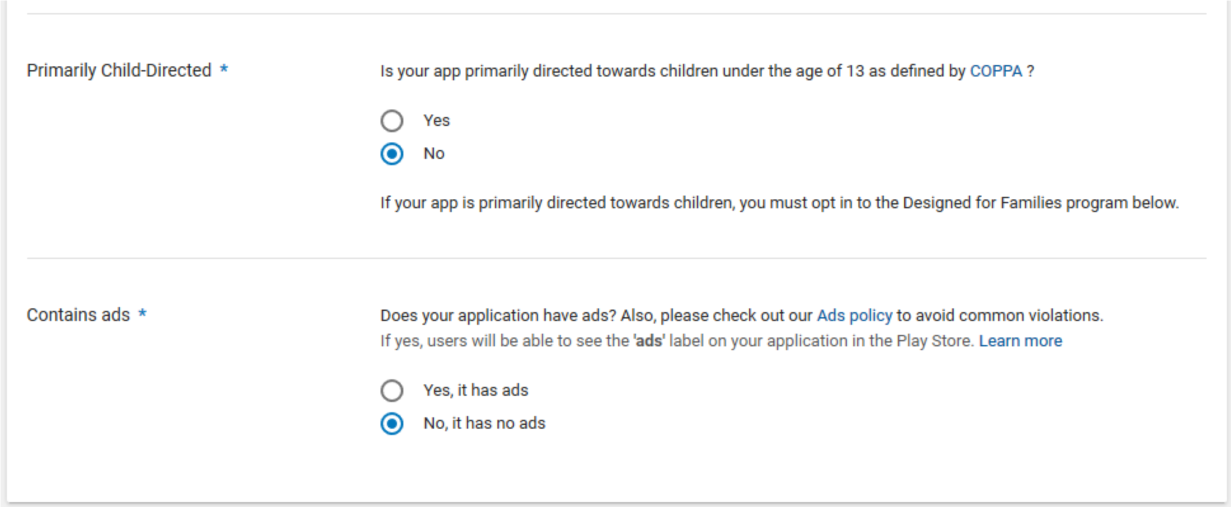
19. The go to “Content rating” and click on “CALCULATE RATING”:



20. And then “APPLY RATING”:



21. Now go pricing & distribution. Fill the required fields in this part:



Primarily Child-Directed *

Is your app primarily directed towards children under the age of 13 as defined by COPPA ?

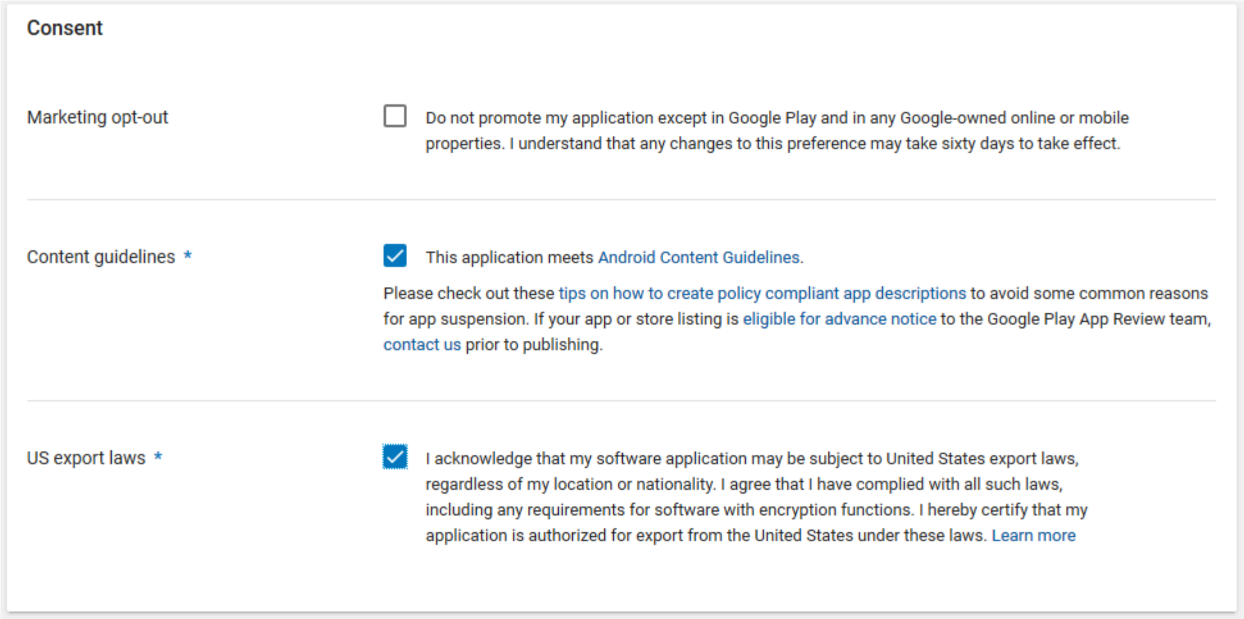
○ Yes
● No

If your app is primarily directed towards children, you must opt in to the Designed for Families program below.

Contains ads *

Does your application have ads? Also, please check out our Ads policy to avoid common violations.
If yes, users will be able to see the **'ads'** label on your application in the Play Store. Learn more

○ Yes, it has ads
● No, it has no ads



Consent

Marketing opt-out

☐ Do not promote my application except in Google Play and in any Google-owned online or mobile properties. I understand that any changes to this preference may take sixty days to take effect.

Content guidelines *

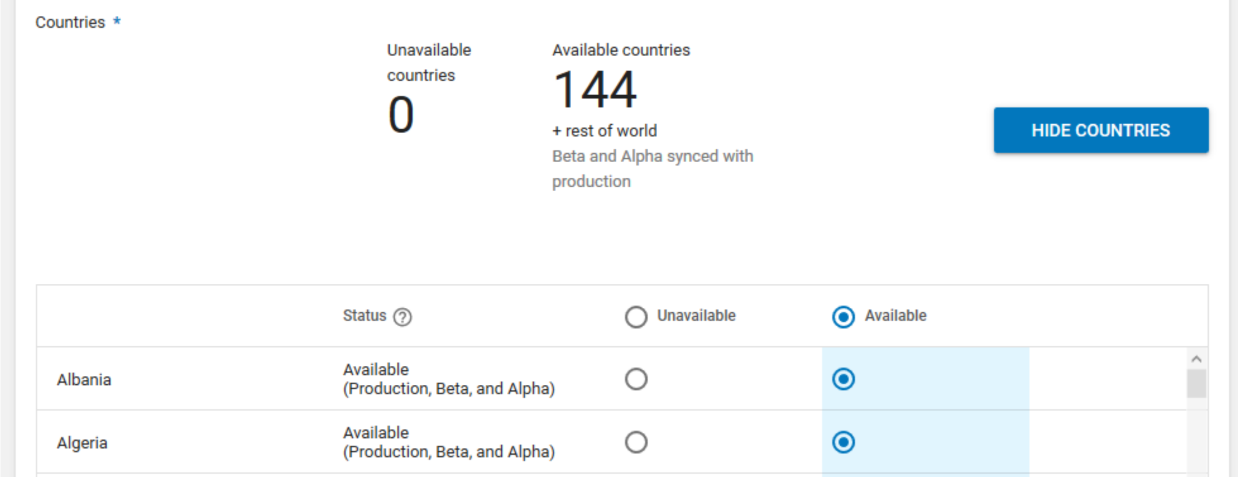
☑ This application meets Android Content Guidelines.

Please check out these tips on how to create policy compliant app descriptions to avoid some common reasons for app suspension. If your app or store listing is eligible for advance notice to the Google Play App Review team, contact us prior to publishing.

US export laws *

☑ I acknowledge that my software application may be subject to United States export laws, regardless of my location or nationality. I agree that I have complied with all such laws, including any requirements for software with encryption functions. I hereby certify that my application is authorized for export from the United States under these laws. Learn more

22. Make the available countries:



Countries *

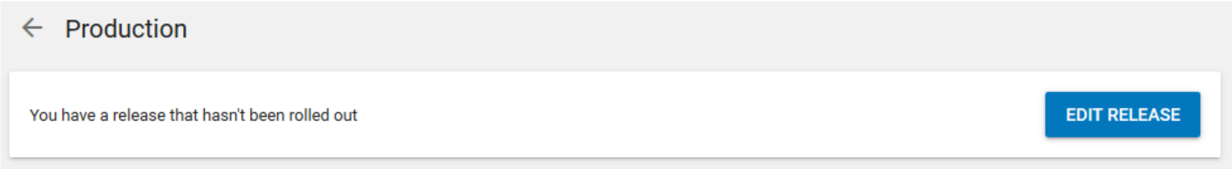
Unavailable countries
0

Available countries
144
+ rest of world
Beta and Alpha synced with production

HIDE COUNTRIES

	Status ⓘ	○ Unavailable	● Available
Albania	Available (Production, Beta, and Alpha)	○	●
Algeria	Available (Production, Beta, and Alpha)	○	●

23. Save and click on “Ready to publish”. Click on “EDIT RELEASE”:

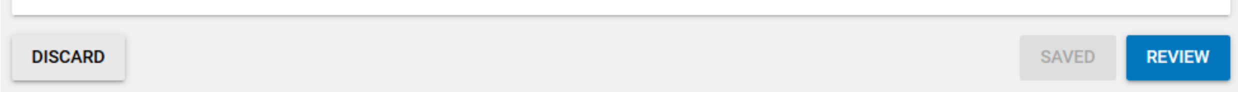


← Production

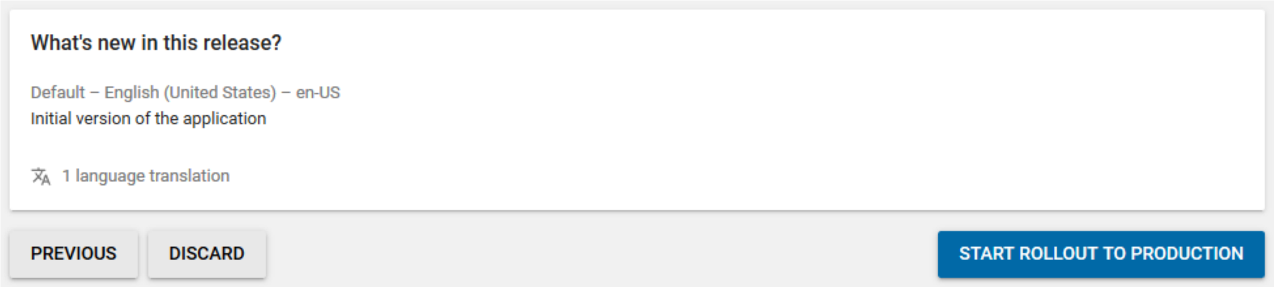
You have a release that hasn't been rolled out

EDIT RELEASE

24. The click on “REVIEW”:

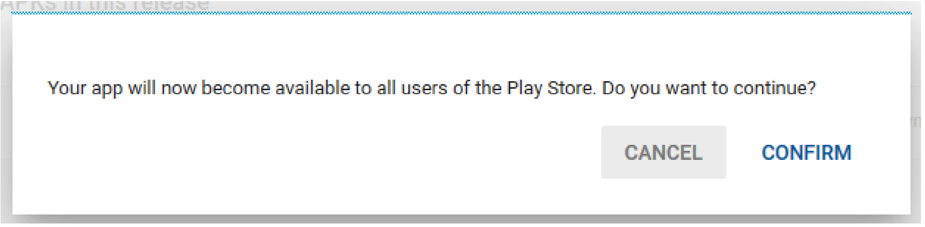


25. And finally in “START ROLLOUT TO PRODUCTION”:



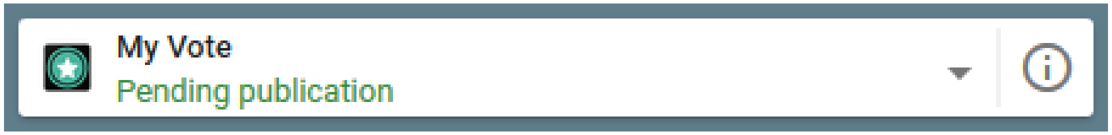
What's new in this release?

Default – English (United States) – en-US
Initial version of the application



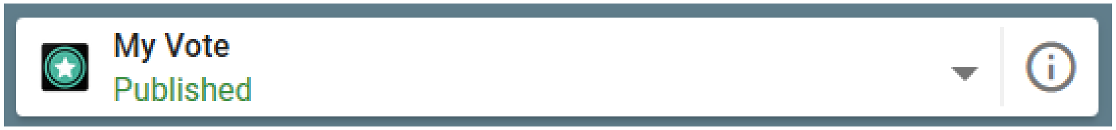
Your app will now become available to all users of the Play Store. Do you want to continue?

26. Then wait some time, about 40 minutes to be able to download from Google Play.



My Vote
Pending publication

27. Wait until appears this message:



My Vote
Published

28. Now you can download the app and test it!

Starting With Xamarin Android Classic

1. In **layout** folder add the **LoginPage**:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <LinearLayout
        android:paddingTop="10dp"
        android:paddingLeft="10dp"
        android:paddingRight="10dp"
        android:orientation="vertical"
        android:minWidth="25px"
        android:minHeight="25px"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <TextView
        android:text="Email"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minWidth="25px"
        android:minHeight="25px"/>
        <EditText
        android:inputType="textEmailAddress"
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/emailText" />
    <TextView
    android:text="Password"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:minWidth="25px"
    android:minHeight="25px"/>
    <EditText
    android:inputType="textPassword"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/passwordText" />
    <ProgressBar
    android:layout_height="wrap_content"
    android:layout_width="match_parent"
    android:id="@+id/activityIndicatorProgressBar"
    android:indeterminateOnly="true"
    android:keepScreenOn="true"/>
    <Button
    android:text="Login"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/loginButton" />
    </LinearLayout>
</RelativeLayout>
```

19. Add the folder **Helpers** and inside it, add the **DiaglogService**:

```
using global::Android.App;
using global::Android.Content;

public static class DiaglogService
{
        public static void ShowMessage(Context context, string title, string message, string button)
        {
        new AlertDialog.Builder(context)
        .SetPositiveButton(button, (sent, args) => { })
        .SetMessage(message)
        .SetTitle(title)
        .SetCancelable(false)
        .Show();
        }
}
```

20. Add the folder **Activities** and inside it, add the **LoginActivity**:

```
using System;
using global::Android.App;
using global::Android.Content;
using global::Android.OS;
using global::Android.Support.V7.App;
using global::Android.Views;
using global::Android.Widget;
using Newtonsoft.Json;
using Shop.Common.Models;
using Shop.Common.Services;
using Shop.UIClassic.Android.Helpers;

[Activity(Label = "@string/app_name", Theme = "@style/AppTheme", MainLauncher = true)]
```

```
public class LoginActivity : AppCompatActivity
{
    private EditText emailText;
    private EditText passwordText;
    private Button loginButton;
    private ApiService apiService;
    private ProgressBar activityIndicatorProgressBar;

    protected override void OnCreate(Bundle savedInstanceState)
    {
    base.OnCreate(savedInstanceState);
    this.SetContentView(Resource.Layout.LoginPage);
    this.FindViews();
    this.HandleEvents();
    this.SetInitialData();
    }

    private void SetInitialData()
    {
    this.apiService = new ApiService();
    this.emailText.Text = "jzuluaga55@gmail.com";
    this.passwordText.Text = "123456";
    this.activityIndicatorProgressBar.Visibility = ViewStates.Invisible;
    }

    private void HandleEvents()
    {
    this.loginButton.Click += this.LoginButton_Click;
    }

    private async void LoginButton_Click(object sender, EventArgs e)
```

```
{
if (string.IsNullOrEmpty(this.emailText.Text))
{
DiaglogService.ShowMessage(this, "Error", "You must enter an email.", "Accept");
return;
}

if (string.IsNullOrEmpty(this.passwordText.Text))
{
DiaglogService.ShowMessage(this, "Error", "You must enter a password.", "Accept");
return;
}

this.activityIndicatorProgressBar.Visibility = ViewStates.Visible;

var request = new TokenRequest
{
Password = this.passwordText.Text,
Username = this.emailText.Text
};

var response = await this.apiService.GetTokenAsync(
"https://shopzulu.azurewebsites.net",
"/Account",
"/CreateToken",
request);

this.activityIndicatorProgressBar.Visibility = ViewStates.Invisible;

if (!response.IsSuccess)
{
    DiaglogService.ShowMessage(this, "Error", "User or password incorrect.", "Accept");
    return;
    }

    DiaglogService.ShowMessage(this, "Ok", "Fuck Yeah!", "Accept");
    }

    private void FindViews()
    {
    this.emailText = this.FindViewById<EditText>(Resource.Id.emailText);
    this.passwordText = this.FindViewById<EditText>(Resource.Id.passwordText);
    this.loginButton = this.FindViewById<Button>(Resource.Id.loginButton);
    this.activityIndicatorProgressBar = this.FindViewById<ProgressBar>(Resource.Id.activityIndicatorProgressBar);
    }
}
```

21. Put the application name in **strings.xml** (in folder **Resources.values**).

```
<resources>
    <string name="app_name">Shop</string>
    <string name="action_settings">Settings</string>
</resources>
```

22. Delete the **MainActivity** and original layout **activity_main.axml**.

23. Test it.

24. In **layout** add the **ProductsPage**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <ListView
        android:minWidth="25px"
        android:minHeight="25px"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/productsListView" />
</LinearLayout>
```

25. In **layout** add the **ProductRow**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:padding="8dp"
        android:orientation="horizontal">
        <ImageView
        android:id="@+id/productImageView"
        android:layout_width="80dp"
        android:layout_height="80dp"
        android:padding="5dp" />
        <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingLeft="10dp">
        <TextView
```

```
    android:id="@+id/nameTextView"
    android:layout_width="match_parent"
    android:textSize="24dp"
    android:layout_height="wrap_content"
    android:textColor="@android:color/black"
    android:layout_alignParentLeft="true"
    android:textStyle="bold"
    android:gravity="left" />
    <TextView
    android:id="@+id/priceTextView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:gravity="left"
    android:textSize="18dp"
    android:textColor="@android:color/black" />
    </LinearLayout>
</LinearLayout>
```

26. In folder **Helpers** add the **ImageHelper**:

```
using System.Net;
using global::Android.Graphics;

public class ImageHelper
{
    public static Bitmap GetImageBitmapFromUrl(string url)
    {
    if (string.IsNullOrEmpty(url))
    {
    return null;
    }

    Bitmap imageBitmap = null;

    using (var webClient = new WebClient())
    {
    var imageBytes = webClient.DownloadData(url);
    if (imageBytes != null && imageBytes.Length > 0)
    {
        imageBitmap = BitmapFactory.DecodeByteArray(imageBytes, 0, imageBytes.Length);
    }
    }

    return imageBitmap;
    }
}
```

27. Create the folder **Adapters** and inside it add the class **ProductsListAdapter**:

```
using System.Collections.Generic;
using Common.Models;
using global::Android.App;
using global::Android.Views;
using global::Android.Widget;
using Helpers;

public class ProductsListAdapter : BaseAdapter<Product>
{
    private readonly List<Product> items;
    private readonly Activity context;
```

```
public ProductsListAdapter(Activity context, List<Product> items) : base()
{
this.context = context;
this.items = items;
}

public override long GetItemId(int position)
{
return position;
}

public override Product this[int position] => items[position];

public override int Count => items.Count;

public override View GetView(int position, View convertView, ViewGroup parent)
{
var item = items[position];

var imageBitmap = ImageHelper.GetImageBitmapFromUrl(item.ImageFullPath);

if (convertView == null)
{
convertView = context.LayoutInflater.Inflate(Resource.Layout.ProductRow, null);
}

convertView.FindViewById<TextView>(Resource.Id.nameTextView).Text = item.Name;
convertView.FindViewById<TextView>(Resource.Id.priceTextView).Text = $"{item.Price:C2}";
convertView.FindViewById<ImageView>(Resource.Id.productImageView).SetImageBitmap(imageBitmap);

return convertView;
```

```
    }
}
```

28. In **Activities** folder add the **ProductsActivity**:

```
using System.Collections.Generic;
using Adapters;
using Common.Models;
using Common.Services;
using global::Android.App;
using global::Android.Content;
using global::Android.OS;
using global::Android.Support.V7.App;
using global::Android.Widget;
using Helpers;
using Newtonsoft.Json;

[Activity(Label = "@string/app_name", Theme = "@style/AppTheme", MainLauncher = true)]
public class ProductsActivity : AppCompatActivity
{
    private TokenResponse token;
    private string email;
    private ApiService apiService;
    private ListView productsListView;

    protected override void OnCreate(Bundle savedInstanceState)
    {
    base.OnCreate(savedInstanceState);
    this.SetContentView(Resource.Layout.ProductsPage);

    this.productsListView = FindViewById<ListView>(Resource.Id.productsListView);
    this.email = Intent.Extras.GetString("email");
    var tokenString = Intent.Extras.GetString("token");
    this.token = JsonConvert.DeserializeObject<TokenResponse>(tokenString);

    this.apiService = new ApiService();
    this.LoadProducts();
    }

    private async void LoadProducts()
    {
    var response = await this.apiService.GetListAsync<Product>(
    "https://shopzulu.azurewebsites.net",
    "/api",
    "/Products",
    "bearer",
    this.token.Token);

    if (!response.IsSuccess)
    {
    DiaglogService.ShowMessage(this, "Error", response.Message, "Accept");
    return;
    }

    var products = (List<Product>)response.Result;
    this.productsListView.Adapter = new ProductsListAdapter(this, products);
    this.productsListView.FastScrollEnabled = true;
    }
}
```

29. Modify the **LoginActivity**:

```
this.activityIndicatorProgressBar.Visibility = ViewStates.Gone;

//DiaglogService.ShowMessage(this, "Ok", "Fuck Yeah!", "Accept");

var token = (TokenResponse)response.Result;
var intent = new Intent(this, typeof(ProductsActivity));
intent.PutExtra("token", JsonConvert.SerializeObject(token));
intent.PutExtra("email", this.emailText.Text);
this.StartActivity(intent);
```

30. Test it.

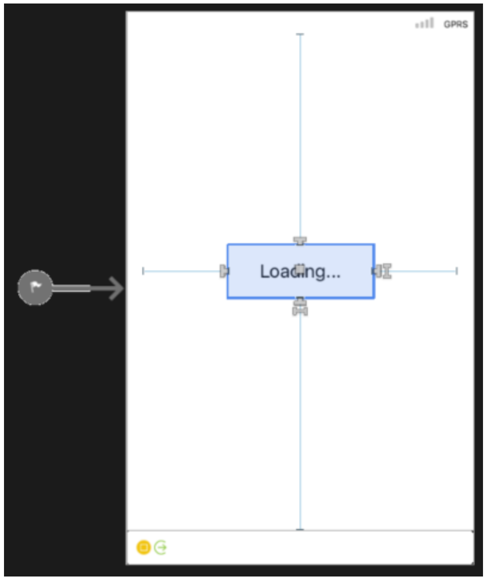
Starting With Xamarin iOS Classic

1. In classic folder delete the previous **Shop.UIClassic.iOS** and add the Xamarin iOS project **Shop.UIClassic.iOS**, using single view app template:

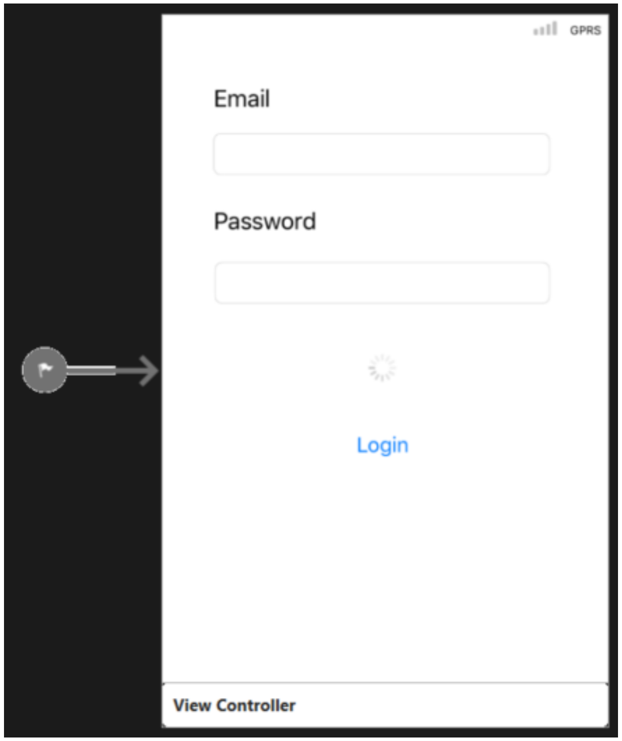
2. Add a reference to **Shop.Common**.

3. Add the NuGets **Newtonsoft.Json** and **Xam.Plugins.Settings**.

4. Modify the **LaunchScreen.storyboard**:



5. Modify the **Main.storyboard**.



6. Modify the **ViewController**:

```
using System;
using UIKit;

public partial class ViewController : UIViewController
{
        public ViewController(IntPtr handle) : base(handle)
        {
        }
```

```
public override void ViewDidLoad()
{
base.ViewDidLoad();
}

public override void DidReceiveMemoryWarning()
{
base.DidReceiveMemoryWarning();
// Release any cached data, images, etc that aren't in use.
}

partial void LoginButton_TouchUpInside(UIButton sender)
{
if (string.IsNullOrEmpty(this.EmailText.Text))
{
var alert = UIAlertController.Create("Error", "You must enter an email.", UIAlertControllerStyle.Alert);
alert.AddAction(UIAlertAction.Create("Accept", UIAlertActionStyle.Default, null));
this.PresentViewController(alert, true, null);
return;
}

if (string.IsNullOrEmpty(this.PasswordText.Text))
{
var alert = UIAlertController.Create("Error", "You must enter a password.", UIAlertControllerStyle.Alert);
alert.AddAction(UIAlertAction.Create("Accept", UIAlertActionStyle.Default, null));
this.PresentViewController(alert, true, null);
return;
}

var ok = UIAlertController.Create("Ok", "Fuck yeah!", UIAlertControllerStyle.Alert);
ok.AddAction(UIAlertAction.Create("Accept", UIAlertActionStyle.Default, null));
```

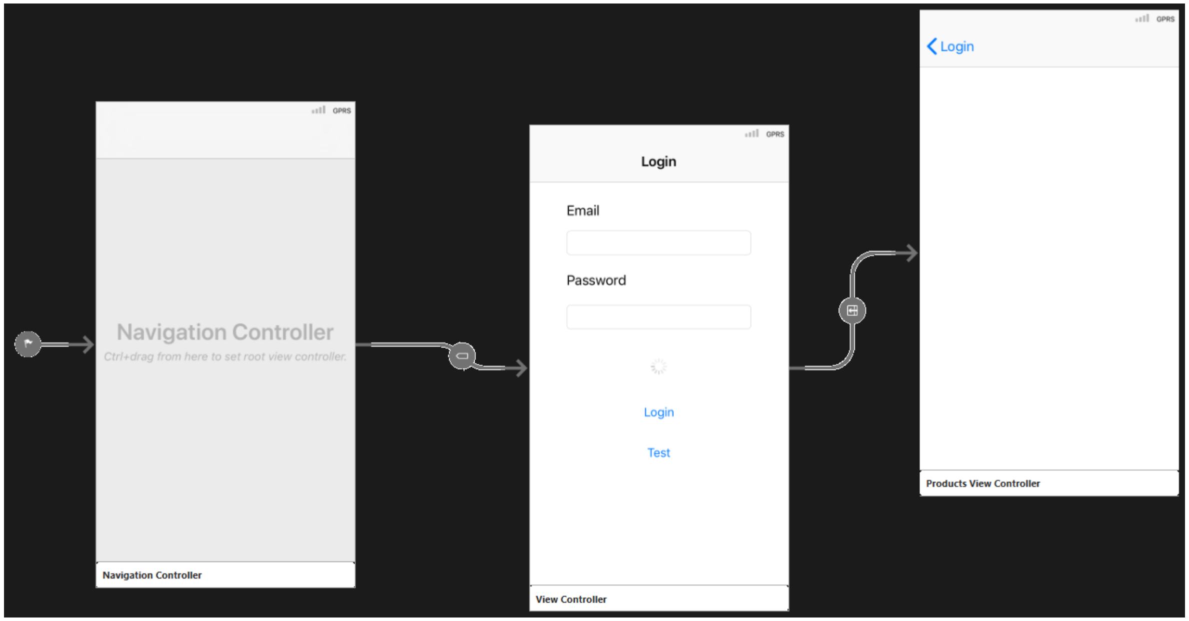
```
    this.PresentViewController(ok, true, null);
    }
}
```

7. Test it.

8. Modify the ViewController:

```
    //var ok = UIAlertController.Create("Ok", "Fuck yeah!", UIAlertControllerStyle.Alert);
    //ok.AddAction(UIAlertAction.Create("Accept", UIAlertActionStyle.Default, null));
    //this.PresentViewController(ok, true, null);

    this.DoLogin();
}

private async void DoLogin()
{
    this.ActivityIndicator.StartAnimating();
    var request = new TokenRequest
    {
    Username = this.EmailText.Text,
    Password = this.PasswordText.Text
    };

    var response = await this.apiService.GetTokenAsync(
    "https://shopzulu.azurewebsites.net",
    "/Account",
    "/CreateToken",
    request);

    if (!response.IsSuccess)
    {
    this.ActivityIndicator.StopAnimating();
    var alert = UIAlertController.Create("Error", "User or password incorrect.", UIAlertControllerStyle.Alert);
    alert.AddAction(UIAlertAction.Create("Accept", UIAlertActionStyle.Default, null));
    this.PresentViewController(alert, true, null);
    return;
    }

    var token = (TokenResponse)response.Result;
    this.ActivityIndicator.StopAnimating();
    var ok = UIAlertController.Create("Ok", "Fuck yeah!", UIAlertControllerStyle.Alert);
    ok.AddAction(UIAlertAction.Create("Accept", UIAlertActionStyle.Default, null));
    this.PresentViewController(ok, true, null);
}
```

9. Modify the **Main.storyboard**:



10. Test the button **Test**.

11. Add the new property **Products** in **Settings**.

12. Delete the **Test** button and modify the **ViewController**:

```
//var ok = UIAlertController.Create("Ok", "Fuck yeah!", UIAlertControllerStyle.Alert);
//ok.AddAction(UIAlertAction.Create("Accept", UIAlertActionStyle.Default, null));
//this.PresentViewController(ok, true, null);

var token = (TokenResponse)response.Result;

var response2 = await this.apiService.GetListAsync<Product>(
        "https://shopzulu.azurewebsites.net",
        "/api",
        "/Products",
        "bearer",
        token.Token);

if (!response2.IsSuccess)
{
        var alert = UIAlertController.Create("Error", response.Message, UIAlertControllerStyle.Alert);
        alert.AddAction(UIAlertAction.Create("Accept", UIAlertActionStyle.Default, null));
        this.PresentViewController(alert, true, null);
        return;
}

var products = (List<Product>)response2.Result;

Settings.UserEmail = this.EmailText.Text;
Settings.Token = JsonConvert.SerializeObject(token);
Settings.Products = JsonConvert.SerializeObject(products);
this.ActivityIndicator.StopAnimating();

var board = UIStoryboard.FromName("Main", null);
var productsViewController = board.InstantiateViewController("ProductsViewController");
productsViewController.Title = "Products";
this.NavigationController.PushViewController(productsViewController, true);
```

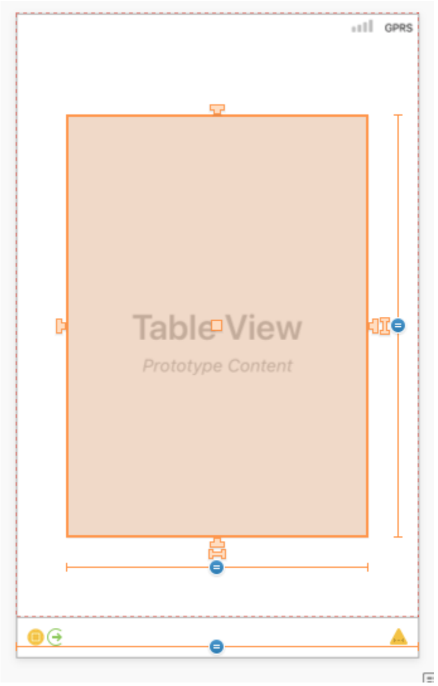
13. Put the property **ProductsViewController** in **Storyboard ID** and test it.

14. Add the folder **DataSources** and inside it, add the class **ProductsDataSource**:

```
using System;
using System.Collections.Generic;
using Common.Models;
using Foundation;
using UIKit;

public class ProductsDataSource : UITableViewSource
{
    private readonly List<Product> products;
    private readonly NSString cellIdentifier = new NSString("ProductCell");

    public ProductsDataSource(List<Product> products)
    {
    this.products = products;
    }

    public override UITableViewCell GetCell(UITableView tableView, NSIndexPath indexPath)
    {
    var cell = tableView.DequeueReusableCell(cellIdentifier) as UITableViewCell;

    if (cell == null)
    {
    cell = new UITableViewCell(UITableViewCellStyle.Default, cellIdentifier);
    }

    var product = products[indexPath.Row];
    cell.TextLabel.Text = product.Name;
    cell.ImageView.Image = UIImage.FromFile(product.ImageFullPath);

    return cell;
    }
```

```
        public override nint RowsInSection(UITableView tableview, nint section)
        {
        return this.products.Count;
        }
}
```

15. Modify the Main Story Board:



16. Modify the **ProductsViewController**:

```
using System;
using System.Collections.Generic;
using Common.Helpers;
using Common.Models;
using DataSources;
using Newtonsoft.Json;
using UIKit;

public partial class ProductsViewController : UIViewController
{
        public ProductsViewController(IntPtr handle) : base(handle)
        {
        }

        public override void ViewDidLoad()
        {
        base.ViewDidLoad();
        var products = JsonConvert.DeserializeObject<List<Product>>(Settings.Products);
        var datasource = new ProductsDataSource(products);
        this.TableView.Source = datasource;
        }
}
```

17. Test what we did until this moment.

18. Add the folder **Cells** and inside it add the class **ProductCell**:

```
using System.Drawing;
using Foundation;
using UIKit;

public class ProductCell : UITableViewCell
{
        private readonly UILabel nameLabel;
        private readonly UILabel priceLabel;
        private readonly UIImageView imageView;

        public ProductCell(NSString cellId) : base(UITableViewCellStyle.Default, cellId)
        {
        this.SelectionStyle = UITableViewCellSelectionStyle.Gray;

        this.imageView = new UIImageView();
        this.nameLabel = new UILabel();
        this.priceLabel = new UILabel()
        {
        TextAlignment = UITextAlignment.Right
        };

        this.ContentView.Add(this.nameLabel);
        this.ContentView.Add(this.priceLabel);
        this.ContentView.Add(this.imageView);
        }

        public void UpdateCell(string caption, string subtitle, UIImage image)
        {
        this.imageView.Image = image;
        this.nameLabel.Text = caption;
        this.priceLabel.Text = subtitle;
        }

        public override void LayoutSubviews()
        {
        base.LayoutSubviews();

        this.imageView.Frame = new RectangleF((float)this.ContentView.Bounds.Width - 63, 5, 33, 33);
```

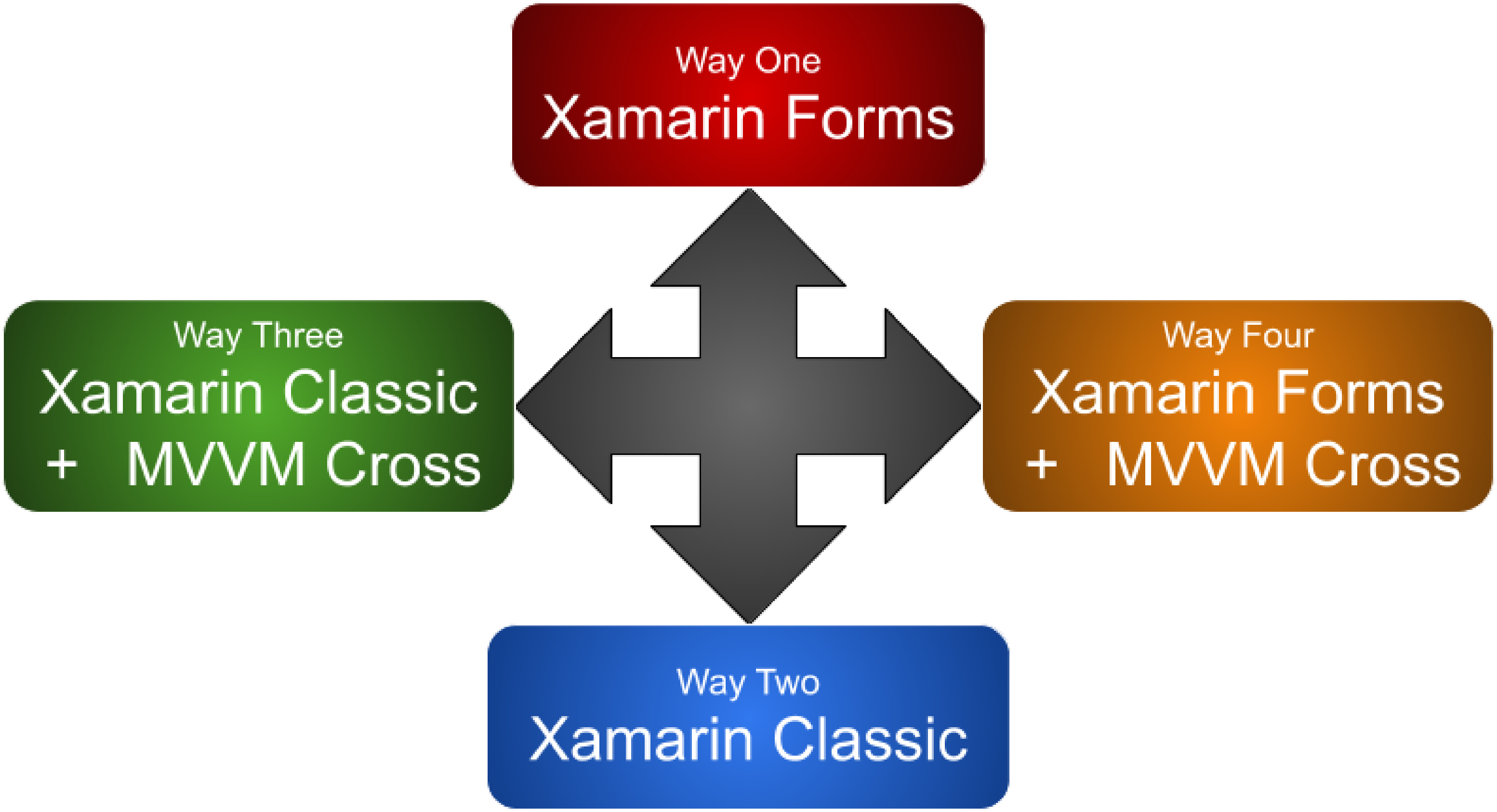
```
        this.nameLabel.Frame = new RectangleF(5, 4, (float)this.ContentView.Bounds.Width - 63, 25);
        this.priceLabel.Frame = new RectangleF(200, 10, 100, 20);
        }
}
```

19. Modify the **GetCell** method in **ProductsDataSource**:

```
public override UITableViewCell GetCell(UITableView tableView, NSIndexPath indexPath)
{
        var cell = tableView.DequeueReusableCell(cellIdentifier) as ProductCell;

        if (cell == null)
        {
        cell = new ProductCell(cellIdentifier);
        }

        var product = products[indexPath.Row];
        cell.UpdateCell(product.Name, $"{product.Price:C2}", UIImage.FromFile(product.ImageUrl));

        return cell;
}
```

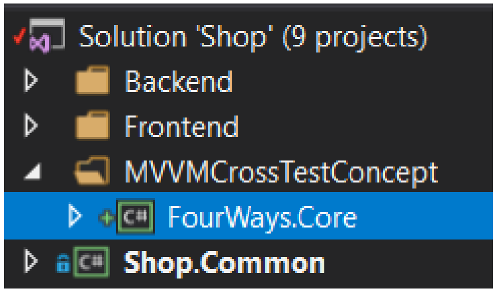
20. Test it.

Starting With MVVM Cross, Test Concept



MVVM Cross Core Project (initial)

1. Add a standard protect call **FourWays.Core**.



2. Delete **Class1**.

3. Add **Services** folder and add **ICalculationService** interface inside it.

```
public interface ICalculationService
{
        decimal TipAmount(decimal subTotal, double generosity);
}
```

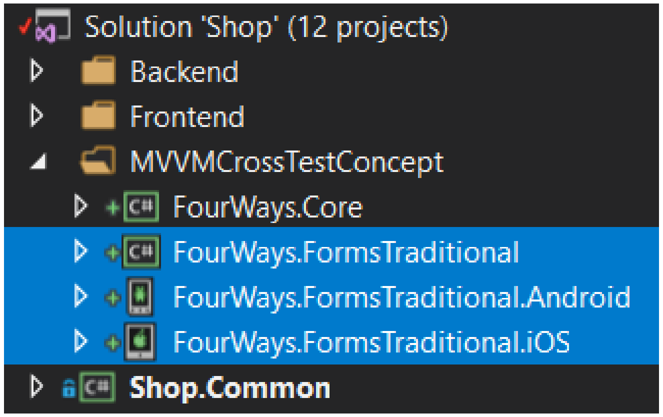
4. Then, add the implementation (**CalculationService**):

```
public class CalculationService : ICalculationService
{
        public decimal TipAmount(decimal subTotal, double generosity)
        {
        return subTotal * (decimal)(generosity / 100);
        }
}
```

5. Congratulations you have ready the initial foundation for the solution.

Forms Traditional Project - Way ONE

1. Add a traditional Xamarin Forms project call: **FourWays.FormsTraditional** and move all the projects to the correct folder:



Solution 'Shop' (12 projects)
Backend
Frontend
MVVMCrossTestConcept
FourWays.Core
FourWays.FormsTraditional
FourWays.FormsTraditional.Android
FourWays.FormsTraditional.iOS
Shop.Common

2. Add the reference to **ThreeWays.Core**:

3. Delete the **MainPage.xaml**.

4. Add the folder **ViewModels** and inside it the class **BaseViewModel**:

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Runtime.CompilerServices;

public class BaseViewModel : INotifyPropertyChanged
{
        public event PropertyChangedEventHandler PropertyChanged;
```

```
    protected void OnPropertyChanged([CallerMemberName] string propertyName = null)
    {
    this.PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }

    protected void SetValue<T>(ref T backingField, T value, [CallerMemberName] string propertyName = null)
    {
    if (EqualityComparer<T>.Default.Equals(backingField, value))
    {
    return;
    }

    backingField = value;
    this.OnPropertyChanged(propertyName);
    }
}
```

5. Now in the same folder, add the **MainViewModel**:

```
using FourWays.Core.Services;

public class MainViewModel : BaseViewModel
{
    private ICalculationService calculationService;
    private decimal amount;
    private double generosity;
    private decimal tip;

    public decimal Amount
    {
    get { return this.amount; }
```

```
set
{
this.SetValue(ref this.amount, value);
this.Recalculate();
}
}

public double Generosity
{
get { return this.generosity; }
set
{
this.SetValue(ref this.generosity, value);
this.Recalculate();
}
}

public decimal Tip
{
get { return this.tip; }
set
{
this.SetValue(ref this.tip, value);
}
}

public MainViewModel()
{
this.calculationService = new CalculationService();
this.Amount = 100;
this.Generosity = 10;
```

```
    }

    private void Recalculate()
    {
    this.Tip = this.calculationService.TipAmount(this.Amount, this.Generosity);
    }
}
```

6. Add the folder **Infrastructure** and inside it the class **InstanceLocator**:

```
using ViewModels;

public class InstanceLocator
{
    public MainViewModel Main { get; set; }

    public InstanceLocator()
    {
    this.Main = new MainViewModel();
    }
}
```

7. Add the folder **Views** and inside it add the **TipPage**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    x:Class="FourWays.FormsTraditional.Views.TipPage"
    BackgroundColor="Black"
    BindingContext="{Binding Main, Source={StaticResource Locator}}"
    Title="Tip Calculator">
```

```
<ContentPage.Content>
<StackLayout
Padding="5">
<Label
        TextColor="White"
        Text="Amount:">
</Label>
<Entry
        Keyboard="Numeric"
        BackgroundColor="White"
        TextColor="Black"
        Text="{Binding Amount, Mode=TwoWay}">
</Entry>
<Label
        TextColor="White"
        Text="Generosity:">
</Label>
<Slider
        Minimum="0"
        Maximum="100"
        Value="{Binding Generosity, Mode=TwoWay}">
</Slider>
<Label
        TextColor="White"
        Text="Tip:">
</Label>
<Label
        TextColor="Yellow"
        FontAttributes="Bold"
        FontSize="Large"
        HorizontalTextAlignment="Center"
```

```
                Text="{Binding Tip, Mode=TwoWay}">
        </Label>
        </StackLayout>
        </ContentPage.Content>
</ContentPage>
```

8. Modify the **App.xaml**:

```
<?xml version="1.0" encoding="utf-8" ?>
<Application
        xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:infra="clr-namespace:FourWays.FormsTraditional.Infrastructure"
        x:Class="FourWays.FormsTraditional.App">
        <Application.Resources>
        <ResourceDictionary>
        <infra:InstanceLocator x:Key="Locator"/>
        </ResourceDictionary>
        </Application.Resources>
</Application>
```

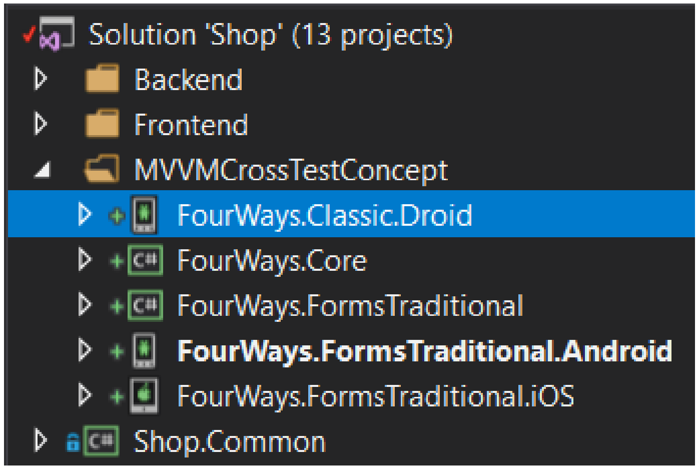
9. Modify the **App.xaml.cs**:

```
using Views;
using Xamarin.Forms;

public partial class App : Application
{
        public App()
        {
        InitializeComponent();
        this.MainPage = new NavigationPage(new TipPage());
        }
}
```

10. Test it on Android and iOS.

Xamarin Android Classic - Way TWO

1. Add a Xamarin Android project call: **FourWays.Classic.Droid**, using blank template.



```
Solution 'Shop' (13 projects)
  Backend
  Frontend
  MVVMCrossTestConcept
    FourWays.Classic.Droid
    FourWays.Core
    FourWays.FormsTraditional
    FourWays.FormsTraditional.Android
    FourWays.FormsTraditional.iOS
  Shop.Common
```

2. Add a reference to **FourWays.Core**.

3. Modify the **activity_main.axml**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
```

```
android:padding="20dp"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:textSize="24dp"
android:text="SubTotal" />
<EditText
android:text="100"
android:id="@+id/amountEditText"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:inputType="number|numberDecimal"
android:textSize="24dp"
android:gravity="right" />
<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginTop="10dp"
android:textSize="24dp"
android:text="Generosity" />
<SeekBar
android:id="@+id/generositySeekBar"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:max="100"
android:min="0"
android:progress="10" />
<View
android:layout_width="match_parent"
```

```
        android:layout_height="1dp"
        android:layout_margin="30dp"
        android:background="@android:color/darker_gray" />
        <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="24dp"
        android:text="Tip to leave" />
        <TextView
        android:id="@+id/tipTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="@android:color/holo_blue_dark"
        android:textSize="24dp"
        android:gravity="center" />
</LinearLayout>
```

4. Modify the **strings.xml**:

```
<resources>
        <string name="app_name">Tip Calc</string>
        <string name="action_settings">Settings</string>
</resources>
```

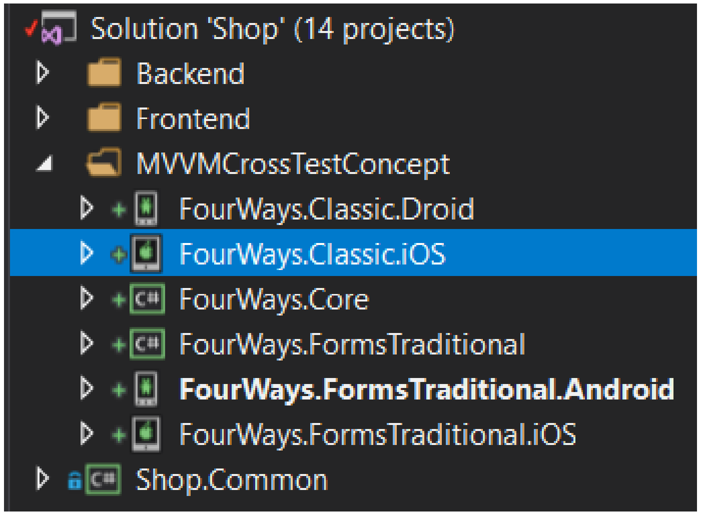
5. Modify the **MainActivity**:

```
using System;
using Android.App;
using Android.OS;
using Android.Support.V7.App;
using Android.Widget;
using Core.Services;

[Activity(
    Label = "@string/app_name",
    Theme = "@style/AppTheme",
    MainLauncher = true)]
public class MainActivity : AppCompatActivity
{
    private EditText amountEditText;
    private SeekBar generositySeekBar;
    private TextView tipTextView;
    private ICalculationService calculationService;

    protected override void OnCreate(Bundle savedInstanceState)
    {
    base.OnCreate(savedInstanceState);

    this.SetContentView(Resource.Layout.activity_main);
    this.calculationService = new CalculationService();
    this.FindViews();
    this.SetupEvents();
    }

    private void SetupEvents()
    {
    this.amountEditText.TextChanged += AmountEditText_TextChanged;
    this.generositySeekBar.ProgressChanged += GenerositySeekBar_ProgressChanged;
    }

    private void GenerositySeekBar_ProgressChanged(object sender, SeekBar.ProgressChangedEventArgs e)
    {
    this.RefreshTip();
    }

    private void AmountEditText_TextChanged(object sender, Android.Text.TextChangedEventArgs e)
    {
    this.RefreshTip();
    }

    private void RefreshTip()
    {
    var amount = Convert.ToDecimal(this.amountEditText.Text);
    var generosity = (double)this.generositySeekBar.Progress;
    this.tipTextView.Text = $"{this.calculationService.TipAmount(amount, generosity):C2}";
    }

    private void FindViews()
    {
    this.amountEditText = this.FindViewById<EditText>(Resource.Id.amountEditText);
    this.generositySeekBar = this.FindViewById<SeekBar>(Resource.Id.generositySeekBar);
    this.tipTextView = this.FindViewById<TextView>(Resource.Id.tipTextView);
    }
}
```

6. Test it.

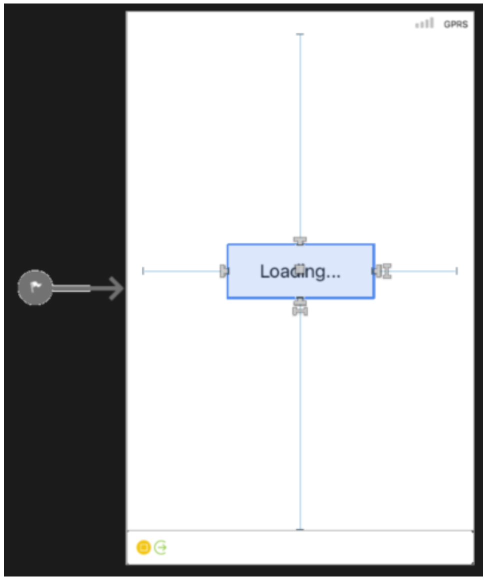
Xamarin iOS Classic - Way TWO

1. Add a Xamarin iOS Project call: **FourWays.Classic.iOS**, using “Single View App” template.

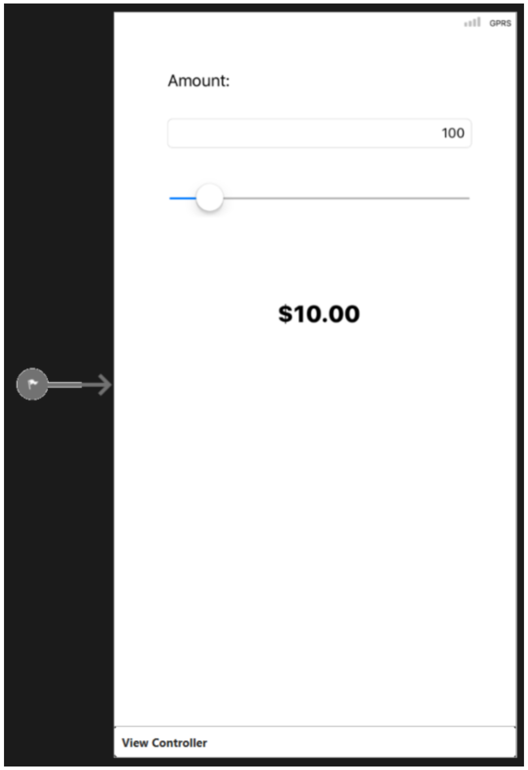

Solution 'Shop' (14 projects)
Backend
Frontend
MVVMCrossTestConcept
FourWays.Classic.Droid
FourWays.Classic.iOS
FourWays.Core
FourWays.FormsTraditional
FourWays.FormsTraditional.Android
FourWays.FormsTraditional.iOS
Shop.Common

2. Add a reference to **FourWays.Core**.

3. Modify the **LaunchScreen.storyboard**:



4. Modify the **Main.storyboard**.



5. Modify the **ViewController**.

```
using System;
using FourWays.Core.Services;
using UIKit;

public partial class ViewController : UIViewController
{
        private readonly ICalculationService calculationService;
```

```
public ViewController(IntPtr handle) : base(handle)
{
this.calculationService = new CalculationService();
}

public override void ViewDidLoad()
{
base.ViewDidLoad();

this.AmountText.EditingChanged += AmountText_EditingChanged;
this.GenerositySlider.ValueChanged += GenerositySlider_ValueChanged;
}

private void GenerositySlider_ValueChanged(object sender, EventArgs e)
{
this.RefreshTip();
}

private void RefreshTip()
{
var amount = Convert.ToDecimal(this.AmountText.Text);
var generosity = (double)this.GenerositySlider.Value;
this.TipLabel.Text = $"{this.calculationService.TipAmount(amount, generosity):C2}";
}

private void AmountText_EditingChanged(object sender, EventArgs e)
{
this.RefreshTip();
}

public override void DidReceiveMemoryWarning()
```

```
        {
        base.DidReceiveMemoryWarning();
        }
}
```

6. Test it.

MVVM Cross Core Project (definitive)

1. Add NuGet **MvvmCross**.

2. Add the folder **ViewModels** and the class **TipViewModel** inside it.

```
using System.Threading.Tasks;
using MvvmCross.ViewModels;
using Services;

public class TipViewModel : MvxViewModel
{
        #region Attributes
        private readonly ICalculationService calculationService;
        private decimal subTotal;
        private int generosity;
        private decimal tip;
        #endregion

        #region Properties
        public decimal SubTotal
        {
        get
```

```
{
return this.subTotal;
}
set
{
this.subTotal = value;
this.RaisePropertyChanged(() => this.SubTotal);
this.Recalculate();
}
}

public decimal Tip
{
get
{
return this.tip;
}
set
{
this.tip = value;
this.RaisePropertyChanged(() => this.Tip);
}
}

public int Generosity
{
get
{
return this.generosity;
}
set
```

```
    {
    this.generosity = value;
    this.RaisePropertyChanged(() => this.Generosity);
    this.Recalculate();
    }
    }
    #endregion

    #region Constructors
    public TipViewModel(ICalculationService calculationService)
    {
    this.calculationService = calculationService;
    }
    #endregion

    #region Methods
    public override async Task Initialize()
    {
    await base.Initialize();

    this.SubTotal = 100;
    this.Generosity = 10;
    this.Recalculate();
    }

    private void Recalculate()
    {
    this.Tip = this.calculationService.TipAmount(this.SubTotal, this.Generosity);
    }
    #endregion
}
```

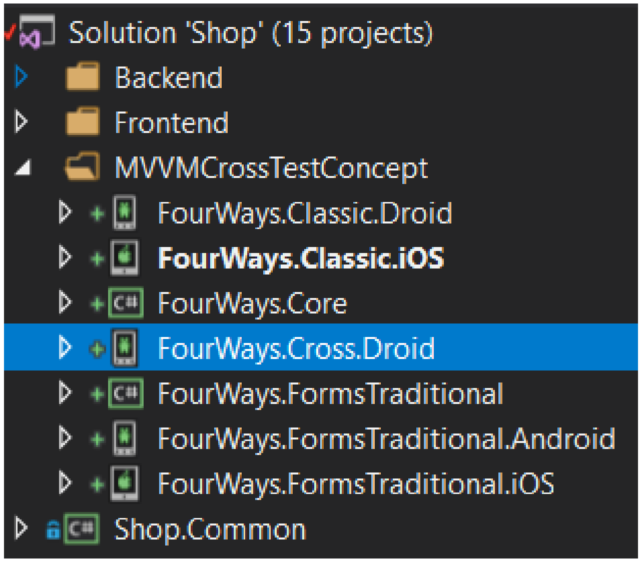
3. In the root project add the **App** class.

```
using MvvmCross.IoC;
using MvvmCross.ViewModels;
using ViewModels;

public class App : MvxApplication
{
        public override void Initialize()
        {
        this.CreatableTypes()
        .EndingWith("Service")
        .AsInterfaces()
        .RegisterAsLazySingleton();

        this.RegisterAppStart<TipViewModel>();
        }
}
```

4. Congratulations you have ready the complete foundation for the solution.

MVVM Cross Android Project - Way THREE

1. Now add the android project and call **FourWays.Cross.Droid**, use blank application template.



Solution 'Shop' (15 projects)
- Backend
- Frontend
- MVVMCrossTestConcept
  - FourWays.Classic.Droid
  - **FourWays.Classic.iOS**
  - FourWays.Core
  - FourWays.Cross.Droid
  - FourWays.FormsTraditional
  - FourWays.FormsTraditional.Android
  - FourWays.FormsTraditional.iOS
- Shop.Common

2. Add the reference to **Core** project and add the NuGet **MvvmCross**.

3. Add a reference to **Mono.Android.Export.dll**.

4. Delete the **MainActivity** activity and the **activity_main** layout.

5. Into **Resources** folder, add the folder **drawable** and inside it add the files **Icon.png** and **splash.png** (you can get it for my repository https://github.com/Zulu55/Shop select a branch different to master).

6. Into **layout** folder add the **SplashPage** layout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Loading...." />
</LinearLayout>
```

7. Into layout folder add the **TipPage** layout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:local="http://schemas.android.com/apk/res-auto"
        android:orientation="vertical"
        android:padding="20dp"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="24dp"
        android:text="SubTotal" />
        <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="number|numberDecimal"
        android:textSize="24dp"
        android:gravity="right"
        local:MvxBind="Text SubTotal" />
        <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
    android:layout_marginTop="10dp"
    android:textSize="24dp"
    android:text="Generosity" />
    <SeekBar
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:max="100"
    local:MvxBind="Progress Generosity" />
    <View
    android:layout_width="match_parent"
    android:layout_height="1dp"
    android:layout_margin="30dp"
    android:background="@android:color/darker_gray" />
    <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="24dp"
    android:text="Tip to leave" />
    <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="@android:color/holo_blue_dark"
    android:textSize="24dp"
    android:gravity="center"
    local:MvxBind="Text Tip" />
</LinearLayout>
```

8. In **strings.xml** modify the application name.

```
<resources>
    <string name="app_name">Tip Calc</string>
```

```
        <string name="action_settings">Settings</string>
</resources>
```

9. In **values** folder add the **SplashStyle.xml** file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="Theme.Splash" parent="android:Theme">
        <item name="android:windowBackground">@drawable/splash</item>
        <item name="android:windowNoTitle">true</item>
  </style>
</resources>
```

10. Add the folder **Views** and inside it add the class **SplashView**.

```
using Android.App;
using Android.Content.PM;
using Core;
using MvvmCross.Platforms.Android.Core;
using MvvmCross.Platforms.Android.Views;

[Activity(
        Label = "@string/app_name",
        MainLauncher = true,
        Icon = "@drawable/icon",
        Theme = "@style/Theme.Splash",
        NoHistory = true,
        ScreenOrientation = ScreenOrientation.Portrait)]
public class SplashView : MvxSplashScreenActivity<MvxAndroidSetup<App>, App>
{
        public SplashView() : base(Resource.Layout.SplashPage)
        {
        }
}
```

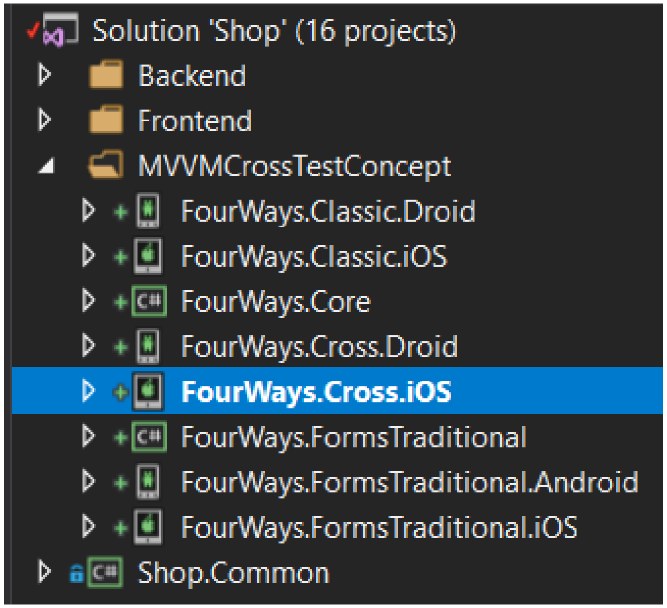
11. In the folder **Views** add **TipView**.

```
using Android.App;
using Android.OS;
using Core.ViewModels;
using MvvmCross.Platforms.Android.Views;

[Activity(Label = "@string/app_name")]
public class TipView : MvxActivity<TipViewModel>
{
        protected override void OnCreate(Bundle savedInstanceState)
        {
        base.OnCreate(savedInstanceState);
        this.SetContentView(Resource.Layout.TipPage);
        }
}
```

12. You’re ready to test the real cross project in android!

MVVM Cross iOS Project - Way THREE

1. Now add the iOS project and call **FourWays.Cross.iOS**, use **Single View App**.



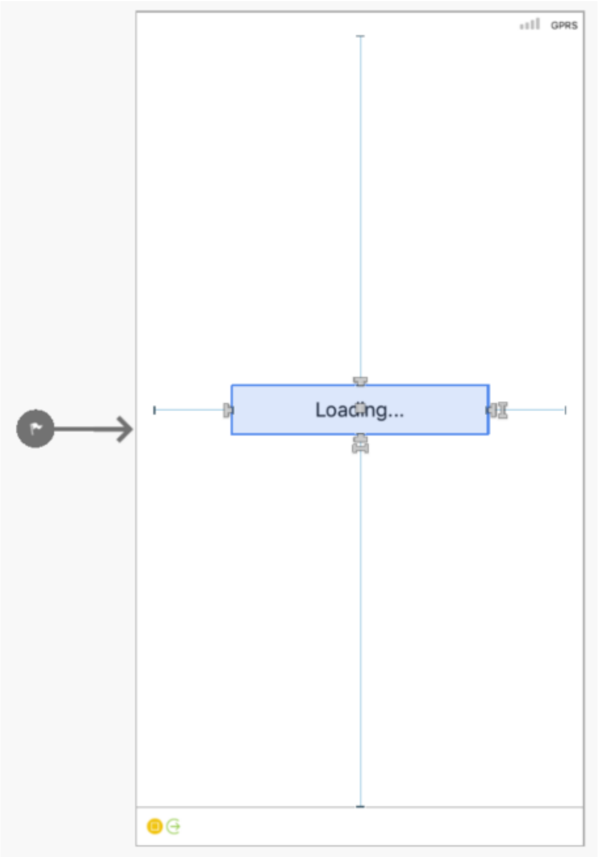
```
Solution 'Shop' (16 projects)
  Backend
  Frontend
  MVVMCrossTestConcept
    FourWays.Classic.Droid
    FourWays.Classic.iOS
    FourWays.Core
    FourWays.Cross.Droid
    FourWays.Cross.iOS
    FourWays.FormsTraditional
    FourWays.FormsTraditional.Android
    FourWays.FormsTraditional.iOS
  Shop.Common
```

2. Add the reference to **Core** project and add the NuGet **MvvmCross**.

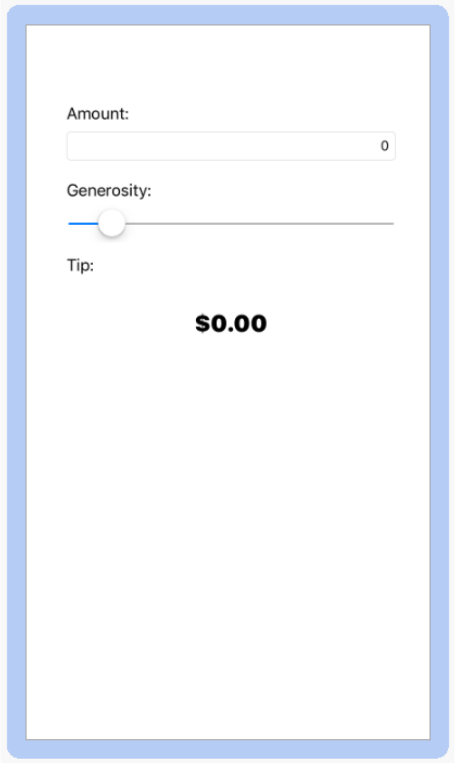
3. Modify the **AppDelegate** by:

```
using Core;
using Foundation;
using MvvmCross.Platforms.Ios.Core;

[Register("AppDelegate")]
public class AppDelegate : MvxApplicationDelegate<MvxIosSetup<App>, App>
{
}
```

4. Modify the **LaunchScreen.storyboard** by:



5. Modify the view **Main.storyboard** similar to this:



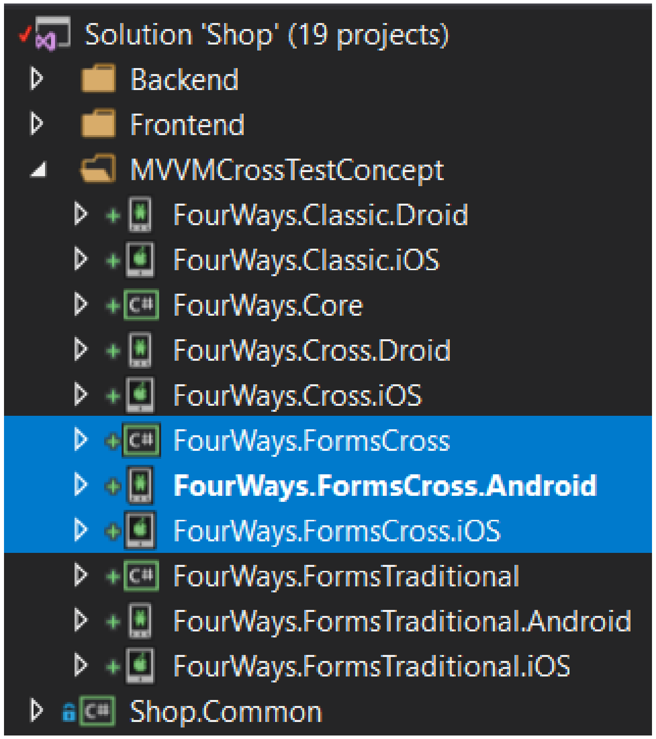
6. Modify the class **ViewController**:

```
using Core.ViewModels;
using MvvmCross.Binding.BindingContext;
using MvvmCross.Platforms.Ios.Presenters.Attributes;
using MvvmCross.Platforms.Ios.Views;

[MvxRootPresentation(WrapInNavigationController = true)]
public partial class ViewController : MvxViewController<TipViewModel>
{
    public override void ViewDidLoad()
    {
        base.ViewDidLoad();

        var set = this.CreateBindingSet<ViewController, TipViewModel>();
        set.Bind(this.AmountText).To(vm => vm.SubTotal);
        set.Bind(this.GenerositySlider).To(vm => vm.Generosity);
        set.Bind(this.TipLabel).To(vm => vm.Tip);
        set.Apply();
    }
}
```

7. You’re ready to test the project on iOS!

Forms Cross Project - Way FOUR

1. Add a traditional Xamarin Forms project call: **FourWays.FormsCross** and move all the projects to the correct folder:



```
Solution 'Shop' (19 projects)
  Backend
  Frontend
  MVVMCrossTestConcept
    FourWays.Classic.Droid
    FourWays.Classic.iOS
    FourWays.Core
    FourWays.Cross.Droid
    FourWays.Cross.iOS
    FourWays.FormsCross
    FourWays.FormsCross.Android
    FourWays.FormsCross.iOS
    FourWays.FormsTraditional
    FourWays.FormsTraditional.Android
    FourWays.FormsTraditional.iOS
  Shop.Common
```

2. Add the reference to **FourWays.Core** in **FourWays.FormCross**, **FourWays.FormCross.Android** and **FourWays.FormCross.iOS**.

3. Add the NuGet **MvvmCross** and **MvvmCross.Forms** to all **FormsCross** projects.

4. Add a reference to **Mono.Android.Export.dll** only to **FormsCross.Android**.

5. Delete the **MainPage.xaml** and **App.xaml**.

6. Add the **FormsApp.xaml**.

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="FourWays.FormsCross.FormsApp">
        <Application.Resources>
        </Application.Resources>
</Application>
```

7. Modify the **FormsApp.xaml.cs**.

```
using Xamarin.Forms;

public partial class FormsApp : Application
{
        public FormsApp()
        {
        InitializeComponent();
        }
}
```

8. Add the folder **Views** and inside it, create the **TipView.xaml**:

```
<?xml version="1.0" encoding="utf-8" ?>
<views:MvxContentPage
        x:TypeArguments="viewModels:TipViewModel"
        xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:views="clr-namespace:MvvmCross.Forms.Views;assembly=MvvmCross.Forms"
        xmlns:mvx="clr-namespace:MvvmCross.Forms.Bindings;assembly=MvvmCross.Forms"
        xmlns:viewModels="clr-namespace:FourWays.Core.ViewModels;assembly=FourWays.Core"
        x:Class="FourWays.FormsCross.Views.TipView"
```

```
    Title="Tip Calc">
    <ContentPage.Content>
    <StackLayout Margin="10">
    <Label Text="Subtotal" />
    <Entry
        x:Name="SubTotalEntry"
        Keyboard="Numeric"
        mvx:Bi.nd="Text SubTotal, Mode=TwoWay">
    </Entry>
    <Label
        Text="Generosity">
    </Label>
    <Slider
        x:Name="GenerositySlider"
        Maximum="100"
        mvx:Bi.nd="Value Generosity, Mode=TwoWay">
    </Slider>
    <Label
        Text="Tip to leave">
    </Label>
    <Label
        x:Name="TipLabel"
        mvx:Bi.nd="Text Tip">
    </Label>
    </StackLayout>
    </ContentPage.Content>
</views:MvxContentPage>
```

9. Modify the **TipView.xaml.cs**:

```
using Core.ViewModels;
using MvvmCross.Forms.Views;

public partial class TipView : MvxContentPage<TipViewModel>
{
        public TipView()
        {
        InitializeComponent();
        }
}
```

10. Modify the **MainActivity**:

```
using Android.App;
using Android.Content.PM;
using Android.OS;
using Core;
using MvvmCross.Forms.Platforms.Android.Core;
using MvvmCross.Forms.Platforms.Android.Views;

[Activity(
        Label = "Tip Calc",
        Icon = "@mipmap/icon",
        Theme = "@style/MainTheme",
        MainLauncher = true,
        ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation,
        LaunchMode = LaunchMode.SingleTask)]
public class MainActivity : MvxFormsAppCompatActivity<MvxFormsAndroidSetup<App, FormsApp>, App, FormsApp>
{
        protected override void OnCreate(Bundle bundle)
        {
        TabLayoutResource = Resource.Layout.Tabbar;
        ToolbarResource = Resource.Layout.Toolbar;
        base.OnCreate(bundle);
        }
}
```

11. Test **FourWays.FormCross.Android**.

12. Now modify the **AppDelegate** in **FourWays.FormsCross.iOS**.

```
using Core;
using Foundation;
using MvvmCross.Forms.Platforms.Ios.Core;
using UIKit;

[Register(nameof(AppDelegate))]
public partial class AppDelegate : MvxFormsApplicationDelegate<MvxFormsIosSetup<App, FormsApp>, App, FormsApp>
{
        public override bool FinishedLaunching(UIApplication uiApplication, NSDictionary launchOptions)
        {
        return base.FinishedLaunching(uiApplication, launchOptions);
        }
}
```

13. Test **FourWays.FormCross.iOS**.

I recommend to watch this video: https://www.youtube.com/watch?v=c8dwpnN3sl8
The official site is: https://www.mvvmcross.com/

MVVM Cross Value Converters

Core Project

1. Add the folder **Converters** and inside it, create the class: **DecimalToStringValueConverter**, it's very important that the class name ends with **ValueConverter**.

```
using MvvmCross.Converters;
using System;
using System.Globalization;

public class DecimalToStringValueConverter : MvxValueConverter<decimal, string>
{
        protected override string Convert(decimal value, Type targetType, object parameter, CultureInfo culture)
        {
        return $"{value:C2}";
        }
}
```

Android Project

1. Change the Page to call the converter:

```
<TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textColor="@android:color/holo_blue_dark"
        android:textSize="24dp"
        android:gravity="center"
        local:MvxBind="Text Tip,Converter=DecimalToString" />
```

2. Test it.

iOS Project

1. Change the controller to call the converter:

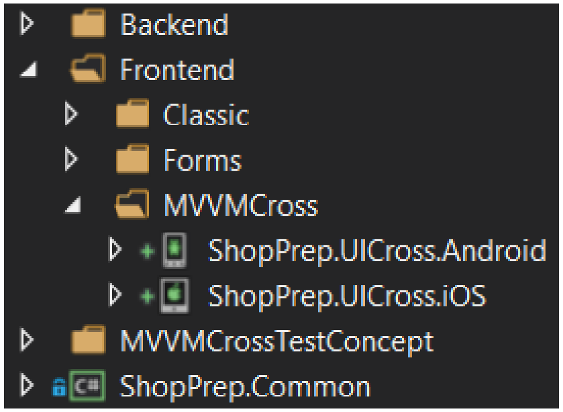
```
set.Bind(this.TipLabel).To(vm => vm.Tip).WithConversion("DecimalToString");
```

2. Test it.

Making the Shop Project With MVVM Cross

Core First Part

1. Create this projects folder structure, add a project Xamarin Android and Xamarin iOS:



```
▷ Backend
◢ Frontend
    ▷ Classic
    ▷ Forms
    ◢ MVVMCross
        ▷ ShopPrep.UICross.Android
        ▷ ShopPrep.UICross.iOS
▷ MVVMCrossTestConcept
▷ ShopPrep.Common
```

2. Add the NuGet **MvvmCross** to **Common** project.

3. Extract the interface to **ApiService**:

```
using System.Threading.Tasks;
using Models;

public interface IApiService
{
        Task<Response> DeleteAsync(
        string urlBase,
        string servicePrefix,
        string controller,
        int id,
        string tokenType,
        string accessToken);

        Task<Response> GetListAsync<T>(
        string urlBase,
        string servicePrefix,
        string controller);

        Task<Response> GetListAsync<T>(
        string urlBase,
        string servicePrefix,
        string controller,
        string tokenType,
        string accessToken);

        Task<Response> GetTokenAsync(
        string urlBase,
        string servicePrefix,
        string controller,
        TokenRequest request);

        Task<Response> PostAsync<T>(
        string urlBase,
        string servicePrefix,
        string controller,
        T model,
        string tokenType,
        string accessToken);

        Task<Response> PutAsync<T>(
        string urlBase,
        string servicePrefix,
        string controller,
        int id,
        T model,
        string tokenType,
        string accessToken);
}
```

4. Add the folder **Interfaces** and inside it add the interface **IDialogService**:

```
public interface IDialogService
{
    void Alert(string message, string title, string okbtnText);
}
```

5. Add the folder **ViewModels** and inside it add the class **LoginViewModel**:

```
using System.Windows.Input;
using Interfaces;
using Models;
using MvvmCross.Commands;
using MvvmCross.ViewModels;
using Services;

public class LoginViewModel : MvxViewModel
{
        private string email;
        private string password;
        private MvxCommand loginCommand;
        private readonly IApiService apiService;
        private readonly IDialogService dialogService;
        private bool isLoading;

        public bool IsLoading
        {
        get => this.isLoading;
        set => this.SetProperty(ref this.isLoading, value);
        }

        public string Email
        {
        get => this.email;
        set => this.SetProperty(ref this.email, value);
        }

        public string Password
        {
        get => this.password;
```

```
set => this.SetProperty(ref this.password, value);
}

public ICommand LoginCommand
{
get
{
this.loginCommand = this.loginCommand ?? new MvxCommand(this.DoLoginCommand);
return this.loginCommand;
}
}

public LoginViewModel(
IApiService apiService,
IDialogService dialogService)
{
this.apiService = apiService;
this.dialogService = dialogService;

this.Email = "jzuluaga55@gmail.com";
this.Password = "123456";
this.IsLoading = false;
}

private async void DoLoginCommand()
{
if (string.IsNullOrEmpty(this.Email))
{
this.dialogService.Alert("Error", "You must enter an email.", "Accept");
return;
}
```

```
if (string.IsNullOrEmpty(this.Email))
{
this.dialogService.Alert("Error", "You must enter a password.", "Accept");
return;
}

this.IsLoading = true;

var request = new TokenRequest
{
Password = this.Password,
Username = this.Email
};

var response = await this.apiService.GetTokenAsync(
"https://shopzulu.azurewebsites.net",
"/Account",
"/CreateToken",
request);

if (!response.IsSuccess)
{
this.IsLoading = false;
this.dialogService.Alert("Error", "User or password incorrect.", "Accept");
return;
}

this.IsLoading = false;
this.dialogService.Alert("Ok", "Fuck yeah!", "Accept");
}
```

```
}
```

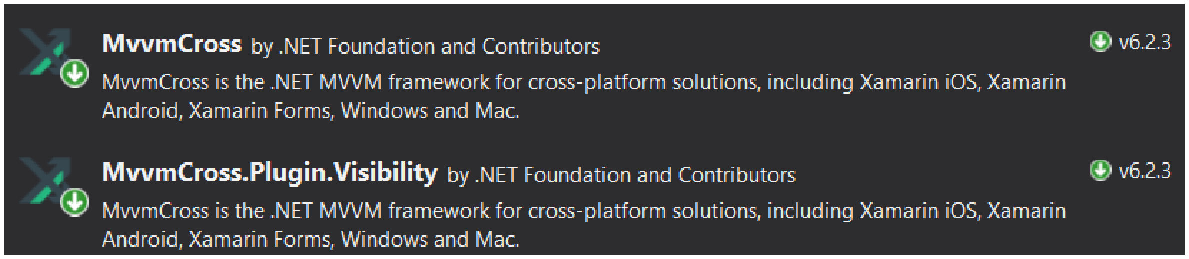
6. Add the **App** class to **Common** project:

```
using MvvmCross.IoC;
using MvvmCross.ViewModels;
using ViewModels;

public class App : MvxApplication
{
        public override void Initialize()
        {
        this.CreatableTypes()
        .EndingWith("Service")
        .AsInterfaces()
        .RegisterAsLazySingleton();

        this.RegisterAppStart<LoginViewModel>();
        }
}
```

7. Ready the first part to **Common** project:

Android First Part

1. Now to the project **UICross.Android** add the NuGets: **MvvmCross** and **MvvmCross.Plugin.Visibility**:



MvvmCross by .NET Foundation and Contributors v6.2.3
MvvmCross is the .NET MVVM framework for cross-platform solutions, including Xamarin iOS, Xamarin Android, Xamarin Forms, Windows and Mac.

MvvmCross.Plugin.Visibility by .NET Foundation and Contributors v6.2.3
MvvmCross is the .NET MVVM framework for cross-platform solutions, including Xamarin iOS, Xamarin Android, Xamarin Forms, Windows and Mac.

2. Add a reference to **Mono.Android.Export.dll**.

3. Add a reference to **Common** project.

4. Delete the **MainActivity** activity and the **activity_main** layout.

5. Into **Resources** folder, add the folder **drawable** and inside it add the files **Icon.png** and **splash.png** (you can get it from the repository: https://github.com/Zulu55/Shop, select the branch Group 3)

6. Into **layout** folder add the **SplashPage** layout.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Loading...." />
</LinearLayout>
```

7. Into **layout** folder add the **LoginPage** layout:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:local="http://schemas.android.com/apk/res-auto"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <LinearLayout
        android:paddingTop="10dp"
        android:paddingLeft="10dp"
        android:paddingRight="10dp"
        android:orientation="vertical"
        android:minWidth="25px"
        android:minHeight="25px"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <TextView
        android:text="Email"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minWidth="25px"
        android:minHeight="25px"/>
        <EditText
        android:inputType="textEmailAddress"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        local:MvxBind="Text Email" />
        <TextView
        android:text="Password"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minWidth="25px"
        android:minHeight="25px"/>
        <EditText
        android:inputType="textPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        local:MvxBind="Text Password" />
        <ProgressBar
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        local:MvxBind="Visibility Visibility(IsLoading)"
        android:indeterminateOnly="true"
        android:keepScreenOn="true"/>
        <Button
        android:text="Login"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        local:MvxBind="Click LoginCommand" />
        </LinearLayout>
</RelativeLayout>
```

8. In **strings.xml** modify the application name.

```
<resources>
        <string name="app_name">Shop</string>
        <string name="action_settings">Settings</string>
</resources>
```

8. In values folder add the **SplashStyle.xml** file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <style name="Theme.Splash" parent="android:Theme">
          <item name="android:windowBackground">@drawable/splash</item>
          <item name="android:windowNoTitle">true</item>
  </style>
</resources>
```

9. Add the folder **Views** and inside it add the class **SplashView**.

```
using global::Android.App;
using global::Android.Content.PM;
using MvvmCross.Platforms.Android.Views;

[Activity(
          Label = "@string/app_name",
          MainLauncher = true,
          Icon = "@drawable/icon",
          Theme = "@style/Theme.Splash",
          NoHistory = true,
          ScreenOrientation = ScreenOrientation.Portrait)]
public class SplashView : MvxSplashScreenActivity
{
          public SplashView() : base(Resource.Layout.SplashPage)
          {
          }
}
```

10. In the folder **Views** add the **LoginView**.

```
using Common.ViewModels;
using global::Android.App;
using global::Android.OS;
using MvvmCross.Platforms.Android.Views;

[Activity(Label = "@string/app_name")]
public class LoginView : MvxActivity<LoginViewModel>
{
        protected override void OnCreate(Bundle savedInstanceState)
        {
        base.OnCreate(savedInstanceState);
        this.SetContentView(Resource.Layout.LoginPage);
        }
}
```

11. Add the folder **Services** and inside it add the class **DialogService**:

```
using Common.Interfaces;
using global::Android.App;
using MvvmCross;
using MvvmCross.Platforms.Android;

public class DialogService : IDialogService
{
        public void Alert(string title, string message, string okbtnText)
        {
        var top = Mvx.Resolve<IMvxAndroidCurrentTopActivity>();
        var act = top.Activity;
        var adb = new AlertDialog.Builder(act);
        adb.SetTitle(title);
        adb.SetMessage(message);
        adb.SetPositiveButton(okbtnText, (sender, args) => { /* some logic */ });
        adb.Create().Show();
        }
}
```

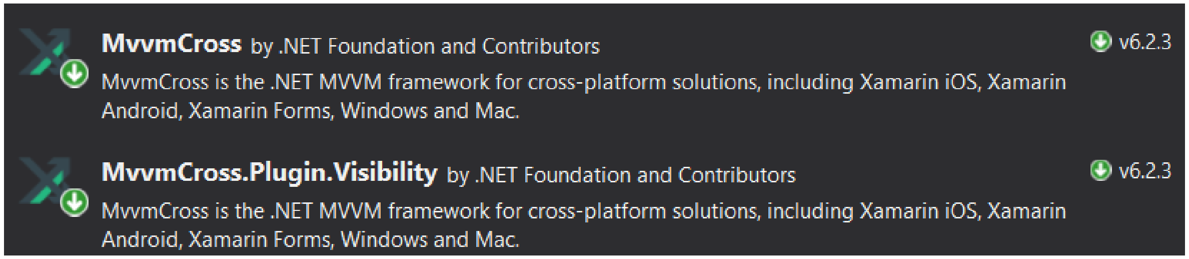
12. To the root **UICross.Android** project, add the class **Setup**:

```
using Common;
using Common.Interfaces;
using MvvmCross;
using MvvmCross.Platforms.Android.Core;
using Services;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;

public class Setup : MvxAndroidSetup<App>
{
        protected override void InitializeFirstChance()
        {
        Mvx.IoCProvider.RegisterType<IDialogService, DialogService>();

        base.InitializeFirstChance();
        }

        public override IEnumerable<Assembly> GetPluginAssemblies()
        {
        var assemblies = base.GetPluginAssemblies().ToList();
        assemblies.Add(typeof(MvvmCross.Plugin.Visibility.Platforms.Android.Plugin).Assembly);
        return assemblies;
        }
}
```

13. Now you can test the first part on Android.

iOS First Part

1. Now to the project **UICross.iOS** add the NuGets: **MvvmCross** and **MvvmCross.Plugin.Visibility**:



MvvmCross by .NET Foundation and Contributors v6.2.3
MvvmCross is the .NET MVVM framework for cross-platform solutions, including Xamarin iOS, Xamarin Android, Xamarin Forms, Windows and Mac.

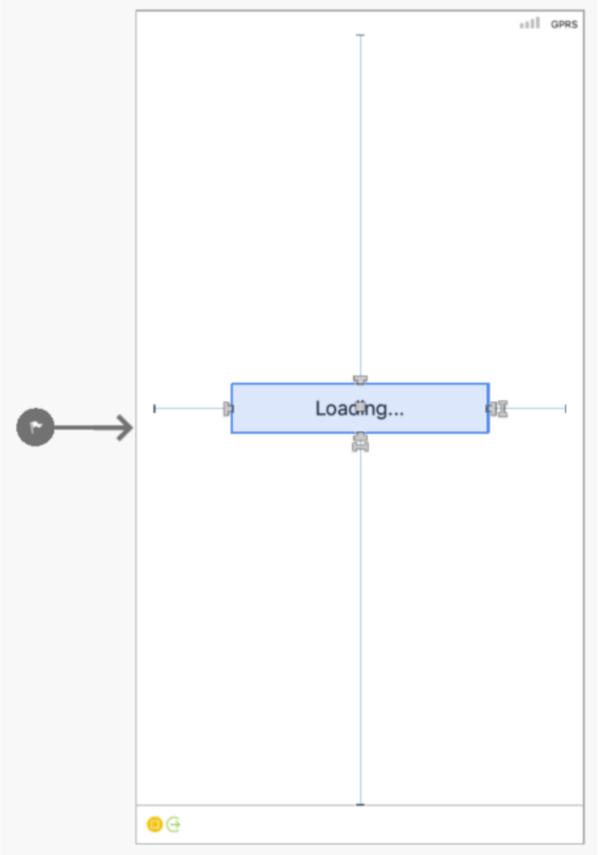
MvvmCross.Plugin.Visibility by .NET Foundation and Contributors v6.2.3
MvvmCross is the .NET MVVM framework for cross-platform solutions, including Xamarin iOS, Xamarin Android, Xamarin Forms, Windows and Mac.

2. Add the reference to **Common** project.

3. Modify the **AppDelegate** by:

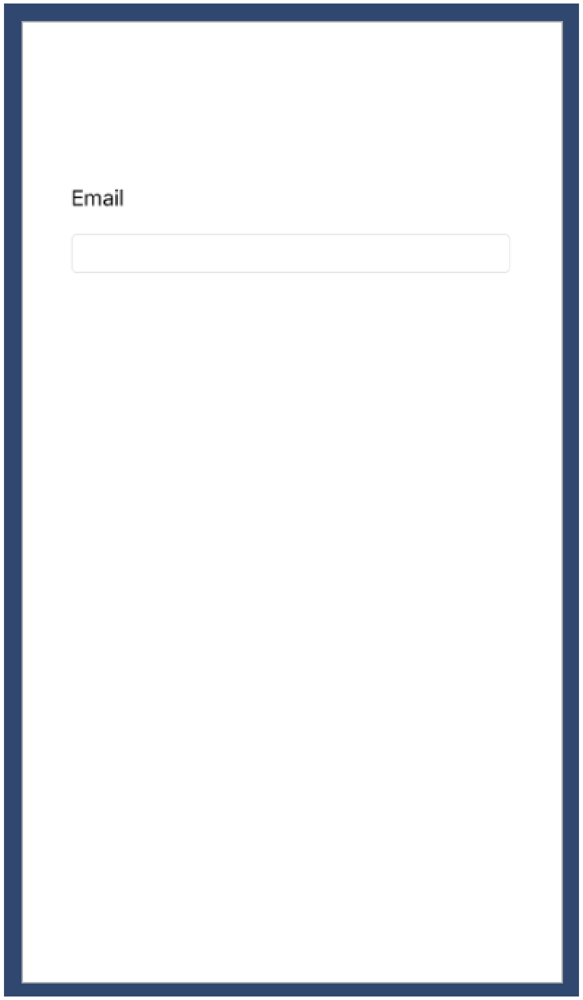
```
using Foundation;
using MvvmCross.Platforms.Ios.Core;

[Register("AppDelegate")]
public class AppDelegate : MvxApplicationDelegate
{
}
```

4. Modify the **LaunchScreen.storyboard** by:



5. Add the folder **Views** and inside it add the view **HomeView.xib, HomeView.cs** and **HomeView.designer.cs**. And fix the namespaces.

6. Modify the view **HomeView.xib** similar to this:



7. Modify the class **HomeView**:

```
using Common.ViewModels;
using MvvmCross.Binding.BindingContext;
using MvvmCross.Platforms.Ios.Presenters.Attributes;
using MvvmCross.Platforms.Ios.Views;

[MvxRootPresentation(WrapInNavigationController = true)]
public partial class HomeView : MvxViewController<LoginViewModel>
{
    public HomeView() : base("HomeView", null)
    {
    }

    public override void ViewDidLoad()
    {
    base.ViewDidLoad();

    var set = this.CreateBindingSet<HomeView, LoginViewModel>();
    set.Bind(this.EmailText).To(vm => vm.Email);
    //set.Bind(Button).To(vm => vm.ResetTextCommand);
    set.Apply();
    }
}
```

8. Add the **Setup** class in root:

```
using MvvmCross.Platforms.Ios.Core;
using MvvmCross.ViewModels;

public class Setup : MvxIosSetup
{
    protected override IMvxApplication CreateApp()
    {
    return new Common.App();
```

```
        }
}
```

9. You’re ready to test the project on iOS!

Core Second Part

1. Add the **ProductsViewModel** in **Common.ViewModels**:

```
using System.Collections.Generic;
using Helpers;
using Interfaces;
using Models;
using MvvmCross.ViewModels;
using Newtonsoft.Json;
using Services;

public class ProductsViewModel : MvxViewModel
{
        private List<Product> products;
        private readonly IApiService apiService;
        private readonly IDialogService dialogService;

        public List<Product> Products
        {
        get => this.products;
        set => this.SetProperty(ref this.products, value);
        }

        public ProductsViewModel(
        IApiService apiService,
        IDialogService dialogService)
        {
        this.apiService = apiService;
        this.dialogService = dialogService;
        this.LoadProducts();
        }

        private async void LoadProducts()
        {
        var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);
        var response = await this.apiService.GetListAsync<Product>(
        "https://shopzulu.azurewebsites.net",
        "/api",
        "/Products",
        "bearer",
        token.Token);

        if (!response.IsSuccess)
        {
        this.dialogService.Alert("Error", response.Message, "Accept");
        return;
        }

        this.Products = (List<Product>)response.Result;
        }
}
```

2. Add the folder **Converters** in inside it add the class **DecimalToStringValueConverter**:

```
using System;
using System.Globalization;
using MvvmCross.Converters;

public class DecimalToStringValueConverter : MvxValueConverter<decimal, string>
{
        protected override string Convert(decimal value, Type targetType, object parameter, CultureInfo culture)
        {
        return $"{value:C2}";
        }
}
```

3. Modify the **LoginViewModel**.

```
using System.Windows.Input;
using Interfaces;
using Models;
using MvvmCross.Commands;
using MvvmCross.Navigation;
using MvvmCross.ViewModels;
using Newtonsoft.Json;
using Services;
using Shop.Common.Helpers;

public class LoginViewModel : MvxViewModel
{
        private string email;
        private string password;
        private MvxCommand loginCommand;
        private readonly IApiService apiService;
        private readonly IDialogService dialogService;
        private readonly IMvxNavigationService navigationService;
        private bool isLoading;
```

```
public bool IsLoading
{
get => this.isLoading;
set => this.SetProperty(ref this.isLoading, value);
}

public string Email
{
get => this.email;
set => this.SetProperty(ref this.email, value);
}

public string Password
{
get => this.password;
set => this.SetProperty(ref this.password, value);
}

public ICommand LoginCommand
{
get
{
this.loginCommand = this.loginCommand ?? new MvxCommand(this.DoLoginCommand);
return this.loginCommand;
}
}

public LoginViewModel(
IApiService apiService,
IDialogService dialogService,
IMvxNavigationService navigationService)
{
this.apiService = apiService;
this.dialogService = dialogService;
this.navigationService = navigationService;

this.Email = "jzuluaga55@gmail.com";
this.Password = "123456";
this.IsLoading = false;
}

private async void DoLoginCommand()
{
if (string.IsNullOrEmpty(this.Email))
{
this.dialogService.Alert("Error", "You must enter an email.", "Accept");
return;
}

if (string.IsNullOrEmpty(this.Email))
{
this.dialogService.Alert("Error", "You must enter a password.", "Accept");
return;
}

this.IsLoading = true;

var request = new TokenRequest
{
Password = this.Password,
Username = this.Email
```

```
        };

        var response = await this.apiService.GetTokenAsync(
        "https://shopzulu.azurewebsites.net",
        "/Account",
        "/CreateToken",
        request);

        if (!response.IsSuccess)
        {
        this.IsLoading = false;
        this.dialogService.Alert("Error", "User or password incorrect.", "Accept");
        return;
        }

        var token = (TokenResponse)response.Result;
        Settings.UserEmail = this.Email;
        Settings.Token = JsonConvert.SerializeObject(token);
        this.IsLoading = false;

        //this.dialogService.Alert("Ok", "Fuck yeah!", "Accept");
        await this.navigationService.Navigate<ProductsViewModel>();
        }
}
```

4. We’ve finished the second part on core.

Android Second Part

1. Add the NuGet **Xamarin.FFImageLoading**.

2. In **layout** add the **ProductRow** layout:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:local="http://schemas.android.com/apk/res-auto"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ffimageloading.cross.MvxCachedImageView
    android:layout_width="75dp"
    android:layout_height="75dp"
    android:layout_margin="10dp"
    local:MvxBind="ImagePath ImageFullPath" />
    <LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="30dp"
    local:MvxBind="Text Name" />
    <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="20dp"
    local:MvxBind="Text Price,Converter=DecimalToString" />
    </LinearLayout>
</LinearLayout>
```

3. In **layout** add the **ProductsPage** layout:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:local="http://schemas.android.com/apk/res-auto"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <MvxListView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        local:MvxBind="ItemsSource Products"
        local:MvxItemTemplate="@layout/productrow"/>
</LinearLayout>
```

4. In **Views**, add the **ProductView** class:

```
using Common.ViewModels;
using global::Android.App;
using global::Android.OS;
using MvvmCross.Platforms.Android.Views;

[Activity(Label = "@string/app_name")]
public class KittensView : MvxActivity<ProductsViewModel>
{
        protected override void OnCreate(Bundle bundle)
        {
        base.OnCreate(bundle);
        this.SetContentView(Resource.Layout.ProductsPage);
        }
}
```

5. Test it in Android.

iOS Second Part

1. Now to the project **UICross.iOS** add the NuGets: **MvvmCross** and **MvvmCross.Plugin.Visibility**:

Core Third Part

1. Add the **AppProductViewModel** in **Common.ViewModels**:

```
using System.Windows.Input;
using Helpers;
using Interfaces;
using Models;
using MvvmCross.Commands;
using MvvmCross.Navigation;
using MvvmCross.ViewModels;
using Newtonsoft.Json;
using Services;

public class AppProductViewModel : MvxViewModel
{
	private string name;
	private string price;
	private MvxCommand addProductCommand;
	private readonly IApiService apiService;
	private readonly IDialogService dialogService;
	private readonly IMvxNavigationService navigationService;
	private bool isLoading;
```

```
public bool IsLoading
{
get => this.isLoading;
set => this.SetProperty(ref this.isLoading, value);
}

public string Name
{
get => this.name;
set => this.SetProperty(ref this.name, value);
}

public string Price
{
get => this.price;
set => this.SetProperty(ref this.price, value);
}

public ICommand AddProductCommand
{
get
{
this.addProductCommand = this.addProductCommand ?? new MvxCommand(this.AddProduct);
return this.addProductCommand;
}
}

public AppProductViewModel(
IApiService apiService,
IDialogService dialogService,
IMvxNavigationService navigationService)
{
this.apiService = apiService;
this.dialogService = dialogService;
this.navigationService = navigationService;
}

private async void AddProduct()
{
if (string.IsNullOrEmpty(this.Name))
{
this.dialogService.Alert("Error", "You must enter a product name.", "Accept");
return;
}

if (string.IsNullOrEmpty(this.Price))
{
this.dialogService.Alert("Error", "You must enter a product price.", "Accept");
return;
}

var price = decimal.Parse(this.Price);
if (price <= 0)
{
this.dialogService.Alert("Error", "The price must be a number greather than zero.", "Accept");
return;
}

this.IsLoading = true;

//TODO: Image pending
```

```
    var product = new Product
    {
    IsAvailabe = true,
    Name = this.Name,
    Price = price,
    User = new User { UserName = Settings.UserEmail },
    };

    var token = JsonConvert.DeserializeObject<TokenResponse>(Settings.Token);

    var response = await this.apiService.PostAsync(
    "https://shopzulu.azurewebsites.net",
    "/api",
    "/Products",
    product,
    "bearer",
    token.Token);

    this.IsLoading = false;

    if (!response.IsSuccess)
    {
    this.dialogService.Alert("Error", response.Message, "Accept");
    return;
    }

    var newProduct = (Product)response.Result;
    await this.navigationService.Navigate<ProductsViewModel>();
    }
}
```

2. Modify the **ProductsViewModel** in **Common.ViewModels**:

```
private MvxCommand addProductCommand;
…
public ICommand AddProductCommand
{
        get
        {
        this.addProductCommand = this.addProductCommand ?? new MvxCommand(this.AddProduct);
        return this.addProductCommand;
        }
}
…
private async void AddProduct()
{
        await this.navigationService.Navigate<AppProductViewModel>();
}
```

Android Third Part

1. Add the **noImage.png** in folder **drawable** on **UICross.Android**.

2. Add the image **shop.png**:

3. Modify the **LoginPage.axml**:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:local="http://schemas.android.com/apk/res-auto"
```

```
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent">
<LinearLayout
android:paddingTop="10dp"
android:paddingLeft="10dp"
android:paddingRight="10dp"
android:orientation="vertical"
android:minWidth="25px"
android:minHeight="25px"
android:layout_width="match_parent"
android:layout_height="match_parent">
<ImageView
android:layout_gravity="center"
android:src="@drawable/shop"
android:layout_width="300dp"
android:layout_height="200dp" />
<TextView
android:text="Email"
android:textAppearance="?android:attr/textAppearanceLarge"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:minWidth="25px"
android:minHeight="25px"/>
<EditText
android:inputType="textEmailAddress"
android:layout_width="match_parent"
android:layout_height="wrap_content"
local:MvxBind="Text Email" />
<TextView
android:text="Password"
```

```
          android:textAppearance="?android:attr/textAppearanceLarge"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:minWidth="25px"
          android:minHeight="25px"/>
          <EditText
          android:inputType="textPassword"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          local:MvxBind="Text Password" />
          <ProgressBar
          android:layout_height="wrap_content"
          android:layout_width="match_parent"
          android:layout_gravity="center_vertical"
          local:MvxBind="Visibility Visibility(IsLoading)"
          android:indeterminateOnly="true"
          android:keepScreenOn="true"/>
          </LinearLayout>
          <Button
          android:text="Login"
          android:layout_alignParentBottom="true"
          android:layout_width="match_parent"
          android:layout_height="wrap_content"
          local:MvxBind="Click LoginCommand" />
</RelativeLayout>
```

4. Modify the **ProductsPage.axml**:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
          xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:local="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <android.support.design.widget.FloatingActionButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="end|bottom"
    android:src="@drawable/ic_add"
    android:layout_margin="16dp"
    local:layout_anchorGravity="bottom|right|end"
    local:MvxBind="Click AddProductCommand" />
    <MvxListView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    local:MvxBind="ItemsSource Products"
    local:MvxItemTemplate="@layout/productrow"/>
</android.support.design.widget.CoordinatorLayout>
```

3. Add the **AddProductPage.axml** in **UICross.Android.Resources.layout**:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:local="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
    android:paddingTop="10dp"
    android:paddingLeft="10dp"
```

```
android:paddingRight="10dp"
android:orientation="vertical"
android:minWidth="25px"
android:minHeight="25px"
android:layout_width="match_parent"
android:layout_height="wrap_content">
<TextView
android:text="Name"
android:textAppearance="?android:attr/textAppearanceLarge"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:minWidth="25px"
android:minHeight="25px"/>
<EditText
android:inputType="textEmailAddress"
android:layout_width="match_parent"
android:layout_height="wrap_content"
local:MvxBind="Text Name" />
<TextView
android:text="Price"
android:textAppearance="?android:attr/textAppearanceLarge"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:minWidth="25px"
android:minHeight="25px"/>
<EditText
android:inputType="numberDecimal"
android:layout_width="match_parent"
android:layout_height="wrap_content"
local:MvxBind="Text Price" />
<ProgressBar
```

```
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        local:MvxBind="Visibility Visibility(IsLoading)"
        android:indeterminateOnly="true"
        android:keepScreenOn="true"/>
        <Button
        android:text="Add Product"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        local:MvxBind="Click AddProductCommand" />
        </LinearLayout>
</RelativeLayout>
```

4. Add the **AddProductView** in **UICross.Android.Views**:

```
using global::Android.App;
using global::Android.OS;
using MvvmCross.Platforms.Android.Views;
using Shop.Common.ViewModels;

[Activity(Label = "@string/add_product")]
public class AddProductView : MvxActivity<AppProductViewModel>
{
        protected override void OnCreate(Bundle savedInstanceState)
        {
        base.OnCreate(savedInstanceState);
        this.SetContentView(Resource.Layout.AddProductPage);
        }
}
```

5. Test it.

iOS Third Part

1. Add the **AppProductViewModel** in **Common.ViewModels**: