On Sale - WEB Parte I

Juan Carlos Zuluaga Cardona
Medellín
2020

Tabla de contenido

¿Que vamos hacer?

En tiempos de pandemia vamos hacer una solución donde un comerciante pueda matricular sus productos y luego venderlos mediante una aplicación móvil.

Matriz de funcionalidad

Funcionalidad	**Web**		**App**
	Admin	**User**	**User**
Login	X	X	X
Registrarse como usuario		X	X
Modificar el perfil	X	X	X
Recordar contraseña	X	X	X
Administrar administradores	X		
Administrar países, departamentos, ciudades	X		
Administrar productos	X		
Ver y buscar productos		X	X
Agregar productos al carrito de compras		X	X
Confirmar orden		X	X
Administrar los pedidos	X		
Ver estado de mis pedidos		X	X

Arquitectura

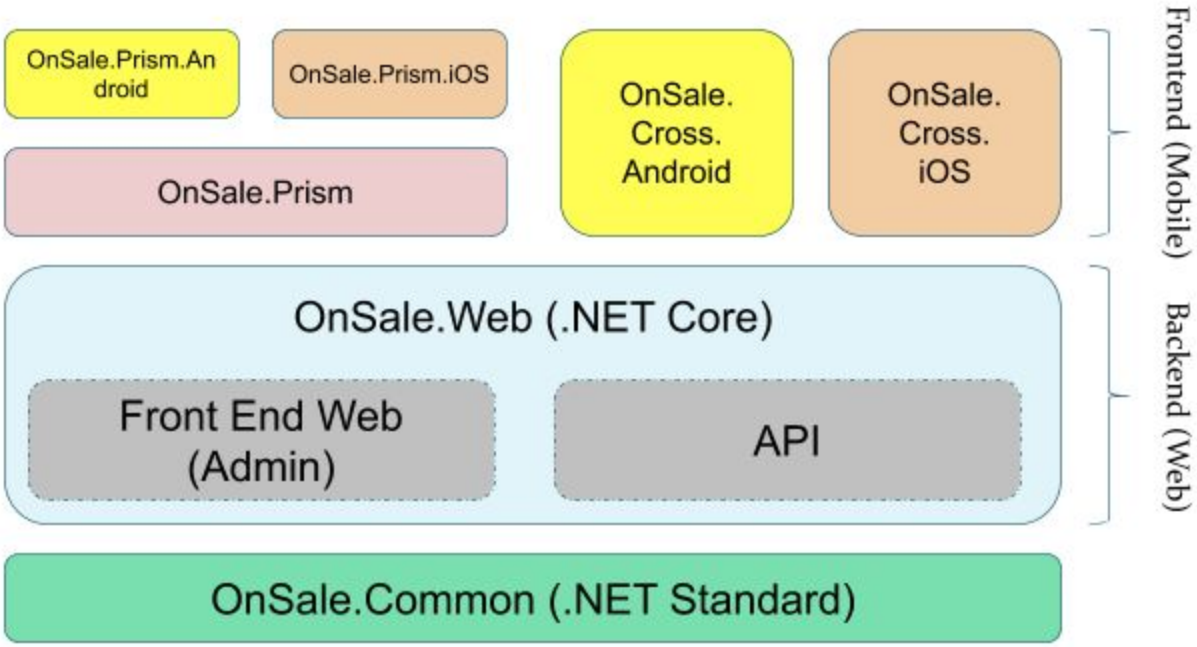
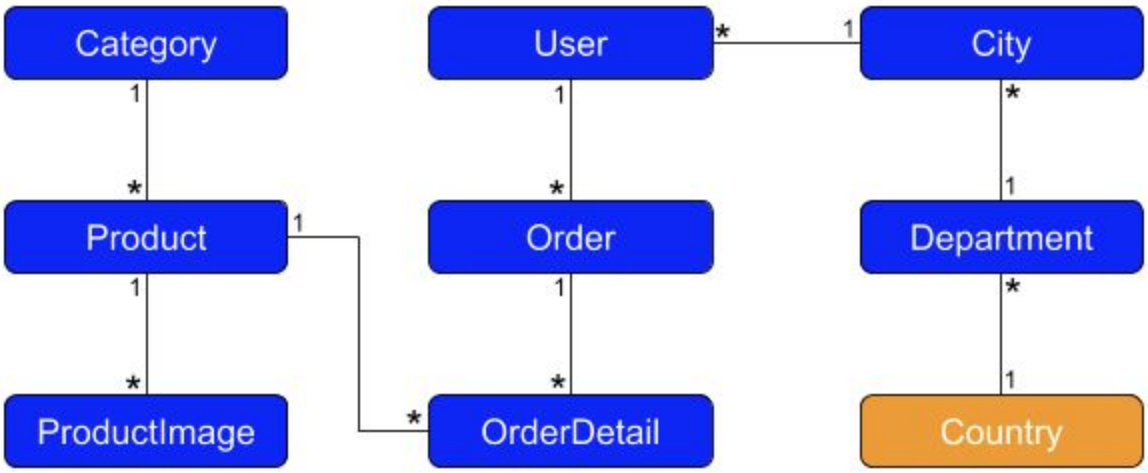
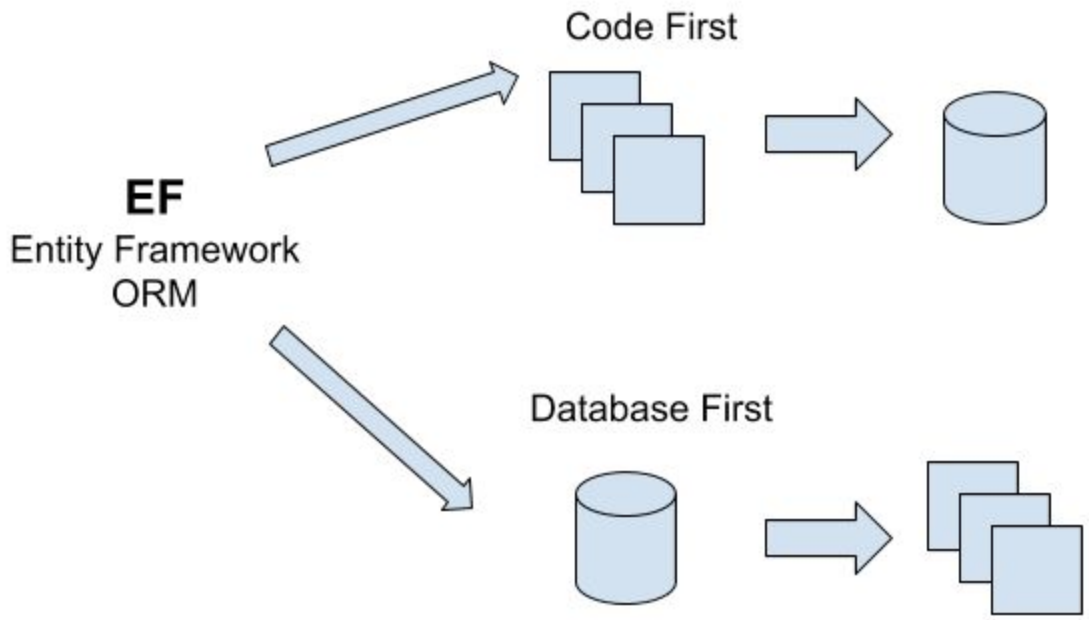


Diagrama Entidad Relación (Candidato)



Creación de la Base de Datos con Entity Framework



Recomiendo buscar y leer documentación sobre Code First y Database First. En este curso trabajaremos con EF Code First, si están interesados en conocer más sobre EF Database First acá les dejo un enlace:
https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/existing-db

1. Cree los proyectos **Common** y **Web**.

2. En el proyecto **Common** cree la carpeta **Entities** y dentro de esta la clase **Country**:

```
public class Country
{
    public int Id { get; set; }

    [MaxLength(50)]
    [Required]
    public string Name { get; set; }
}
```

3. En el proyecto **Web** cree la carpeta **Data** y dentro de esta la clase **DataContext**:

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

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

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

        modelBuilder.Entity<Country>()
            .HasIndex(t => t.Name)
            .IsUnique();
    }
}
```

4. Agregue una cadena de conexión al archivo **appsettings.json**:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection":
"Server=(localdb)\\MSSQLLocalDB;Database=OnSale;Trusted_Connection=True;MultipleActive
ResultSets=true"
  }
}
```

5. Inyectamos la conexión a la base de datos en el archivo **Startup** en el método **ConfigureServices**:

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

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

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

6. Guardamos los cambios y corremos los comandos para crear la base de datos de forma local:

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

7. Creamos un controlador con el asistente para países y modificamos el menú para poder probar lo que llevamos.

```
<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="Countries" asp-action="Index">Countries</a></li>
</ul>
```

8. Mejoremos el CRUD colocando un aspecto más profesional. Para esto adicionamos la vista parcial **_DeleteDialog**:

```
<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 Record</h5>
         </div>
         <div class="modal-body">
            <p>Are you sure to wanto to delete the record?</p>
         </div>
         <div class="modal-footer">
            <button type="button" class="btn btn-primary" data-dismiss="modal">No</button>
            <button type="button" class="btn btn-danger" id="btnYesDelete">Yes</button>
         </div>
      </div>
   </div>
</div>
```

9. Adicional a esto, agregamos el archivo **deleteDialog.js** en **wwwroot/js**:

```
(function (soccerDeleteDialog) {

   var methods = {
      "openModal": openModal,
      "deleteItem": deleteItem
   };

   var item_to_delete;
```

```
    /**
     * Open a modal by class name or Id.
     *
     * @return string id item.
     */
    function openModal(modalName, classOrId, sourceEvent, deletePath, eventClassOrId) {
        var textEvent;
        if (classOrId) {
            textEvent = "." + modalName;
        } else {
            textEvent = "#" + modalName;
        }
        $(textEvent).click((e) => {
            item_to_delete = e.currentTarget.dataset.id;
            deleteItem(sourceEvent, deletePath, eventClassOrId);
        });
    }

    /**
     * Path to delete an item.
     *
     * @return void.
     */
    function deleteItem(sourceEvent, deletePath, eventClassOrId) {
        var textEvent;
        if (eventClassOrId) {
            textEvent = "." + sourceEvent;
        } else {
            textEvent = "#" + sourceEvent;
        }
        $(textEvent).click(function () {
            window.location.href = deletePath + item_to_delete;
        });
    }

    soccerDeleteDialog.sc_deleteDialog = methods;

})(window);
```

10. Modificamos la acción **Delete**:

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

```
    Country country = await _context.Countries
        .FirstOrDefaultAsync(m => m.Id == id);
    if (country == null)
    {
        return NotFound();
    }

    _context.Countries.Remove(country);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}
```

11. Modificamos el resto de vistas colocando los botones con estilos.

12. Modificar la vista **Index**:

```
@model IEnumerable<OnSale.Common.Entities.Country>

@{
    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> Add
New</a>
</p>

<div class="row">
    <div class="col-md-12">
        <div class="panel panel-default">
            <div class="panel-heading">
                <h3 class="panel-title">Countries</h3>
            </div>
            <div class="panel-body">
                <table class="table table-hover table-responsive table-striped" id="MyTable">
                    <thead>
                        <tr>
                            <th>
                                @Html.DisplayNameFor(model => model.Name)
                            </th>
                            <th></th>
                        </tr>
                    </thead>
                    <tbody>
                        @foreach (var item in Model)
```

```
                    {
                        <tr>
                            <td>
                                @Html.DisplayFor(modelItem => item.Name)
                            </td>
                            <td>
                                <a asp-action="Edit" asp-route-id="@item.Id" class="btn btn-warning"><i
class="glyphicon glyphicon-pencil"></i></a>
                                <a asp-action="Details" asp-route-id="@item.Id" class="btn btn-info"><i
class="glyphicon glyphicon-align-justify"></i></a>
                                <button data-id="@item.Id" class="btn btn-danger deleteItem"
data-toggle="modal" data-target="#deleteDialog"><i class="glyphicon
glyphicon-trash"></i></button>
                            </td>
                        </tr>
                    }
                </tbody>
            </table>
        </div>
    </div>
  </div>
</div>

<partial name="_DeleteDialog" />

@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
  <script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>
  <script src="/js/deleteDialog.js"></script>

  <script type="text/javascript">
    $(document).ready(function () {
        $('#MyTable').DataTable();

        // Delete item
        sc_deleteDialog.openModal('deleteItem', true, 'btnYesDelete', '/Countries/Delete/', false);
    });
  </script>
}
```

13. Adicionamos una validación al controlador para evitar errores de duplicados en el **Create**:

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(Country country)
{
  if (ModelState.IsValid)
```

```
    {
        try
        {
            _context.Add(country);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        catch (DbUpdateException dbUpdateException)
        {
            if (dbUpdateException.InnerException.Message.Contains("duplicate"))
            {
                ModelState.AddModelError(string.Empty, "There are a record with the same name.");
            }
            else
            {
                ModelState.AddModelError(string.Empty,
dbUpdateException.InnerException.Message);
            }
        }
        catch (Exception exception)
        {
            ModelState.AddModelError(string.Empty, exception.Message);
        }
    }

    return View(country);
}
```

14. Adicionamos una validación al controlador para evitar errores de duplicados en el **Edit**:

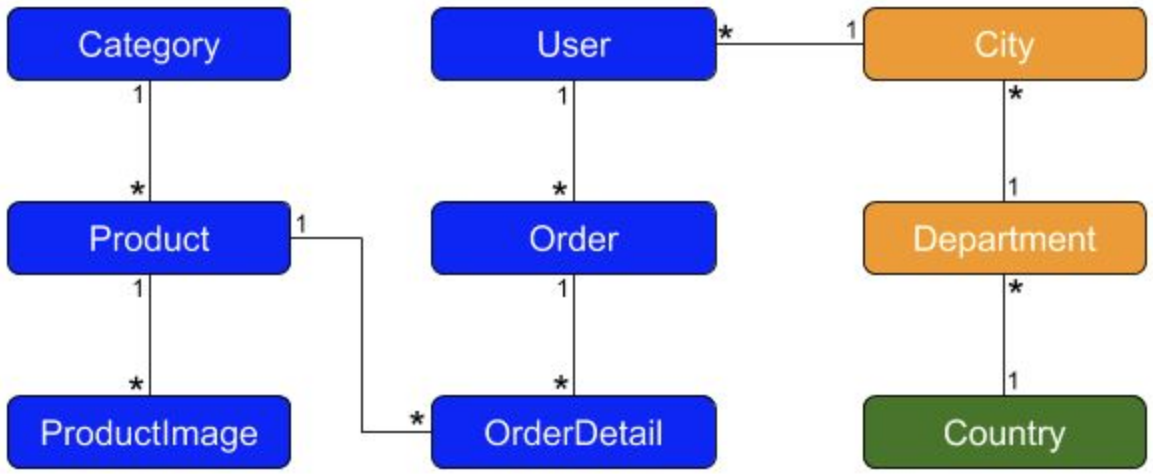
```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, Country country)
{
    if (id != country.Id)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(country);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        catch (DbUpdateException dbUpdateException)
```

```
        {
            if (dbUpdateException.InnerException.Message.Contains("duplicate"))
            {
                ModelState.AddModelError(string.Empty, "There are a record with the same name.");
            }
            else
            {
                ModelState.AddModelError(string.Empty,
dbUpdateException.InnerException.Message);
            }
        }
        catch (Exception exception)
        {
            ModelState.AddModelError(string.Empty, exception.Message);
        }
    }

    return View(country);
}
```

15. Probamos.

Modificación de la Base de Datos

1. Completamos estas entidades:



2. Adicionar **City**:

```
public class City
{
    public int Id { get; set; }
```

```
    [MaxLength(50)]
    [Required]
    public string Name { get; set; }
}
```

3. Adicionamos **Department**:

```
public class Department
{
    public int Id { get; set; }

    [MaxLength(50)]
    [Required]
    public string Name { get; set; }

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

    [DisplayName("Cities Number")]
    public int CitiesNumber => Cities == null ? 0 : Cities.Count;
}
```

4. Modificamos **Country**:

```
public class Country
{
    public int Id { get; set; }

    [MaxLength(50)]
    [Required]
    public string Name { get; set; }

    public ICollection<Department> Departments { get; set; }

    [DisplayName("Departments Number")]
    public int DepartmentsNumber => Departments == null ? 0 : Departments.Count;
}
```

5. Modificamos **DataContext**:

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

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

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

```
    public DbSet<Department> Departments { get; set; }

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

        modelBuilder.Entity<City>()
            .HasIndex(t => t.Name)
            .IsUnique();

        modelBuilder.Entity<Country>()
            .HasIndex(t => t.Name)
            .IsUnique();

        modelBuilder.Entity<Department>()
            .HasIndex(t => t.Name)
            .IsUnique();
    }
}
```

6. Guardamos los cambios y corremos los comandos para actualizar la base de datos de forma local:

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

Creación de un maestro detalle MVC

1. Modificamos la vista **Index** de **Countries**:

```
<thead>
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Name)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.DepartmentsNumber)
        </th>
        <th></th>
    </tr>
</thead>
<tbody>
    @foreach (var item in Model)
    {
    <tr>
        <td>
```

```
            @Html.DisplayFor(modelItem => item.Name)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.DepartmentsNumber)
        </td>
        <td>
            <a asp-action="Edit" asp-route-id="@item.Id" class="btn btn-warning"><i
class="glyphicon glyphicon-pencil"></i></a>
            <a asp-action="Details" asp-route-id="@item.Id" class="btn btn-info"><i
class="glyphicon glyphicon-align-justify"></i></a>
            <button data-id="@item.Id" class="btn btn-danger deleteItem" data-toggle="modal"
data-target="#deleteDialog"><i class="glyphicon glyphicon-trash"></i></button>
        </td>
    </tr>
    }
</tbody>
```

2. Modificamos los métodos **Index** y **Details** del controlador **Countries**:

```
public async Task<IActionResult> Index()
{
    return View(await _context.Countries
        .Include(c => c.Departments)
        .ToListAsync());
}

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

    Country country = await _context.Countries
        .Include(c => c.Departments)
        .ThenInclude(d => d.Cities)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (country == null)
    {
        return NotFound();
    }

    return View(country);
}
```

3. Modificamos la vista **Details** del controlador **Countries**:

```
@model OnSale.Common.Entities.Country
```

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

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />
<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>
        <dt>
            @Html.DisplayNameFor(model => model.DepartmentsNumber)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.DepartmentsNumber)
        </dd>
    </dl>
</div>
<div>
    <a asp-action="AddDepartment"  asp-route-id="@Model.Id" class="btn btn-primary"><i
class="glyphicon glyphicon-plus"></i> Department</a>
    <a asp-action="Edit" asp-route-id="@Model.Id" class="btn btn-warning">Edit</a>
    <a asp-action="Index" class="btn btn-success">Back to List</a>
</div>
<br />

<div class="row">
    <div class="col-md-12">
        <div class="panel panel-default">
            <div class="panel-heading">
                <h3 class="panel-title">Departments</h3>
            </div>
            <div class="panel-body">
                <table class="table table-hover table-responsive table-striped" id="MyTable">
                    <thead>
                        <tr>
                            <th>
                                @Html.DisplayNameFor(model =>
model.Departments.FirstOrDefault().Name)
                            </th>
```

```
                    <th>
                        @Html.DisplayNameFor(model =>
model.Departments.FirstOrDefault().CitiesNumber)
                    </th>
                    <th></th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Model.Departments)
                {
                    <tr>
                        <td>
                            @Html.DisplayFor(modelItem => item.Name)
                        </td>
                        <td>
                            @Html.DisplayFor(modelItem => item.CitiesNumber)
                        </td>
                        <td>
                            <a asp-action="EditDepartment" asp-route-id="@item.Id" class="btn
btn-warning"><i class="glyphicon glyphicon-pencil"></i></a>
                            <a asp-action="DetailsDepartment" asp-route-id="@item.Id" class="btn
btn-info"><i class="glyphicon glyphicon-align-justify"></i></a>
                            <button data-id="@item.Id" class="btn btn-danger deleteItem"
data-toggle="modal" data-target="#deleteDialog"><i class="glyphicon
glyphicon-trash"></i></button>
                        </td>
                    </tr>
                }
            </tbody>
        </table>
    </div>
  </div>
 </div>
</div>

<partial name="_DeleteDialog" />

@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
  <script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>
  <script src="/js/deleteDialog.js"></script>

  <script type="text/javascript">
    $(document).ready(function () {
        $('#MyTable').DataTable();

        // Delete item
```

```
            sc_deleteDialog.openModal('deleteItem', true, 'btnYesDelete',
'/Countries/DeleteDepartment/', false);
        });
    </script>
}
```

4. Modificamos la entidad **Department**:

```
public class Department
{
    public int Id { get; set; }

    [MaxLength(50)]
    [Required]
    public string Name { get; set; }

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

    [DisplayName("Cities Number")]
    public int CitiesNumber => Cities == null ? 0 : Cities.Count;

    [NotMapped]
    public int IdCountry { get; set; }
}
```

5. Iniciamos creando nuevos departamentos, para esto, adicionamos estos métodos al controlador **Countries**:

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

    Country country = await _context.Countries.FindAsync(id);
    if (country == null)
    {
        return NotFound();
    }

    Department model = new Department { IdCountry = country.Id };
    return View(model);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> AddDepartment(Department department)
```

```
{
    if (ModelState.IsValid)
    {
        Country country = await _context.Countries
            .Include(c => c.Departments)
            .FirstOrDefaultAsync(c => c.Id == department.IdCountry);
        if (country == null)
        {
            return NotFound();
        }

        try
        {
            department.Id = 0;
            country.Departments.Add(department);
            _context.Update(country);
            await _context.SaveChangesAsync();
            return RedirectToAction($"{nameof(Details)}/{country.Id}");

        }
        catch (DbUpdateException dbUpdateException)
        {
            if (dbUpdateException.InnerException.Message.Contains("duplicate"))
            {
                ModelState.AddModelError(string.Empty, "There are a record with the same name.");
            }
            else
            {
                ModelState.AddModelError(string.Empty,
dbUpdateException.InnerException.Message);
            }
        }
        catch (Exception exception)
        {
            ModelState.AddModelError(string.Empty, exception.Message);
        }
    }

    return View(department);
}
```

6. Adicionamos la vista parcial **_Department**:

```
@model OnSale.Common.Entities.Department

<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>
```

7. Adicionamos la vista **AddDepartment**:

```
@model OnSale.Common.Entities.Department

@{
    ViewData["Title"] = "Add Department";
}

<h2>Add</h2>

<h4>Department</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="AddDepartment">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="IdCountry" />

            <partial name="_Department" />

            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-primary" />
                <a asp-action="Details" asp-route-id="@Model.IdCountry" class="btn
btn-success">Back to List</a>
            </div>
        </form>
    </div>
</div>

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

8. Probamos.

9. Ahora vamos con la edición de departamentos, agregamos estos métodos a al controlador de **Countries**:

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

```
    Department department = await _context.Departments.FindAsync(id);
    if (department == null)
    {
        return NotFound();
    }

    Country country = await _context.Countries.FirstOrDefaultAsync(c =>
c.Departments.FirstOrDefault(d => d.Id == department.Id) != null);
    department.IdCountry = country.Id;
    return View(department);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> EditDepartment(Department department)
{
    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(department);
            await _context.SaveChangesAsync();
            return RedirectToAction($"{nameof(Details)}/{department.IdCountry}");

        }
        catch (DbUpdateException dbUpdateException)
        {
            if (dbUpdateException.InnerException.Message.Contains("duplicate"))
            {
                ModelState.AddModelError(string.Empty, "There are a record with the same name.");
            }
            else
            {
                ModelState.AddModelError(string.Empty,
dbUpdateException.InnerException.Message);
            }
        }
        catch (Exception exception)
        {
            ModelState.AddModelError(string.Empty, exception.Message);
        }
    }
    return View(department);
}
```

10. Adicionamos la vista **EditDepartment**:

```
@model OnSale.Common.Entities.Department
```

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

<h2>Edit</h2>

<h4>Department</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="EditDepartment">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="Id" />
            <input type="hidden" asp-for="IdCountry" />

            <partial name="_Department" />

            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-primary" />
                <a asp-action="Details" asp-route-id="@Model.IdCountry" class="btn
btn-success">Back to List</a>
            </div>
        </form>
    </div>
</div>

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

11. Vamos ahora con el borrado, adicionamos este método al controlador de **Countries**:

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

    Department department = await _context.Departments
        .Include(d => d.Cities)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (department == null)
    {
        return NotFound();
    }
```

```
    Country country = await _context.Countries.FirstOrDefaultAsync(c =>
c.Departments.FirstOrDefault(d => d.Id == department.Id) != null);
    _context.Departments.Remove(department);
    await _context.SaveChangesAsync();
    return RedirectToAction($"{nameof(Details)}/{country.Id}");
}
```

12. Para evitar que saque error al borrar un país que tenga matriculados departamentos, modificamos el método **Delete**:

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

    Country country = await _context.Countries
        .Include(c => c.Departments)
        .ThenInclude(d => d.Cities)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (country == null)
    {
        return NotFound();
    }

    _context.Countries.Remove(country);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}
```

13. Ahora vamos con la administración de ciudades de un departamento. Adicionamos este método al controlador **Countries**:

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

    Department department = await _context.Departments
        .Include(d => d.Cities)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (department == null)
    {
        return NotFound();
    }
```

```
    Country country = await _context.Countries.FirstOrDefaultAsync(c =>
c.Departments.FirstOrDefault(d => d.Id == department.Id) != null);
    department.IdCountry = country.Id;
    return View(department);
}
```

14. Adicionamos la vista **DetailsDepartment**:

```
@model OnSale.Common.Entities.Department

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

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />
<h2>Details</h2>

<div>
    <h4>Department</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Name)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.CitiesNumber)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.CitiesNumber)
        </dd>
    </dl>
</div>
<div>
    <a asp-action="AddCity" asp-route-id="@Model.Id" class="btn btn-primary"><i
class="glyphicon glyphicon-plus"></i> City</a>
    <a asp-action="Edit" asp-route-id="@Model.Id" class="btn btn-warning">Edit</a>
    <a asp-action="Details" asp-route-id="@Model.IdCountry" class="btn btn-success">Back to
List</a>
</div>
<br />

<div class="row">
    <div class="col-md-12">
        <div class="panel panel-default">
```

```
            <div class="panel-heading">
                <h3 class="panel-title">Cities</h3>
            </div>
            <div class="panel-body">
                <table class="table table-hover table-responsive table-striped" id="MyTable">
                    <thead>
                        <tr>
                            <th>
                                @Html.DisplayNameFor(model => model.Cities.FirstOrDefault().Name)
                            </th>
                            <th></th>
                        </tr>
                    </thead>
                    <tbody>
                        @foreach (var item in Model.Cities)
                        {
                            <tr>
                                <td>
                                    @Html.DisplayFor(modelItem => item.Name)
                                </td>
                                <td>
                                    <a asp-action="EditCity" asp-route-id="@item.Id" class="btn
btn-warning"><i class="glyphicon glyphicon-pencil"></i></a>
                                    <button data-id="@item.Id" class="btn btn-danger deleteItem"
data-toggle="modal" data-target="#deleteDialog"><i class="glyphicon
glyphicon-trash"></i></button>
                                </td>
                            </tr>
                        }
                    </tbody>
                </table>
            </div>
        </div>
    </div>
</div>

<partial name="_DeleteDialog" />

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
    <script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>
    <script src="/js/deleteDialog.js"></script>

    <script type="text/javascript">
        $(document).ready(function () {
            $('#MyTable').DataTable();

            // Delete item
```

```
            sc_deleteDialog.openModal('deleteItem', true, 'btnYesDelete', '/Countries/DeleteCity/',
false);
        });
    </script>
}
```

15. Probamos.

16. Ahora vamos a adicionar una ciudad, primero modificamos la entidad **City**:

```
public class City
{
    public int Id { get; set; }

    [MaxLength(50)]
    [Required]
    public string Name { get; set; }

    [NotMapped]
    public int IdDepartment { get; set; }
}
```

17. Adicionamos estos métodos en el controlador de **Country**:

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

    Department department = await _context.Departments.FindAsync(id);
    if (department == null)
    {
        return NotFound();
    }

    City model = new City { IdDepartment = department.Id };
    return View(model);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> AddCity(City city)
{
    if (ModelState.IsValid)
    {
        Department department = await _context.Departments
```

```
            .Include(d => d.Cities)
            .FirstOrDefaultAsync(c => c.Id == city.IdDepartment);
        if (department == null)
        {
            return NotFound();
        }

        try
        {
            city.Id = 0;
            department.Cities.Add(city);
            _context.Update(department);
            await _context.SaveChangesAsync();
            return RedirectToAction($"{nameof(DetailsDepartment)}/{department.Id}");

        }
        catch (DbUpdateException dbUpdateException)
        {
            if (dbUpdateException.InnerException.Message.Contains("duplicate"))
            {
                ModelState.AddModelError(string.Empty, "There are a record with the same name.");
            }
            else
            {
                ModelState.AddModelError(string.Empty,
dbUpdateException.InnerException.Message);
            }
        }
        catch (Exception exception)
        {
            ModelState.AddModelError(string.Empty, exception.Message);
        }
    }

    return View(city);
}
```

18. Adicionamos la vista parcial **_City**:

```
@model OnSale.Common.Entities.City

<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>
```

19. Adicionamos la vista **AddCity**:

```
@model OnSale.Common.Entities.City

@{
   ViewData["Title"] = "Add City";
}

<h2>Add</h2>

<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="IdDepartment" />

         <partial name="_City" />

         <div class="form-group">
            <input type="submit" value="Save" class="btn btn-primary" />
            <a asp-action="DetailsDepartment" asp-route-id="@Model.IdDepartment" class="btn
btn-success">Back to List</a>
         </div>
      </form>
   </div>
</div>

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

20. Probamos.

21. Para editar la ciudad, adicionamos estos métodos al controlador **Country**:

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

   City city = await _context.Cities.FindAsync(id);
   if (city == null)
   {
      return NotFound();
   }
```

```
    Department department = await _context.Departments.FirstOrDefaultAsync(d =>
d.Cities.FirstOrDefault(c => c.Id == city.Id) != null);
    city.IdDepartment = department.Id;
    return View(city);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> EditCity(City city)
{
    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(city);
            await _context.SaveChangesAsync();
            return RedirectToAction($"{nameof(DetailsDepartment)}/{city.IdDepartment}");

        }
        catch (DbUpdateException dbUpdateException)
        {
            if (dbUpdateException.InnerException.Message.Contains("duplicate"))
            {
                ModelState.AddModelError(string.Empty, "There are a record with the same name.");
            }
            else
            {
                ModelState.AddModelError(string.Empty,
dbUpdateException.InnerException.Message);
            }
        }
        catch (Exception exception)
        {
            ModelState.AddModelError(string.Empty, exception.Message);
        }
    }
    return View(city);
}
```

22. Y adicionamos la vista **EditCity**:

```
@model OnSale.Common.Entities.City

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

```
<h2>Edit</h2>

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

            <partial name="_City" />

            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-primary" />
                <a asp-action="DetailsDepartment" asp-route-id="@Model.IdDepartment" class="btn
btn-success">Back to List</a>
            </div>
        </form>
    </div>
</div>

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

23. Probamos.

24. Por último, adicionamos el método **DeleteCity** al controlador **Country**:

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

    City city = await _context.Cities
        .FirstOrDefaultAsync(m => m.Id == id);
    if (city == null)
    {
        return NotFound();
    }

    Department department = await _context.Departments.FirstOrDefaultAsync(d =>
d.Cities.FirstOrDefault(c => c.Id == city.Id) != null);
    _context.Cities.Remove(city);
    await _context.SaveChangesAsync();
```

```
    return RedirectToAction($"{nameof(DetailsDepartment)}/{department.Id}");
}
```

25. Probamos y hemos terminado un maestro detalle de tres niveles.

Adición de un “Seeder”

1. Vamos a necesitar un método que nos llene información fija en la base de datos y que se verifique esta información exista cada que ejecutemos nuestro sitio WEB. Para tal motivo adicionamos la clase **SeedDb** e la carpeta **Data**:

```
public class SeedDb
{
    private readonly DataContext _context;

    public SeedDb(DataContext context)
    {
        _context = context;
    }

    public async Task SeedAsync()
    {
        await _context.Database.EnsureCreatedAsync();
        await CheckCountriesAsync();
    }

    private async Task CheckCountriesAsync()
    {
        if (!_context.Countries.Any())
        {
            _context.Countries.Add(new Country
            {
                Name = "Colombia",
                Departments = new List<Department>
                {
                    new Department
                    {
                        Name = "Antioquia",
                        Cities = new List<City>
                        {
                            new City { Name = "Medellín" },
                            new City { Name = "Envigado" },
                            new City { Name = "Itagüí" }
                        }
                    },
                    new Department
                    {
```

```
                    Name = "Bogotá",
                    Cities = new List<City>
                    {
                        new City { Name = "Bogotá" }
                    }
                },
                new Department
                {
                    Name = "Valle del Cauca",
                    Cities = new List<City>
                    {
                        new City { Name = "Cali" },
                        new City { Name = "Buenaventura" },
                        new City { Name = "Palmira" }
                    }
                }
            }
        });
        _context.Countries.Add(new Country
        {
            Name = "USA",
            Departments = new List<Department>
            {
                new Department
                {
                    Name = "California",
                    Cities = new List<City>
                    {
                        new City { Name = "Los Angeles" },
                        new City { Name = "San Diego" },
                        new City { Name = "San Francisco" }
                    }
                },
                new Department
                {
                    Name = "Illinois",
                    Cities = new List<City>
                    {
                        new City { Name = "Chicago" },
                        new City { Name = "Springfield" }
                    }
                }
            }
        });
        await _context.SaveChangesAsync();
    }
  }
}
```

2. Modificamos el **Startup** para inyectar esta clase:

```
public void ConfigureServices(IServiceCollection services)
{
   services.Configure<CookiePolicyOptions>(options =>
   {
      options.CheckConsentNeeded = context => true;
      options.MinimumSameSitePolicy = SameSiteMode.None;
   });

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

   services.AddTransient<SeedDb>();
   services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

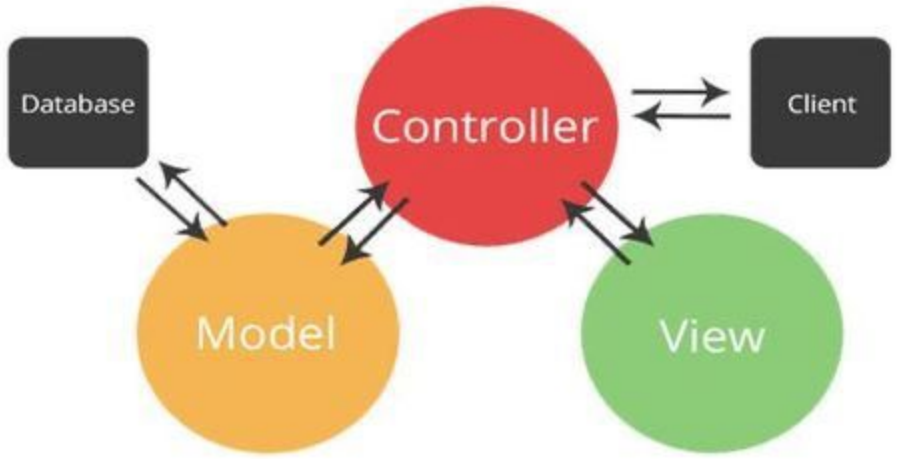
3. Modificamos el **Program** para llamar el seeder cada que inicie nuestro sitio WEB:

```
public class Program
{
   public static void Main(string[] args)
   {
      IWebHost host = CreateWebHostBuilder(args).Build();
      RunSeeding(host);
      host.Run();
   }

   private static void RunSeeding(IWebHost host)
   {
      IServiceScopeFactory scopeFactory = host.Services.GetService<IServiceScopeFactory>();
      using (IServiceScope scope = scopeFactory.CreateScope())
      {
         SeedDb seeder = scope.ServiceProvider.GetService<SeedDb>();
         seeder.SeedAsync().Wait();
      }
   }

   public static IWebHostBuilder CreateWebHostBuilder(string[] args)
   {
      return WebHost.CreateDefaultBuilder(args).UseStartup<Startup>();
   }
}
```

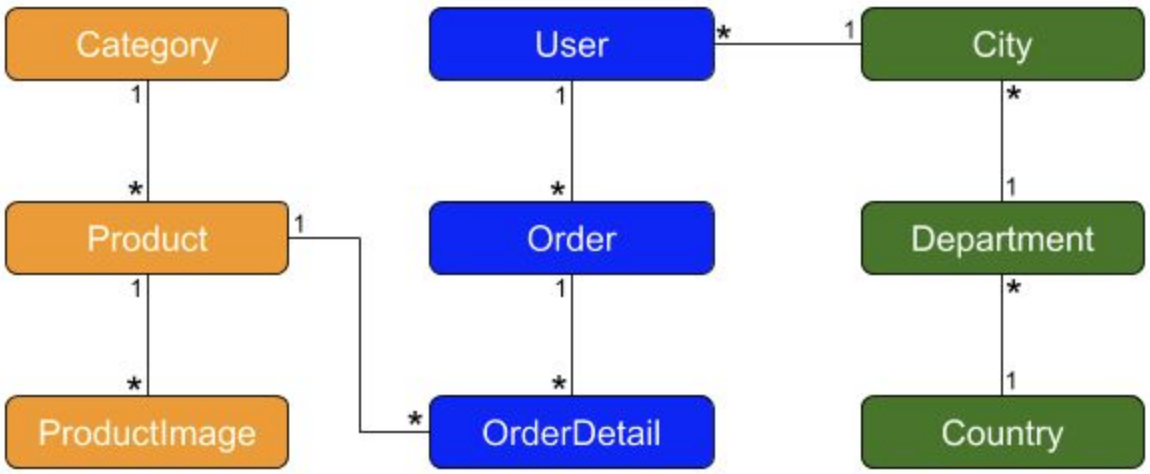
4. Borramos la base de datos con el comando: **drop-database**

5. Corremos el proyecto y probamos.

Creación de un CRUD donde el Model no es el mismo Entity (Categories)



1. Continuamos completando la base de datos, y en esta ocasión vamos a añadir unas tablas que incluyen imágenes. Específicamente vamos a adicionar las tablas de categoría, productos e imágenes.



2. Iniciamos con la entidad **Category**:

```
public class Category
{
```

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

    [MaxLength(50)]
    [Required]
    public string Name { get; set; }

    [Display(Name = "Image")]
    public Guid ImageId { get; set; }

    //TODO: Pending to put the correct paths
    [Display(Name = "Image")]
    public string ImageFullPath => ImageId == Guid.Empty
        ? $"https://localhost:44390/images/noimage.png"
        : $"https://onsale.blob.core.windows.net/categories/{ImageId}";
}
```

3. Ahora con la entidad **ProductImage**:

```
public class ProductImage
{
    public int Id { get; set; }

    [Display(Name = "Image")]
    public Guid ImageId { get; set; }

    //TODO: Pending to put the correct paths
    [Display(Name = "Image")]
    public string ImageFullPath => ImageId == Guid.Empty
        ? $"https://localhost:44390/images/noimage.png"
        : $"https://onsale.blob.core.windows.net/products/{ImageId}";
}
```

4. Finalizamos con la entidad **Product**:

```
public class Product
{
    public int Id { get; set; }

    [MaxLength(50)]
    [Required]
    public string Name { get; set; }

    [DataType(DataType.MultilineText)]
    public string Description { get; set; }

    [DisplayFormat(DataFormatString = "{0:C2}")]
    public decimal Price { get; set; }
```

```
    [DisplayName("Is Active")]
    public bool IsActive { get; set; }

    [DisplayName("Is Starred")]
    public bool IsStarred { get; set; }

    public Category Category { get; set; }

    public ICollection<ProductImage> ProductImages { get; set; }

    [DisplayName("Product Images Number")]
    public int ProductImagesNumber => ProductImages == null ? 0 : ProductImages.Count;

    //TODO: Pending to put the correct paths
    [Display(Name = "Image")]
    public string ImageFullPath => ProductImages == null || ProductImages.Count == 0
        ? $"https://localhost:44390/images/noimage.png"
        : ProductImages.FirstOrDefault().ImageFullPath;
}
```

5. Actualizamos el **DataContext**:

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

    public DbSet<Category> Categories { get; set; }

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

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

    public DbSet<Department> Departments { get; set; }

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

    public DbSet<ProductImage> ProductImages { get; set; }

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

        modelBuilder.Entity<Category>()
            .HasIndex(t => t.Name)
            .IsUnique();
```

```
        modelBuilder.Entity<City>()
            .HasIndex(t => t.Name)
            .IsUnique();

        modelBuilder.Entity<Country>()
            .HasIndex(t => t.Name)
            .IsUnique();

        modelBuilder.Entity<Department>()
            .HasIndex(t => t.Name)
            .IsUnique();

        modelBuilder.Entity<Product>()
            .HasIndex(t => t.Name)
            .IsUnique();
    }
}
```

6. Guardamos los cambios y corremos los comandos para actualizar la base de datos de forma local:

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

Nota: si te genera error, borrar la BD con el comando **drop-database** y volverlo a intentar.

7. Ahora vamos hacer un CRUD para **Category**, pero como tenemos captura de imagen. Debemos crear primero la **CategoryViewModel** en la carpeta **Models**:

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

8. Creamos **CategoriesController**:

```
public class CategoriesController : Controller
{
    private readonly DataContext _context;

    public CategoriesController(DataContext context)
    {
        _context = context;
    }

    public async Task<IActionResult> Index()
```

```
    {
        return View(await _context.Categories.ToListAsync());
    }
}
```

9. Creamos la vista **Index** de **CategoriesController**:

```
@model IEnumerable<OnSale.Common.Entities.Category>

@{
    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> Add
New</a>
</p>

<div class="row">
    <div class="col-md-12">
        <div class="panel panel-default">
            <div class="panel-heading">
                <h3 class="panel-title">Categories</h3>
            </div>
            <div class="panel-body">
                <table class="table table-hover table-responsive table-striped" id="MyTable">
                    <thead>
                        <tr>
                            <th>
                                @Html.DisplayNameFor(model => model.Name)
                            </th>
                            <th>
                                @Html.DisplayNameFor(model => model.ImageFullPath)
                            </th>
                            <th></th>
                        </tr>
                    </thead>
                    <tbody>
                        @foreach (var item in Model)
                        {
                        <tr>
                            <td>
                                @Html.DisplayFor(modelItem => item.Name)
                            </td>
                            <td>
```

```
                        <img src="@item.ImageFullPath"
style="width:100px;height:100px;max-width: 100%; height: auto;" />
                    </td>
                    <td>
                        <a asp-action="Edit" asp-route-id="@item.Id" class="btn btn-warning"><i
class="glyphicon glyphicon-pencil"></i></a>
                        <button data-id="@item.Id" class="btn btn-danger deleteItem"
data-toggle="modal" data-target="#deleteDialog"><i class="glyphicon
glyphicon-trash"></i></button>
                    </td>
                </tr>
                }
            </tbody>
        </table>
    </div>
  </div>
 </div>
</div>

<partial name="_DeleteDialog" />

@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
  <script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>
  <script src="/js/deleteDialog.js"></script>

  <script type="text/javascript">
    $(document).ready(function () {
      $('#MyTable').DataTable();

      // Delete item
      sc_deleteDialog.openModal('deleteItem', true, 'btnYesDelete', '/Categories/Delete/',
false);
    });
  </script>
}
```

10. Probamos.

11. Vamos a crear un Blob Storage en Azure para poder almacenar nuestras imágenes:

Create storage account

Basics | Networking | Data protection | Advanced | Tags | Review + create

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below.
Learn more about Azure storage accounts

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	Visual Studio Enterprise
Resource group *	(New) OnSale Create new

Instance details

The default deployment model is Resource Manager, which supports the latest Azure features. You may choose to deploy using the classic deployment model instead. Choose classic deployment model

Storage account name *	onsale
Location *	(US) Central US
Performance	Standard / Premium
Account kind	BlobStorage
Replication	Read-access geo-redundant storage (RA-GRS)
	Accounts with the selected kind, replication and performance type only support block and append blobs. Page blobs, file shares, tables, and queues will not be available.
Access tier (default)	Cool / Hot

Luego creamos los contenedores donde vamos a almacenar nuestras imágenes: categories, products, users:

+ Container | Change access level | Refresh | Delete

Search containers by prefix

Name	Last modified	Public access level	Lease state
categories	8/4/2020, 2:44:12 PM	Container	Available
products	8/4/2020, 2:44:24 PM	Container	Available
users	8/4/2020, 2:44:32 PM	Container	Available

12. Agregamos la cadena de conexión del blob al **appsettings.json**:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection":
"Server=(localdb)\\MSSQLLocalDB;Database=OnSale;Trusted_Connection=True;MultipleActive
ResultSets=true"
  },
  "Blob": {
    "ConnectionString":
"DefaultEndpointsProtocol=https;AccountName=onsale;AccountKey=u4Ds+6uYoz5qfaejvPRRQ
Cg3PI5HVaDaLj1rfx/UVSB68trksZ37YRTxawCr8tSZmNxXIxzW2VqNgE5vvactYg==;EndpointS
uffix=core.windows.net"
  }
}
```

13. Vamos a crear la carpeta **Helpers** y dentro de esta vamos a crear el **IBlobHelper** para subir archivos a nuestro blob:

```
public interface IBlobHelper
{
    Task<Guid> UploadBlobAsync(IFormFile file, string containerName);

    Task<Guid> UploadBlobAsync(byte[] file, string containerName);

    Task<Guid> UploadBlobAsync(string image, string containerName);
}
```

14. Creamos la implementación de la interfaz **BlobHelper**:

```
public class BlobHelper : IBlobHelper
{
    private readonly CloudBlobClient _blobClient;

    public BlobHelper(IConfiguration configuration)
    {
        string keys = configuration["Blob:ConnectionString"];
        CloudStorageAccount storageAccount = CloudStorageAccount.Parse(keys);
        _blobClient = storageAccount.CreateCloudBlobClient();
    }

    public async Task<Guid> UploadBlobAsync(byte[] file, string containerName)
```

```
    {
        MemoryStream stream = new MemoryStream(file);
        Guid name = Guid.NewGuid();
        CloudBlobContainer container = _blobClient.GetContainerReference(containerName);
        CloudBlockBlob blockBlob = container.GetBlockBlobReference($"{name}");
        await blockBlob.UploadFromStreamAsync(stream);
        return name;
    }

    public async Task<Guid> UploadBlobAsync(IFormFile file, string containerName)
    {
        Stream stream = file.OpenReadStream();
        Guid name = Guid.NewGuid();
        CloudBlobContainer container = _blobClient.GetContainerReference(containerName);
        CloudBlockBlob blockBlob = container.GetBlockBlobReference($"{name}");
        await blockBlob.UploadFromStreamAsync(stream);
        return name;
    }

    public async Task<Guid> UploadBlobAsync(string image, string containerName)
    {
        Stream stream = File.OpenRead(image);
        Guid name = Guid.NewGuid();
        CloudBlobContainer container = _blobClient.GetContainerReference(containerName);
        CloudBlockBlob blockBlob = container.GetBlockBlobReference($"{name}");
        await blockBlob.UploadFromStreamAsync(stream);
        return name;
    }
}
```

15. Dentro de la misma carpeta **Helpers** adicionamos la interfaz **IConverterHelper** para convertir los objetos:

```
public interface IConverterHelper
{
    Category ToCategory(CategoryViewModel model, Guid imageId, bool isNew);

    CategoryViewModel ToCategoryViewModel(Category category);
}
```

16. Implementamos la interfaz:

```
public class ConverterHelper : IConverterHelper
{
    public Category ToCategory(CategoryViewModel model, Guid imageId, bool isNew)
    {
        return new Category
        {
```

```
            Id = isNew ? 0 : model.Id,
            ImageId = imageId,
            Name = model.Name
        };
    }

    public CategoryViewModel ToCategoryViewModel(Category category)
    {
        return new CategoryViewModel
        {
            Id = category.Id,
            ImageId = category.ImageId,
            Name = category.Name
        };
    }
}
```

17. Adicioamos la inyección en el **Startup** de los helpers que acabamos de crear:

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

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

    services.AddTransient<SeedDb>();
    services.AddScoped<IBlobHelper, BlobHelper>();
    services.AddScoped<IConverterHelper, ConverterHelper>();
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

18. Modificamos el **CategoriesController**:

```
public class CategoriesController : Controller
{
    private readonly DataContext _context;
    private readonly IBlobHelper _blobHelper;
    private readonly IConverterHelper _converterHelper;

    public CategoriesController(DataContext context, IBlobHelper blobHelper, IConverterHelper
converterHelper)
```

```
    {
        _context = context;
        _blobHelper = blobHelper;
        _converterHelper = converterHelper;
    }

    public async Task<IActionResult> Index()
    {
        return View(await _context.Categories.ToListAsync());
    }

    public IActionResult Create()
    {
        CategoryViewModel model = new CategoryViewModel();
        return View(model);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create(CategoryViewModel model)
    {
        if (ModelState.IsValid)
        {
            Guid imageId = Guid.Empty;

            if (model.ImageFile != null)
            {
                imageId = await _blobHelper.UploadBlobAsync(model.ImageFile, "categories");
            }

            try
            {
                Category category = _converterHelper.ToCategory(model, imageId, true);
                _context.Add(category);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            catch (DbUpdateException dbUpdateException)
            {
                if (dbUpdateException.InnerException.Message.Contains("duplicate"))
                {
                    ModelState.AddModelError(string.Empty, "There are a record with the same
name.");
                }
                else
                {
                    ModelState.AddModelError(string.Empty,
dbUpdateException.InnerException.Message);
```

```
            }
        }
        catch (Exception exception)
        {
            ModelState.AddModelError(string.Empty, exception.Message);
        }
    }

    return View(model);
  }
}
```

19. Creamos la vista parcial **_Category**:

```
@model OnSale.Web.Models.CategoryViewModel

<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="ImageFile" class="control-label"></label>
  <input asp-for="ImageFile" type="file" class="form-control" />
  <span asp-validation-for="ImageFile" class="text-danger"></span>
</div>
```

20. Creamos la vista **Create** del controlador **Categories**:

```
@model OnSale.Web.Models.CategoryViewModel

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

<h2>Create</h2>

<h4>Category</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>

      <partial name="_Category" />

      <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");}
}
```

21. Probamos.

22. Ahora implementemos el **Edit** de la categoría, para eso, agregamos estos métodos al controlador de **Categories**:

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

  Category category = await _context.Categories.FindAsync(id);
  if (category == null)
  {
    return NotFound();
  }

  CategoryViewModel model = _converterHelper.ToCategoryViewModel(category);
  return View(model);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(CategoryViewModel model)
{
  if (ModelState.IsValid)
  {
    Guid imageId = model.ImageId;

    if (model.ImageFile != null)
    {
      imageId = await _blobHelper.UploadBlobAsync(model.ImageFile, "categories");
    }

    try
    {
```

```
            Category category = _converterHelper.ToCategory(model, imageId, false);
            _context.Update(category);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));

        }
        catch (DbUpdateException dbUpdateException)
        {
            if (dbUpdateException.InnerException.Message.Contains("duplicate"))
            {
                ModelState.AddModelError(string.Empty, "There are a record with the same name.");
            }
            else
            {
                ModelState.AddModelError(string.Empty,
dbUpdateException.InnerException.Message);
            }
        }
        catch (Exception exception)
        {
            ModelState.AddModelError(string.Empty, exception.Message);
        }
    }

    return View(model);
}
```

23. Agregamos la vista **Edit**:

```
@model OnSale.Web.Models.CategoryViewModel

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

    <h2>Edit</h2>

<h4>Category</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="ImageId" />

            <partial name="_Category" />
```

```
            <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">
        <img src="@Model.ImageFullPath" style="width:200px;height:200px;max-width: 100%;
height: auto;" />
    </div>
</div>

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

24. Probamos.

25. Por último para terminar este CRUD agregamos el método del **Delete**:

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

    Category category = await _context.Categories
        .FirstOrDefaultAsync(m => m.Id == id);
    if (category == null)
    {
        return NotFound();
    }

    try
    {
        _context.Categories.Remove(category);
        await _context.SaveChangesAsync();
    }
    catch (Exception ex)
    {
        ModelState.AddModelError(string.Empty, ex.Message);
    }

    return RedirectToAction(nameof(Index));
}
```

26. Probamos.

Creación de un CRUD con lista desplegable (Products)

1. Vamos ahora a crear el CRUD de productos el cual tiene una lista desplegable para seleccionar la categoría del producto. Empezamos creado la **ProductViewModel**:

```
public class ProductViewModel : Product
{
    [Display(Name = "Category")]
    [Range(1, int.MaxValue, ErrorMessage = "You must select a category.")]
    [Required]
    public int CategoryId { get; set; }

    public IEnumerable<SelectListItem> Categories { get; set; }
}
```

2. Creamos el **ICombosHelper** dentro de la carpeta **Helpers**:

```
public interface ICombosHelper
{
    IEnumerable<SelectListItem> GetComboCategories();
}
```

3. Implementamos la interfaz:

```
public class CombosHelper : ICombosHelper
{
    private readonly DataContext _context;

    public CombosHelper(DataContext context)
    {
        _context = context;
    }

    public IEnumerable<SelectListItem> GetComboCategories()
    {
        List<SelectListItem> list = _context.Categories.Select(t => new SelectListItem
        {
            Text = t.Name,
            Value = $"{t.Id}"
        })
            .OrderBy(t => t.Text)
            .ToList();

        list.Insert(0, new SelectListItem
```

```
        {
            Text = "[Select a category...]",
            Value = "0"
        });

        return list;
    }
}
```

4. Configuramos la inyección de la nueva interfaz en el **Startup**:

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

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

    services.AddTransient<SeedDb>();
    services.AddScoped<IBlobHelper, BlobHelper>();
    services.AddScoped<IConverterHelper, ConverterHelper>();
    services.AddScoped<ICombosHelper, CombosHelper>();

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

5. Adicionamos estos métodos a la interfaz **IConverterHelper**:

```
Task<Product> ToProductAsync(ProductViewModel model, bool isNew);

ProductViewModel ToProductViewModel(Product product);
```

6. Implementamos los nuevos métodos:

```
public async Task<Product> ToProductAsync(ProductViewModel model, bool isNew)
{
    return new Product
    {
        Category = await _context.Categories.FindAsync(model.CategoryId),
        Description = model.Description,
        Id = isNew ? 0 : model.Id,
        IsActive = model.IsActive,
```

```
        IsStarred = model.IsStarred,
        Name = model.Name,
        Price = model.Price,
        ProductImages = model.ProductImages
    };
}

public ProductViewModel ToProductViewModel(Product product)
{
    return new ProductViewModel
    {
        Categories = _combosHelper.GetComboCategories(),
        Category = product.Category,
        CategoryId = product.Category.Id,
        Description = product.Description,
        Id = product.Id,
        IsActive = product.IsActive,
        IsStarred = product.IsStarred,
        Name = product.Name,
        Price = product.Price,
        ProductImages = product.ProductImages
    };
}
```

7. Creamos el **ProductsController**:

```
public class ProductsController : Controller
{
    private readonly DataContext _context;
    private readonly IBlobHelper _blobHelper;
    private readonly ICombosHelper _combosHelper;
    private readonly IConverterHelper _converterHelper;

    public ProductsController(DataContext context, IBlobHelper blobHelper, ICombosHelper
combosHelper, IConverterHelper converterHelper)
    {
        _context = context;
        _blobHelper = blobHelper;
        _combosHelper = combosHelper;
        _converterHelper = converterHelper;
    }

    public async Task<IActionResult> Index()
    {
        return View(await _context.Products
            .Include(p => p.Category)
            .Include(p => p.ProductImages)
            .ToListAsync());
```

```
    }
}
```

8. Creamos la vista **Index** en el **ProductsController**:

```
@model IEnumerable<OnSale.Common.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> Add
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="MyTable">
                    <thead>
                        <tr>
                            <th>
                                @Html.DisplayNameFor(model => model.Name)
                            </th>
                            <th>
                                @Html.DisplayNameFor(model => model.ImageFullPath)
                            </th>
                            <th>
                                @Html.DisplayNameFor(model => model.Price)
                            </th>
                            <th>
                                @Html.DisplayNameFor(model => model.IsActive)
                            </th>
                            <th>
                                @Html.DisplayNameFor(model => model.IsStarred)
                            </th>
                            <th>
                                Category
                            </th>
                            <th>
```

```
                        @Html.DisplayNameFor(model => model.ProductImagesNumber)
                    </th>
                    <th></th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Model)
                {
                <tr>
                    <td>
                        @Html.DisplayFor(modelItem => item.Name)
                    </td>
                    <td>
                        <img src="@item.ImageFullPath"
style="width:100px;height:100px;max-width: 100%; height: auto;" />
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.Price)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.IsActive)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.IsStarred)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.Category.Name)
                    </td>
                    <td>
                        @Html.DisplayFor(modelItem => item.ProductImagesNumber)
                    </td>
                    <td>
                        <a asp-action="Edit" asp-route-id="@item.Id" class="btn btn-warning"><i
class="glyphicon glyphicon-pencil"></i></a>
                        <a asp-action="Details" asp-route-id="@item.Id" class="btn btn-info"><i
class="glyphicon glyphicon-align-justify"></i></a>
                        <button data-id="@item.Id" class="btn btn-danger deleteItem"
data-toggle="modal" data-target="#deleteDialog"><i class="glyphicon
glyphicon-trash"></i></button>
                    </td>
                </tr>
                }
            </tbody>
        </table>
    </div>
  </div>
 </div>
</div>
```

```
<partial name="_DeleteDialog" />

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
    <script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>
    <script src="/js/deleteDialog.js"></script>

    <script type="text/javascript">
        $(document).ready(function () {
            $('#MyTable').DataTable();

            // Delete item
            sc_deleteDialog.openModal('deleteItem', true, 'btnYesDelete', '/Products/Delete/', false);
        });
    </script>
}
```

9. Adicionamos la nueva opción al menú:

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

10. Probamos lo que llevamos hasta el momento.

11. Para poder crear nuevos productos adicionamos estos métodos al **ProductsController**:

```
public IActionResult Create()
{
    ProductViewModel model = new ProductViewModel
    {
        Categories = _combosHelper.GetComboCategories(),
        IsActive = true
    };

    return View(model);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(ProductViewModel model)
{
    if (ModelState.IsValid)
    {
```

```
        try
        {
            Product product = await _converterHelper.ToProductAsync(model, true);

            if (model.ImageFile != null)
            {
                Guid imageId = await _blobHelper.UploadBlobAsync(model.ImageFile, "products");
                product.ProductImages = new List<ProductImage>
                {
                    new ProductImage { ImageId = imageId }
                };
            }

            _context.Add(product);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        catch (DbUpdateException dbUpdateException)
        {
            if (dbUpdateException.InnerException.Message.Contains("duplicate"))
            {
                ModelState.AddModelError(string.Empty, "There are a record with the same name.");
            }
            else
            {
                ModelState.AddModelError(string.Empty,
dbUpdateException.InnerException.Message);
            }
        }
        catch (Exception exception)
        {
            ModelState.AddModelError(string.Empty, exception.Message);
        }
    }

    model.Categories = _combosHelper.GetComboCategories();
    return View(model);
}
```

12. Adicionamos la vista parcial **_Product** en el **ProductsController**:

```
@model OnSale.Web.Models.ProductViewModel

<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="Description" class="control-label"></label>
    <textarea asp-for="Description" class="form-control"></textarea>
    <span asp-validation-for="Description" class="text-danger"></span>
</div>

<div class="form-group">
    <label asp-for="CategoryId" class="control-label"></label>
    <select asp-for="CategoryId" asp-items="Model.Categories" class="form-control"></select>
    <span asp-validation-for="CategoryId" 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" type="file" class="form-control" />
    <span asp-validation-for="ImageFile" class="text-danger"></span>
</div>

<div class="form-group">
    <div class="checkbox">
        <label>
            <input asp-for="IsActive" /> @Html.DisplayNameFor(model => model.IsActive)
        </label>
    </div>
</div>

<div class="form-group">
    <div class="checkbox">
        <label>
            <input asp-for="IsStarred" /> @Html.DisplayNameFor(model => model.IsStarred)
        </label>
    </div>
</div>
```

13. Adicionamos la vista **Create** en el **ProductsController**:

```
@model OnSale.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>

            <partial name="_Product" />

            <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");}
}
```

14. Probamos.

15. Continuamos con la edición de productos. Adicionamos estos métodos en el controlador **Products**:

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

    Product product = await _context.Products
        .Include(p => p.Category)
        .Include(p => p.ProductImages)
        .FirstOrDefaultAsync(p => p.Id == id);
    if (product == null)
    {
        return NotFound();
    }

    ProductViewModel model = _converterHelper.ToProductViewModel(product);
    return View(model);
}
```

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(ProductViewModel model)
{
    if (ModelState.IsValid)
    {
        try
        {
            Product product = await _converterHelper.ToProductAsync(model, false);

            if (model.ImageFile != null)
            {
                Guid imageId = await _blobHelper.UploadBlobAsync(model.ImageFile, "products");
                if (product.ProductImages == null)
                {
                    product.ProductImages = new List<ProductImage>();
                }

                product.ProductImages.Add(new ProductImage { ImageId = imageId });
            }

            _context.Update(product);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));

        }
        catch (DbUpdateException dbUpdateException)
        {
            if (dbUpdateException.InnerException.Message.Contains("duplicate"))
            {
                ModelState.AddModelError(string.Empty, "There are a record with the same name.");
            }
            else
            {
                ModelState.AddModelError(string.Empty,
dbUpdateException.InnerException.Message);
            }
        }
        catch (Exception exception)
        {
            ModelState.AddModelError(string.Empty, exception.Message);
        }
    }

    model.Categories = _combosHelper.GetComboCategories();
    return View(model);
}
```

16. Adicionamos la vista **Edit** en el **ProductsController**:

```
@model OnSale.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" />

         <partial name="_Product" />

         <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">
      <img src="@Model.ImageFullPath" style="width:200px;height:200px;max-width: 100%;
height: auto;" />
   </div>
</div>

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

17. Probamos.

18. Continuamos con el borrado de productos. Adicionamos este método al **ProductsController**:

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

```
    Product product = await _context.Products
        .Include(p => p.ProductImages)
        .FirstOrDefaultAsync(p => p.Id == id);
    if (product == null)
    {
        return NotFound();
    }

    try
    {
        _context.Products.Remove(product);
        await _context.SaveChangesAsync();
    }
    catch (Exception ex)
    {
        ModelState.AddModelError(string.Empty, ex.Message);
    }

    return RedirectToAction(nameof(Index));
}
```

19. Probamos.

20. Para terminar con este CRUD vamos a implementar el botón **Details** para administrar las diferentes imágenes de un producto. Adicionemos este método al **ProdcutsController**:

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

    Product product = await _context.Products
        .Include(c => c.Category)
        .Include(c => c.ProductImages)
        .FirstOrDefaultAsync(m => m.Id == id);
    if (product == null)
    {
        return NotFound();
    }

    return View(product);
}
```

21. Adicionamos la vista **Details** en el **ProductsController**:

```
@model OnSale.Common.Entities.Product

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

<link rel="stylesheet" href="https://cdn.datatables.net/1.10.19/css/jquery.dataTables.min.css" />
<h2>Details</h2>

<div>
    <h4>Product</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Name)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Name)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Description)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Description)
        </dd>
        <dt>
            Category
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Category.Name)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.Price)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.Price)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.IsActive)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.IsActive)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.IsStarred)
        </dt>
        <dd>
```

```
            @Html.DisplayFor(model => model.IsStarred)
        </dd>
        <dt>
            @Html.DisplayNameFor(model => model.ProductImagesNumber)
        </dt>
        <dd>
            @Html.DisplayFor(model => model.ProductImagesNumber)
        </dd>
    </dl>
</div>
<div>
    <a asp-action="AddImage" asp-route-id="@Model.Id" class="btn btn-primary"><i
class="glyphicon glyphicon-plus"></i> Image</a>
    <a asp-action="Edit" asp-route-id="@Model.Id" class="btn btn-warning">Edit</a>
    <a asp-action="Index" class="btn btn-success">Back to List</a>
</div>
<br />

<div class="row">
    <div class="col-md-12">
        <div class="panel panel-default">
            <div class="panel-heading">
                <h3 class="panel-title">Product Images</h3>
            </div>
            <div class="panel-body">
                <table class="table table-hover table-responsive table-striped" id="MyTable">
                    <thead>
                        <tr>
                            <th>
                                @Html.DisplayNameFor(model =>
model.ProductImages.FirstOrDefault().ImageFullPath)
                            </th>
                            <th></th>
                        </tr>
                    </thead>
                    <tbody>
                        @foreach (var item in Model.ProductImages)
                        {
                            <tr>
                                <td>
                                    <img src="@item.ImageFullPath"
style="width:200px;height:200px;max-width: 100%; height: auto;" />
                                </td>
                                <td>
                                    <button data-id="@item.Id" class="btn btn-danger deleteItem"
data-toggle="modal" data-target="#deleteDialog"><i class="glyphicon
glyphicon-trash"></i></button>
                                </td>
```

```
                    </tr>
                }
            </tbody>
        </table>
    </div>
  </div>
</div>
</div>

<partial name="_DeleteDialog" />

@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
  <script src="//cdn.datatables.net/1.10.19/js/jquery.dataTables.min.js"></script>
  <script src="/js/deleteDialog.js"></script>

  <script type="text/javascript">
    $(document).ready(function () {
      $('#MyTable').DataTable();

      // Delete item
      sc_deleteDialog.openModal('deleteItem', true, 'btnYesDelete', '/Products/DeleteImage/',
false);
    });
  </script>
}
```

22. Probamos.

23. Ahora para poder adicionar varias imágenes a un producto adicionamos la clase **AddProductImageViewModel**:

```
public class AddProductImageViewModel
{
  public int ProductId { get; set; }

  [Display(Name = "Image")]
  [Required]
  public IFormFile ImageFile { get; set; }
}
```

24. Adicionamos estos métodos en el **ProductsController**:

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

```
    }

    Product product = await _context.Products.FindAsync(id);
    if (product == null)
    {
        return NotFound();
    }

    AddProductImageViewModel model = new AddProductImageViewModel { ProductId =
product.Id };
    return View(model);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> AddImage(AddProductImageViewModel model)
{
    if (ModelState.IsValid)
    {
        Product product = await _context.Products
            .Include(p => p.ProductImages)
            .FirstOrDefaultAsync(p => p.Id == model.ProductId);
        if (product == null)
        {
            return NotFound();
        }

        try
        {
            Guid imageId = await _blobHelper.UploadBlobAsync(model.ImageFile, "products");
            if (product.ProductImages == null)
            {
                product.ProductImages = new List<ProductImage>();
            }

            product.ProductImages.Add(new ProductImage { ImageId = imageId });
            _context.Update(product);
            await _context.SaveChangesAsync();
            return RedirectToAction($"{nameof(Details)}/{product.Id}");

        }
        catch (Exception exception)
        {
            ModelState.AddModelError(string.Empty, exception.Message);
        }
    }

    return View(model);
```

```
}
```

25. Adicionamos la vista **AddImage** el **ProductsController**:

```
@model OnSale.Web.Models.AddProductImageViewModel

@{
    ViewData["Title"] = "Add Image";
}

<h2>Add</h2>

<h4>Image</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="AddImage" enctype="multipart/form-data">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="ProductId" />

            <div class="form-group">
                <label asp-for="ImageFile" class="control-label"></label>
                <input asp-for="ImageFile" type="file" class="form-control" />
                <span asp-validation-for="ImageFile" class="text-danger"></span>
            </div>

            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-primary" />
                <a asp-action="Details" asp-route-id="@Model.ProductId" class="btn
btn-success">Back to List</a>
            </div>
        </form>
    </div>
</div>

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

26. Probamos.

27. Por último para finalizar este CRUD creamos el método **DeleteImage**:

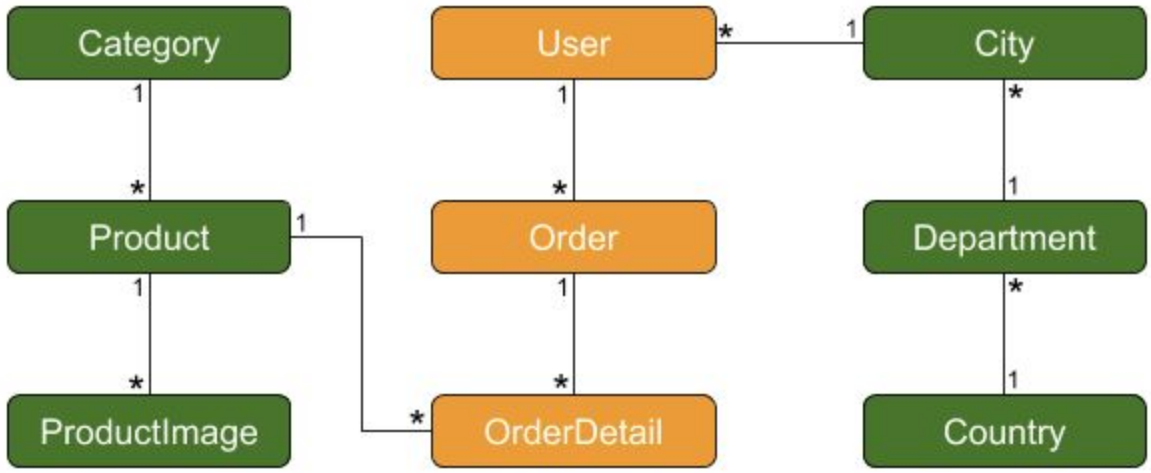
```
public async Task<IActionResult> DeleteImage(int? id)
{
    if (id == null)
    {
        return NotFound();
```

```
    }

    ProductImage productImage = await _context.ProductImages
        .FirstOrDefaultAsync(m => m.Id == id);
    if (productImage == null)
    {
        return NotFound();
    }

    Product product = await _context.Products.FirstOrDefaultAsync(p =>
p.ProductImages.FirstOrDefault(pi => pi.Id == productImage.Id) != null);
    _context.ProductImages.Remove(productImage);
    await _context.SaveChangesAsync();
    return RedirectToAction($"{nameof(Details)}/{product.Id}");
}
```

28. Probamos.

Adición de usuarios y roles

Una característica muy importante de nuestra aplicación, y de la mayoría de las aplicaciones es la utilización de usuarios. Procedemos a actualizar nuestro sitio con las tablas que nos faltan:



1. Como vamos a tener dos tipos de usuarios; administradores y usuarios. Vamos a crear una enumeración para diferenciarlos. Creamos la carpeta **Enums** en el proyecto **Common** y dentro de esta carpeta la enumeración **UserType**:

```
public enum UserType
{
    Admin,
    User
}
```

2. En el proyecto **Web** en la carpeta **Data**, crear la carpeta **Entities** y dentro de esta, crear la entidad **User**:

```
public class User : IdentityUser
{
    [MaxLength(20)]
    [Required]
    public string Document { get; set; }

    [Display(Name = "First Name")]
    [MaxLength(50)]
    [Required]
    public string FirstName { get; set; }

    [Display(Name = "Last Name")]
    [MaxLength(50)]
    [Required]
    public string LastName { get; set; }

    [MaxLength(100)]
    public string Address { get; set; }

    [Display(Name = "Image")]
    public Guid ImageId { get; set; }

    //TODO: Pending to put the correct paths
    [Display(Name = "Image")]
    public string ImageFullPath => ImageId == Guid.Empty
        ? $"https://localhost:44390/images/noimage.png"
        : $"https://onsale.blob.core.windows.net/users/{ImageId}";

    [Display(Name = "User Type")]
    public UserType UserType { get; set; }

    public City City { get; set; }

    [Display(Name = "User")]
    public string FullName => $"{FirstName} {LastName}";

    [Display(Name = "User")]
    public string FullNameWithDocument => $"{FirstName} {LastName} - {Document}";
}
```

3. Modificar el **DataContext**:

```
public class DataContext : IdentityDbContext<User>
{
```

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

4. Crear la interfaz **IUserHelper**:

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

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

    Task CheckRoleAsync(string roleName);

    Task AddUserToRoleAsync(User user, string roleName);

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

5. Creamos la implementación de la interfaz **UserHelper**:

```
public class UserHelper : IUserHelper
{
    private readonly DataContext _context;
    private readonly UserManager<User> _userManager;
    private readonly RoleManager<IdentityRole> _roleManager;

    public UserHelper(DataContext context, UserManager<User> userManager,
RoleManager<IdentityRole> roleManager)
    {
        _context = context;
        _userManager = userManager;
        _roleManager = roleManager;
    }

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

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

    public async Task CheckRoleAsync(string roleName)
    {
```

```
        bool roleExists = await _roleManager.RoleExistsAsync(roleName);
        if (!roleExists)
        {
            await _roleManager.CreateAsync(new IdentityRole
            {
                Name = roleName
            });
        }
    }

    public async Task<User> GetUserAsync(string email)
    {
        return await _context.Users
            .Include(u => u.City)
            .FirstOrDefaultAsync(u => u.Email == email);
    }

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

6. Modificamos el método **ConfigureServices** del **Startup**:

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    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(Configuration.GetConnectionString("DefaultConnection"));
    });
```

```
    services.AddTransient<SeedDb>();
    services.AddScoped<IBlobHelper, BlobHelper>();
    services.AddScoped<IConverterHelper, ConverterHelper>();
    services.AddScoped<ICombosHelper, CombosHelper>();
    services.AddScoped<IUserHelper, UserHelper>();

    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

7. Modificamos el método **Configure** del **Startup**:

```
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?}");
    });
}
```

8. Modificamos el **SeedDb**:

```
public class SeedDb
{
    private readonly DataContext _context;
    private readonly IUserHelper _userHelper;

    public SeedDb(DataContext context, IUserHelper userHelper)
    {
        _context = context;
        _userHelper = userHelper;
    }
```

```
    public async Task SeedAsync()
    {
        await _context.Database.EnsureCreatedAsync();
        await CheckCountriesAsync();
        await CheckRolesAsync();
        await CheckUserAsync("1010", "Juan", "Zuluaga", "jzuluaga55@hotmail.com", "322 311
4620", "Calle Luna Calle Sol", UserType.Admin);
    }

    private async Task CheckRolesAsync()
    {
        await _userHelper.CheckRoleAsync(UserType.Admin.ToString());
        await _userHelper.CheckRoleAsync(UserType.User.ToString());
    }

    private async Task<User> CheckUserAsync(
        string document,
        string firstName,
        string lastName,
        string email,
        string phone,
        string address,
        UserType userType)
    {
        User user = await _userHelper.GetUserAsync(email);
        if (user == null)
        {
            user = new User
            {
                FirstName = firstName,
                LastName = lastName,
                Email = email,
                UserName = email,
                PhoneNumber = phone,
                Address = address,
                Document = document,
                City = _context.Cities.FirstOrDefault(),
                UserType = userType
            };

            await _userHelper.AddUserAsync(user, "123456");
            await _userHelper.AddUserToRoleAsync(user, userType.ToString());
        }

        return user;
    }
```

```
    private async Task CheckCountriesAsync()
…
```

9. Salvamos los cambios y ejecutamos los siguientes comandos para actualizar la BD:

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

10. Probamos.

11. Para finalizar con este capítulo vamos a completar las entidades. Iniciamos creando la enumeración **OrderStatus**:

```
public enum OrderStatus
{
    Pending,
    Spreading,
    Sent,
    Confirmed
}
```

12. En el proyecto **Common** creamos la entidad **OrderDetail**:

```
public class OrderDetail
{
    public int Id { get; set; }

    public Product Product { get; set; }

    public float Quantity { get; set; }

    public decimal Price { get; set; }

    [DataType(DataType.MultilineText)]
    public string Remarks { get; set; }

    public decimal Value => (decimal)Quantity * Price;
}
```

13. En el proyecto **Web** creamos la entidad **Order**:

```
public class Order
{
    public int Id { get; set; }

    public DateTime Date { get; set; }

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

```
    public OrderStatus OrderStatus { get; set; }

    [Display(Name = "Date Sent")]
    public DateTime? DateSent { get; set; }

    [Display(Name = "Date Confirmed")]
    public DateTime? DateConfirmed { get; set; }

    [DataType(DataType.MultilineText)]
    public string Remarks { get; set; }

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

    public int Lines => OrderDetails == null ? 0 : OrderDetails.Count;

    public float Quantity => OrderDetails == null ? 0 : OrderDetails.Sum(od => od.Quantity);

    public decimal Value => OrderDetails == null ? 0 : OrderDetails.Sum(od => od.Value);
}
```

14. Actualizamos el **DataContext**:

```
…
public DbSet<Department> Departments { get; set; }

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

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

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

15. Grabamos los cambios y corremos los siguientes comandos:

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

Implementando Login/Logout

1. Creamos la **LoginViewModel**:

```
public class LoginViewModel
{
    [Required]
    [EmailAddress]
    public string Username { get; set; }
```

```
    [Required]
    [MinLength(6)]
    public string Password { get; set; }

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

2. Adicionamos estos métodos a la **IUserHelper**:

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

Task LogoutAsync();
```

3. Y agregamos su implementación en el **UserHelper:**

```
…
private readonly DataContext _context;
private readonly UserManager<User> _userManager;
private readonly RoleManager<IdentityRole> _roleManager;
private readonly SignInManager<User> _signInManager;

public UserHelper(DataContext context, UserManager<User> userManager,
RoleManager<IdentityRole> roleManager, SignInManager<User> signInManager)
{
    _context = context;
    _userManager = userManager;
    _roleManager = roleManager;
    _signInManager = signInManager;
}

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

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

4. Creamos el **AccountController**:

```
public class AccountController : Controller
{
    private readonly IUserHelper _userHelper;

    public AccountController(IUserHelper userHelper)
    {
        _userHelper = userHelper;
    }

    public IActionResult Login()
    {
        if (User.Identity.IsAuthenticated)
        {
            return RedirectToAction("Index", "Home");
        }

        return View(new LoginViewModel());
    }

    [HttpPost]
    public async Task<IActionResult> Login(LoginViewModel model)
    {
        if (ModelState.IsValid)
        {
            Microsoft.AspNetCore.Identity.SignInResult result = await
_userHelper.LoginAsync(model);
            if (result.Succeeded)
            {
                if (Request.Query.Keys.Contains("ReturnUrl"))
                {
                    return Redirect(Request.Query["ReturnUrl"].First());
                }

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

            ModelState.AddModelError(string.Empty, "Email or password incorrect.");
        }

        return View(model);
    }

    public async Task<IActionResult> Logout()
    {
        await _userHelper.LogoutAsync();
        return RedirectToAction("Index", "Home");
    }
}
```

5. Adicioamos la vista **Login**:

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

6. Adicionamos la anotación authorize a los controladores previos:

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

7. Modificamos nuestro menú **_Layout**:

```
…
<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 (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
      {
         <li><a asp-area="" asp-controller="Countries" asp-action="Index">Countries</a></li>
         <li><a asp-area="" asp-controller="Categories" asp-action="Index">Categories</a></li>
         <li><a asp-area="" asp-controller="Products" asp-action="Index">Products</a></li>
      }
   </ul>
   <ul class="nav navbar-nav navbar-right">
      @if (User.Identity.IsAuthenticated)
      {
         <li><a asp-area="" asp-controller="Account"
asp-action="ChangeUser">@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>
```

8. Probamos.

Creando API sin seguridad

1. Creamos la carpeta **API** dentro de la carpeta Controllers y dentro de API creamos el **CountriesController** con el siguiente código:

```
[ApiController]
[Route("api/[controller]")]
public class CountriesController : ControllerBase
{
   private readonly DataContext _context;

   public CountriesController(DataContext context)
```

```
    {
        _context = context;
    }

    [HttpGet]
    public IActionResult GetCountries()
    {
        return Ok(_context.Countries
            .Include(c => c.Departments)
            .ThenInclude(d => d.Cities));
    }
}
```

2. Probamos.

3. Para evitar campos innecesarios podemos agregar la anotación **JsonIgnore**, modificamos las entidades **Department** y **City**:

```
[JsonIgnore]
[NotMapped]
public int IdCountry { get; set; }

[JsonIgnore]
[NotMapped]
public int IdDepartment { get; set; }
```

4. Probamos.

5. Hacemos lo propio pero para **ProductsController**:

```
[ApiController]
[Route("api/[controller]")]
public class ProductsController : ControllerBase
{
    private readonly DataContext _context;

    public ProductsController(DataContext context)
    {
        _context = context;
    }

    [HttpGet]
    public IActionResult GetProducts()
    {
        return Ok(_context.Products
            .Include(p => p.Category)
            .Include(p => p.ProductImages)
            .Where(p => p.IsActive));
```

```
    }
}
```

6. Probamos.

Generando Token de seguridad

1. Agregamos estos valores a nuestro **appsettings**:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection":
"Server=(localdb)\\MSSQLLocalDB;Database=OnSale;Trusted_Connection=True;MultipleActive
ResultSets=true"
  },
  "Blob": {
    "ConnectionString":
"DefaultEndpointsProtocol=https;AccountName=onsale;AccountKey=u4Ds+6uYoz5qfaejvPRRQ
Cg3PI5HVaDaLj1rfx/UVSB68trksZ37YRTxawCr8tSZmNxXIxzW2VqNgE5vvactYg==;EndpointS
uffix=core.windows.net"
  },
  "Tokens": {
    "Key":
"asdfghjikbnRRUREDFJDLKJF69877vcgfdsrtfyLKJHGFRTGVC543FDhgcvgfx_dg708ctrreSSss
SwwsswrrrRRRRWYUIY",
    "Issuer": "localhost",
    "Audience": "users"
  }
}
```

2. Agregamos este método al **IUserHelper**:

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

3. Y su implementación en el **UserHelper**:

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

4. Dentro de la carpeta **API/Controllers** creamos el **AccountController**:

```
[ApiController]
[Route("api/[controller]")]
public class AccountController : ControllerBase
{
    private readonly IUserHelper _userHelper;
    private readonly IConfiguration _configuration;

    public AccountController(IUserHelper userHelper, IConfiguration configuration)
    {
        _userHelper = userHelper;
        _configuration = configuration;
    }

    [HttpPost]
    [Route("CreateToken")]
    public async Task<IActionResult> CreateToken([FromBody] LoginViewModel model)
    {
        if (ModelState.IsValid)
        {
            User user = await _userHelper.GetUserAsync(model.Username);
            if (user != null)
            {
                Microsoft.AspNetCore.Identity.SignInResult result = await
_userHelper.ValidatePasswordAsync(user, model.Password);

                if (result.Succeeded)
                {
                    Claim[] claims = new[]
                    {
                        new Claim(JwtRegisteredClaimNames.Sub, user.Email),
                        new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
                    };

                    SymmetricSecurityKey key = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["Tokens:Key"]));
                    SigningCredentials credentials = new SigningCredentials(key,
SecurityAlgorithms.HmacSha256);
                    JwtSecurityToken token = new JwtSecurityToken(
                        _configuration["Tokens:Issuer"],
                        _configuration["Tokens:Audience"],
                        claims,
                        expires: DateTime.UtcNow.AddDays(99),
                        signingCredentials: credentials);
                    var results = new
                    {
                        token = new JwtSecurityTokenHandler().WriteToken(token),
```

```
                    expiration = token.ValidTo,
                    user
                };

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

    return BadRequest();
  }
}
```

5. Agregamos la configuración en el **Startup**:

```
…
}).AddEntityFrameworkStores<DataContext>();

services.AddAuthentication()
   .AddCookie()
   .AddJwtBearer(cfg =>
   {
      cfg.TokenValidationParameters = new TokenValidationParameters
      {
         ValidIssuer = Configuration["Tokens:Issuer"],
         ValidAudience = Configuration["Tokens:Audience"],
         IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Tokens:Key"]))
      };
   });

services.AddDbContext<DataContext>(cfg =>
…
```

6. Probamos.

Creando API con seguridad

Dentro de nuestra App van haber métodos que necesitan seguridad y otros métodos que no necesitan seguridad. Este es un ejemplo de cómo colocarle seguridad a un método de nuestra API.

1. Dentro de la carpeta **API/Controllers** agregamos este método al **AccountController**:

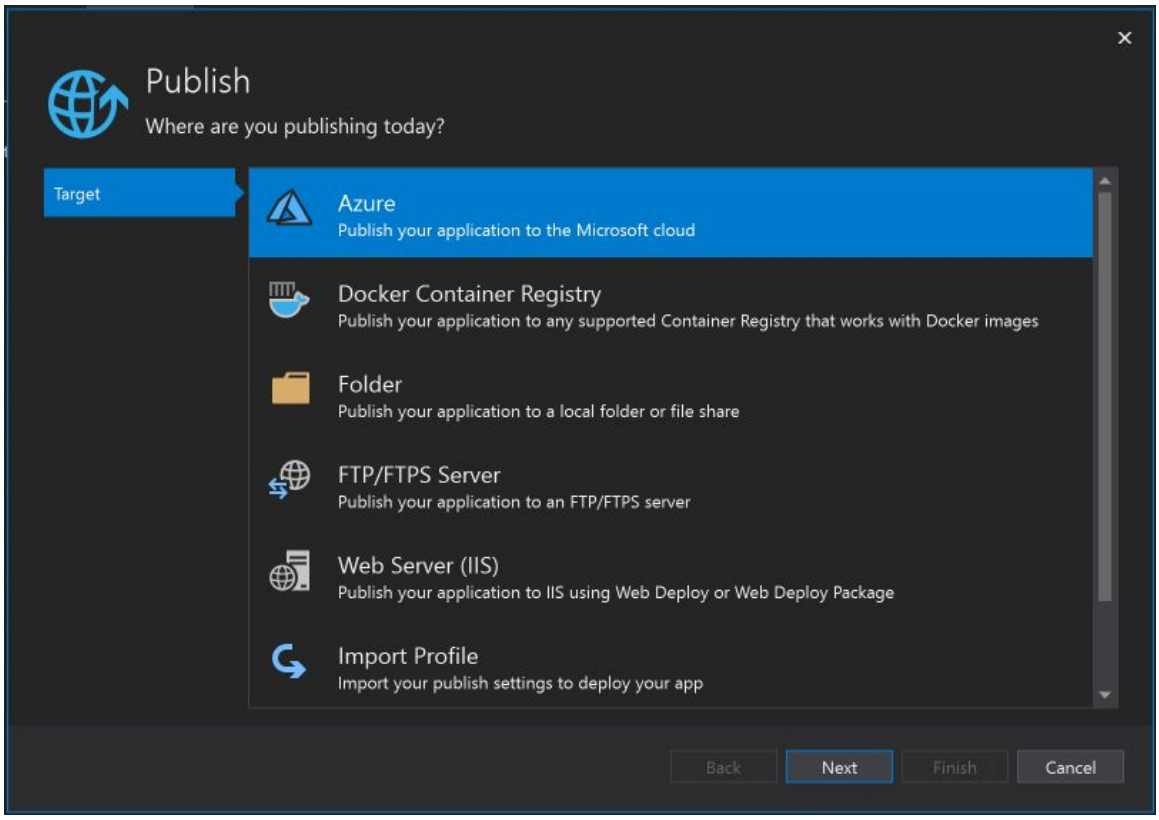
```
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
[HttpPost]
[Route("GetUserByEmail")]
```

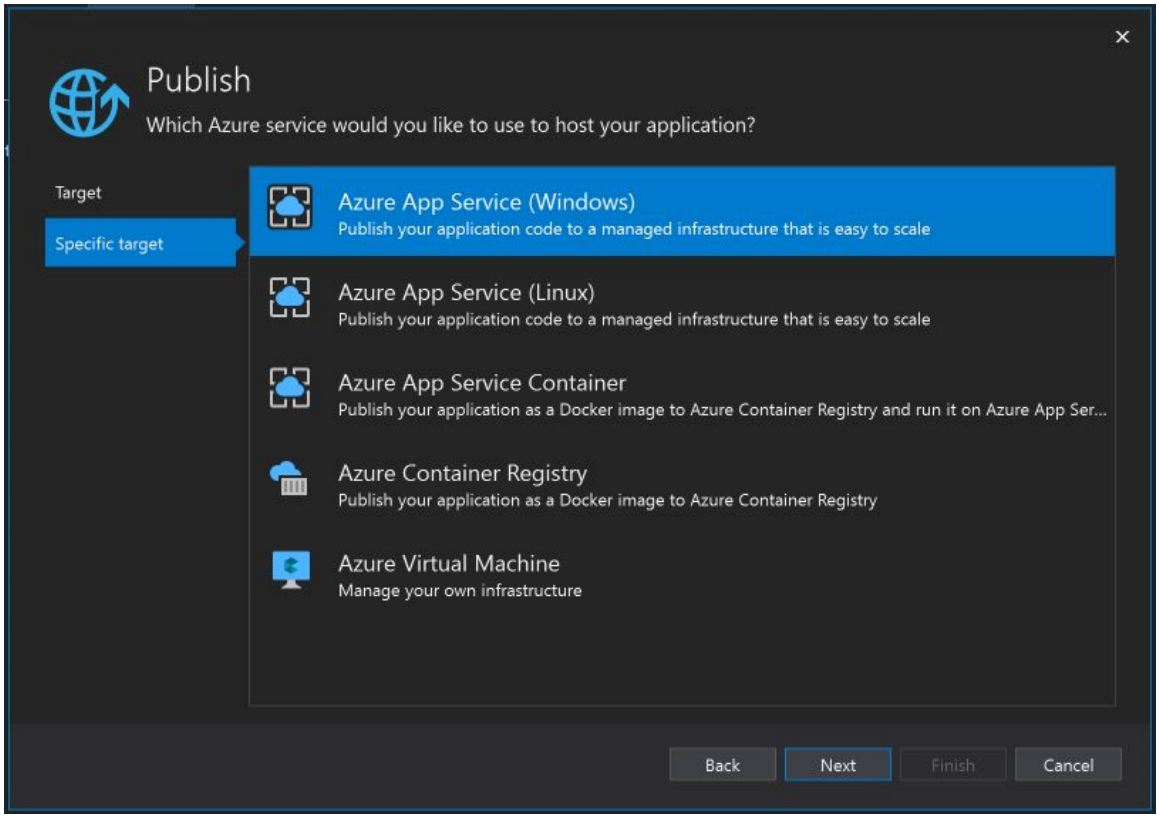
```
public async Task<IActionResult> GetUserByEmail([FromBody] EmailRequest request)
{
    if (!ModelState.IsValid)
    {
        return BadRequest();
    }

    User user = await _userHelper.GetUserAsync(request.Email);
    if (user == null)
    {
        return NotFound("Error001");
    }

    return Ok(user);
}
```

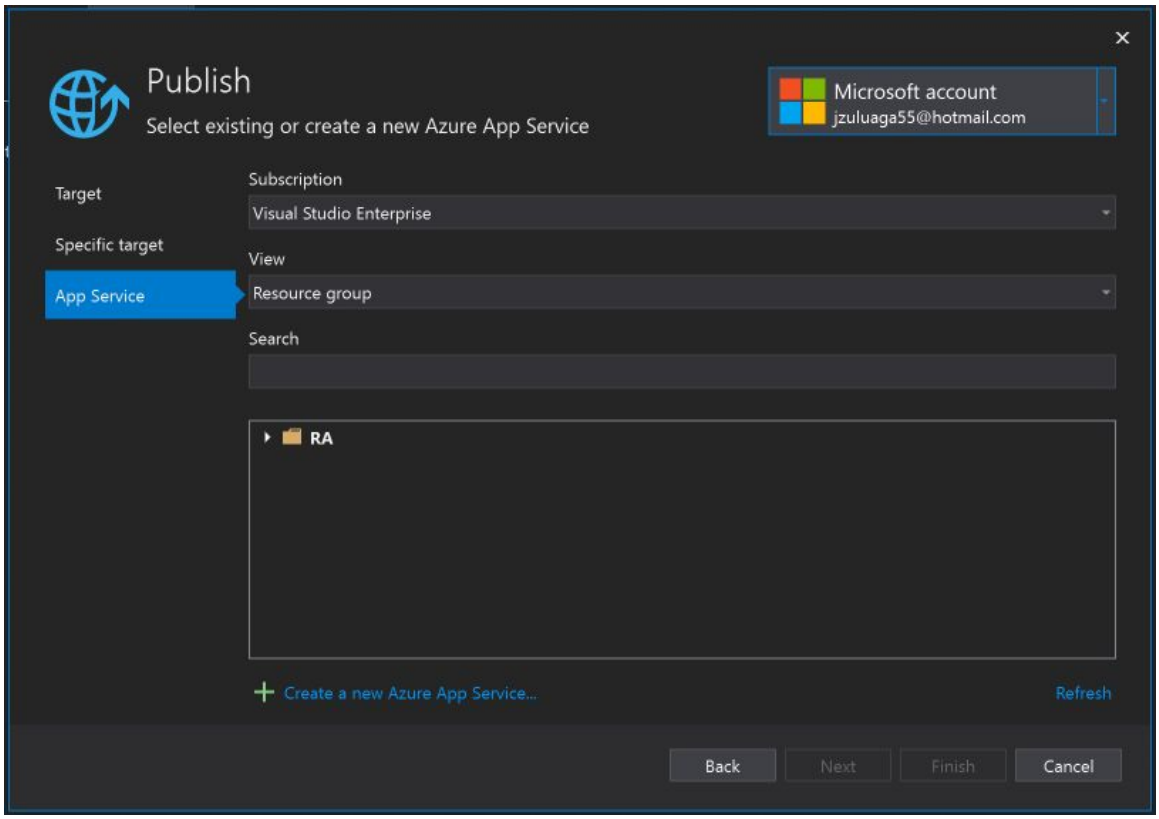
2. Probamos.

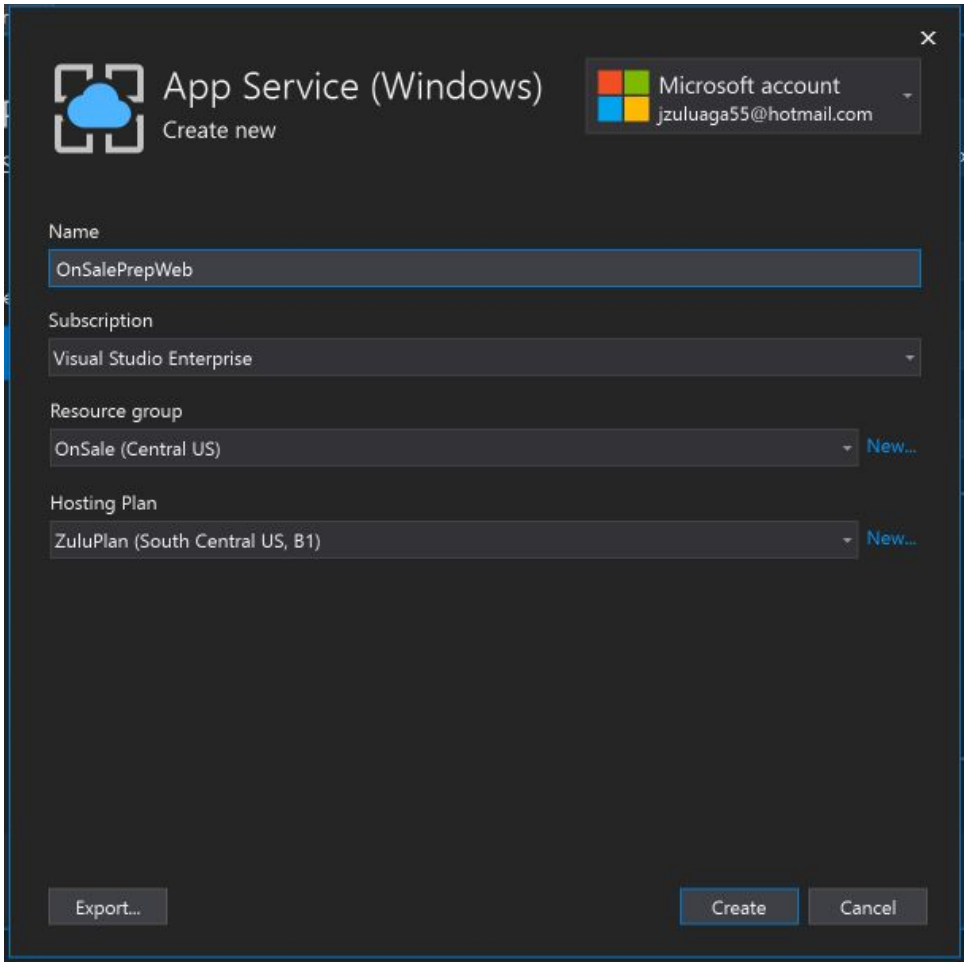
Publicando en Azure

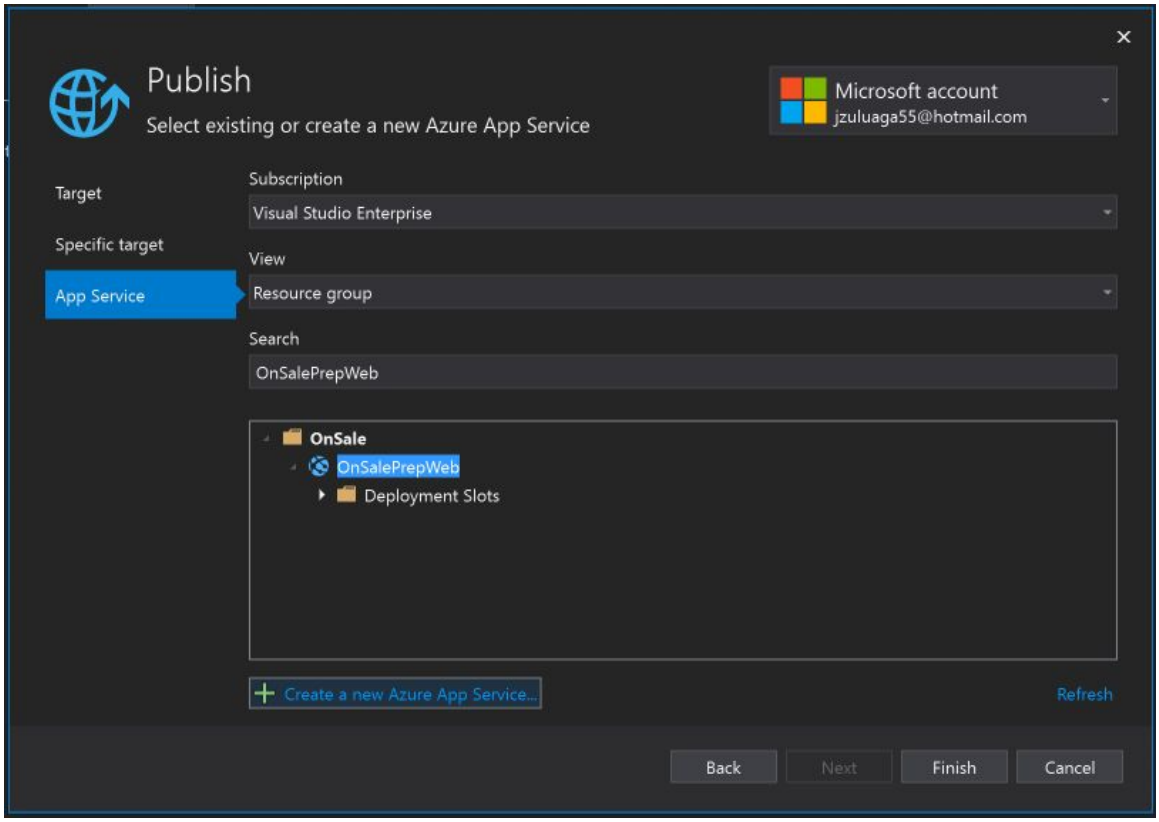
Para poder accesar nuestra API desde los dispositivos móviles necesitamos publicar nuestro sitio. Este es un ejemplo de como publicar en Azure.

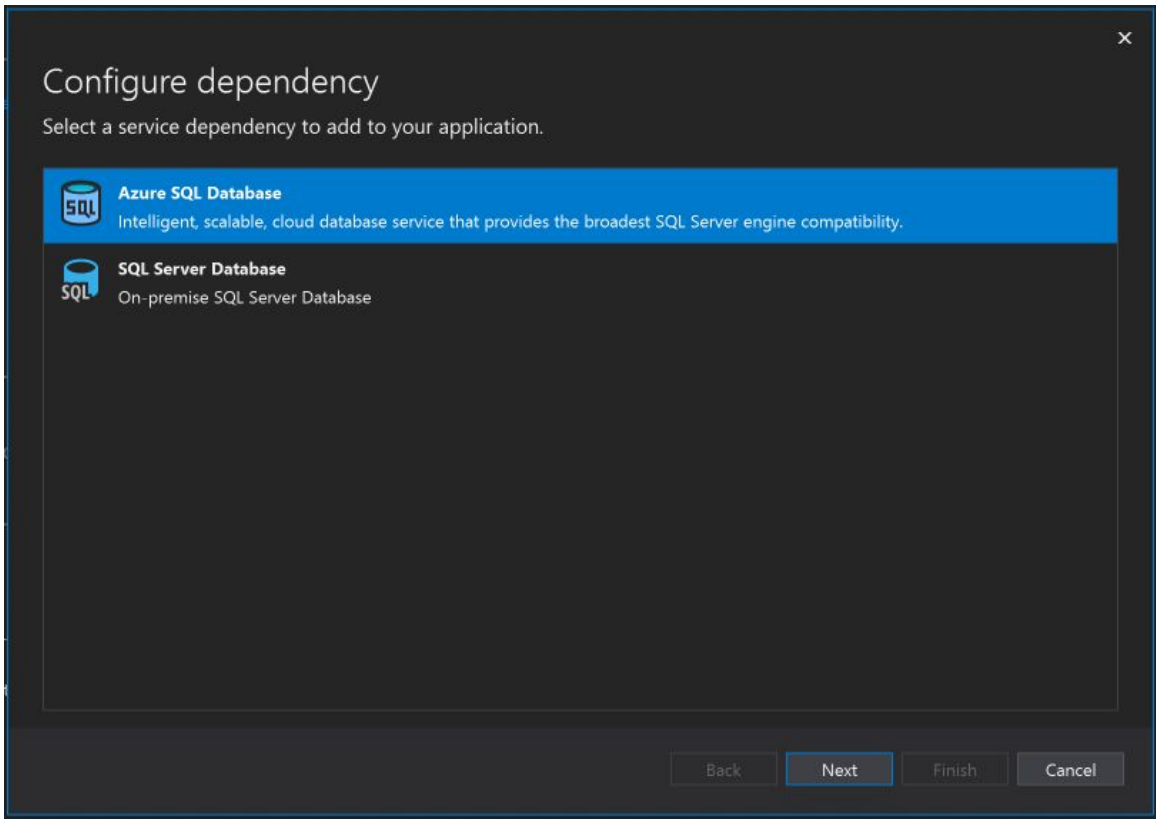


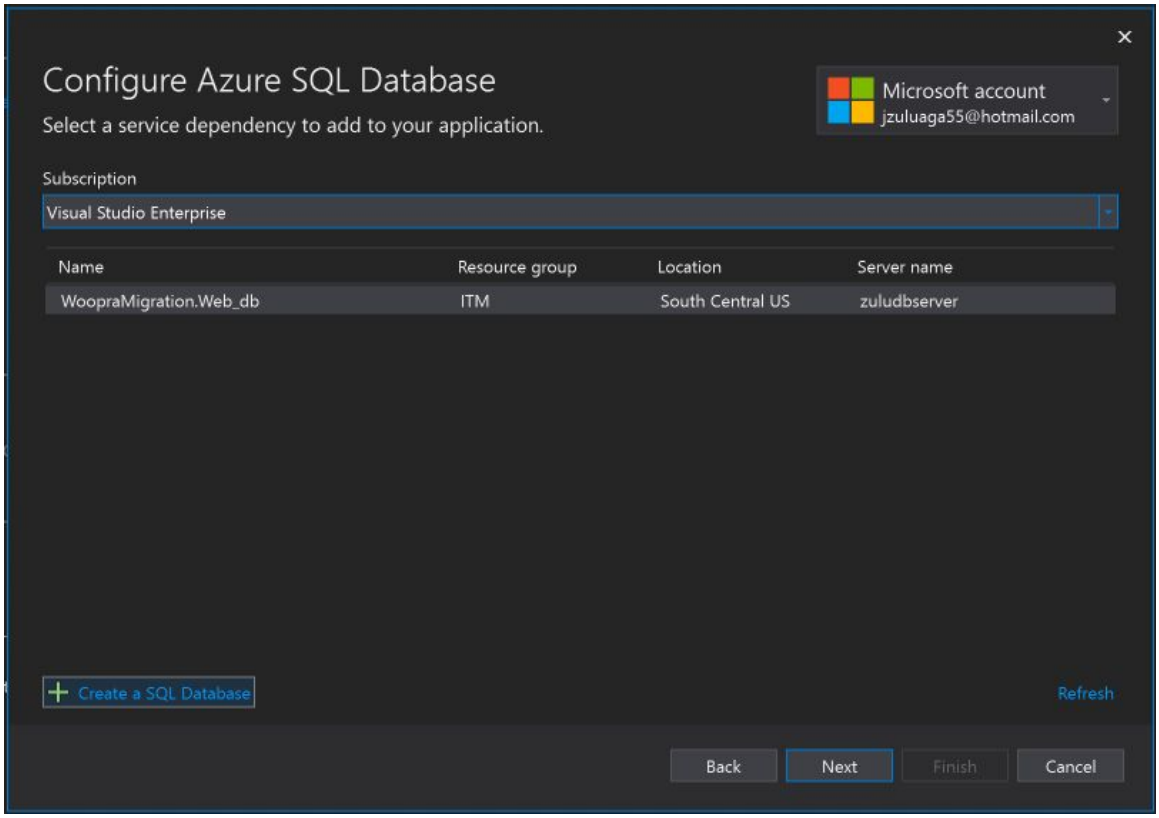

Publish
Which Azure service would you like to use to host your application?
Target
Specific target
Azure App Service (Windows)
Publish your application code to a managed infrastructure that is easy to scale
Azure App Service (Linux)
Publish your application code to a managed infrastructure that is easy to scale
Azure App Service Container
Publish your application as a Docker image to Azure Container Registry and run it on Azure App Ser...
Azure Container Registry
Publish your application as a Docker image to Azure Container Registry
Azure Virtual Machine
Manage your own infrastructure
Back
Next
Finish
Cancel

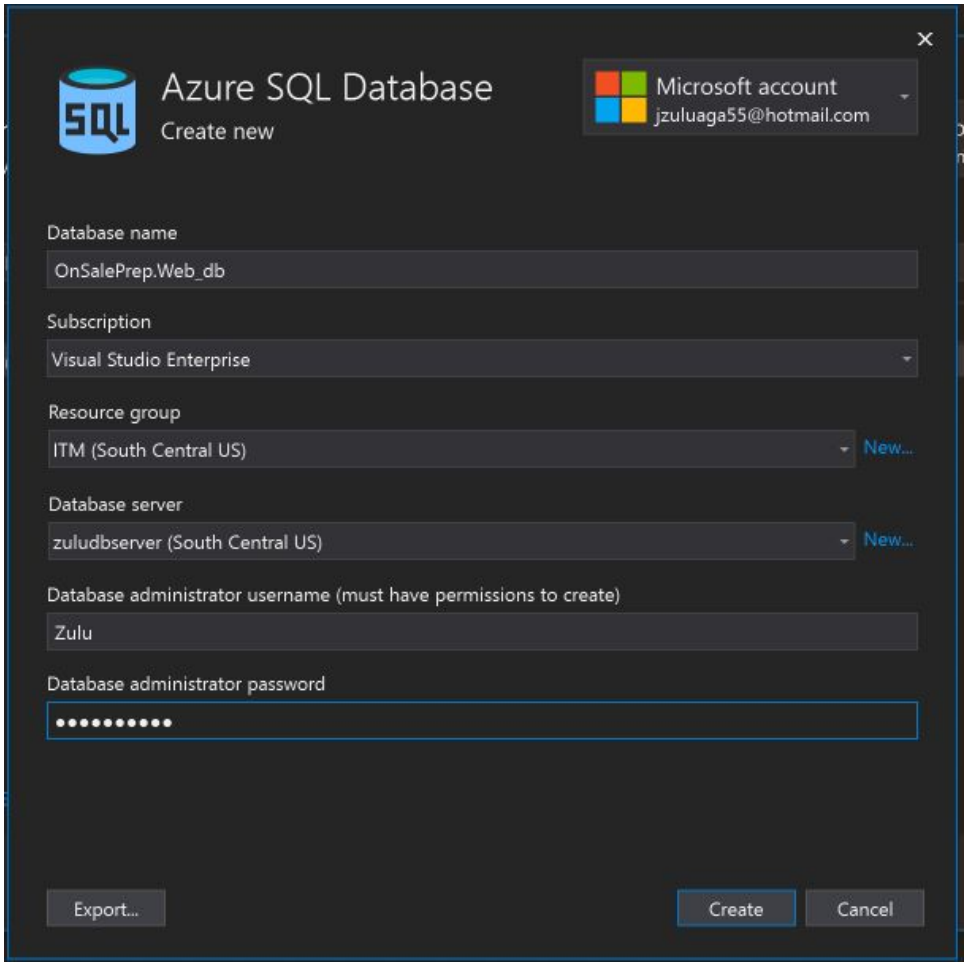

Publish
Select existing or create a new Azure App Service
Microsoft account
jzuluaga55@hotmail.com
Target
Specific target
App Service
Subscription
Visual Studio Enterprise
View
Resource group
Search
RA
Create a new Azure App Service...
Refresh
Back
Next
Finish
Cancel

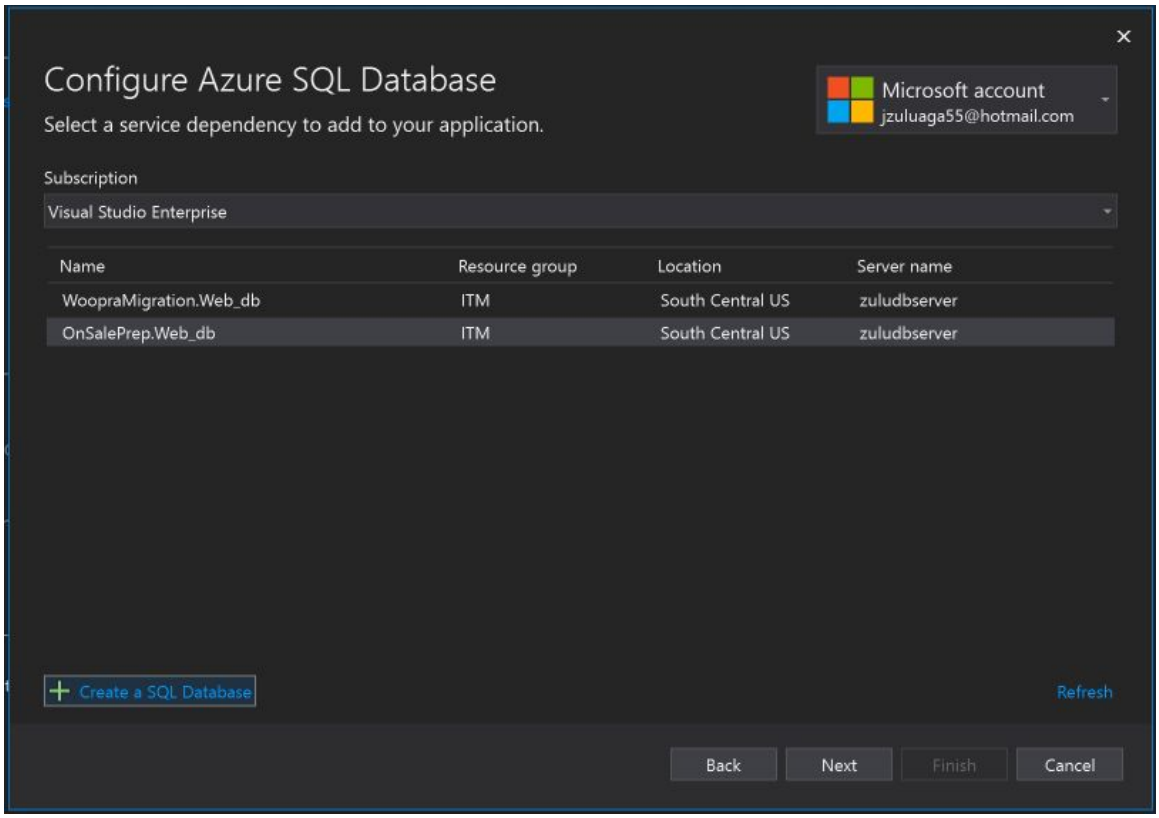

App Service (Windows)
Create new
Microsoft account
jzuluaga55@hotmail.com
Name
OnSalePrepWeb
Subscription
Visual Studio Enterprise
Resource group
OnSale (Central US)
New...
Hosting Plan
ZuluPlan (South Central US, B1)
New...
Export...
Create
Cancel

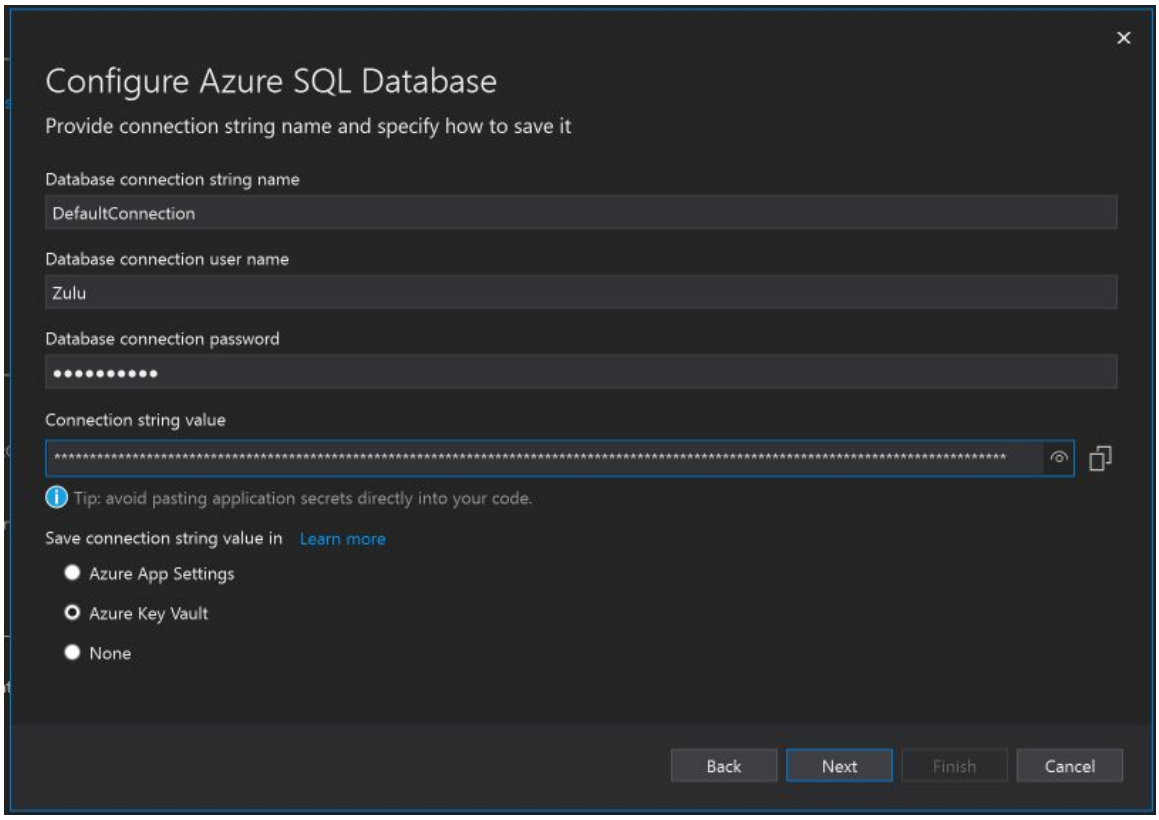

Publish
Select existing or create a new Azure App Service
Microsoft account
jzuluaga55@hotmail.com
Target
Specific target
App Service
Subscription
Visual Studio Enterprise
View
Resource group
Search
OnSalePrepWeb
OnSale
OnSalePrepWeb
Deployment Slots
Create a new Azure App Service...
Refresh
Back
Next
Finish
Cancel

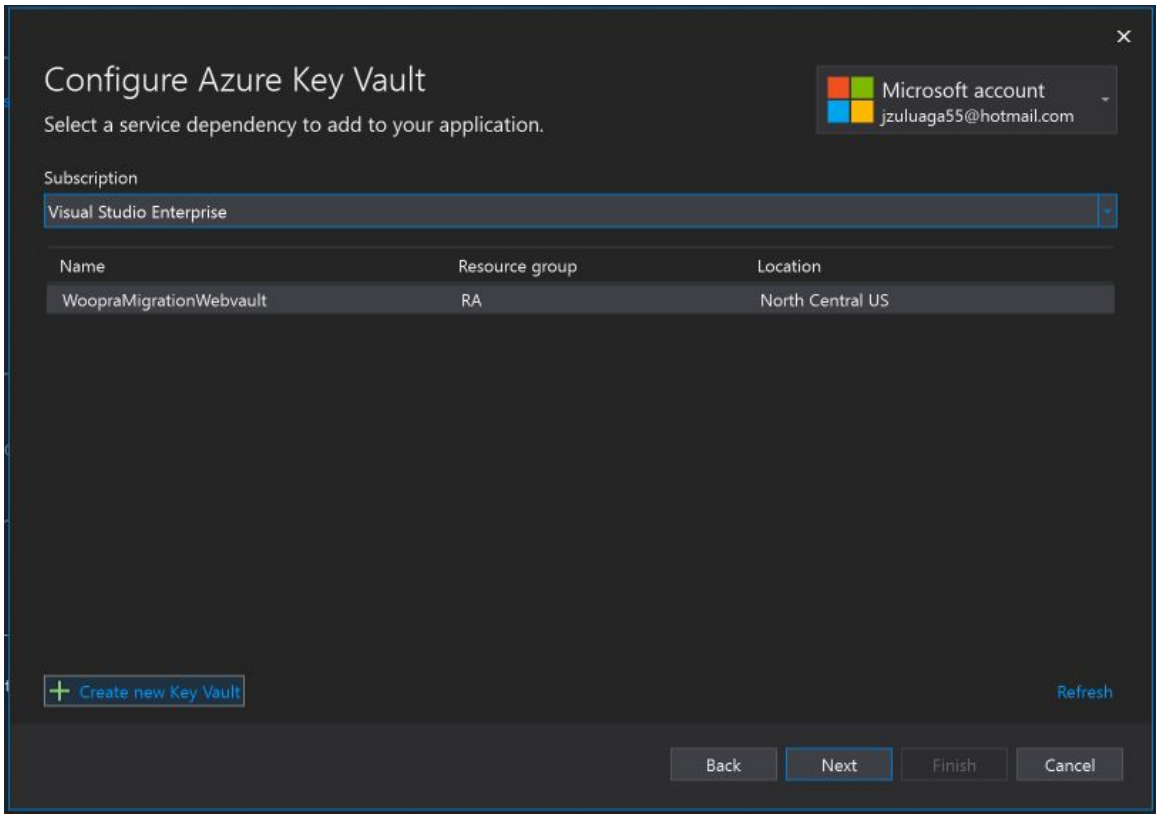

Configure dependency
Select a service dependency to add to your application.
Azure SQL Database
Intelligent, scalable, cloud database service that provides the broadest SQL Server engine compatibility.
SQL Server Database
On-premise SQL Server Database
Back
Next
Finish
Cancel

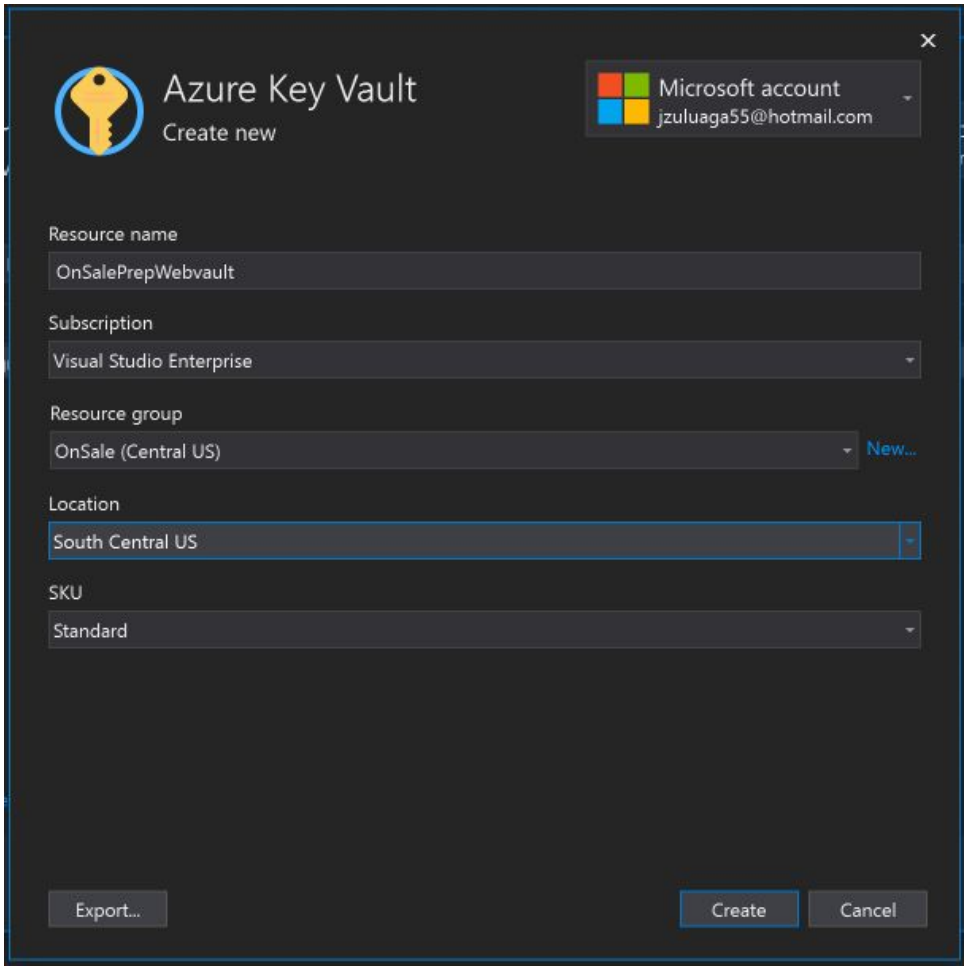

Configure Azure SQL Database
Select a service dependency to add to your application.
Microsoft account
jzuluaga55@hotmail.com
Subscription
Visual Studio Enterprise
Name
Resource group
Location
Server name
WoopraMigration.Web_db
ITM
South Central US
zuludbserver
Create a SQL Database
Refresh
Back
Next
Finish
Cancel

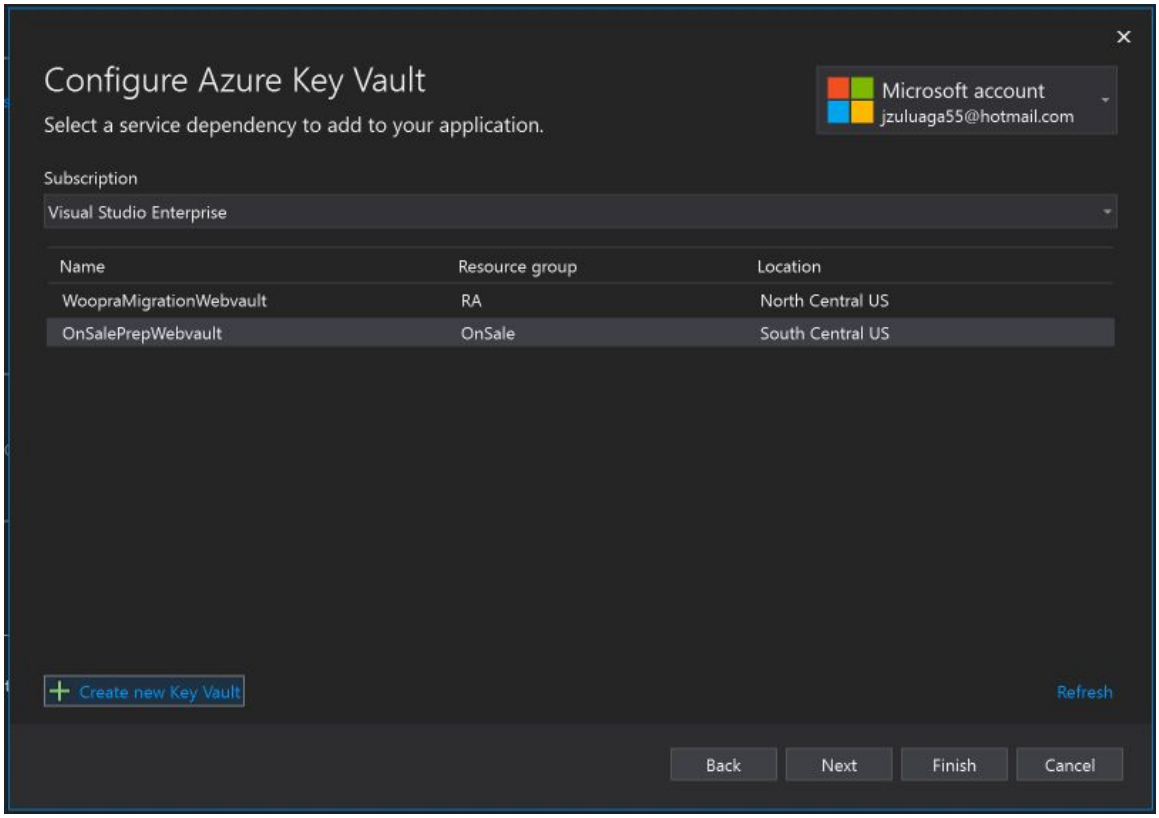

Azure SQL Database
Create new
Microsoft account
jzuluaga55@hotmail.com
Database name
OnSalePrep.Web_db
Subscription
Visual Studio Enterprise
Resource group
ITM (South Central US)
New...
Database server
zuludbserver (South Central US)
New...
Database administrator username (must have permissions to create)
Zulu
Database administrator password
Export...
Create
Cancel

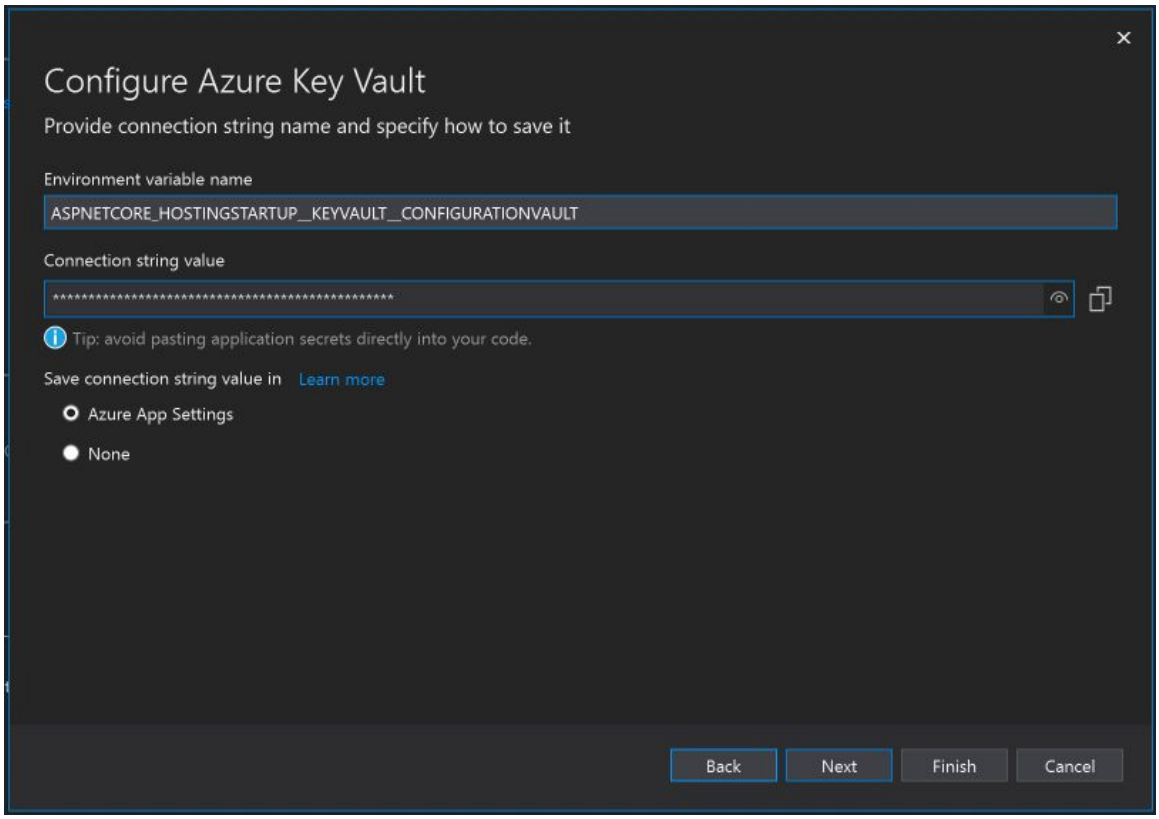

Configure Azure SQL Database
Select a service dependency to add to your application.
Microsoft account
jzuluaga55@hotmail.com
Subscription
Visual Studio Enterprise
Name
Resource group
Location
Server name
WoopraMigration.Web_db
ITM
South Central US
zuludbserver
OnSalePrep.Web_db
ITM
South Central US
zuludbserver
Create a SQL Database
Refresh
Back
Next
Finish
Cancel

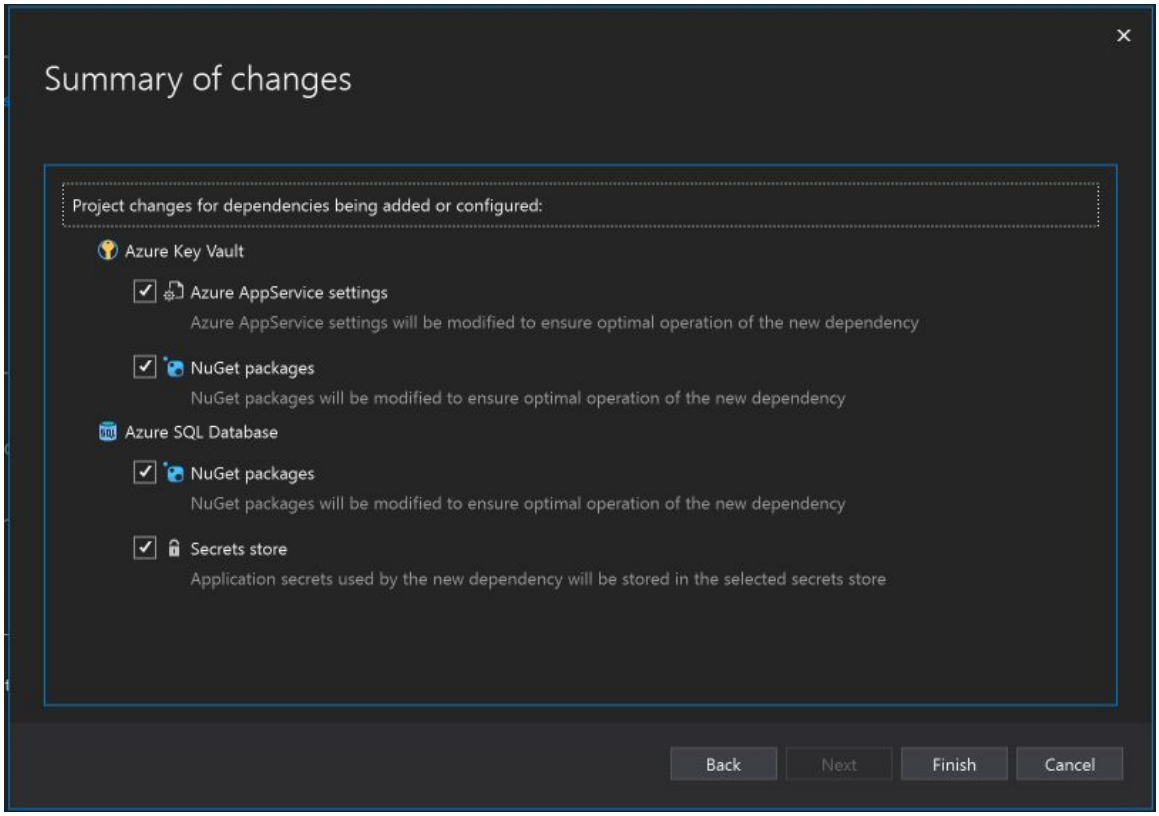

Configure Azure SQL Database
Provide connection string name and specify how to save it
Database connection string name
DefaultConnection
Database connection user name
Zulu
Database connection password
Connection string value
Tip: avoid pasting application secrets directly into your code.
Save connection string value in
Learn more
Azure App Settings
Azure Key Vault
None
Back
Next
Finish
Cancel

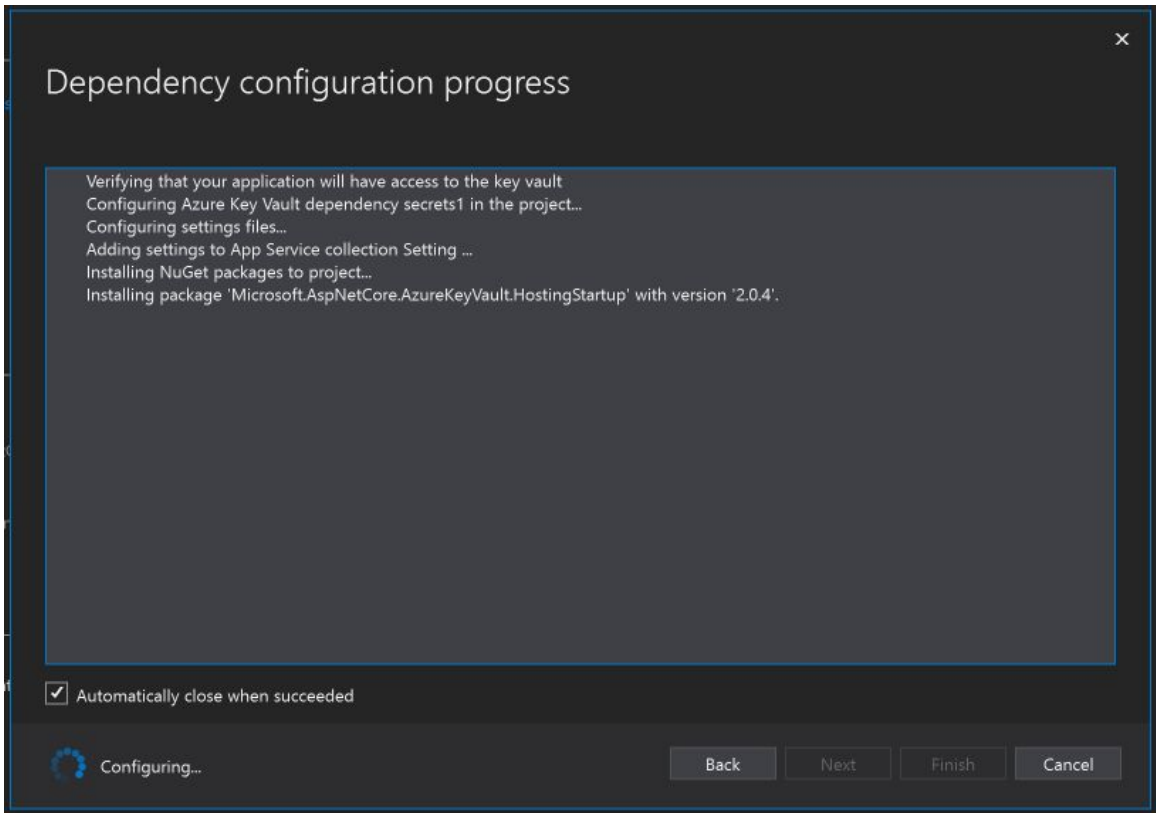

Configure Azure Key Vault
Select a service dependency to add to your application.
Microsoft account
jzuluaga55@hotmail.com
Subscription
Visual Studio Enterprise
Name
Resource group
Location
WoopraMigrationWebvault
RA
North Central US
Create new Key Vault
Refresh
Back
Next
Finish
Cancel


Azure Key Vault
Create new
Microsoft account
jzuluaga55@hotmail.com
Resource name
OnSalePrepWebvault
Subscription
Visual Studio Enterprise
Resource group
OnSale (Central US)
New...
Location
South Central US
SKU
Standard
Export...
Create
Cancel


Configure Azure Key Vault
Select a service dependency to add to your application.
Microsoft account
jzuluaga55@hotmail.com
Subscription
Visual Studio Enterprise
Name
Resource group
Location
WoopraMigrationWebvault
RA
North Central US
OnSalePrepWebvault
OnSale
South Central US
Create new Key Vault
Refresh
Back
Next
Finish
Cancel


Configure Azure Key Vault
Provide connection string name and specify how to save it
Environment variable name
ASPNETCORE_HOSTINGSTARTUP__KEYVAULT__CONFIGURATIONVAULT
Connection string value
Tip: avoid pasting application secrets directly into your code.
Save connection string value in
Learn more
Azure App Settings
None
Back
Next
Finish
Cancel


Summary of changes
Project changes for dependencies being added or configured:
Azure Key Vault
Azure AppService settings
Azure AppService settings will be modified to ensure optimal operation of the new dependency
NuGet packages
NuGet packages will be modified to ensure optimal operation of the new dependency
Azure SQL Database
NuGet packages
NuGet packages will be modified to ensure optimal operation of the new dependency
Secrets store
Application secrets used by the new dependency will be stored in the selected secrets store
Back
Next
Finish
Cancel


Dependency configuration progress
Verifying that your application will have access to the key vault
Configuring Azure Key Vault dependency secrets1 in the project...
Configuring settings files...
Adding settings to App Service collection Setting ...
Installing NuGet packages to project...
Installing package 'Microsoft.AspNetCore.AzureKeyVault.HostingStartup' with version '2.0.4'.
Automatically close when succeeded
Configuring...
Back
Next
Finish
Cancel

OnSalePrep.Web | EmailRequest.cs | AccountController.cs | AccountController.cs

Overview
Connected Services
Publish

Publish

Deploy your app to a folder, IIS, Azure, or another destination. More info

OnSalePrepWeb - Web Deploy | Publish

New Edit Rename Delete

Summary

Site URL	https://onsaleprepweb.azurewebsites.net
Resource group	OnSale
Configuration	Release
Target framework	netcoreapp2.1
Deployment mode	Framework-dependent
Target runtime	Portable

Actions

Preview changes
Manage in Cloud Explorer
Manage Azure App Service settings
Manage in Azure portal
View streaming logs
Open troubleshooting guide

Service Dependencies

Storage
Storage dependency with connectionId 'Blob/ConnectionString' is detected — Configure

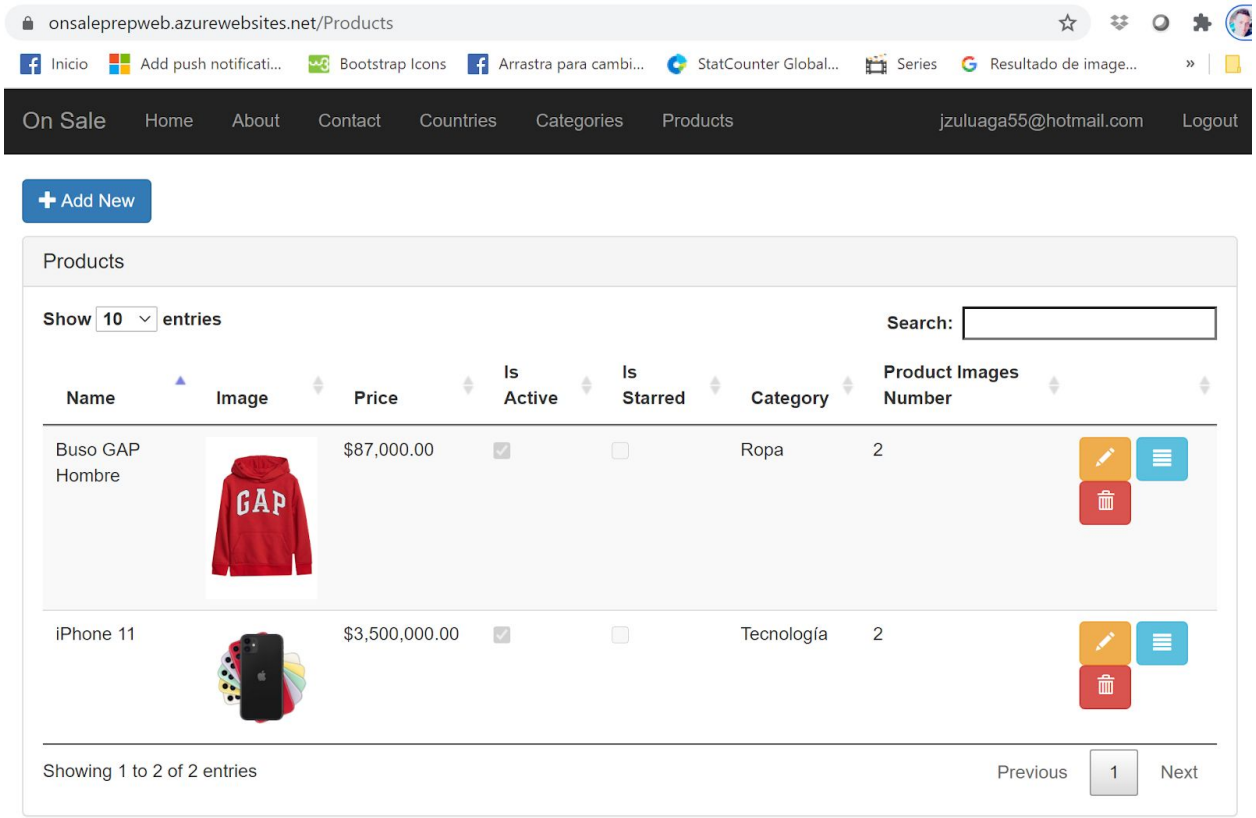
Azure Key Vault: OnSalePrepWebvault
Environment variable: ASPNETCORE_HOSTINGSTARTUP__KEYVAULT__CONFIGURATIONVAULT — Configured

Azure SQL Database: OnSalePrep.Web_db
Connection string name: DefaultConnection — Configured

Continuous delivery

Automatically publish your application to Azure with continuous delivery. Click Configure to begin setup.

Configure



Solución de Problemas

Tenemos un lío con la configuración regional, podemos dar una solución rápida con:

```
app.UseRequestLocalization(new RequestLocalizationOptions
{
    DefaultRequestCulture = new RequestCulture("en-US"),
    SupportedCultures = new[] { new CultureInfo("en-US") },
    SupportedUICultures = new[] { new CultureInfo("en-US") }
});
```

O podemos dar una solución más efectiva:

```
public class ProductViewModel : Product
{
    [Display(Name = "Category")]
    [Range(1, int.MaxValue, ErrorMessage = "You must select a category.")]
    [Required]
    public int CategoryId { get; set; }

    [Display(Name = "Price")]
    [MaxLength(12)]
```

```
        [RegularExpression(@"^\d+([\.\,]?\d+)?$", ErrorMessage = "Use only numbers and . or , to
put decimals")]
        [Required]
        public string PriceString { get; set; }

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

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

Luego:

```
<div class="form-group">
    <label asp-for="PriceString" class="control-label"></label>
    <input asp-for="PriceString" class="form-control" />
    <span asp-validation-for="PriceString" class="text-danger"></span>
</div>
```

Luego:

```
        public async Task<Product> ToProductAsync(ProductViewModel model, bool isNew)
        {
            return new Product
            {
                Category = await _context.Categories.FindAsync(model.CategoryId),
                Description = model.Description,
                Id = isNew ? 0 : model.Id,
                IsActive = model.IsActive,
                IsStarred = model.IsStarred,
                Name = model.Name,
                Price = ToPrice(model.PriceString),
                ProductImages = model.ProductImages
            };
        }

        private decimal ToPrice(string priceString)
        {
            string nds = CultureInfo.CurrentCulture.NumberFormat.NumberDecimalSeparator;
            if (nds == ".")
            {
                priceString = priceString.Replace(',', '.');

            }
            else
            {
                priceString = priceString.Replace('.', ',');
            }
```

```
        return decimal.Parse(priceString);
    }

    public ProductViewModel ToProductViewModel(Product product)
    {
        return new ProductViewModel
        {
            Categories = _combosHelper.GetComboCategories(),
            Category = product.Category,
            CategoryId = product.Category.Id,
            Description = product.Description,
            Id = product.Id,
            IsActive = product.IsActive,
            IsStarred = product.IsStarred,
            Name = product.Name,
            Price = product.Price,
            PriceString = $"{product.Price}",
            ProductImages = product.ProductImages
        };
    }
```

Probemos.

El otro error, es que tenemos un problema con la eliminación en cascada. Primero vamos a organizar nuestras relaciones en las entities:

```
public class City
{
    public int Id { get; set; }

    [MaxLength(50)]
    [Required]
    public string Name { get; set; }

    [JsonIgnore]
    [NotMapped]
    public int IdDepartment { get; set; }

    [JsonIgnore]
    public Department Department { get; set; }
}


public class Department
{
    public int Id { get; set; }
```

```
        [MaxLength(50)]
        [Required]
        public string Name { get; set; }

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

        [DisplayName("Cities Number")]
        public int CitiesNumber => Cities == null ? 0 : Cities.Count;

        [JsonIgnore]
        [NotMapped]
        public int IdCountry { get; set; }

        [JsonIgnore]
        public Country Country { get; set; }
    }
```

Esto para hacer las relaciones en el datacontext de manera explícita y así poder definir el borrado en cascada:

```
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<Category>()
                .HasIndex(t => t.Name)
                .IsUnique();

            modelBuilder.Entity<Country>(cou =>
            {
                cou.HasIndex("Name").IsUnique();
                cou.HasMany(c => c.Departments).WithOne(d =>
d.Country).OnDelete(DeleteBehavior.Cascade);
            });

            modelBuilder.Entity<Department>(dep =>
            {
                dep.HasIndex("Name", "CountryId").IsUnique();
                dep.HasOne(d => d.Country).WithMany(c =>
c.Departments).OnDelete(DeleteBehavior.Cascade);
            });

            modelBuilder.Entity<City>(cit =>
            {
                cit.HasIndex("Name", "DepartmentId").IsUnique();
                cit.HasOne(c => c.Department).WithMany(d =>
d.Cities).OnDelete(DeleteBehavior.Cascade);
            });
```

```
        modelBuilder.Entity<Product>()
            .HasIndex(t => t.Name)
            .IsUnique();
    }
```

El tercer problema tiene que ver con el Internet Explorer pero….

Redirect Pages

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

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

2. Create correspondent view with this lines:

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

<br />
<br />
<img src="~/images/gopher_head-min.png" />
<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. 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?}");
        });
    }
```

5. Inside the **HomeController** create the following action.

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

6. Create the correspondent view.

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

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

7. Test it!.

Self-registration of users

1. Add the **EditUserViewModel**:

```
public class EditUserViewModel
{
    public string Id { get; set; }

    [MaxLength(20)]
    [Required]
    public string Document { get; set; }

    [Display(Name = "First Name")]
    [MaxLength(50)]
    [Required]
    public string FirstName { get; set; }
```

```
    [Display(Name = "Last Name")]
    [MaxLength(50)]
    [Required]
    public string LastName { get; set; }

    [MaxLength(100)]
    public string Address { get; set; }

    [Display(Name = "Phone Number")]
    [MaxLength(20)]
    public string PhoneNumber { get; set; }

    [Display(Name = "Image")]
    public Guid ImageId { get; set; }

    [Display(Name = "Image")]
    public string ImageFullPath => ImageId == Guid.Empty
        ? $"https://OnSaleweb.azurewebsites.net/images/noimage.png"
        : $"https://onsale.blob.core.windows.net/users/{ImageId}";

    [Display(Name = "Image")]
    public IFormFile ImageFile { get; set; }

    [Required]
    [Display(Name = "Country")]
    [Range(1, int.MaxValue, ErrorMessage = "You must select a country.")]
    public int CountryId { get; set; }

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

    [Required]
    [Display(Name = "Department")]
    [Range(1, int.MaxValue, ErrorMessage = "You must select a department.")]
    public int DepartmentId { get; set; }

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

    [Required]
    [Display(Name = "City")]
    [Range(1, int.MaxValue, ErrorMessage = "You must select a city.")]
    public int CityId { get; set; }
```

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

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

```
public class AddUserViewModel : EditUserViewModel
{
    [Display(Name = "Email")]
    [Required(ErrorMessage = "The field {0} is mandatory.")]
    [MaxLength(100, ErrorMessage = "The {0} field can not have more than {1} characters.")]
    [EmailAddress]
    public string Username { get; set; }

    [Display(Name = "Password")]
    [Required(ErrorMessage = "The field {0} is mandatory.")]
    [DataType(DataType.Password)]
    [StringLength(20, MinimumLength = 6, ErrorMessage = "The {0} field must contain between
{2} and {1} characters.")]
    public string Password { get; set; }

    [Display(Name = "Password Confirm")]
    [Required(ErrorMessage = "The field {0} is mandatory.")]
    [DataType(DataType.Password)]
    [StringLength(20, MinimumLength = 6, ErrorMessage = "The {0} field must contain between
{2} and {1} characters.")]
    [Compare("Password")]
    public string PasswordConfirm { get; set; }
}
```

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

```
Task<User> AddUserAsync(AddUserViewModel model, Guid imageId, UserType userType);
```

4. Add this method to **UserHelper**:

```
public async Task<User> AddUserAsync(AddUserViewModel model, Guid imageId, UserType
userType)
{
    User user = new User
    {
        Address = model.Address,
        Document = model.Document,
        Email = model.Username,
```

```
        FirstName = model.FirstName,
        LastName = model.LastName,
        ImageId = imageId,
        PhoneNumber = model.PhoneNumber,
        City = await _context.Cities.FindAsync(model.CityId),
        UserName = model.Username,
        UserType = userType
    };

    IdentityResult result = await _userManager.CreateAsync(user, model.Password);
    if (result != IdentityResult.Success)
    {
        return null;
    }

    User newUser = await GetUserAsync(model.Username);
    await AddUserToRoleAsync(newUser, user.UserType.ToString());
    return newUser;
}
```

5. Add those methods to **ICombosHelper**:

```
IEnumerable<SelectListItem> GetComboCountries();

IEnumerable<SelectListItem> GetComboDepartments(int countryId);

IEnumerable<SelectListItem> GetComboCities(int departmentId);
```

6. Add those methods to **CombosHelper**:

```
public IEnumerable<SelectListItem> GetComboCities(int departmentId)
{
    List<SelectListItem> list = new List<SelectListItem>();
    Department department = _context.Departments
        .Include(d => d.Cities)
        .FirstOrDefault(d => d.Id == departmentId);
    if (department != null)
    {
        list = department.Cities.Select(t => new SelectListItem
        {
            Text = t.Name,
            Value = $"{t.Id}"
        })
```

```
            .OrderBy(t => t.Text)
            .ToList();
    }

    list.Insert(0, new SelectListItem
    {
        Text = "[Select a city...]",
        Value = "0"
    });

    return list;
}

public IEnumerable<SelectListItem> GetComboCountries()
{
    List<SelectListItem> list = _context.Countries.Select(t => new SelectListItem
    {
        Text = t.Name,
        Value = $"{t.Id}"
    })
        .OrderBy(t => t.Text)
        .ToList();

    list.Insert(0, new SelectListItem
    {
        Text = "[Select a country...]",
        Value = "0"
    });

    return list;
}

public IEnumerable<SelectListItem> GetComboDepartments(int countryId)
{
    List<SelectListItem> list = new List<SelectListItem>();
    Country country = _context.Countries
        .Include(c => c.Departments)
        .FirstOrDefault(c => c.Id == countryId);
    if (country != null)
    {
        list = country.Departments.Select(t => new SelectListItem
        {
            Text = t.Name,
```

```
            Value = $"{t.Id}"
        })
            .OrderBy(t => t.Text)
            .ToList();
    }

    list.Insert(0, new SelectListItem
    {
        Text = "[Select a department...]",
        Value = "0"
    });

    return list;
}
```

7. Modify the **AccountController**:

```
public class AccountController : Controller
{
    private readonly DataContext _context;
    private readonly IUserHelper _userHelper;
    private readonly ICombosHelper _combosHelper;
    private readonly IBlobHelper _blobHelper;

    public AccountController(
        DataContext context,
        IUserHelper userHelper,
        ICombosHelper combosHelper,
        IBlobHelper blobHelper)
    {
        _context = context;
        _userHelper = userHelper;
        _combosHelper = combosHelper;
        _blobHelper = blobHelper;
    }

    public IActionResult Login()
    {
        if (User.Identity.IsAuthenticated)
        {
            return RedirectToAction("Index", "Home");
        }
```

```
        return View(new LoginViewModel());
    }

    [HttpPost]
    public async Task<IActionResult> Login(LoginViewModel model)
    {
        if (ModelState.IsValid)
        {
            Microsoft.AspNetCore.Identity.SignInResult result = await
_userHelper.LoginAsync(model);
            if (result.Succeeded)
            {
                if (Request.Query.Keys.Contains("ReturnUrl"))
                {
                    return Redirect(Request.Query["ReturnUrl"].First());
                }

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

            ModelState.AddModelError(string.Empty, "Email or password incorrect.");
        }

        return View(model);
    }

    public async Task<IActionResult> Logout()
    {
        await _userHelper.LogoutAsync();
        return RedirectToAction("Index", "Home");
    }

    public IActionResult NotAuthorized()
    {
        return View();
    }

    public IActionResult Register()
    {
        AddUserViewModel model = new AddUserViewModel
        {
            Countries = _combosHelper.GetComboCountries(),
            Departments = _combosHelper.GetComboDepartments(0),
```

```
            Cities = _combosHelper.GetComboCities(0),
        };

        return View(model);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Register(AddUserViewModel model)
    {
        if (ModelState.IsValid)
        {
            Guid imageId = Guid.Empty;

            if (model.ImageFile != null)
            {
                imageId = await _blobHelper.UploadBlobAsync(model.ImageFile, "users");
            }

            User user = await _userHelper.AddUserAsync(model, imageId, UserType.User);
            if (user == null)
            {
                ModelState.AddModelError(string.Empty, "This email is already used.");
                model.Countries = _combosHelper.GetComboCountries();
                model.Departments = _combosHelper.GetComboDepartments(model.CountryId);
                model.Cities = _combosHelper.GetComboCities(model.DepartmentId);
                return View(model);
            }

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

            var result2 = await _userHelper.LoginAsync(loginViewModel);

            if (result2.Succeeded)
            {
                return RedirectToAction("Index", "Home");
            }
        }
```

```
        model.Countries = _combosHelper.GetComboCountries();
        model.Departments = _combosHelper.GetComboDepartments(model.CountryId);
        model.Cities = _combosHelper.GetComboCities(model.DepartmentId);
        return View(model);
    }


    public JsonResult GetDepartments(int countryId)
    {
        Country country = _context.Countries
            .Include(c => c.Departments)
            .FirstOrDefault(c => c.Id == countryId);
        if (country == null)
        {
            return null;
        }

        return Json(country.Departments.OrderBy(d => d.Name));
    }

    public JsonResult GetCities(int departmentId)
    {
        Department department = _context.Departments
            .Include(d => d.Cities)
            .FirstOrDefault(d => d.Id == departmentId);
        if (department == null)
        {
            return null;
        }

        return Json(department.Cities.OrderBy(c => c.Name));
    }
}
```

8. Add the partial view **_User** on **AccountController**:

```
@model OnSale.Web.Models.EditUserViewModel

<div class="form-group">
    <label asp-for="Document" class="control-label"></label>
    <input asp-for="Document" class="form-control" />
    <span asp-validation-for="Document" class="text-danger"></span>
```

```
</div>

<div class="form-group">
    <label asp-for="FirstName" class="control-label"></label>
    <input asp-for="FirstName" class="form-control" />
    <span asp-validation-for="FirstName" class="text-danger"></span>
</div>

<div class="form-group">
    <label asp-for="LastName" class="control-label"></label>
    <input asp-for="LastName" class="form-control" />
    <span asp-validation-for="LastName" class="text-danger"></span>
</div>

<div class="form-group">
    <label asp-for="Address" class="control-label"></label>
    <input asp-for="Address" class="form-control" />
    <span asp-validation-for="Address" class="text-danger"></span>
</div>

<div class="form-group">
    <label asp-for="PhoneNumber" class="control-label"></label>
    <input asp-for="PhoneNumber" class="form-control" />
    <span asp-validation-for="PhoneNumber" 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="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="DepartmentId" class="control-label"></label>
    <select asp-for="DepartmentId" asp-items="Model.Departments"
class="form-control"></select>
    <span asp-validation-for="DepartmentId" 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>
```

9. Add the view **Register** on **AccountController**:

```
@model OnSale.Web.Models.AddUserViewModel

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

<h2>Register</h2>

<h4>User</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <form asp-action="Register" enctype="multipart/form-data">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>

            <div class="form-group">
                <label asp-for="Username" class="control-label"></label>
                <input asp-for="Username" class="form-control" />
                <span asp-validation-for="Username" class="text-danger"></span>
            </div>

            <partial name="_User" />

            <div class="form-group">
                <label asp-for="Password" class="control-label"></label>
                <input asp-for="Password" class="form-control" />
                <span asp-validation-for="Password" class="text-danger"></span>
            </div>

            <div class="form-group">
                <label asp-for="PasswordConfirm" class="control-label"></label>
                <input asp-for="PasswordConfirm" class="form-control" />
                <span asp-validation-for="PasswordConfirm" class="text-danger"></span>
```

```
            </div>

            <div class="form-group">
                <input type="submit" value="Register" class="btn btn-primary" />
            </div>
        </form>
    </div>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
    <script type="text/javascript">
        $(document).ready(function () {
            $("#CountryId").change(function () {
                $("#DepartmentId").empty();
                $("#DepartmentId").append('<option value="0">[Select a department...]</option>');
                $("#CityId").empty();
                $("#CityId").append('<option value="0">[Select a city...]</option>');
                $.ajax({
                    type: 'POST',
                    url: '@Url.Action("GetDepartments")',
                    dataType: 'json',
                    data: { countryId: $("#CountryId").val() },
                    success: function (subcategories) {
                        $.each(subcategories, function (i, department) {
                            debugger;
                            $("#DepartmentId").append('<option value="'
                                + department.id + '">'
                                + department.name + '</option>');
                        });
                    },
                    error: function (ex) {
                        alert('Failed to retrieve departments.' + ex);
                    }
                });
                return false;
            })

            $("#DepartmentId").change(function () {
                $("#CityId").empty();
                $("#CityId").append('<option value="0">[Select a city...]</option>');
                $.ajax({
                    type: 'POST',
```

```
                    url: '@Url.Action("GetCities")',
                    dataType: 'json',
                    data: { departmentId: $("#DepartmentId").val() },
                    success: function (cities) {
                        $.each(cities, function (i, city) {
                            debugger;
                            $("#CityId").append('<option value="'
                                + city.id + '">'
                                + city.name + '</option>');
                        });
                    },
                    error: function (ex) {
                        alert('Failed to retrieve cities.' + ex);
                    }
                });
                return false;
            })
        });
    </script>
}
```

10. Test it.

Modifying users

1. Add the **ChangePasswordViewModel** class:

```
public class ChangePasswordViewModel
{
    [Display(Name = "Current password")]
    [Required(ErrorMessage = "The field {0} is mandatory.")]
    [DataType(DataType.Password)]
    [StringLength(20, MinimumLength = 6, ErrorMessage = "The {0} field must contain between
{2} and {1} characters.")]
    public string OldPassword { get; set; }

    [Display(Name = "New password")]
    [Required(ErrorMessage = "The field {0} is mandatory.")]
    [DataType(DataType.Password)]
    [StringLength(20, MinimumLength = 6, ErrorMessage = "The {0} field must contain between
{2} and {1} characters.")]
    public string NewPassword { get; set; }
```

```
    [Display(Name = "Password confirm")]
    [Required(ErrorMessage = "The field {0} is mandatory.")]
    [DataType(DataType.Password)]
    [StringLength(20, MinimumLength = 6, ErrorMessage = "The {0} field must contain between
{2} and {1} characters.")]
    [Compare("NewPassword")]
    public string Confirm { get; set; }
}
```

2. Add those methods in **IUserHelper** interface:

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

Task<IdentityResult> UpdateUserAsync(User user);

Task<User> GetUserAsync(Guid userId);
```

3. Add the implementation in **UserHelper** class:

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

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

public async Task<User> GetUserAsync(Guid userId)
{
    return await _context.Users
        .Include(u => u.City)
        .FirstOrDefaultAsync(u => u.Id == userId.ToString());
}
```

4. Add those methods to **AccountController** class:

```
public async Task<IActionResult> ChangeUser()
{
```

```
    User user = await _userHelper.GetUserAsync(User.Identity.Name);
    if (user == null)
    {
        return NotFound();
    }

    Department department = await _context.Departments.FirstOrDefaultAsync(d =>
d.Cities.FirstOrDefault(c => c.Id == user.City.Id) != null);
    if (department == null)
    {
        department = await _context.Departments.FirstOrDefaultAsync();
    }

    Country country = await _context.Countries.FirstOrDefaultAsync(c =>
c.Departments.FirstOrDefault(d => d.Id == department.Id) != null);
    if (country == null)
    {
        country = await _context.Countries.FirstOrDefaultAsync();
    }

    EditUserViewModel model = new EditUserViewModel
    {
        Address = user.Address,
        FirstName = user.FirstName,
        LastName = user.LastName,
        PhoneNumber = user.PhoneNumber,
        ImageId = user.ImageId,
        Cities = _combosHelper.GetComboCities(department.Id),
        CityId = user.City.Id,
        Countries = _combosHelper.GetComboCountries(),
        CountryId = country.Id,
        DepartmentId = department.Id,
        Departments = _combosHelper.GetComboDepartments(country.Id),
        Id = user.Id,
        Document = user.Document
    };

    return View(model);
}

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> ChangeUser(EditUserViewModel model)
```

```
{
    if (ModelState.IsValid)
    {
        Guid imageId = model.ImageId;

        if (model.ImageFile != null)
        {
            imageId = await _blobHelper.UploadBlobAsync(model.ImageFile, "users");
        }

        User user = await _userHelper.GetUserAsync(User.Identity.Name);

        user.FirstName = model.FirstName;
        user.LastName = model.LastName;
        user.Address = model.Address;
        user.PhoneNumber = model.PhoneNumber;
        user.ImageId = imageId;
        user.City = await _context.Cities.FindAsync(model.CityId);
        user.Document = model.Document;

        await _userHelper.UpdateUserAsync(user);
        return RedirectToAction("Index", "Home");
    }

    model.Cities = _combosHelper.GetComboCities(model.DepartmentId);
    model.Countries = _combosHelper.GetComboCountries();
    model.Departments = _combosHelper.GetComboDepartments(model.CityId);
    return View(model);
}
```

5. Add the view **ChangeUser** in **AccountController**:

```
@model OnSale.Web.Models.EditUserViewModel
@{
    ViewData["Title"] = "Edit";
}

<h2>Edit</h2>

<h4>User</h4>
<hr />
<div class="row">
    <div class="col-md-6">
```

```
        <form asp-action="ChangeUser" enctype="multipart/form-data">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <input type="hidden" asp-for="Id" />
            <input type="hidden" asp-for="ImageId" />

            <partial name="_User" />

            <div class="form-group">
                <input type="submit" value="Save" class="btn btn-primary" />
                <a asp-action="ChangePassword" class="btn btn-warning">Change Password</a>
            </div>
        </form>
    </div>
    <div class="col-md-4">
        <img src="@Model.ImageFullPath" style="width:250px;height:250px;border-radius:50%" />
    </div>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
    <script type="text/javascript">
        $(document).ready(function () {
            $("#CountryId").change(function () {
                $("#DepartmentId").empty();
                $("#DepartmentId").append('<option value="0">[Select a department...]</option>');
                $("#CityId").empty();
                $("#CityId").append('<option value="0">[Select a city...]</option>');
                $.ajax({
                    type: 'POST',
                    url: '@Url.Action("GetDepartments")',
                    dataType: 'json',
                    data: { countryId: $("#CountryId").val() },
                    success: function (subcategories) {
                        $.each(subcategories, function (i, department) {
                            debugger;
                            $("#DepartmentId").append('<option value="'
                                + department.id + '">'
                                + department.name + '</option>');
                        });
                    },
                    error: function (ex) {
                        alert('Failed to retrieve departments.' + ex);
                    }
```

```
            });
            return false;
        })

        $("#DepartmentId").change(function () {
            $("#CityId").empty();
            $("#CityId").append('<option value="0">[Select a city...]</option>');
            $.ajax({
                type: 'POST',
                url: '@Url.Action("GetCities")',
                dataType: 'json',
                data: { departmentId: $("#DepartmentId").val() },
                success: function (cities) {
                    $.each(cities, function (i, city) {
                        debugger;
                        $("#CityId").append('<option value="'
                            + city.id + '">'
                            + city.name + '</option>');
                    });
                },
                error: function (ex) {
                    alert('Failed to retrieve cities.' + ex);
                }
            });
            return false;
        })
    });
  </script>
}
```

6. Test it.

7. Add those methods to **AccountController** class:

```
public IActionResult ChangePassword()
{
        return View();
}

[HttpPost]
public async Task<IActionResult> ChangePassword(ChangePasswordViewModel model)
{
        if (ModelState.IsValid)
```

```
        {
        var user = await _userHelper.GetUserAsync(User.Identity.Name);
        if (user != null)
        {
        var result = await _userHelper.ChangePasswordAsync(user, model.OldPassword,
model.NewPassword);
        if (result.Succeeded)
        {
                return RedirectToAction("ChangeUser");
        }
        else
        {
                ModelState.AddModelError(string.Empty,
result.Errors.FirstOrDefault().Description);
        }
        }
        else
        {
        ModelState.AddModelError(string.Empty, "User no found.");
        }
        }

        return View(model);
}
```

8. Add the view **ChangePassword** to **AccountController** class:

```
@model OnSale.Web.Models.ChangePasswordViewModel
@{
        ViewData["Title"] = "Register";
}

<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>

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

9. Test it.

Confirm Email Registration

1. First, change the setup project:

```
services.AddIdentity<UserEntity, 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 and https://accounts.google.com/DisplayUnlockCaptcha.

3. Add this parameters to the configuration file:

```
"Mail": {
  "From": "onsalezulu@gmail.com",
  "Smtp": "smtp.gmail.com",
  "Port": 587,
  "Password": "Zulu1234."
}
```

4. Add the nuget "**Mailkit**".

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

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

6. In **Helpers** folder add the implementation **MailHelper**:

```
public class MailHelper : IMailHelper
{
    private readonly IConfiguration _configuration;

    public MailHelper(IConfiguration configuration)
    {
        _configuration = configuration;
    }

    public Response SendMail(string to, string subject, string body)
    {
        try
        {
            string from = _configuration["Mail:From"];
            string smtp = _configuration["Mail:Smtp"];
```

```
            string port = _configuration["Mail:Port"];
            string password = _configuration["Mail:Password"];

            MimeMessage message = new MimeMessage();
            message.From.Add(new MailboxAddress(from));
            message.To.Add(new MailboxAddress(to));
            message.Subject = subject;
            BodyBuilder bodyBuilder = new BodyBuilder
            {
                HtmlBody = body
            };
            message.Body = bodyBuilder.ToMessageBody();

            using (SmtpClient client = new SmtpClient())
            {
                client.Connect(smtp, int.Parse(port), false);
                client.Authenticate(from, password);
                client.Send(message);
                client.Disconnect(true);
            }

            return new Response { IsSuccess = true };

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

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);
```

And the implementation:

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

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

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

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Register(AddUserViewModel model)
{
   if (ModelState.IsValid)
   {
      Guid imageId = Guid.Empty;

      if (model.ImageFile != null)
      {
         imageId = await _blobHelper.UploadBlobAsync(model.ImageFile, "users");
      }

      User user = await _userHelper.AddUserAsync(model, imageId, UserType.User);
      if (user == null)
      {
         ModelState.AddModelError(string.Empty, "This email is already used.");
         model.Countries = _combosHelper.GetComboCountries();
         model.Departments = _combosHelper.GetComboDepartments(model.CountryId);
         model.Cities = _combosHelper.GetComboCities(model.DepartmentId);
         return View(model);
      }

      string myToken = await _userHelper.GenerateEmailConfirmationTokenAsync(user);
      string tokenLink = Url.Action("ConfirmEmail", "Account", new
      {
```

```
            userid = user.Id,
            token = myToken
        }, protocol: HttpContext.Request.Scheme);

        Response response = _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>");
        if (response.IsSuccess)
        {
            ViewBag.Message = "The instructions to allow your user has been sent to email.";
            return View(model);
        }

        ModelState.AddModelError(string.Empty, response.Message);
    }

    model.Countries = _combosHelper.GetComboCountries();
    model.Departments = _combosHelper.GetComboDepartments(model.CountryId);
    model.Cities = _combosHelper.GetComboCities(model.DepartmentId);
    return View(model);
}
```

10. Add this to the register view ends:

```
<div class="text-success">
    <p>
        @ViewBag.Message
    </p>
</div>
```

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 NotFound();
    }

    User user = await _userHelper.GetUserAsync(new Guid(userId));
    if (user == null)
    {
```

```
        return NotFound();
    }

    IdentityResult result = await _userHelper.ConfirmEmailAsync(user, token);
    if (!result.Succeeded)
    {
        return 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:

```
private async Task<User> CheckUserAsync(
    string document,
    string firstName,
    string lastName,
    string email,
    string phone,
    string address,
    UserType userType)
{
    User user = await _userHelper.GetUserAsync(email);
    if (user == null)
    {
        user = new User
```

```
        {
            FirstName = firstName,
            LastName = lastName,
            Email = email,
            UserName = email,
            PhoneNumber = phone,
            Address = address,
            Document = document,
            City = _context.Cities.FirstOrDefault(),
            UserType = userType
        };

        await _userHelper.AddUserAsync(user, "123456");
        await _userHelper.AddUserToRoleAsync(user, userType.ToString());

        string token = await _userHelper.GenerateEmailConfirmationTokenAsync(user);
        await _userHelper.ConfirmEmailAsync(user, token);
    }

    return user;
}
```

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 **RecoverPasswordViewModel**:

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

3. Add the model **ResetPasswordViewModel**:

```
public class ResetPasswordViewModel
{
    [Required]
    [EmailAddress]
    public string UserName { get; set; }

    [Required]
    [StringLength(20, MinimumLength = 6, ErrorMessage = "The {0} field must contain between
{2} and {1} characters.")]
    [DataType(DataType.Password)]
    public string Password { get; set; }

    [Required]
    [StringLength(20, MinimumLength = 6, ErrorMessage = "The {0} field must contain between
{2} and {1} characters.")]
    [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 _userManager.GeneratePasswordResetTokenAsync(user);
}

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

```
}
```

5. Add this methods to account controller:

```
public IActionResult RecoverPassword()
{
    return View();
}

[HttpPost]
public async Task<IActionResult> RecoverPassword(RecoverPasswordViewModel model)
{
    if (ModelState.IsValid)
    {
        User user = await _userHelper.GetUserAsync(model.Email);
        if (user == null)
        {
            ModelState.AddModelError(string.Empty, "The email doesn't correspont to a registered
user.");
            return View(model);
        }

        string myToken = await _userHelper.GeneratePasswordResetTokenAsync(user);
        string link = Url.Action(
            "ResetPassword",
            "Account",
            new { token = myToken }, protocol: HttpContext.Request.Scheme);
        _mailHelper.SendMail(model.Email, "Password Reset", $"<h1>Password Reset</h1>" +
            $"To reset the password click in this link:</br></br>" +
            $"<a href = \"{link}\">Reset Password</a>");
        ViewBag.Message = "The instructions to recover your password has been sent to email.";
        return View();

    }

    return View(model);
}

public IActionResult ResetPassword(string token)
{
    return View();
}
```

```
[HttpPost]
public async Task<IActionResult> ResetPassword(ResetPasswordViewModel model)
{
    User user = await _userHelper.GetUserAsync(model.UserName);
    if (user != null)
    {
        IdentityResult result = await _userHelper.ResetPasswordAsync(user, model.Token,
model.Password);
        if (result.Succeeded)
        {
            ViewBag.Message = "Password reset successful.";
            return View();
        }

        ViewBag.Message = "Error while resetting the password.";
        return View(model);
    }

    ViewBag.Message = "User not found.";
    return View(model);
}
```

6. Add the view:

```
@model OnSale.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 OnSale.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 the Azure DB, recreate and re-publish the solution.