PROYECTO ASP.Net parte 3

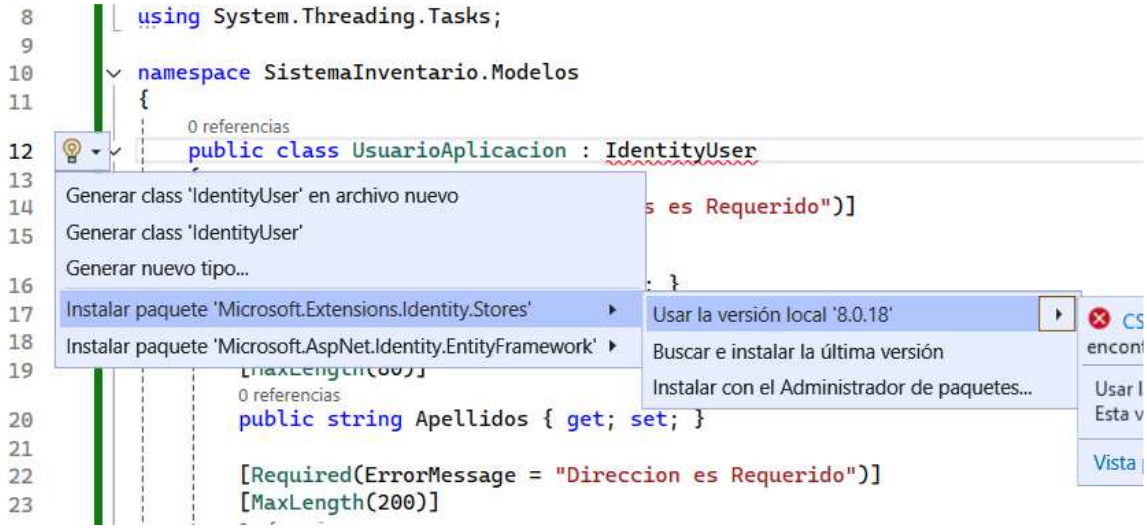
Modelo Usuario

1. Agregar más propiedades a la Tabla Usuarios

- En SistemaInventario.Modelos creamos un nuevo modelo llamado UsuarioAplicacion.cs que herede de IdentityUser

```
using Microsoft.AspNetCore.Identity;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SistemaInventario.Modelos
{
   public class UsuarioAplicacion : IdentityUser
   {
      [Required(ErrorMessage = "Nombres es Requerido")]
      [MaxLength(80)]
      public string Nombres { get; set; }

      [Required(ErrorMessage = "Apellidos es Requerido")]
      [MaxLength(80)]
      public string Apellidos { get; set; }

      [Required(ErrorMessage = "Direccion es Requerido")]
      [MaxLength(200)]
      public string Direccion { get; set; }

      [Required(ErrorMessage = "Ciudad es Requerido")]
      [MaxLength(60)]
      public string Ciudad { get; set; }

      [Required(ErrorMessage = "Pais es Requerido")]
      [MaxLength(60)]
      public string Pais { get; set; }

      [NotMapped]  // No se agrega a la tabla
      public string Role { get; set; }
   }
}
```

- Corregir el error de IdentityUser instalando el paquete



- Agregamos el modelo UsuarioAplicacion a SistemaInventario.AccesoDatos/Data/ApplicationDbContext.cs

```
public DbSet<UsuarioAplicacion> UsuarioAplicacion { get; set; }
```

- Realizamos migración
  - add-migration AgregarCamposTablaUsuariosMigracion
  - update-database

2. Usuario Aplicacion Repositorio

- En SistemaInventario.AccesoDatos/Repositorio/IRepositorio le hacemos una copia a IBodegaRepositorio.cs y le cambiamos el nombre a IUsuarioAplicacionRepositorio.cs

```
using SistemaInventario.Modelos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SistemaInventario.AccesoDatos.Repositorio.IRepositorio
{
   public interface IUsuarioAplicacionRepositorio : IRepositorio<UsuarioAplicacion>
   {
   }
}
```

- En SistemaInventario.AccesoDatos/Repositorio le hacemos una copia a BodegaRepositorio.cs y le cambiamos el nombre a UsuarioAplicacionRepositorio.cs

```
using SistemaInventario.AccesoDatos.Data;
using SistemaInventario.AccesoDatos.Repositorio.IRepositorio;
using SistemaInventario.Modelos;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SistemaInventario.AccesoDatos.Repositorio
{
   public class UsuarioAplicacionRepositorio : Repositorio<UsuarioAplicacion>, IUsuarioAplicacionRepositorio
   {

      private readonly ApplicationDbContext _db;

      public UsuarioAplicacionRepositorio(ApplicationDbContext db) : base(db)
      {
         _db = db;
      }
   }
}
```

- En SistemaInventario.AccesoDatos/Repositorio/IRepositorio modificamos IUnidadTrabajo.cs

```
IUsuarioAplicacionRepositorio UsuarioAplicacion { get; }
```

- En SistemaInventario.AccesoDatos/Repositorio/Repositorio modificamos UnidadTrabajo.cs

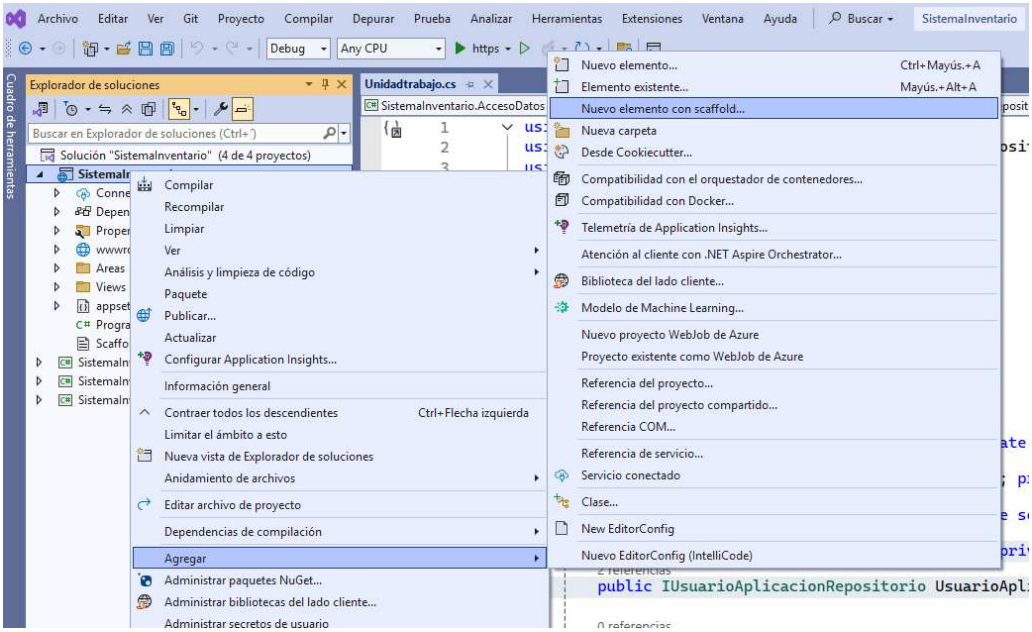
```
.
.
.
public IProductoRepositorio Producto { get; private set; }
 public IUsuarioAplicacionRepositorio UsuarioAplicacion { get; private set;  }

 public UnidadTrabajo(ApplicationDbContext db)
 {
    _db = db;
    Bodega = new BodegaRepositorio(_db);
    Categoria = new CategoriaRepositorio(_db);
    Marca = new MarcaRepositorio(_db);
    Producto = new ProductoRepositorio(_db);
    UsuarioAplicacion = new UsuarioAplicacionRepositorio(_db);
 }
.
.
.
```

- Compilamos para verificar errores

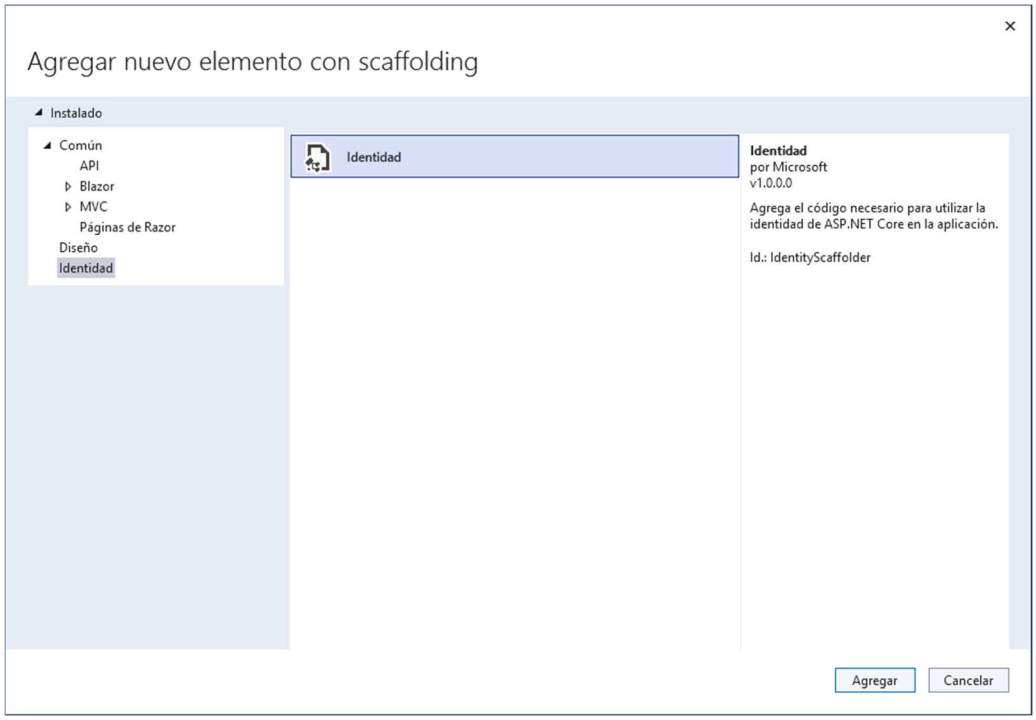
Registro de Usuarios

1. Agregar más campos al Registro

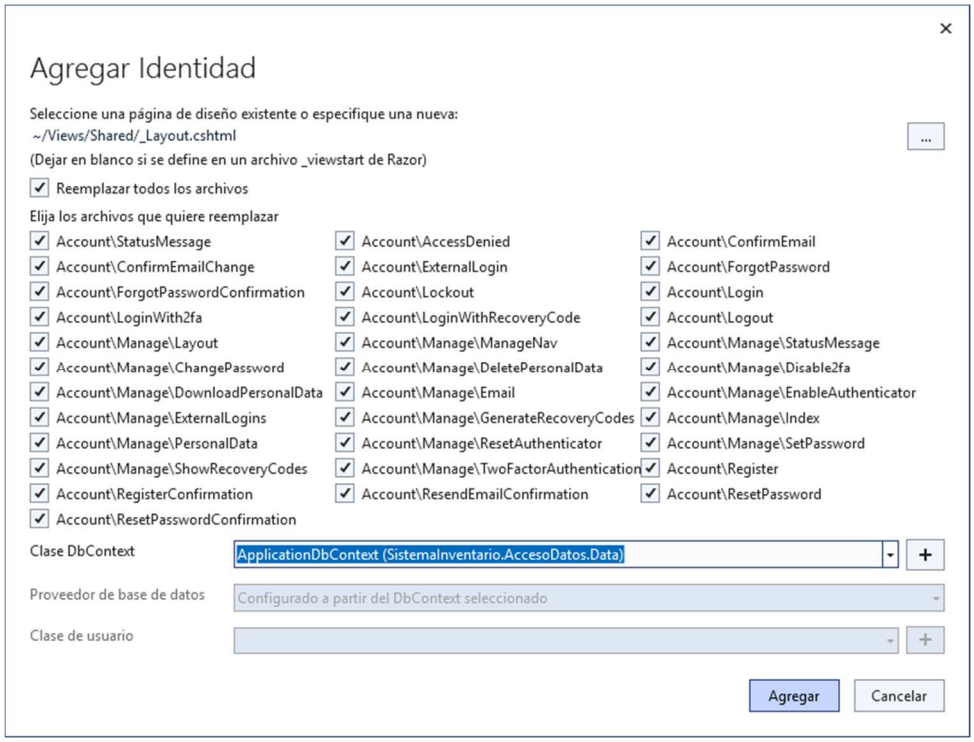
- En el proyecto principal SistemaInventario agregamos “Nuevo elemento con scalfold”



- Escogemos Identidad y Agregar



- Seleccionamos Reemplazar todos los archivos, en Clase DbContext seleccionamos El que está disponible y Agregar para crear las vistas



- Eliminamos la carpeta Data que se creó dentro de Identity

- En SistemaInventario/Areas/Identity/Pages/Account/Register.cshtml.cs modificamos el metodo InputModel

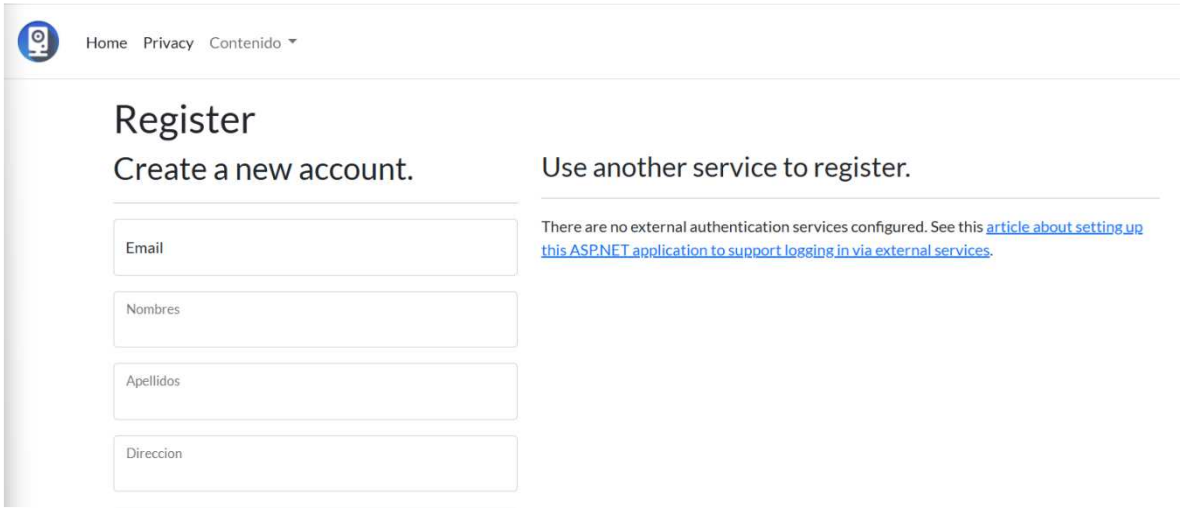
```
public string ConfirmPassword { get; set; }

public string PhoneNumber { get; set; }
[Required]
public string Nombres { get; set; }

[Required]
public string Apellidos { get; set; }

public string Direccion { get; set; }

public string Ciudad { get; set; }
public string Pais { get; set; }
public string Role { get; set; }
```

- Modificamos SistemaInventario/Areas/Identity/Pages/Account/Register.cshtml con los campos adicionados en Register.cshtml.cs

```
<form id="registerForm" asp-route-returnUrl="@Model.ReturnUrl" method="post">
    <h2>Create a new account.</h2>
    <hr />
    <div asp-validation-summary="ModelOnly" class="text-danger" role="alert"></div>
    <div class="form-floating mb-3">
        <input asp-for="Input.Email" class="form-control" autocomplete="username" aria-required="true"
placeholder="name@example.com" />
        <label asp-for="Input.Email">Email</label>
        <span asp-validation-for="Input.Email" class="text-danger"></span>
    </div>
    <div class="form-floating mb-3">
        <input asp-for="Input.Nombres" class="form-control" aria-required="true" />
        <label asp-for="Input.Nombres">Nombres</label>
        <span asp-validation-for="Input.Nombres" class="text-danger"></span>
    </div>
    <div class="form-floating mb-3">
        <input asp-for="Input.Apellidos" class="form-control" aria-required="true" />
        <label asp-for="Input.Apellidos">Apellidos</label>
        <span asp-validation-for="Input.Apellidos" class="text-danger"></span>
    </div>
    <div class="form-floating mb-3">
        <input asp-for="Input.Direccion" class="form-control" aria-required="true" />
        <label asp-for="Input.Direccion">Direccion</label>
        <span asp-validation-for="Input.Direccion" class="text-danger"></span>
    </div>
    <div class="form-floating mb-3">
        <input asp-for="Input.Ciudad" class="form-control" aria-required="true" />
        <label asp-for="Input.Ciudad">Ciudad</label>
        <span asp-validation-for="Input.Ciudad" class="text-danger"></span>
    </div>
    <div class="form-floating mb-3">
        <input asp-for="Input.Pais" class="form-control" aria-required="true" />
        <label asp-for="Input.Pais">Pais</label>
        <span asp-validation-for="Input.Pais" class="text-danger"></span>
    </div>
    <div class="form-floating mb-3">
        <input asp-for="Input.PhoneNumber" class="form-control" aria-required="true" />
        <label asp-for="Input.PhoneNumber">Telefono</label>
        <span asp-validation-for="Input.PhoneNumber" class="text-danger"></span>
    </div>
    <div class="form-floating mb-3">
        <input asp-for="Input.Password" class="form-control" autocomplete="new-password" aria-
required="true" placeholder="password" />
        <label asp-for="Input.Password">Password</label>
        <span asp-validation-for="Input.Password" class="text-danger"></span>
    </div>
    <div class="form-floating mb-3">
        <input asp-for="Input.ConfirmPassword" class="form-control" autocomplete="new-password" aria-
required="true" placeholder="password" />
        <label asp-for="Input.ConfirmPassword">Confirm Password</label>
        <span asp-validation-for="Input.ConfirmPassword" class="text-danger"></span>
    </div>
    <button id="registerSubmit" type="submit" class="w-100 btn btn-lg btn-primary">Register</button>
```

</form>

- Ejecutamos la aplicación para verificar funcionamiento.



2. Registro Admin

- En SistemaInventario.Utilidades/DS.cs verificar los tres roles creados

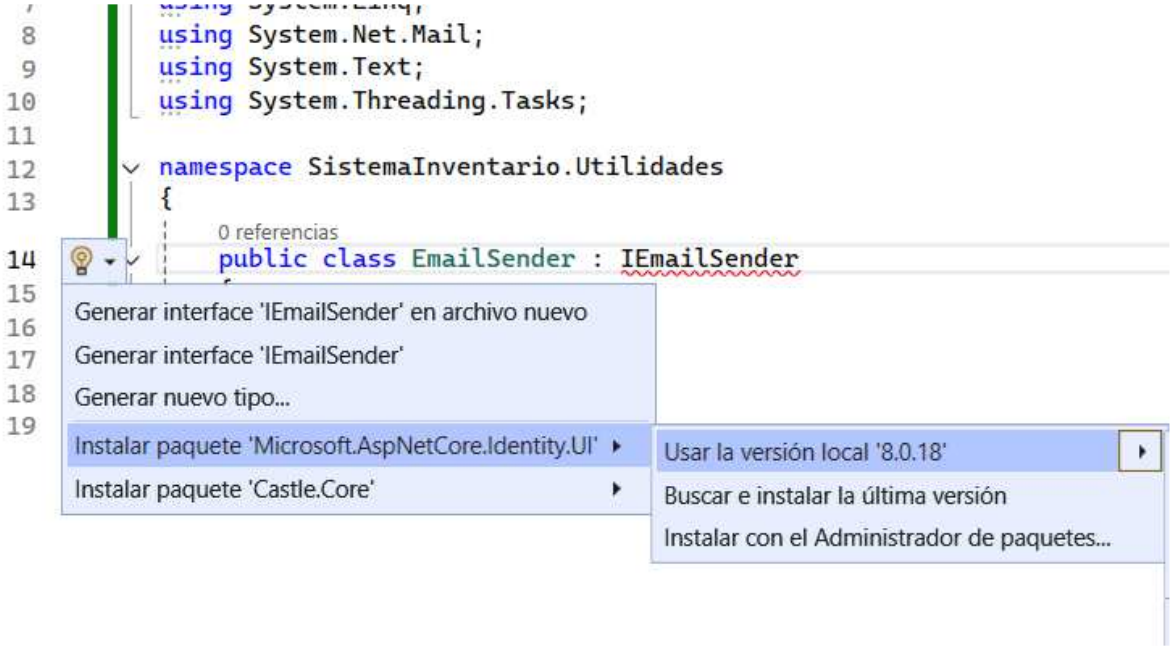
```
public const string Role_Admin = "Admin";
public const string Role_Cliente = "Cliente";
public const string Role_Inventario = "Inventario";
```

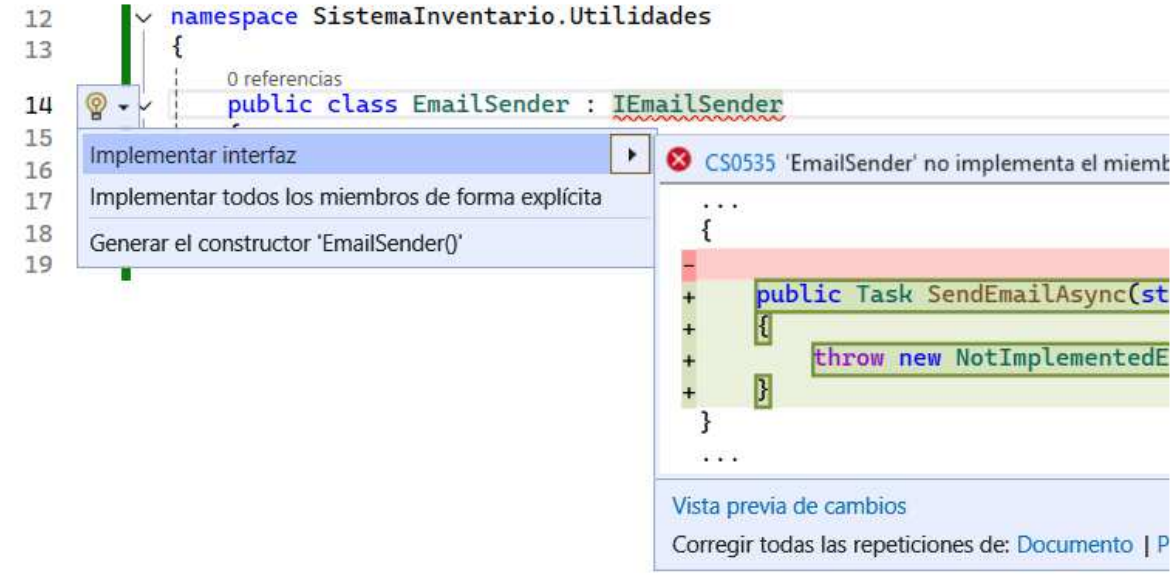
- En SistemaInventario/Areas/Identity/Pages/Account/Register.cshtml.cs modificamos el metodo OnPost

```
public class RegisterModel : PageModel
 {
    private readonly SignInManager<IdentityUser> _signInManager;
    private readonly UserManager<IdentityUser> _userManager;
    private readonly IUserStore<IdentityUser> _userStore;
    private readonly IUserEmailStore<IdentityUser> _emailStore;
    private readonly ILogger<RegisterModel> _logger;
    private readonly IEmailSender _emailSender;
    private readonly RoleManager<IdentityRole> _roleManager;

    public RegisterModel(
        UserManager<IdentityUser> userManager,
        IUserStore<IdentityUser> userStore,
        SignInManager<IdentityUser> signInManager,
        ILogger<RegisterModel> logger,
        IEmailSender emailSender,
        RoleManager<IdentityRole> roleManager)
    {
        _userManager = userManager;
```

```
        _userStore = userStore;
        _emailStore = GetEmailStore();
        _signInManager = signInManager;
        _logger = logger;
        _emailSender = emailSender;
        _roleManager = roleManager;
    }
.
.
.
if (ModelState.IsValid)
{
   // var user = CreateUser();

   var user = new UsuarioAplicacion
   {
      UserName = Input.Email,
      Email = Input.Email,
      PhoneNumber = Input.PhoneNumber,
      Nombres = Input.Nombres,
      Apellidos = Input.Apellidos,
      Direccion = Input.Direccion,
      Ciudad = Input.Ciudad,
      Pais = Input.Pais,
      Role = Input.Role,
   };

await _userStore.SetUserNameAsync(user, Input.Email, CancellationToken.None);
await _emailStore.SetEmailAsync(user, Input.Email, CancellationToken.None);
var result = await _userManager.CreateAsync(user, Input.Password);

if (result.Succeeded)
{
   _logger.LogInformation("User created a new account with password.");

   if (!await _roleManager.RoleExistsAsync(DS.Role_Admin))
   {
      await _roleManager.CreateAsync(new IdentityRole(DS.Role_Admin));
   }
   if (!await _roleManager.RoleExistsAsync(DS.Role_Cliente))
   {
      await _roleManager.CreateAsync(new IdentityRole(DS.Role_Cliente));
   }
   if (!await _roleManager.RoleExistsAsync(DS.Role_Inventario))
   {
      await _roleManager.CreateAsync(new IdentityRole(DS.Role_Inventario));
   }

   await _userManager.AddToRoleAsync(user, DS.Role_Admin);
```

- En SistemaInventario.Utilidades creamos una clase llamada EmailSender.cs y corregimos errores.






```
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.Extensions.Configuration;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Mail;
using System.Text;
```

```
using System.Threading.Tasks;

namespace SistemaInventario.Utilidades
{
   public class EmailSender : IEmailSender
   {
      public Task SendEmailAsync(string email, string subject, string htmlMessage)
      {
         throw new NotImplementedException();
      }
   }
}
```

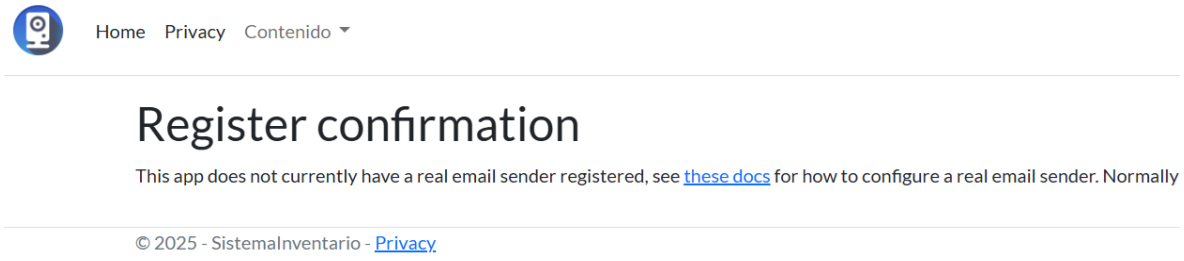
- Modificamos SistemaInventario/Program.cs

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI.Services;
using Microsoft.EntityFrameworkCore;
using SistemaInventario.AccesoDatos.Data;
using SistemaInventario.AccesoDatos.Repositorio;
using SistemaInventario.AccesoDatos.Repositorio.IRepositorio;
using SistemaInventario.Utilidades;
//using SistemaInventario.Data;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new
InvalidOperationException("Connection string 'DefaultConnection' not found.");
builder.Services.AddDbContext<ApplicationDbContext>(options =>
   options.UseSqlServer(connectionString));

builder.Services.AddDatabaseDeveloperPageExceptionFilter();

builder.Services.AddIdentity<IdentityUser,IdentityRole>(options => options.SignIn.RequireConfirmedAccount =
true)
   .AddDefaultTokenProviders()
   .AddEntityFrameworkStores<ApplicationDbContext>();

builder.Services.AddControllersWithViews().AddRazorRuntimeCompilation();

builder.Services.AddScoped<IUnidadTrabajo, UnidadTrabajo>();

builder.Services.AddRazorPages();

builder.Services.AddSingleton<IEmailSender, EmailSender>();

var app = builder.Build();
.
.
.
```

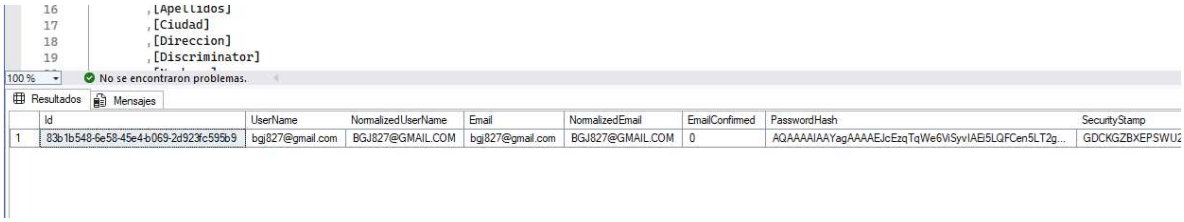
- En SistemaInventario/Areas/Identity/Pages/Account/Register.cshtml.cs comentamos las siguientes líneas:

```
var userId = await _userManager.GetUserIdAsync(user);
//var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
//code = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
//var callbackUrl = Url.Page(
//    "/Account/ConfirmEmail",
//    pageHandler: null,
//    values: new { area = "Identity", userId = userId, code = code, returnUrl = returnUrl },
//    protocol: Request.Scheme);

//await _emailSender.SendEmailAsync(Input.Email, "Confirm your email",
//    $"Please confirm your account by <a href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");

if (_userManager.Options.SignIn.RequireConfirmedAccount)
```

- Ejecutamos la aplicación para verificar registrar un usuario.



- Verificamos en la base de datos



3. Agregar Roles al Registro

- Modificamos SistemaInventario/Areas/Identity/Pages/Account/Register.cshtml.cs agregando una propiedad de tipo Lista.

```
public IEnumerable<SelectListItem> ListaRol { get; set; }
.
.
.
public async Task OnGetAsync(string returnUrl = null)
{
   ReturnUrl = returnUrl;
   Input = new InputModel()
   {
```

```
      ListaRol = _roleManager.Roles.Where(r=> r.Name!=DS.Role_Cliente).Select(n=> n.Name).Select(l=> new
SelectListItem
      {
         Text = l,
         Value =l
      })
   };
   ExternalLogins = (await _signInManager.GetExternalAuthenticationSchemesAsync()).ToList();
}
```

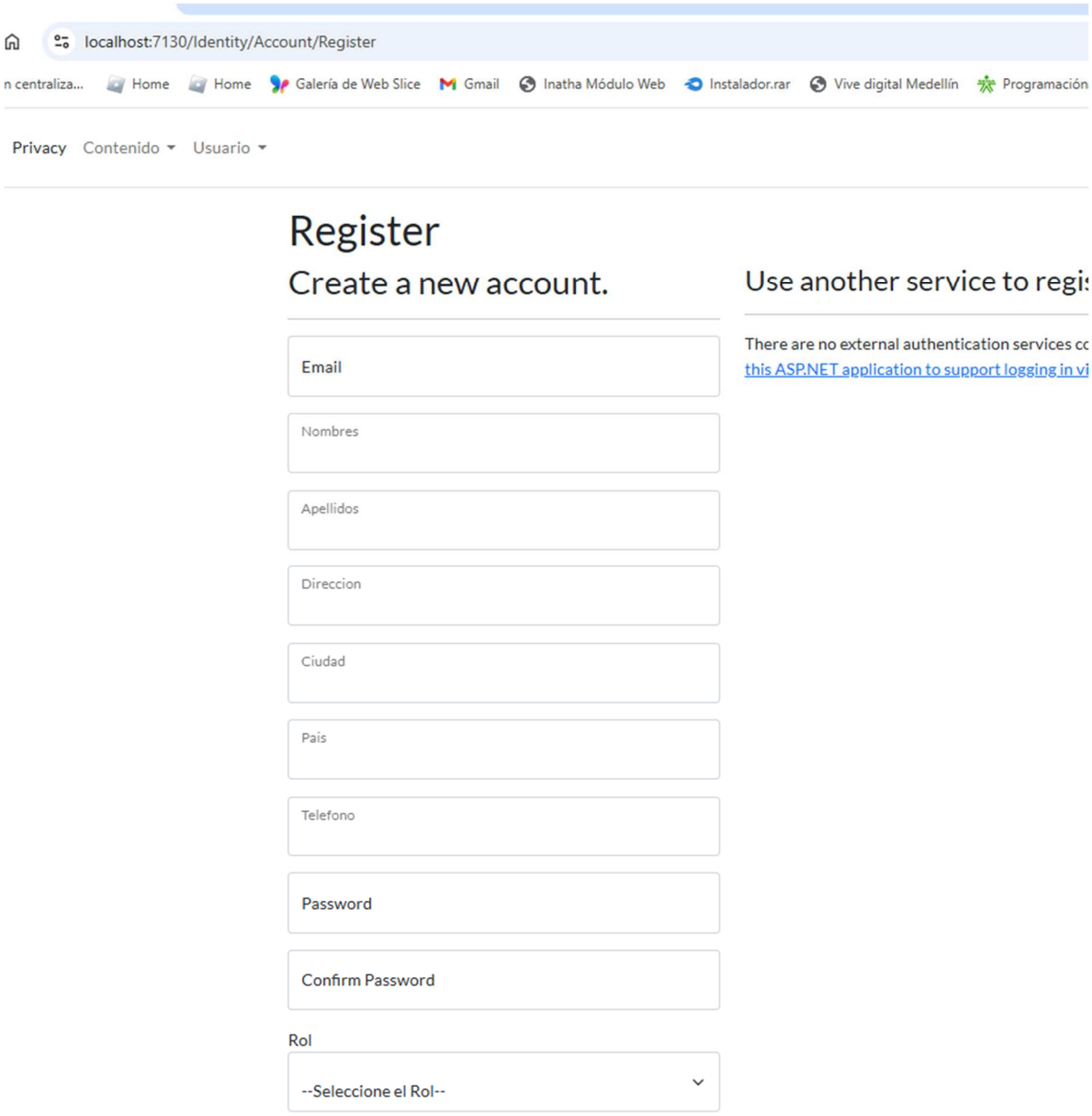
- Modificamos SistemaInventario/Areas/Identity/Pages/Account/Register.cshtml

```
@page
@using SistemaInventario.Utilidades
@model RegisterModel
.
.
.
<div class="form-floating mb-3">
   <input asp-for="Input.ConfirmPassword" class="form-control" autocomplete="new-password" aria-
required="true" placeholder="password" />
   <label asp-for="Input.ConfirmPassword">Confirm Password</label>
   <span asp-validation-for="Input.ConfirmPassword" class="text-danger"></span>
</div>
@if (User.IsInRole(DS.Role_Admin))
{
   <div class="form-floating mb-3">
      <lable asp-for="Input.Role">Rol</lable>
      <select asp-for="Input.Role" class="form-select" asp-items="@Model.Input.ListaRol">
         <option value="">--Seleccione el Rol--</option>
      </select>
   </div>
}
<button id="registerSubmit" type="submit" class="w-100 btn btn-lg btn-primary">Register</button>
```

- En SistemaInventario/Program.cs modificamos el valor del servicio a False para que deje verificar la opción implementada

```
builder.Services.AddIdentity<IdentityUser,IdentityRole>(options => options.SignIn.RequireConfirmedAccount =
false)
```

- Ejecutamos la aplicación e ingresamos como usuario, en el navegador completamos la ruta con /Identity/Account/Register



4. Cambios en la clase de Registro

- Modificamos SistemaInventario/Areas/Identity/Pages/Account/Register.cshtml.cs en el metodo OnPostAsync

```
public async Task<IActionResult> OnPostAsync(string returnUrl = null)
{.
.
.
if (user.Role == null)  // El Valor que recibe desde el Page
 {
    await _userManager.AddToRoleAsync(user, DS.Role_Cliente); //busca esta linea
 }
```

```
 else
 {
    await _userManager.AddToRoleAsync(user, user.Role);
 }.
.
.
.
if (user.Role == null)
{
   await _signInManager.SignInAsync(user, isPersistent: false); //busca esta linea
   return LocalRedirect(returnUrl);
}
else
{
   // Administrador esta registrando un nuevo Usuario
   return RedirectToAction("Index", "Usuario", new { Area = "Admin" });
}
```

Administración de Usuarios

1. Usuario Controlador

- En SistemaInventario/Areas/Admin/Controllers creamos un controlador de tipo Controlador de MVC en blanco llamado UsuarioController.cs

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using SistemaInventario.AccesoDatos.Data;
using SistemaInventario.AccesoDatos.Repositorio.IRepositorio;
using SistemaInventario.Utilidades;
using System.Data;

namespace SistemaInventario.Areas.Admin.Controllers
{
   [Area("Admin")]
   [Authorize(Roles = DS.Role_Admin)]
   public class UsuarioController : Controller
   {

      private readonly IUnidadTrabajo _unidadTrabajo;
      private readonly ApplicationDbContext _db;

      public UsuarioController(IUnidadTrabajo unidadTrabajo, ApplicationDbContext db)
      {
         _unidadTrabajo = unidadTrabajo;
         _db = db;
      }

      public IActionResult Index()
```

```
        {
            return View();
        }

        #region API

        [HttpGet]
        public async Task<IActionResult> ObtenerTodos()
        {
            var usuarioLista = await _unidadTrabajo.UsuarioAplicacion.ObtenerTodos();
            var userRole = await _db.UserRoles.ToListAsync();
            var roles = await _db.Roles.ToListAsync();

            foreach (var usuario in usuarioLista)
            {
                var roleId = userRole.FirstOrDefault(u => u.UserId == usuario.Id).RoleId;
                usuario.Role = roles.FirstOrDefault(u => u.Id == roleId).Name;
            }
            return Json(new { data = usuarioLista });
        }

        #endregion
    }
}
```

2. Mostrar Lista de Usuarios Index

- Desde el método Index de SistemaInventario/Areas/Admin/Controllers/UsuarioController.cs creamos una vista de tipo Razor llamada Index.cshtml

```
@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<div class="row">
    <div class="col-lg-6">
        <h2 class="text-primary">Lista de Usuarios</h2>
    </div>
    <div class="col-lg-6">
        <a class="btn btn-outline-primary" asp-area="Identity" asp-page="/Account/Register">
            <i class="bi bi-plus-square-fill"></i> Crear Nuevo Usuario
        </a>
    </div>
</div>

<br />

<div class="p-4 border rounded bg-light">
    <table id="tblDatos" class="table table-responsive table-hover">
        <thead class="table-dark">
            <tr>
```

```
            <th>Email</th>
            <th>Nombres</th>
            <th>Apellidos</th>
            <th>Telefono</th>
            <th>Role</th>
            <th></th>
          </tr>
        </thead>
    </table>
</div>
@section Scripts {
    <script src="~/js/usuario.js"></script>
}
```

- En SistemaInventario/wwwroot/js creamos usuario.js

```
let datatable;

$(document).ready(function () {
    loadDataTable();
});

function loadDataTable() {
    datatable = $('#tblDatos').DataTable({
        "language": {
            "lengthMenu": "Mostrar _MENU_ Registros Por Pagina",
            "zeroRecords": "Ningun Registro",
            "info": "Mostrar page _PAGE_ de _PAGES_",
            "infoEmpty": "no hay registros",
            "infoFiltered": "(filtered from _MAX_ total registros)",
            "search": "Buscar",
            "paginate": {
                "first": "Primero",
                "last": "Último",
                "next": "Siguiente",
                "previous": "Anterior"
            }
        },
        "ajax": {
            "url": "/Admin/Usuario/ObtenerTodos"
        },
        "columns": [
            { "data": "email" },
            { "data": "nombres" },
            { "data": "apellidos" },
            { "data": "phoneNumber" },
            { "data": "role" },
            //{
            //    "data": "id",
            //    "render": function (data) {
            //        return `
            //            <div>
```

```
        //            <a href="/Admin/Categoria/Upsert/${data}" class="btn btn-success text-white"
style="cursor:pointer">
        //            <i class="bi bi-pencil-square"></i>
        //            </a>
        //            <a onclick=Delete("/Admin/Categoria/Delete/${data}") class="btn btn-danger text-white"
style="cursor:pointer">
        //            <i class="bi bi-trash3-fill"></i>
        //            </a>
        //          </div>
        //      `;
        //   }, "width": "20%"
        //}
      ]
  });
}
```

- Modificamos SistemaInventario/Views/Shared/_Layout.cshtml agregándole lo relacionado al usuario

```
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle" href="#" data-bs-toggle="dropdown" aria-expanded="false">
    Usuario
  </a>
  <ul class="dropdown-menu">
    <li><a class="dropdown-item" asp-area="Admin" asp-controller="Usuario" asp-
action="Index">Usuarios</a></li>
  </ul>
</li>
```

3. Bloquear y Desbloquear botones en JavaScript

- Modificamos el archivo SistemaInventario/wwwroot/js/usuario.js

```
"columns": [
  { "data": "email" },
  { "data": "nombres" },
  { "data": "apellidos" },
  { "data": "phoneNumber" },
  { "data": "role" },
  {
    "data": {
      id: "id", lockoutEnd: "lockoutEnd"
    },
    "render": function (data) {
      let hoy = new Date().getTime();
      let bloqueo = new Date(data.lockoutEnd).getTime();
      if (bloqueo > hoy) {
        // Usuario esta Bloqueado
        return `
          <div class="text-center">
            <a onclick=BloquearDesbloquear('${data.id}') class="btn btn-danger text-white"
style="cursor:pointer", width:150px >
```

```
                    <i class="bi bi-unlock-fill"></i> Desbloquear
                </a>
              </div>
            `;
        }
        else {
          return `
            <div class="text-center">
              <a onclick=BloquearDesbloquear('${data.id}') class="btn btn-success text-white"
style="cursor:pointer", width:150px >
                    <i class="bi bi-lock-fill"></i> Bloquear
                </a>
              </div>
            `;
        }

      }
    }
]
```

4. Metodo BloquearDesbloquear y Llamada API

- En SistemaInventario/Areas/Admin/Controllers/UsuarioController.cs creamos un método llamado BloquerDesbloquear en la región API

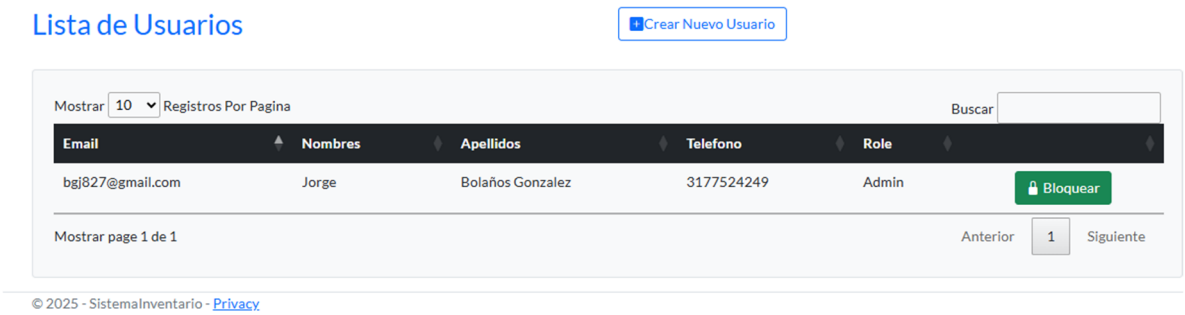
```
[HttpPost]
public async Task<IActionResult> BloquearDesbloquear([FromBody] string id)
{
   var usuario = await _unidadTrabajo.UsuarioAplicacion.ObtenerPrimero(u=> u.Id == id);
   if(usuario==null)
   {
      return Json(new { success = false, message = "Error de Usuario" });
   }
   if(usuario.LockoutEnd != null && usuario.LockoutEnd > DateTime.Now)
   {
      // Usuario Bloqueado
      usuario.LockoutEnd = DateTime.Now;
   }
   else
   {
      usuario.LockoutEnd = DateTime.Now.AddYears(1000);
   }
   await _unidadTrabajo.Guardar();
   return Json(new { success = true, message = "Operacion Exitosa" });

}
```

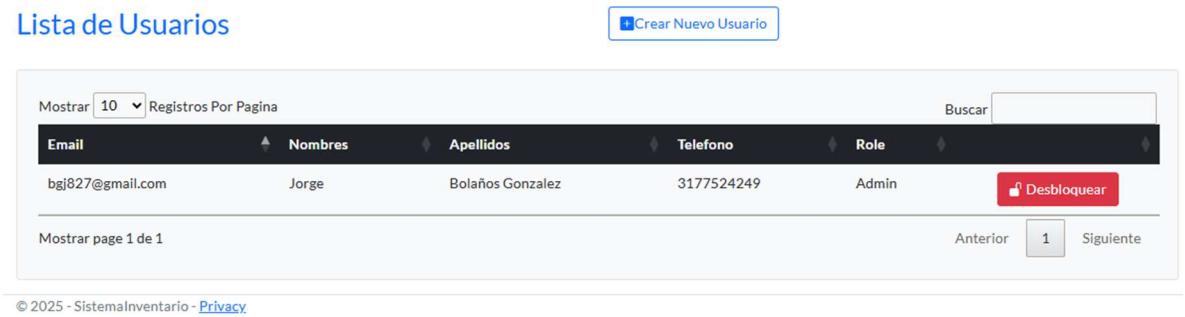
- En SistemaInventario/wwwroot/js/usuario.js invocamos el método BloquearDesbloquear

```
function BloquearDesbloquear(id) {
```

```
    $.ajax({
        type: "POST",
        url: '/Admin/Usuario/BloquearDesbloquear',
        data: JSON.stringify(id),
        contentType: "application/json",
        success: function (data) {
            if (data.success) {
                toastr.success(data.message);
                datatable.ajax.reload();
            }
            else {
                toastr.error(data.message);
            }
        }
    });
}
```

- Ejecutamos la aplicación para verificar funcionalidad del botón Bloquear-Desbloquesar.





5. Cambiar Reglas del Password

- En SistemaInventario/Program.cs adicionamos reglas de password personalizadas

```
builder.Services.Configure<IdentityOptions>(options =>
{
    options.Password.RequireDigit = false;
    options.Password.RequireLowercase = true;
    options.Password.RequireNonAlphanumeric = false;
```

```
    options.Password.RequireUppercase = false;
    options.Password.RequiredLength = 6;
    options.Password.RequiredUniqueChars = 1;
});

builder.Services.AddControllersWithViews().AddRazorRuntimeCompilation();
```

- En SistemaInventario.Utilidades crear una nueva clase llamada ErrorDescriber.cs

```
using Microsoft.AspNetCore.Identity;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SistemaInventario.Utilidades
{
    public class ErrorDescriber : IdentityErrorDescriber
    {
        public override IdentityError PasswordRequiresLower()
        {
            return new IdentityError()
            {
                Code = nameof(PasswordRequiresLower),
                Description = "El password debe tener al menos una Minuscula"
            };
        }
    }
}
```

- Adicionamos ErrorDescriber.cs al servicio AddIdentity de SistemaInventario/Program.cs

```
builder.Services.AddIdentity<IdentityUser,IdentityRole>(options => options.SignIn.RequireConfirmedAccount = false)
    .AddErrorDescriber<ErrorDescriber>()
    .AddDefaultTokenProviders()
    .AddEntityFrameworkStores<ApplicationDbContext>();
```

- Ejecutamos la aplicación e ingresamos un nuevo usuario para verificar el ingreso de Password.

Autorización

1. Autorización en el Proyecto

- Modificamos SistemaInventario/Areas/Identity/Pages/Account/Register.cshtml.cs

```
Input = new InputModel()
{
   ListaRol = _roleManager.Roles.Where(r => r.Name != DS.Role_Cliente).Select(n => n.Name).Select(l => new
SelectListItem
   {
      Text = l,
      Value = l
   })
};

foreach (var error in result.Errors) //Buscar esta línea y pegar lo de arriba
{
   ModelState.AddModelError(string.Empty, error.Description);
}
```

- Adicionamos UseAuthentication en SistemaInventario/Program.cs

```
app.UseAuthentication();
app.UseAuthorization();
```

- Adicionamos atributo Authotize en SistemaInventario/Areas/Admin/Controllers/BodegaController.cs

```
[Authorize(Roles = DS.Role_Admin)]
```

- Adicionamos atributo Authotize en SistemaInventario/Areas/Admin/Controllers/CategoriaController.cs

```
[Authorize(Roles = DS.Role_Admin)]
```

- Adicionamos atributo Authotize en SistemaInventario/Areas/Admin/Controllers/MarcaController.cs

```
[Authorize(Roles = DS.Role_Admin)]
```

- Adicionamos atributo Authotize en SistemaInventario/Areas/Admin/Controllers/ProductoController.cs

```
[Authorize(Roles = DS.Role_Admin + "," + DS.Role_Inventario)]
```

- Adicionamos atributo Authotize en SistemaInventario/Areas/Admin/Controllers/UsuarioController.cs

```
[Authorize(Roles = DS.Role_Admin)]
```

- Modificamos SistemaInventario/Program.cs

```
builder.Services.AddIdentity<IdentityUser,IdentityRole>(options => options.SignIn.RequireConfirmedAccount =
false)
    .AddErrorDescriber<ErrorDescriber>()
    .AddDefaultTokenProviders()
    .AddEntityFrameworkStores<ApplicationDbContext>();

builder.Services.ConfigureApplicationCookie(options =>
{
    options.LoginPath = $"/Identity/Account/Login";
    options.LogoutPath = $"/Identity/Account/Logout";
    options.AccessDeniedPath = $"/Identity/Account/AccessDenied";
});
```

- Verificamos la ejecución de la aplicación con Rol inventario

- Modificamos SistemaInventario/Areas/Identity/Pages/Account/ AccessDenied.cshtml

```
@page
@model AccessDeniedModel
@{
    ViewData["Title"] = "Acceso Negado";
}

<header>
    <div class="container">
        <h1 class="text-danger">@ViewData["Title"]</h1>
        <p class="text-danger">No tienes los suficientes permisos para acceder a este Recurso.</p>
    </div>

</header>
```

- Verificamos la ejecución de la aplicación

2. Mostrar Links en base al Rol

- Modificamos SistemaInventario/Views/Shared/_Layout.cshtml encerrando los dropdown dentro de if.

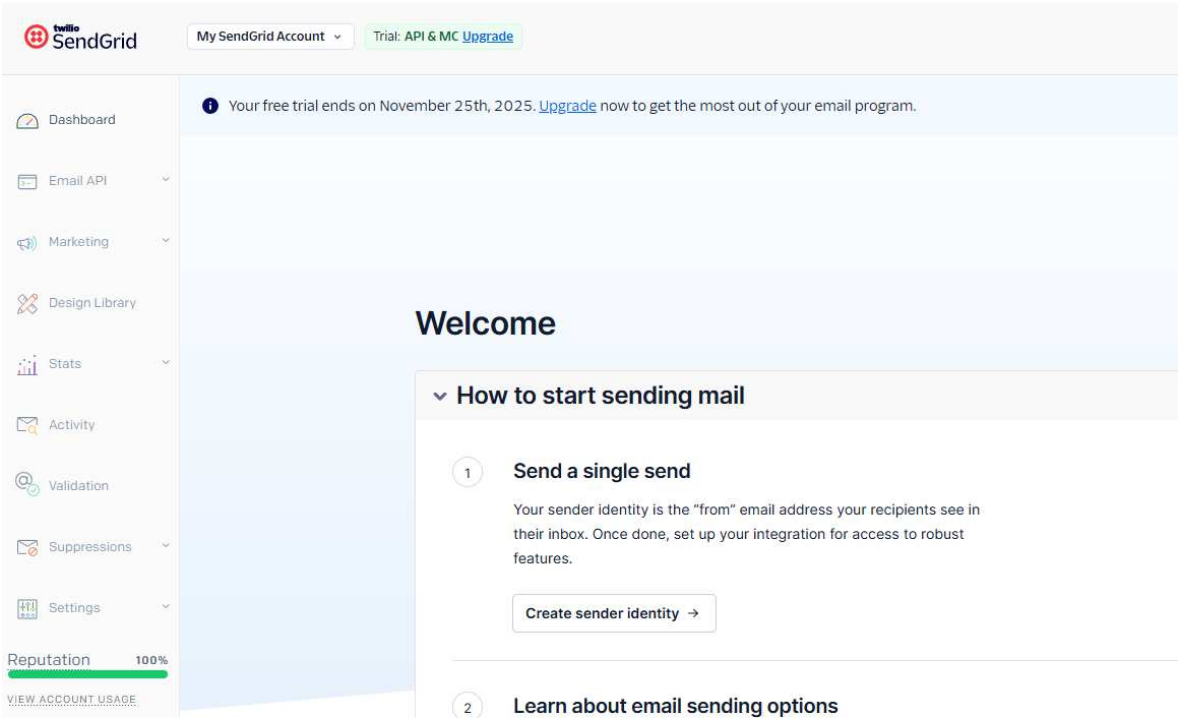
```
@if (User.IsInRole(DS.Role_Admin) || User.IsInRole(DS.Role_Inventario))
{
    <li class="nav-item dropdown">
        <a class="nav-link dropdown-toggle" href="#" data-bs-toggle="dropdown" aria-expanded="false">
            Contenido
        </a>
        <ul class="dropdown-menu">
            @if (User.IsInRole(DS.Role_Admin))
            {
                <li><a class="dropdown-item" asp-area="Admin" asp-controller="Bodega" asp-
action="Index">Bodegas</a></li>
                <li><a class="dropdown-item" asp-area="Admin" asp-controller="Categoria" asp-
action="Index">Categorias</a></li>
```

```
            <li><a class="dropdown-item" asp-area="Admin" asp-controller="Marca" asp-action="Index">Marcas</a></li>
            <li><a class="dropdown-item" asp-area="Admin" asp-controller="Compania" asp-action="Upsert">Compania</a></li>
        }
        <li><a class="dropdown-item" asp-area="Admin" asp-controller="Producto" asp-action="Index">Productos</a></li>
      </ul>
    </li>
}
@if (User.IsInRole(DS.Role_Admin))
{
    <li class="nav-item dropdown">
      <a class="nav-link dropdown-toggle" href="#" data-bs-toggle="dropdown" aria-expanded="false">
        Usuario
      </a>
      <ul class="dropdown-menu">
        <li><a class="dropdown-item" asp-area="Admin" asp-controller="Usuario" asp-action="Index">Usuarios</a></li>
      </ul>
    </li>
}
```

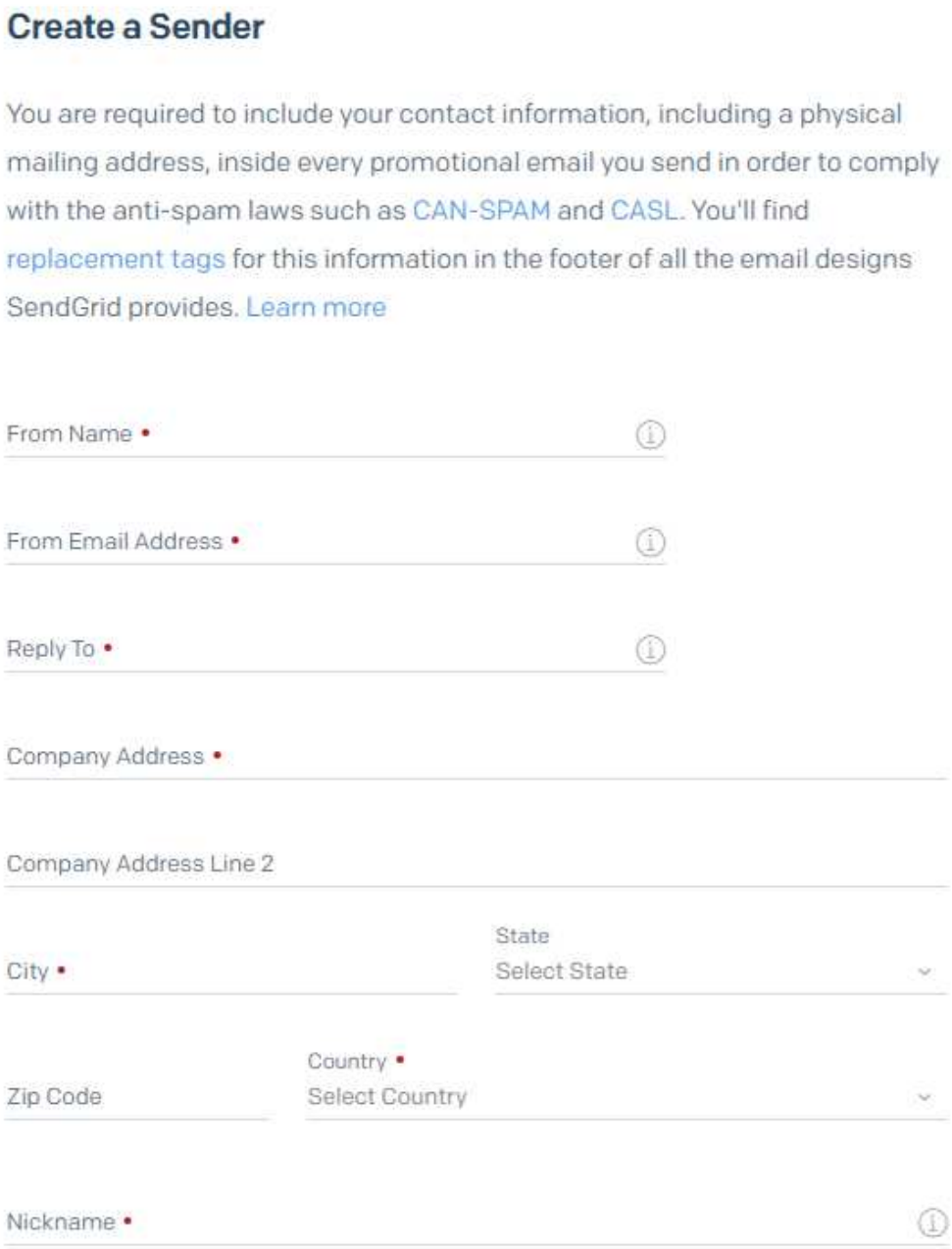
Email

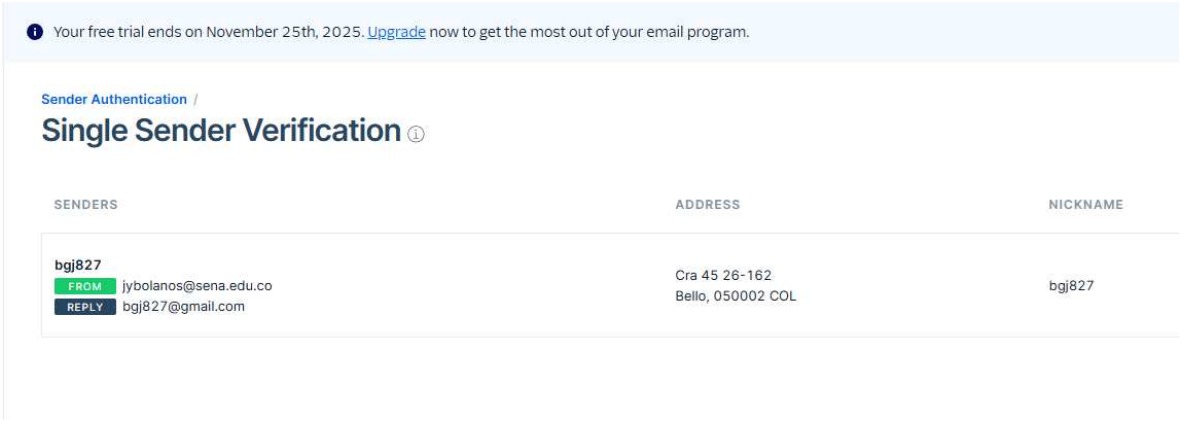
1. Envió de Emails usando SendGrid - Registro

- Nos registramos en el servicio de SendGrid

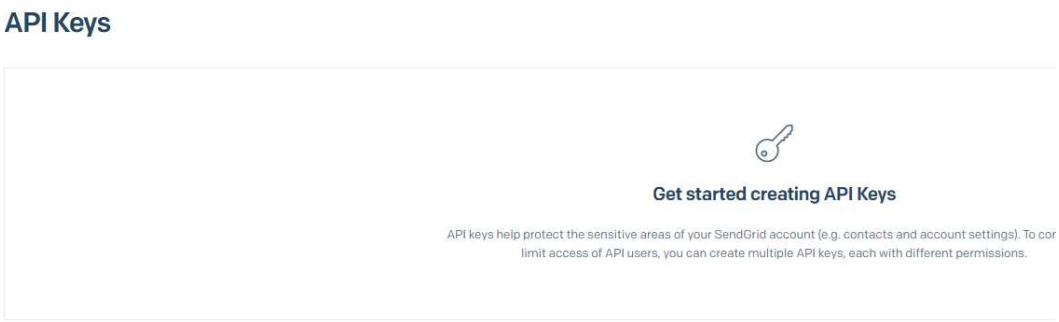


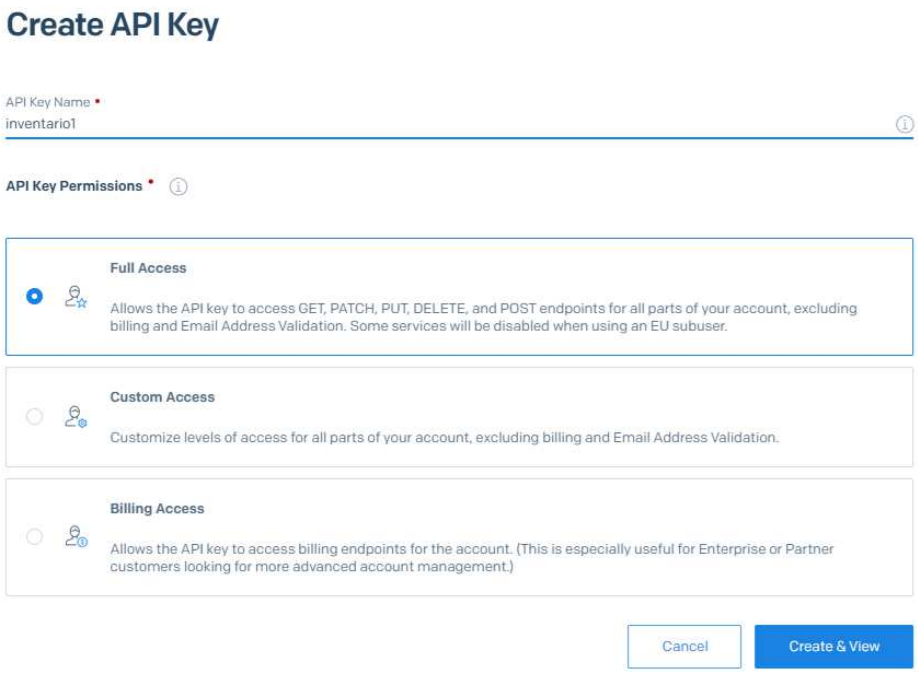
- Configurar Create sender identity

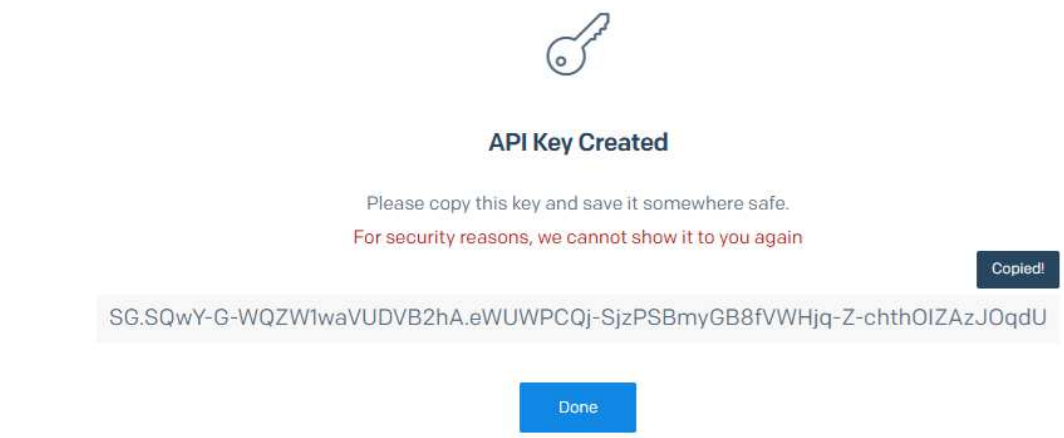




- Confirmar la cuenta en la cuenta de correo electrónico
- Crear API Keys







- Configurar el API key en SistemaInventario/appsettings.json

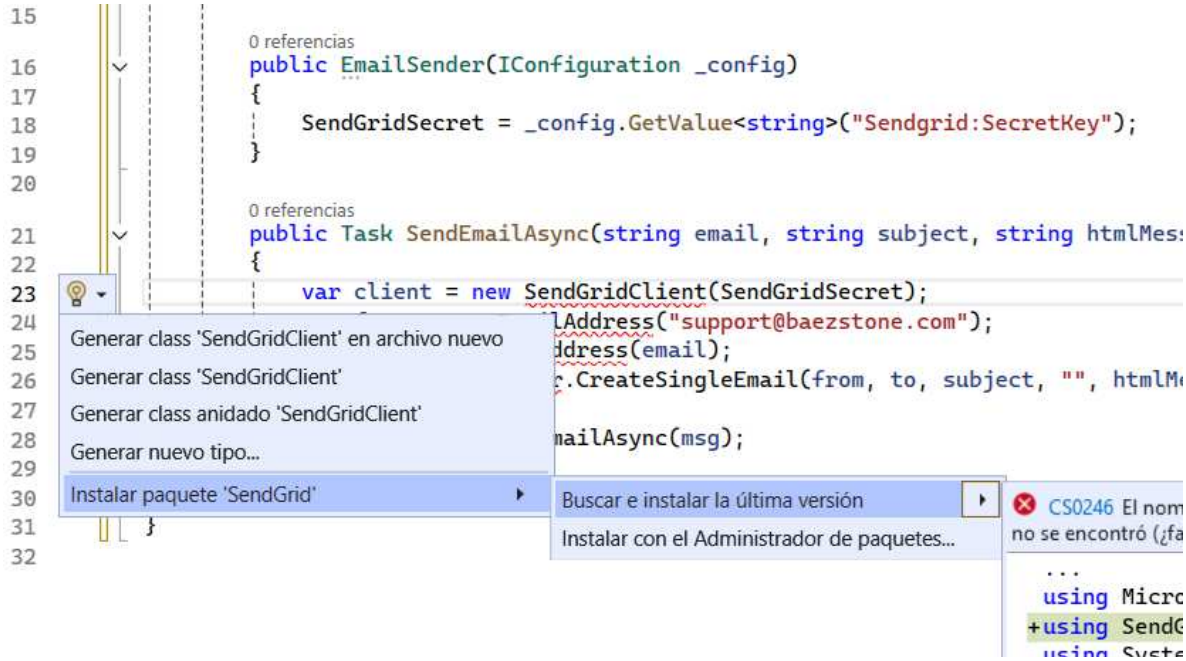
```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Server=<servidor>;Database=SistemaInventarioBD;Trusted_Connection=True;MultipleActiveResultSets=true;
TrustServerCertificate=True",
    "ApplicationDbContextConnection":
"Server=(localdb)\\mssqllocaldb;Database=SistemaInventario;Trusted_Connection=True;MultipleActiveResultS
ets=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "SendGrid": {
    "SecretKey": "<API Key>"
  }
}
```

- Modificamos SistemaInventario.Utilidades/EmailSender.cs

```
namespace SistemaInventario.Utilidades
{
    public class EmailSender : IEmailSender
    {
        public string SendGridSecret { get; set; }

        public EmailSender(IConfiguration _config)
        {
            SendGridSecret = _config.GetValue<string>("Sendgrid:SecretKey");
        }

        public Task SendEmailAsync(string email, string subject, string htmlMessage)
```

```
        {
            var client = new SendGridClient(SendGridSecret);
            var from = new EmailAddress("<correo configurado>");
            var to = new EmailAddress(email);
            var msg = MailHelper.CreateSingleEmail(from, to, subject, "", htmlMessage);

            return client.SendEmailAsync(msg);
        }
    }
}
```

- Instala el paquete SendGrid



2. Implementación Email Sender

- Modificamos SistemaInventario/Areas/Identity/Pages/Account/Register.cshtml.cs para EmailSender,

```
var userId = await _userManager.GetUserIdAsync(user);
//var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
//code = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
//var callbackUrl = Url.Page(
//    "/Account/ConfirmEmail",
//    pageHandler: null,
//    values: new { area = "Identity", userId = userId, code = code, returnUrl = returnUrl },
//    protocol: Request.Scheme);

//await _emailSender.SendEmailAsync(Input.Email, "Confirm your email",
//    $"Please confirm your account by <a href='{HtmlEncoder.Default.Encode(callbackUrl)}'>clicking here</a>.");

if (_userManager.Options.SignIn.RequireConfirmedAccount)
{
    return RedirectToPage("RegisterConfirmation", new { email = Input.Email, returnUrl = returnUrl });
}
else
{
```

```
var userId = await _userManager.GetUserIdAsync(user);
var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
code = WebEncoders.Base64UrlEncode(Encoding.UTF8.GetBytes(code));
var callbackUrl = Url.Page(
   "/Account/ConfirmEmail",
   pageHandler: null,
   values: new { area = "Identity", userId = userId, code = code, returnUrl = returnUrl },
   protocol: Request.Scheme);

await _emailSender.SendEmailAsync(Input.Email, "Confirmar tu Email",
   $"Confirmar tu cuenta <a href='{HtmlEncoder.Default.Encode(callbackUrl)}'>Haz click en el enlace</a>.");
```

- Ejecutamos la aplicación para crear una nueva cuenta.

- Modificamos SistemaInventario/Areas/Identity/Pages/Account/ConfirmEmail.cshtml

```
@page
@model ConfirmEmailModel
@{
   ViewData["Title"] = "Confirmar email";
}

<h1>@ViewData["Title"]</h1>
<partial name="_StatusMessage" model="Model.StatusMessage" />
```

- Modificamos SistemaInventario/Areas/Identity/Pages/Account/ConfirmEmail.cshtml.cs

```
code = Encoding.UTF8.GetString(WebEncoders.Base64UrlDecode(code));
var result = await _userManager.ConfirmEmailAsync(user, code);
StatusMessage = result.Succeeded ? "Gracis por confirmar tu Email." : "Erroral confirmar tu Email.";
return Page();
```