PROYECTO

Simbaña Moreira Adrian Isaee
Socasi Guallichico Moisés Benjamín
Torres Chávez Marlon Pavel

Universidad de las Fuerzas Armadas ESPE
Departamento de Ciencias de la Computación
Aplicaciones Distribuidas
Tabla de Contenido

Introducción

En este informe se describe la implementación de una capa de seguridad robusta para un sistema basado en la arquitectura por capas en .NET. La seguridad es un pilar fundamental en el desarrollo de software, especialmente en sistemas que manejan datos sensibles y requieren controles estrictos de acceso. Este documento detalla los aspectos de autenticación, autorización, protección contra amenazas, gestión de usuarios y auditoría, proporcionando una guía estructurada para su desarrollo e integración.

Objetivos

Objetivo Principal

Diseñar e implementar una capa de seguridad que garantice la protección de datos y el control de acceso en un sistema desarrollado en .NET.

Objetivos específicos

- Implementar métodos seguros de autenticación de usuarios.
- Diseñar un modelo basado en roles para gestionar permisos y accesos.
- Registrar eventos críticos y accesos en un sistema de auditoría.
- Proporcionar una experiencia de usuario segura y eficiente.

Marco Teórico

.NET Framework

.NET es un entorno de desarrollo multiplataforma creado por Microsoft, diseñado para la creación de aplicaciones web, de escritorio y móviles. Ofrece un conjunto de bibliotecas y herramientas que simplifican el desarrollo de software robusto y escalable.

Características principales:

- Multiplataforma: Soporte para Windows, macOS y Linux.
- Productividad: Lenguajes como C# y VB.NET permiten escribir código de manera eficiente.
- Interoperabilidad: Soporte para múltiples tecnologías y lenguajes.

- Herramientas integradas: Compatibilidad con entornos como Visual Studio para facilitar el desarrollo.

2. Metodología de Capas

La arquitectura por capas es un patrón de diseño que divide el sistema en módulos independientes según sus responsabilidades. Este enfoque mejora la organización del código, facilita la escalabilidad y promueve el mantenimiento del software.

Capas comunes:

- Capa de Presentación (UI):
  Interfaz gráfica con la que interactúa el usuario. En .NET, puede desarrollarse con ASP.NET MVC, Blazor o WPF.
- Capa de Lógica de Negocio (BLL):

Implementa las reglas de negocio del sistema y maneja validaciones y procesos antes de interactuar con la base de datos.

- Capa de Acceso a Datos (DAL):

Interactúa directamente con la base de datos y utiliza ORM como Entity Framework para gestionar datos de manera eficiente.

Ventajas:

Separación de responsabilidades: Cada capa tiene un propósito definido.

Escalabilidad: Las capas pueden ser modificadas o ampliadas sin afectar a otras.

Reutilización: Las capas inferiores pueden ser reutilizadas en diferentes aplicaciones.

Especificación de Requisitos Funcionales y No Funcionales

Requisitos Funcionales

- Autenticación basada en credenciales tradicionales (usuario y contraseña).

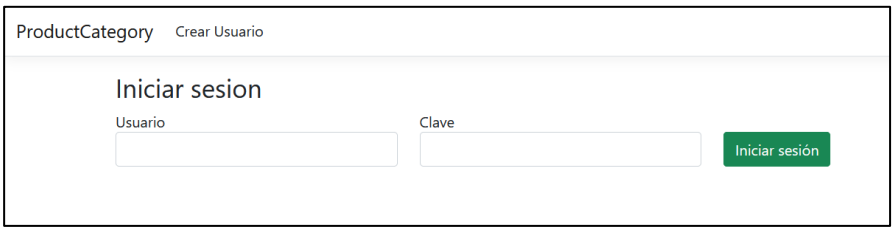


Imagen. Login tradicional

- Gestión de roles y permisos detallada.

Rol	Permiso
Admin	CRUD tabla Categorias
User	CRUD tabla Productos

Rol Admin:

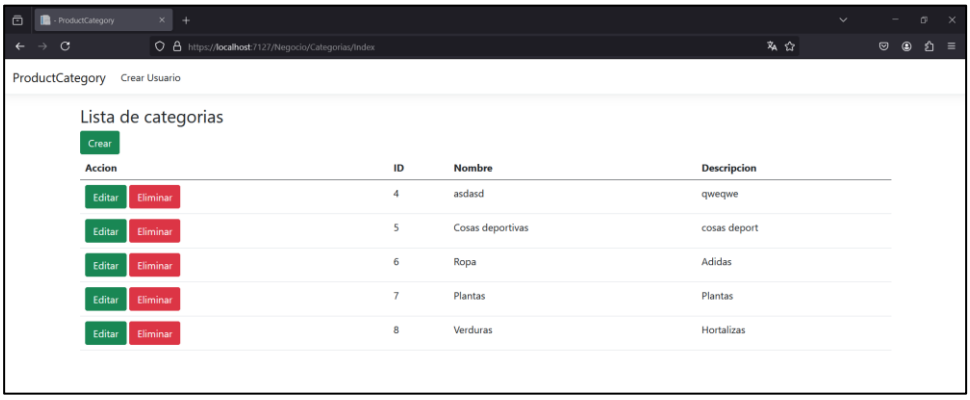


Imagen. CRUD de Categorias

Rol User:

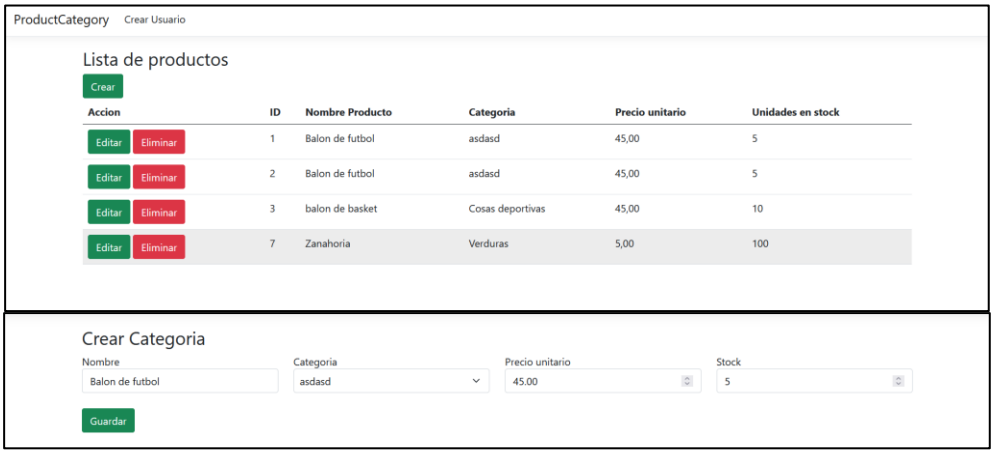


Imagen. CRUD Productos

- Registro de actividades críticas como inicios de sesión, intentos fallidos y cambios.

Imagen. Inicio de Sesión Fallido

	Id	UserName	LockoutEnd	LockoutEnabled	AccessFailedCount
1	307c905a-47f2-4090-97b0-eaa72a069551	asimbana	NULL	1	1

Imagen. Registro del inicio de sesión fallido

Requisitos No Funcionales

- Tiempo de respuesta inferior a 2 segundos para operaciones de autenticación y autorización.
- Soporte para configuraciones escalabilidad.
- Cumplimiento con estándares de seguridad como OWASP Top Ten

Desarrollo

Autenticación:

Para empezar con la autenticación de usuarios es necesario una modificación a la base de datos agregando Usuarios

- Implementa hashing seguro de contraseñas utilizando bcrypt.

	Id	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed	PasswordHash
1	307c905a-47f2-4090-97b0-eaa72a069551	asimbana	ASIMBANA	simbanaa115@gmail.com	SIMBANAA115@GMAIL.COM	0	AQAAAAIAAYagAAAAEM9QY/5ZhcaJaFoMKh/p451tVfZ7AVxstoGq...
2	3acc2926-a3d2-4d87-9136-b00e7b4d3cf8	Xavier	XAVIER	jano3481@hotmail.com	JANO3481@HOTMAIL.COM	0	AQAAAAIAAYagAAAAEG3I8OA3Sz8r1dJRVTBPAG5zy2iFHWLKf3i...
3	5302143a-23c8-440c-adfb-10b78d853f83	test4	TEST4	Test@mail.com	TEST@MAIL.COM	0	AQAAAAIAAYagAAAAEMGQ/4ncRMyJxsmwYGs9SS0BIS6FZKTdh...
4	5c2cfff4-b626-43d4-93e0-6e93ff145438	test2	TEST2	jqwe@mail.com	JQWE@MAIL.COM	0	AQAAAAIAAYagAAAAEJhpalhVKYE3SI7AsXMDpw7S3dTEOSu4+...
5	5f0eed62-bfde-46c0-80b5-a59848a64077	Test12	TEST12	test12@gmail.com	TEST12@GMAIL.COM	0	AQAAAAIAAYagAAAAELw1yB+PTga7+QgmsmBK3p/k0N2/yLrROb...
6	9858df01-a402-4b23-ba48-74e34013f5cc	isaee12	ISAEE12	simbanaa115@gmail.com	SIMBANAA115@GMAIL.COM	0	AQAAAAIAAYagAAAAEHI6QHehXsKyh1Q0iPVZs88rUP5naAcysNO...

Imagen. Contraseñas con Hash

- Configura la expiración de sesiones mediante AuthenticationOptions en .NET.

```
builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme)
    .AddCookie(options =>
    {
        options.LoginPath = "/Account/Login"; // Ruta para la página de inicio de sesión
        options.LogoutPath = "/Account/Logout"; // Ruta para la página de cierre de sesión
        options.AccessDeniedPath = "/Account/AccessDenied"; // Página de acceso denegado
        options.ExpireTimeSpan = TimeSpan.FromMinutes(15); // Tiempo de expiración de la cookie
        options.SlidingExpiration = true; // Renovar automáticamente la cookie si está cerca de expirar
        options.Cookie.HttpOnly = true; // La cookie solo es accesible desde el servidor
        options.Cookie.SecurePolicy = CookieSecurePolicy.SameAsRequest; // Requiere HTTPS
        options.Cookie.SameSite = SameSiteMode.Strict; // Restringe el acceso de terceros a la cookie
    });
```

Imagen. Validación de tiempo de espera de sesión

Autorización:

Definimos roles y permisos en una tabla dentro de la base de base de datos.

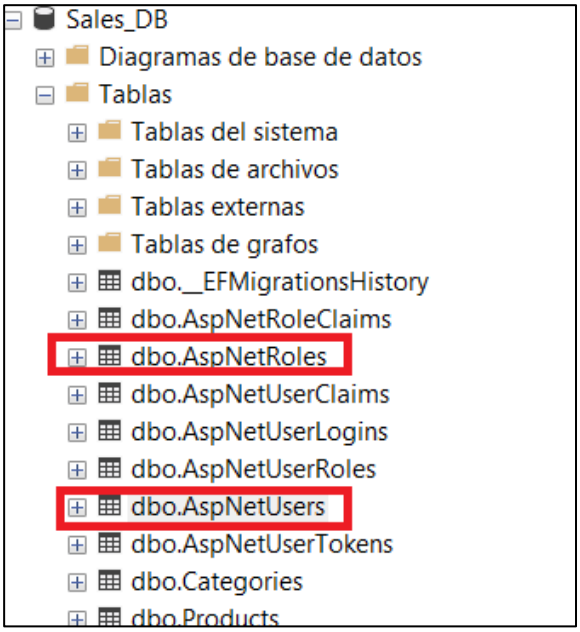


Imagen. Tablas de Usuarios y Roles

Uso de atributos en controladores que permiten la redirección a la respectiva vista.

```
var resultado = await _signInManager.PasswordSignInAsync(model.UserName, model.Passwor
if (resultado.Succeeded)
{
    var roles = await _userManager.GetRolesAsync(user);

    // Redirigir según el rol
    if (roles.Contains("Admin"))
    {
        return RedirectToAction("Index", "Categorias", new {area = "Negocio"});
    }
    else if (roles.Contains("User"))
    {
        return RedirectToAction("Index", "Productos", new { area = "Negocio" });
    }
}
else
{
    ModelState.AddModelError("", "Error al inicar sesion");
    return View("Login", model);
}
```

Imagen. Definición de Roles dentro del controlador para asegurar la ruta a la vista correcta.

Diagrama de Arquitectura.

El diagrama de la arquitectura es por capas el cual esta implementado de la siguiente manera:

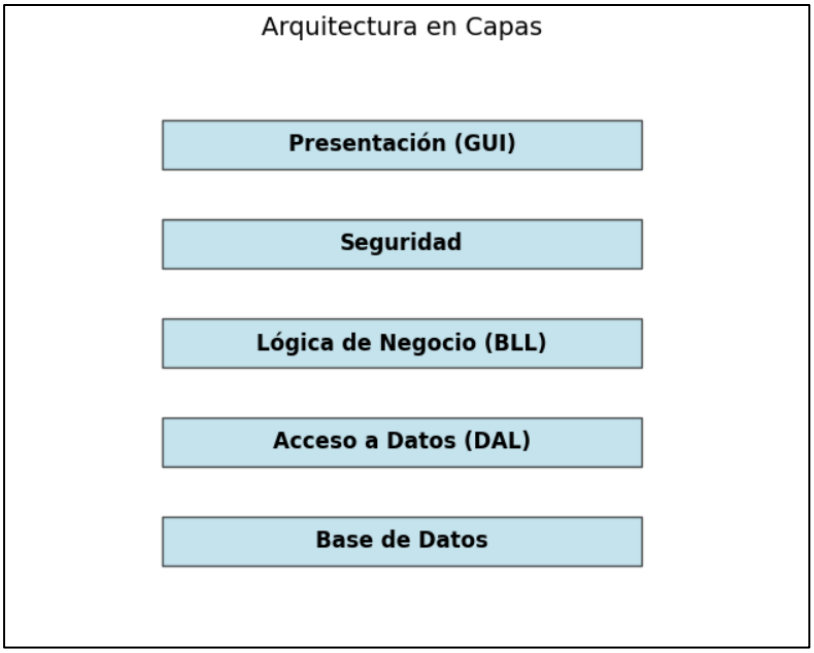


Imagen. Arquitectura del proyecto

Diagrama de Flujo.

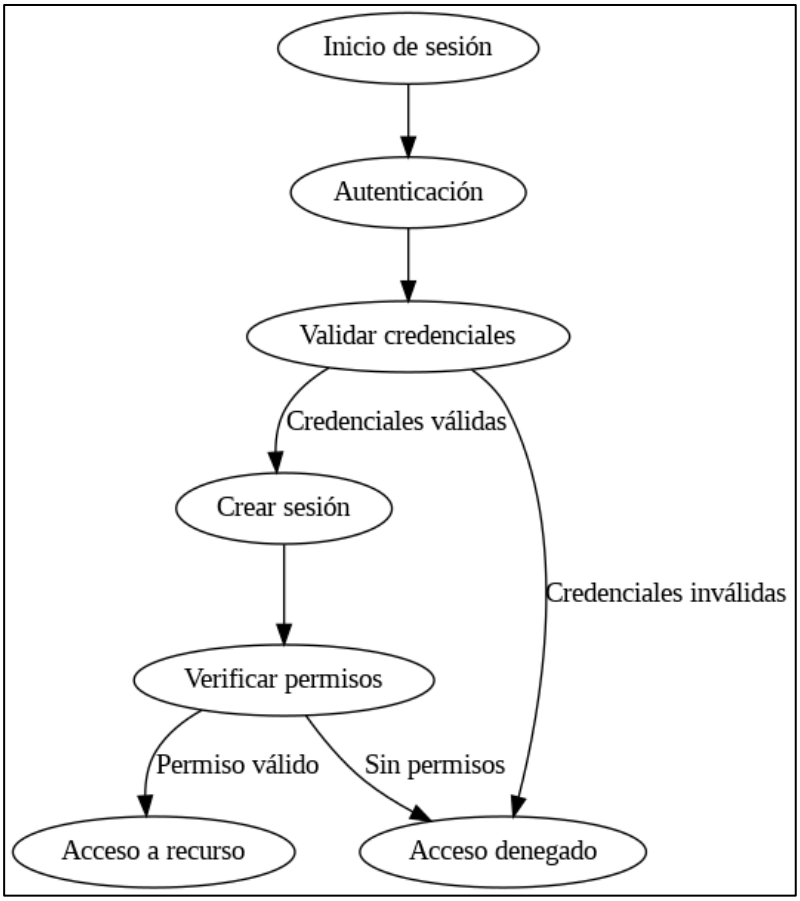


Imagen. Flujo de acceso al Sistema

Matriz de Roles y Permisos

Rol	Permiso
Admin	CRUD tabla Categorias
User	CRUD tabla Productos

Manejo de Errores

El manejo de errores esta programado de tal manera que no se detenga el servicio si no en la vista me nuestre el error.

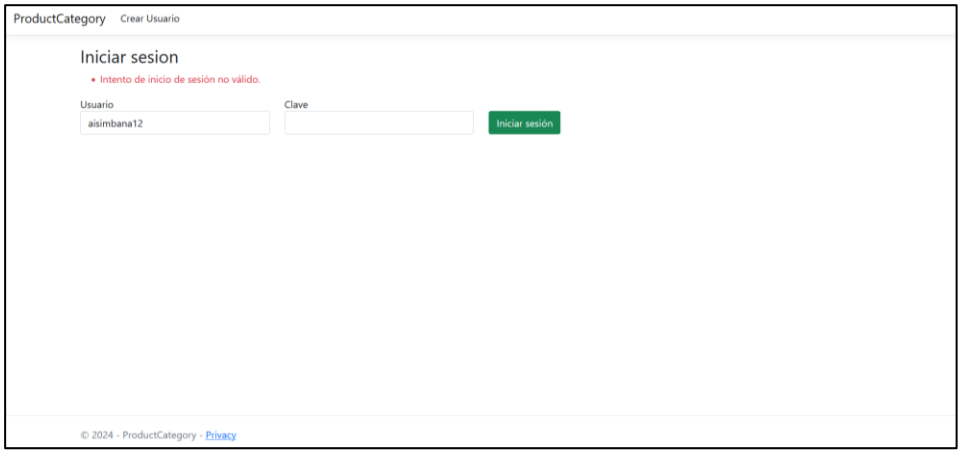


Imagen. Ejemplo de manejo de errores en este caso ese usuario no existe en la base de datos

Manual de Usuario

1. Primero ingresamos al repositorio del github y descargamos los archivos que necesitamos, que son el SQL_Proyecto.sql y la carpeta ProductCategory.

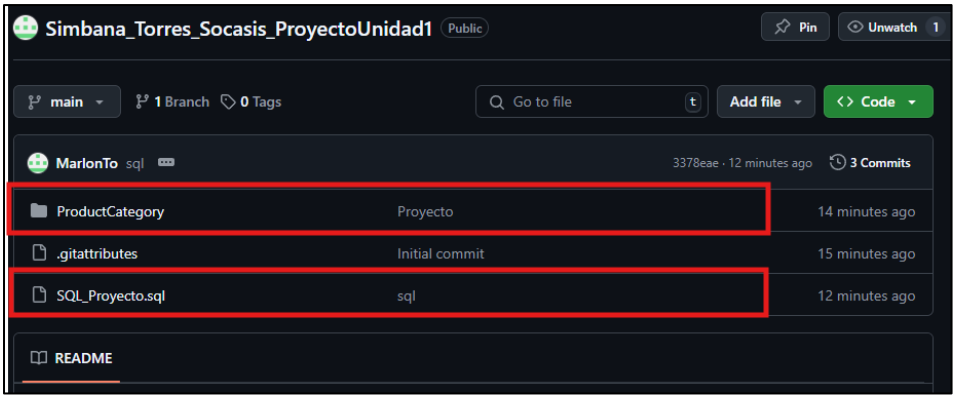


Imagen. Imagen de los archivos del repositorio que se debe descargar

2. Ingresamos al SQL Server Magnament para introducir el sql y creamos una BD llamada Sales_DB

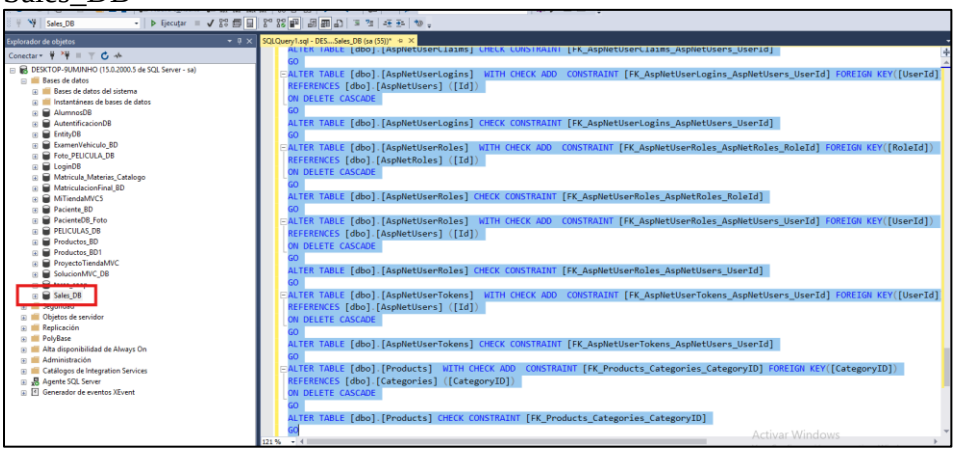


Imagen. Ejecución del Sql.

3. Descomprimimos el archivo descargado del repositorio y le ejecutamos la aplicación en Visual Studio

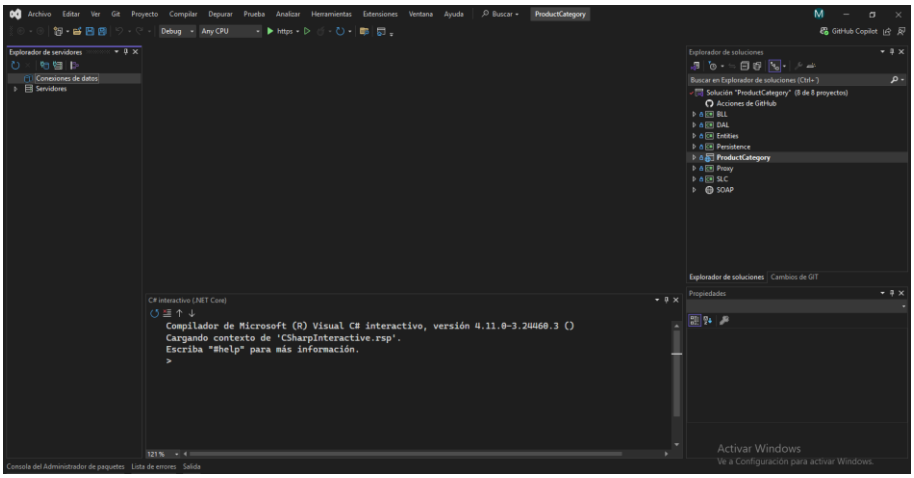


Imagen. Inicio de la solución en la aplicación de Visual Studio

4. Le ejecutamos y probamos que funcione

Conclusiones

La implementación de una capa de seguridad sólida en un sistema por capas es fundamental para proteger la integridad, confidencialidad y disponibilidad de los datos. Al incluir mecanismos como autenticación segura, autorización basada en roles y medidas contra amenazas, se logra un entorno confiable tanto para los usuarios como para los administradores. Además, el monitoreo de eventos críticos y la gestión eficiente de sesiones fortalecen la capacidad de respuesta ante incidentes de seguridad. Estas prácticas no solo refuerzan la seguridad del sistema, sino que también mejoran la experiencia del usuario mediante procesos intuitivos y seguros.

Recomendaciones

Para proyectos futuros que utilicen SOAP, es importante documentar correctamente las operaciones disponibles en el servicio WSDL, lo que agilizará la implementación y permitirá identificar posibles problemas de integración de forma temprana.

Aunque SOAP es útil en muchos casos, se recomienda evaluar alternativas como REST si el proyecto no requiere las características avanzadas de SOAP, como su soporte robusto para transacciones y seguridad. Esto puede simplificar el desarrollo y mejorar el rendimiento en aplicaciones menos críticas.

Bibliografía

Ardalis. (2023, 9 marzo). *Arquitecturas de aplicaciones web comunes - .NET*. Microsoft Learn. https://learn.microsoft.com/es-es/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures#traditional-n-layer-architecture-applications

Küber, G. (2018, 24 junio). *Programación en N Capas .Net* [Diapositivas]. SlideShare. https://es.slideshare.net/slideshow/programacin-en-n-capas-net/102871975

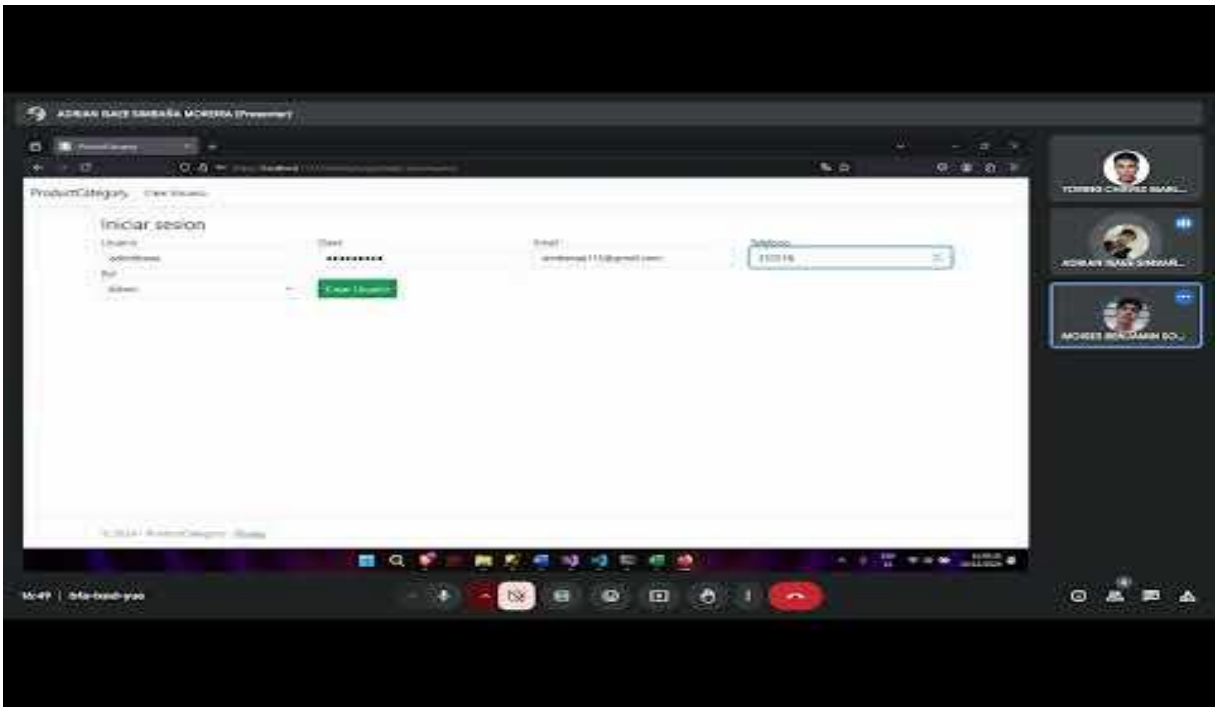
Anexos

Repositorio del Github:

https://github.com/MarlonTo/Simbana_Torres_Socasis_ProyectoUnidad1.git

Link del video en YouTube

https://youtu.be/2sf8nW7lZ1A?si=z-hARd27OYon42OO


Iniciar sesion