



UNIVERSIDAD PRIVADA DE TACNA
FACULTAD DE INGENIERÍA
Escuela Profesional de Ingeniería de Sistemas

Proyecto de Unidad
“Control de Asistencia”

Curso: Calidad y Pruebas de Software

Docente: Ing. Cuadros Quiroga, Patrick Jose

Integrantes:

Paja De la Cruz, Piero Alexander	(2020067576)
Contreras Lipa, Alvaro Javier	(2021070020)
Hernández Cruz, Angel Gadiel	(2021070017)

Tacna – Perú

2024

Resumen

Este proyecto tiene como objetivo desarrollar un sistema de control de asistencia para empleados, utilizando Windows Forms como la interfaz de usuario y SQL Server para la gestión y almacenamiento de datos. El sistema permitirá registrar la asistencia de los empleados, gestionar horarios, áreas, y puestos, y proporcionar un seguimiento detallado del historial de asistencia, faltas, y horarios de trabajo. Además, contará con la funcionalidad para la administración de usuarios, con diferentes niveles de acceso al sistema.

Abstract

This project aims to develop an employee attendance management system, utilizing Windows Forms as the user interface and SQL Server for data storage and management. The system will allow for the recording of employee attendance, management of schedules, areas, and job positions, and provide detailed tracking of attendance history, absences, and work schedules. Additionally, it will include user administration functionality with different access levels within the system.

1. Antecedentes

Cuando el usuario ingresa sus datos para iniciar sesión en la interfaz del sistema de control de asistencia, se le asignará un rol según su nivel de relevancia dentro del sistema (por ejemplo, usuario común, administrador, supervisor). Esta información se almacenará en una base de datos creada anteriormente, asegurando que los nombres de usuario y contraseñas sean únicos.

2. Título

Control Asistencia

3. Autores

1. Contreras Lipa, Alvaro Javier
2. Hernandez Cruz, Angel Gadiel
3. Paja De la Cruz, Piero Alexander

4. Planteamiento del problema

4.1 Problema

La problemática que motiva la implementación de la codificación en el sistema de control de asistencia es la privacidad de los usuarios. La protección de datos como el DNI, número telefónico e incluso los nombres genera preocupación, por lo que se optó por codificar toda la información. De esta manera, solo podrán acceder a los datos aquellos que estén autorizados y tengan una relación directa con el empleado registrado.

4.2 Justificación

La implementación de la codificación en el sistema de control de asistencia se justifica principalmente para proteger la privacidad de los usuarios, asegurando que datos sensibles como el DNI, número telefónico y nombres estén resguardados. Esta medida responde a la necesidad de prevenir el acceso no autorizado y posibles usos indebidos de la información personal, garantizando que solo personas directamente relacionadas con el empleado

registrado puedan acceder a estos datos. De este modo, se preserva la confidencialidad y se cumple con las normativas de protección de datos personales, evitando controversias y posibles daños a los individuos involucrados.

4.3 Alcance

El alcance principal del sistema está limitado únicamente a personas autorizadas que cuenten con un usuario registrado, garantizando así que no haya divulgación o filtración de información.

5. Objetivos

5.1 General

El objetivo principal es un diseño minimalista que proporcione un ambiente amigable, permitiendo al usuario acceder y utilizar el sistema con facilidad, así como ingresar o encontrar rápidamente los datos necesarios.

5.2 Específicos

- Evitar la vulneración de contraseñas mediante ataques de diccionario.
- Prevenir la exposición de datos personales.
- Facilitar la navegación en relación con los usuarios registrados.

6. Referentes teóricos

REQUERIMIENTOS FUNCIONALES

Requerimiento	Descripción	Prioridad
RF1	Autenticación: El sistema debe permitir el inicio de sesión mediante un nombre de usuario y contraseña.	Alta
RF2	Validación de Usuario: El sistema debe verificar la validez del usuario y contraseña utilizando un procedimiento almacenado.	Alta
RF3	Gestión de Empleados: El sistema debe permitir agregar, modificar y eliminar empleados.	Alta
RF4	Visualización de Empleados: El sistema debe listar todos los empleados registrados en un DataGridView.	Alta
RF5	Marcar Asistencia: El sistema debe permitir marcar la asistencia de un empleado usando su nombre y DNI.	Alta

RF6	Listado de Asistencias: El sistema debe mostrar las asistencias marcadas por día.	Alta
RF7	Obtener Faltas: El sistema debe permitir obtener las faltas de los empleados.	Media
RF8	Configuración de Áreas y Puestos: El sistema debe permitir seleccionar áreas y puestos desde un combo box.	Media
RF9	Interacción de UI: El sistema debe proporcionar retroalimentación visual (ej. cambio de color al pasar el mouse) para mejorar la experiencia del usuario.	Media
RF10	Limpiar Campos: El sistema debe permitir limpiar los campos de entrada después de realizar una acción.	Media

DEFINICIÓN, SIGLAS Y ABREVIATURAS

Definiciones:

- **LOGIN:** Proceso de autenticación para usuarios y administradores en el sistema.
- **FrmAsistencias:** Formulario para marcar y gestionar la asistencia de los empleados.
- **FrmEmpleado:** Formulario dedicado a la gestión de información de los empleados, incluyendo agregar, modificar y eliminar registros.
- **FrmEstadísticas:** Formulario que muestra estadísticas relacionadas con la asistencia de los empleados.
- **FrmInicio:** Pantalla principal del sistema que ofrece acceso a diferentes funcionalidades y opciones.
- **FrmLista:** Formulario que muestra una lista de empleados registrados y permite la selección para ver detalles o realizar acciones.

Siglas y Abreviaturas:

- **UI (User Interface):** Interfaz de usuario que debe ser intuitiva y fácil de usar.

- **DB (Database):** Base de datos utilizada por el sistema para almacenar información de los empleados y sus asistencias.
- **CRUD (Create, Read, Update, Delete):** Conjunto de operaciones básicas para la gestión de datos en la base de datos.

DIAGRAMA ENTIDAD-RELACIÓN

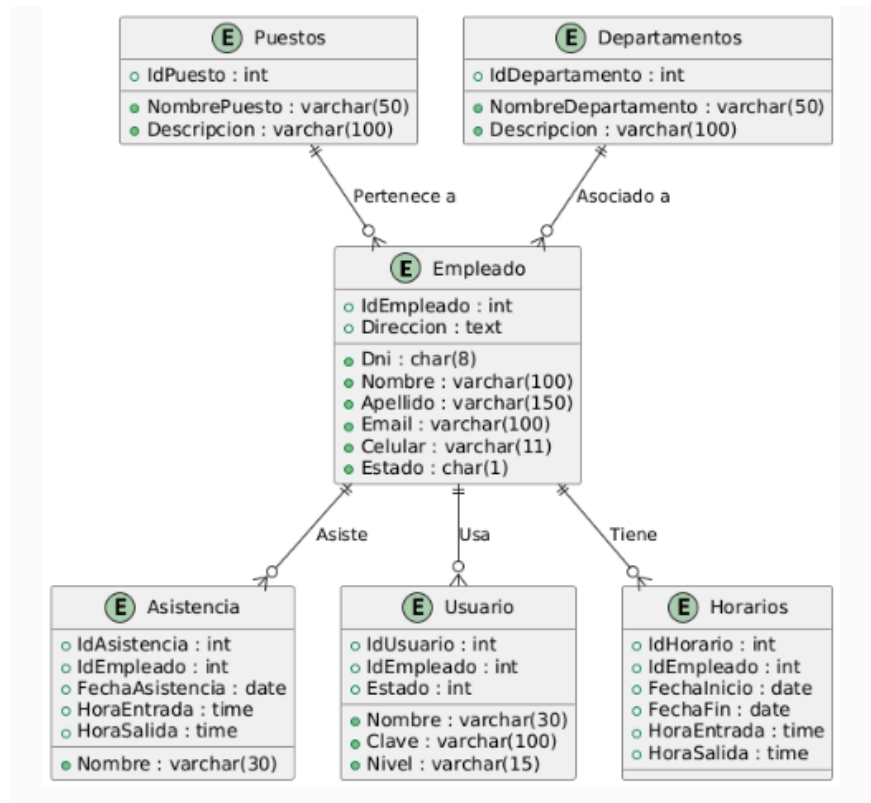


DIAGRAMA DE SECUENCIAS

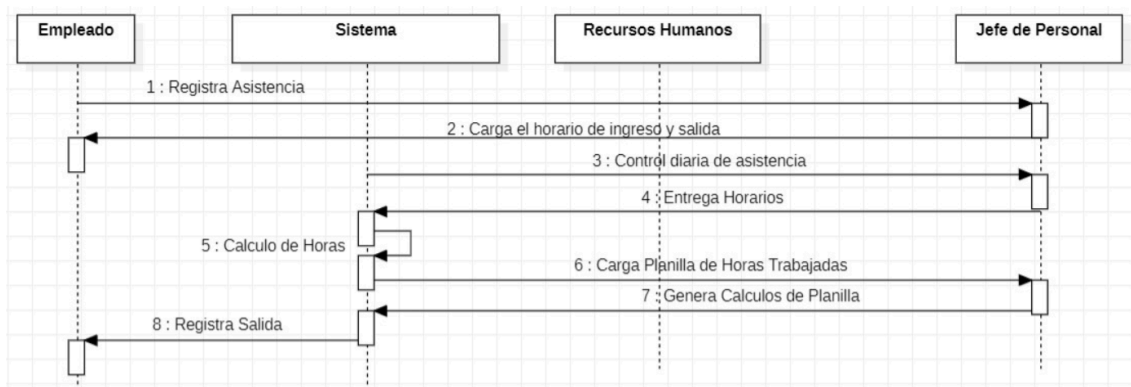


DIAGRAMA DE CASOS DE USO

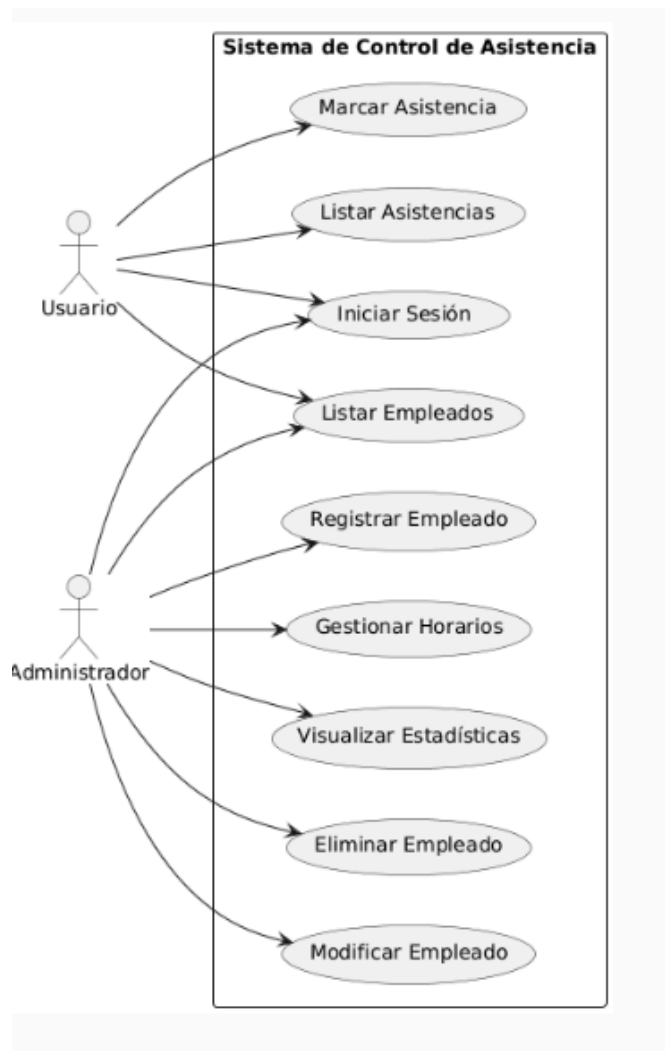


DIAGRAMA DE COMPONENTES

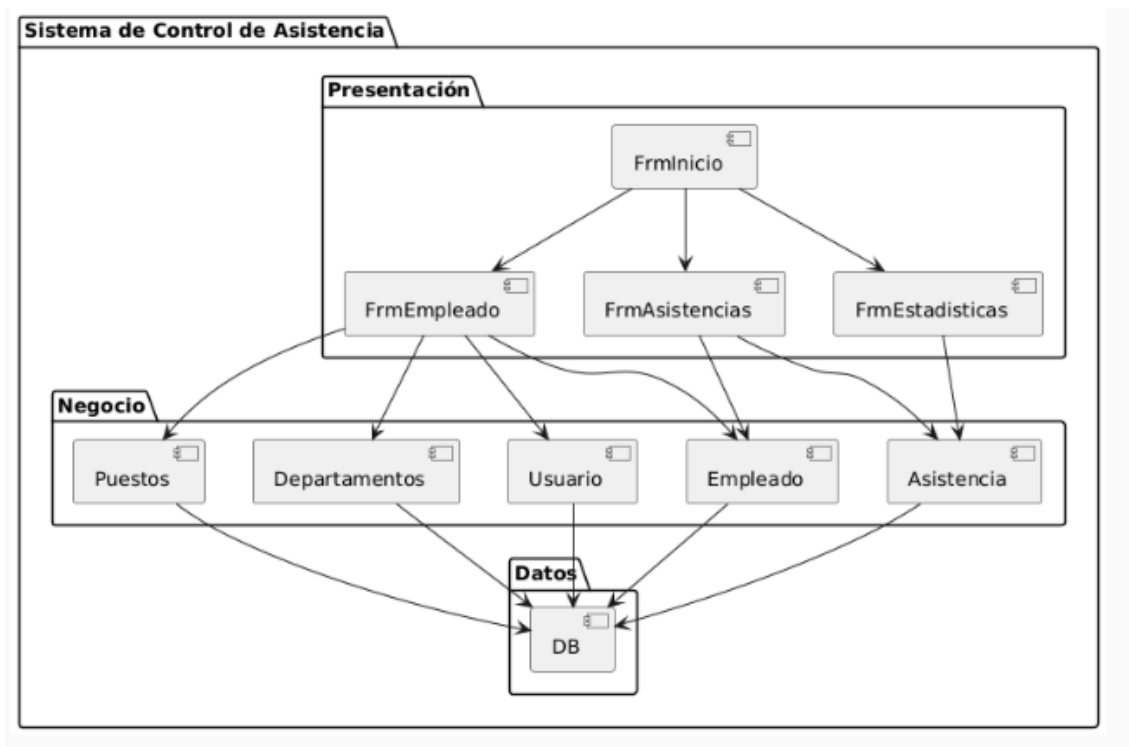
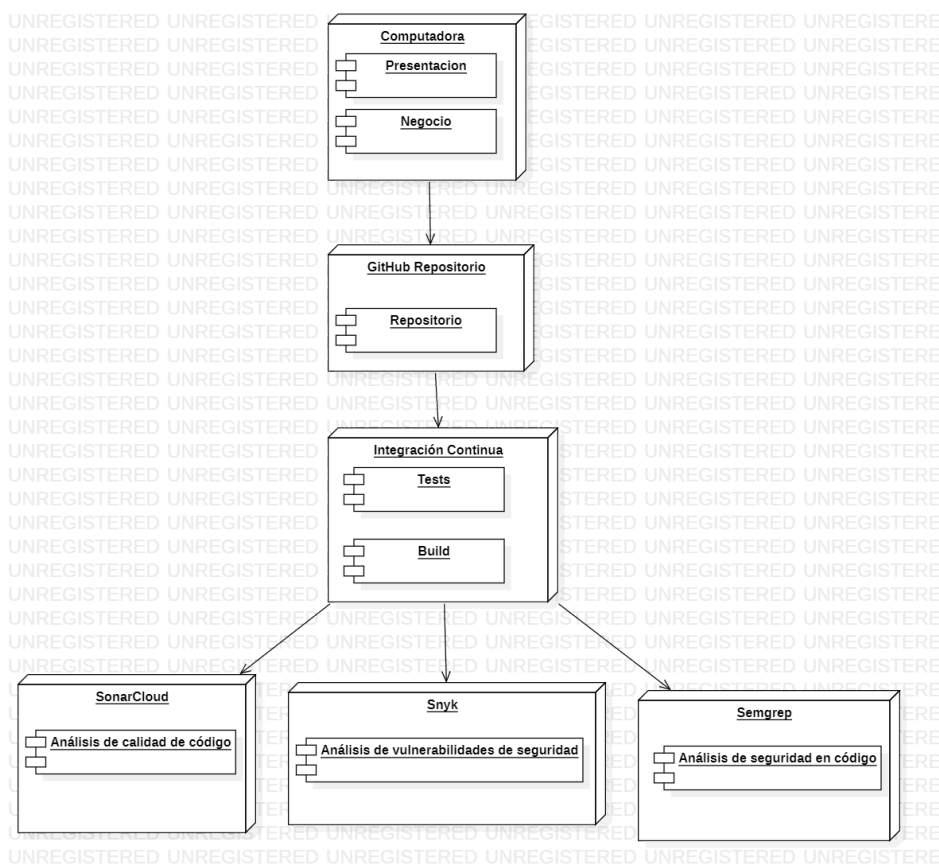


DIAGRAMA DE DESPLIEGUE



7. Desarrollo de la propuesta

SONARCLOUD

ControlAsistencia.WebApi/Program.cs



Intentionality

Await RunAsync instead.

async-await +

Open ▾



AlvaroContreras13 ▾

Reliability ⬆



Code Smell



Major

5min effort

1 day ago

Noncompliant code example

```
public async Task Examples(Stream stream, DbSet<Person> dbSet)
{
    stream.Read(array, 0, 1024);           // Noncompliant
    File.ReadAllLines("path");             // Noncompliant
    dbSet.ToList();                        // Noncompliant in Entity Framework Core queries
    dbSet.FirstOrDefault(x => x.Age >= 18); // Noncompliant in Entity Framework Core queries
}
```

Compliant solution

```
public async Task Examples(Stream stream, DbSet<Person> dbSet)
{
    await stream.ReadAsync(array, 0, 1024);
    await File.ReadAllLinesAsync("path");
    await dbSet.ToListAsync();
    await dbSet.FirstOrDefaultAsync(x => x.Age >= 18);
}
```

Versión Corregida (Con soporte asincrónico):

```
public static async Task Main(string[] args)
{
    var builder = WebApplication.CreateBuilder(args);

    // Add services to the container.
    // Configuración de Swagger/OpenAPI
    builder.Services.AddEndpointsApiExplorer();
    builder.Services.AddSwaggerGen();

    var app = builder.Build();

    // Configurar la tubería de solicitudes HTTP.
    if (app.Environment.IsDevelopment())
    {
        app.UseSwagger();
        app.UseSwaggerUI();
    }

    app.UseHttpsRedirection();

    var summaries = new[]
    {
        "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
    };

    app.MapGet("/weatherforecast", async () =>
    {
        var forecast = Enumerable.Range(1, 5).Select(index =>
            new WeatherForecast
            (
                DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
                Random.Shared.Next(-20, 55),
                summaries[Random.Shared.Next(summaries.Length)]
            ))
            .ToArray();

        // Se utiliza Task.FromResult para simular un resultado asincrónico
        return await Task.FromResult(forecast);
    })
    .WithName("GetWeatherForecast")
    .WithOpenApi();
}
```

Descripción de los cambios:

Cambio de Run() a RunAsync():

- En el código antiguo, el método `app.Run()` se utilizaba de manera sincrónica, lo que no es eficiente para aplicaciones que manejan múltiples solicitudes concurrentes. Ahora, se usa `await app.RunAsync()` para permitir que la aplicación se ejecute de forma asincrónica, mejorando la capacidad de respuesta de la aplicación.

Método principal `async Task Main`:

- El método principal se ha modificado para ser asincrónico, utilizando `async Task Main` en lugar de solo `void Main`. Esto permite el uso de `await` en el código principal del programa, lo que es necesario para las operaciones asincrónicas.

Uso de Task.FromResult en el endpoint:

- El uso de Task.FromResult con await es un patrón que prepara la aplicación para operaciones asincrónicas en endpoints. Aún si el resultado es inmediato, este enfoque permite evitar bloqueos y manejar de forma efectiva operaciones de larga duración, mejorando el rendimiento de la aplicación al escalar con múltiples solicitudes concurrentes.

Adaptability | Not modular

Move 'Program' into a named namespace. [↗](#)

Types should be defined in named namespaces [csharpsquid:S3903](#)

Software qualities impacted: Reliability 

ControlAsistencia.WebApi/Program.cs 

```
1  ac2021... ●2 public class Program
2
3  {
4      public static async Task Main(string[] args)
5      {
6          var builder = WebApplication.CreateBuilder(args);
7      }
8  }
```

Código Corregido:

```
Program.cs X
ControlAsistencia > ControlAsistencia.WebApi > C# Program.cs
1  using Microsoft.AspNetCore.Builder;
2  using Microsoft.Extensions.DependencyInjection;
3
4  namespace ControlAsistencia
5  {
6      public class Program
7      {
8          public static async Task Main(string[] args)
9          {
10             var builder = WebApplication.CreateBuilder(args);
11
12             // Add services to the container.
13             // Configuración de Swagger/OpenAPI
14             builder.Services.AddEndpointsApiExplorer();
15             builder.Services.AddSwaggerGen();
16
17             var app = builder.Build();
18
19             // Configurar la tubería de solicitudes HTTP.
20             if (app.Environment.IsDevelopment())
21             {
22                 app.UseSwagger();
23                 app.UseSwaggerUI();
24             }
25
26             app.UseHttpsRedirection();
27
28             var summaries = new[]
29             {
30                 "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
31             };
32 }
```

Descripción de los cambios:

1. Añadido de un Espacio de Nombres (namespace):

- Se añadió la declaración namespace ControlAsistencia para encapsular la clase Program.

2. Imports Necesarios:

- Se han añadido las declaraciones using para importar los namespaces de Microsoft.AspNetCore.Builder y Microsoft.Extensions.DependencyInjection, que son necesarios para el funcionamiento del código.

Move 'Program' into a named namespace

Código Corregido:

```
{
{
{
// Add services to the container.
// Configuración de Swagger/OpenAPI
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configurar la tubería de solicitudes HTTP.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

var summaries = new[]
{
    "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
};

app.MapGet("/weatherforecast", async () =>
{
    var forecast = Enumerable.Range(1, 5).Select(index =>
        new WeatherForecast
        (
            DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
            Random.Shared.Next(-20, 55),
            summaries[Random.Shared.Next(summaries.Length)]
        ))
        .ToArray();

    // Se utiliza Task.FromResult para simular un resultado asíncrono
    return await Task.FromResult(forecast);
})
.WithName("GetWeatherForecast")
.WithOpenApi();

// Reemplaza Run() con RunAsync para aprovechar la asincronía
await app.RunAsync();
}


// Definición del record para el WeatherForecast sin la propiedad no utilizada
record WeatherForecast(DateOnly Date, int TemperatureC, string? Summary);
```

Descripción de los cambios:

- Eliminación de TemperatureF: Se ha removido la propiedad TemperatureF del record WeatherForecast, ya que no se utilizaba en el código. Esto ayuda a reducir la complejidad y mejora la mantenibilidad del código.


SNYK


Se encontró un error en el SNYK en el SNYK




November 28th 2024, 8:06:15 pm (UTC+00:00)
Source: C:\Users\jange\Documents\Calidad\calidad x2\PROYECTO\snyk\proyecto-si784-2024...

Snyk Code Report

 0 high issues

 0 medium issues

 2 low issues

SCAN COVERAGE
JavaScript files: 15 | .config files: 1 | C# files: 57 | XML files: 3



Snyk test report

November 28th 2024, 7:38:05 pm (UTC+00:00)

Scanned the following path:

- proyectoasistencia-tests ()

0 known vulnerabilities

 |

0 vulnerable dependency paths

 |

dependencies

Path proyectoasistencia-tests

No known vulnerabilities detected.

Controlador EmpleadoController.cs

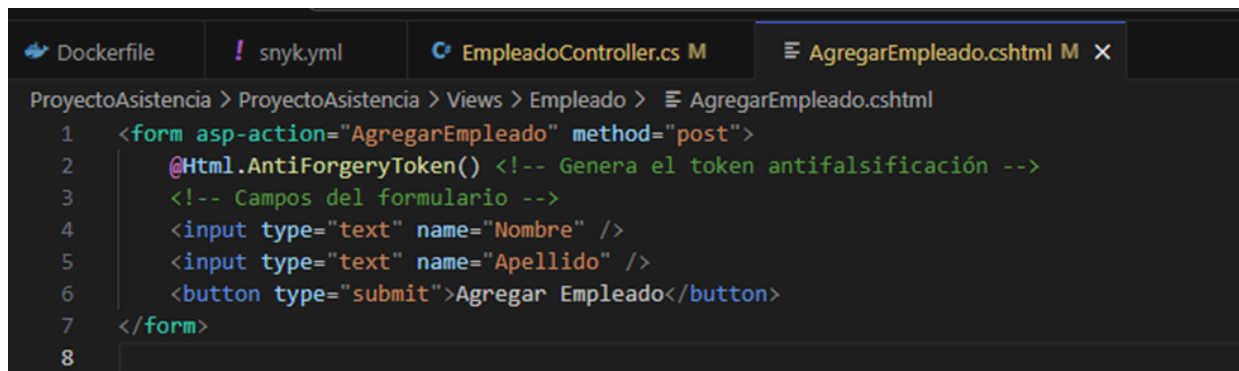
Se utiliza el atributo [ValidateAntiForgeryToken] para proteger la aplicación contra ataques de tipo Cross-Site Request Forgery (CSRF). Este atributo verifica el token antifalsificación generado en la vista antes de procesar la solicitud.

```
8 [HttpPost]
9 [ValidateAntiForgeryToken] // Se añade la validación del token antifalsificación
10 public IActionResult AgregarEmpleado(Empleado empleado)
```

Modificación de Formulario de AgregarEmpleado.cshtml

El formulario utiliza la etiqueta <form> con el atributo asp-action configurado a AgregarEmpleado.

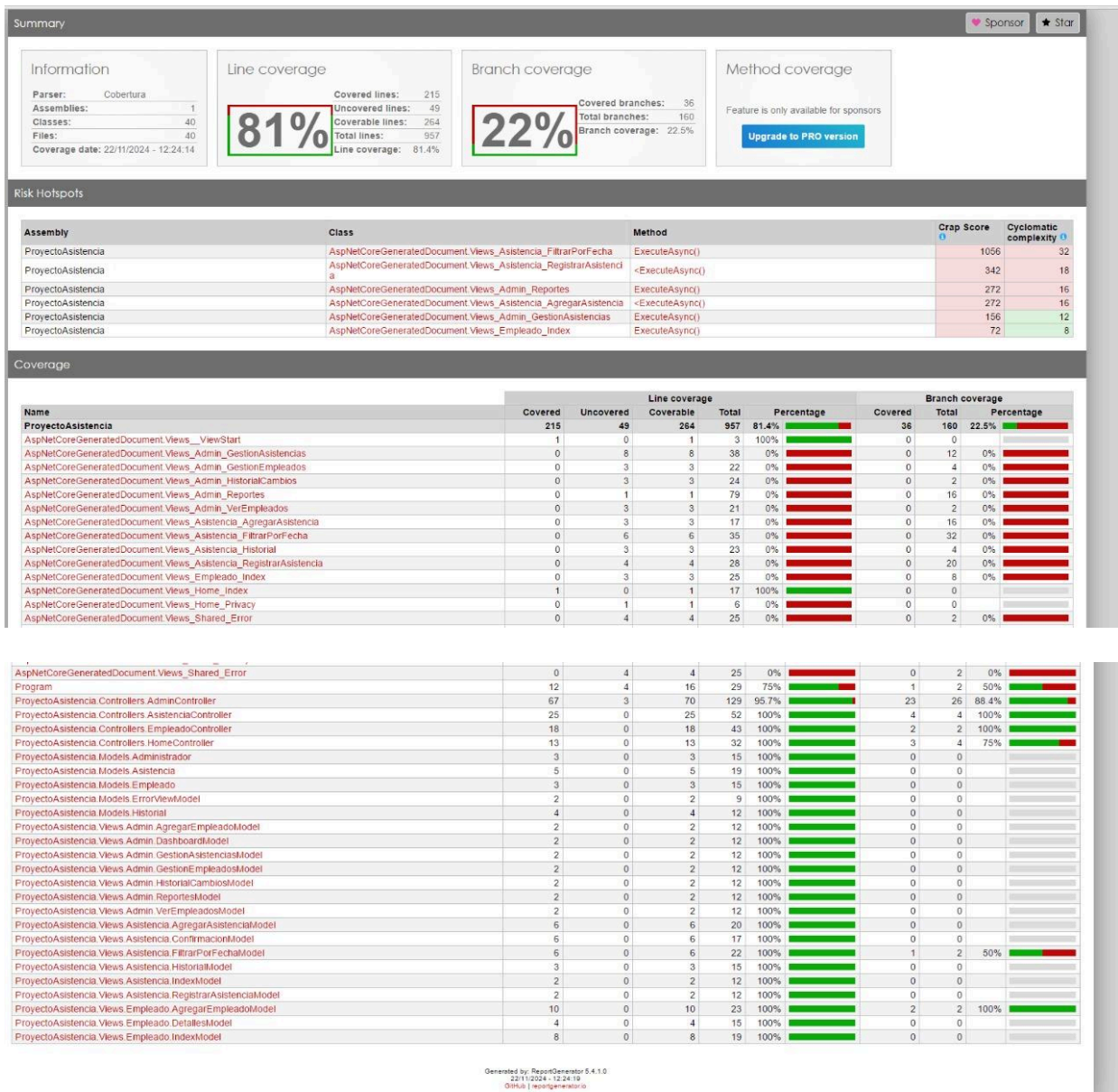
Esto asegura que la solicitud se dirija al método correspondiente en el controlador



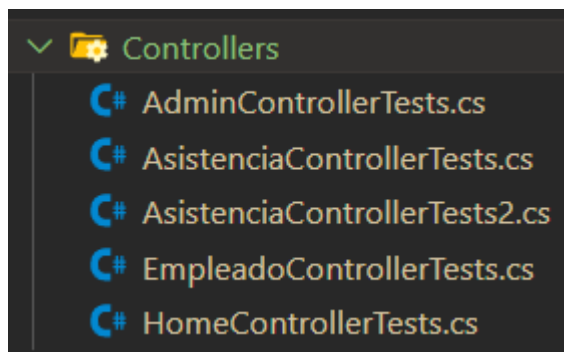
```
Dockerfile  ! snyk.yml  EmpleadoController.cs M  AgregarEmpleado.cshtml M X
ProyectoAsistencia > ProyectoAsistencia > Views > Empleado > AgregarEmpleado.cshtml
1  <form asp-action="AgregarEmpleado" method="post">
2      @Html.AntiForgeryToken() <!-- Genera el token antifalsificación -->
3      <!-- Campos del formulario -->
4      <input type="text" name="Nombre" />
5      <input type="text" name="Apellido" />
6      <button type="submit">Agregar Empleado</button>
7  </form>
8
```


Reporte de Cobertura de Pruebas Unitarias

El objetivo del reporte de cobertura es evaluar qué porcentaje del código fuente ha sido ejecutado durante las pruebas automatizadas, ayudando a identificar áreas no probadas, garantizar la calidad del software, fomentar pruebas más exhaustivas y guiar el mantenimiento del código. Proporciona métricas claras para medir el progreso de las pruebas y priorizar esfuerzos en áreas críticas, reduciendo el riesgo de errores y asegurando que el software sea más robusto y confiable.



Tests Creados



1. HomeControllerTests.cs Mock

Objetivo: Verificar las funcionalidades clave del controlador HomeController.

Cobertura de pruebas:

Método Error: Asegura que el método Error retorna un ViewResult con un modelo de tipo ErrorViewModel y valida el TraceIdentifier proporcionado.

Validación del modelo: Comprueba que el modelo del resultado tiene los datos esperados, lo cual confirma que el controlador procesa correctamente los errores.

2. AdminControllerTests.cs

Objetivo: Probar las funcionalidades principales del controlador AdminController, incluyendo la gestión de empleados, asistencias, historial de cambios y reportes.

Cobertura de pruebas:

Dashboard: Valida que los datos en ViewBag (total de empleados, asistencias, presentes, ausentes) se calculan correctamente en diferentes escenarios.

Gestión de empleados y asistencias: Comprueba que las vistas GestionEmpleados y GestionAsistencias cargan las listas adecuadas de empleados y asistencias respectivamente.

AgregarEmpleado: Verifica tanto los casos válidos como inválidos de agregar un empleado.

Reportes: Asegura que los datos de asistencia por mes, presentes y ausentes son calculados y mostrados correctamente.

3. AsistenciaControllerTests.cs

Objetivo: Validar las acciones del controlador AsistenciaController relacionadas con la gestión de asistencias e historial.

Cobertura de pruebas:

Métodos Index e Historial: Verifica que retornan correctamente una vista.

Registrar asistencia: Comprueba que el registro de asistencia (tanto con solicitudes GET como POST) funciona correctamente y retorna los resultados esperados.

Setup: Asegura que el controlador se inicializa correctamente.

4. EmpleadoControllerTests.cs

Objetivo: Validar las funcionalidades del controlador EmpleadoController, especialmente en la gestión de empleados.

Cobertura de pruebas:

AgregarEmpleado: Valida tanto los casos de éxito como los errores en el registro de empleados (por ejemplo, cuando el modelo es inválido).

VerEmpleados: Comprueba que la lista de empleados se carga correctamente.

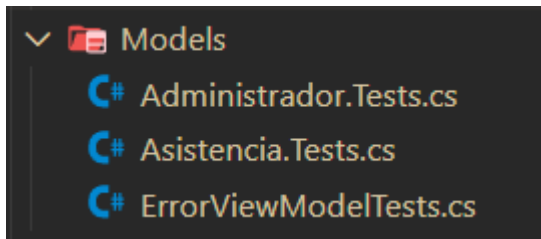
Actualización de la lista: Asegura que, tras agregar un empleado, la lista se actualiza adecuadamente y refleja los cambios esperados.

5. AsistenciaControllerTests2.cs

Objetivo: Validar funcionalidades adicionales del controlador AsistenciaController relacionadas con el manejo de asistencias y vistas complementarias.

Cobertura de pruebas:

- **Index:** Comprueba que el método retorna una vista con el modelo esperado de tipo List<Asistencia>.
- **RegistrarAsistencia (GET):** Valida que se retorna una vista con la lista de empleados en el ViewData.
- **RegistrarAsistencia (POST):**
 - **Caso exitoso:** Verifica que una asistencia se agrega correctamente cuando el empleado existe, redirigiendo al método Historial.
 - **Caso fallido:** Valida que retorna un BadRequest con un mensaje adecuado cuando el empleado no es encontrado.
- **Historial:** Asegura que retorna una vista con un modelo de tipo List<Asistencia>.



1. Administrador.Tests.cs

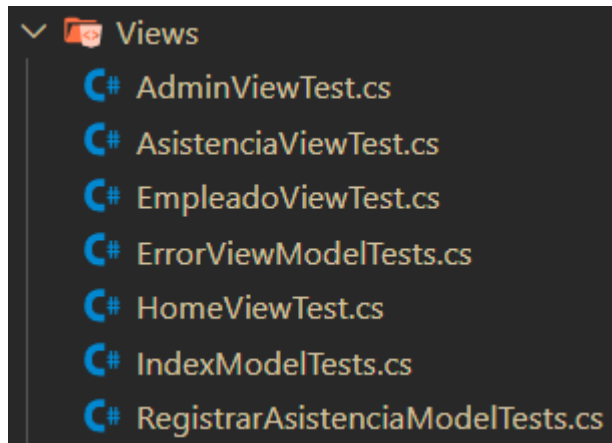
- **Objetivo:** Evaluar las funcionalidades principales del modelo Administrador, verificando que sus propiedades se comporten correctamente y cumplan con las reglas de validación.
- **Cobertura de pruebas:**
 - **Propiedades:** Se verifican los métodos de acceso (get y set) de las propiedades del modelo Administrador.
 - **Validación:** Asegura que los datos cumplen con las restricciones definidas en las anotaciones de validación.
 - **Caso inválido:** Prueba un escenario en el que una propiedad requerida no está configurada, validando que el modelo no sea válido.

2. Asistencia.Tests.cs

- **Objetivo:** Validar la integridad y el comportamiento del modelo Asistencia, asegurando la correcta gestión de sus propiedades y su cumplimiento con las reglas de validación.
- **Cobertura de pruebas:**
 - **Asignación de Propiedades:** Verifica que las propiedades como Id, EmpleadoId, Empleado, Fecha y Estado se asignen y se obtengan correctamente.
 - **Validación Exitosa:** Comprueba que el modelo es válido cuando todas las propiedades requeridas están configuradas adecuadamente.
 - **Caso Inválido:** Valida que el modelo sea inválido cuando una propiedad requerida como Empleado es nula y revisa los mensajes de error generados.

3. ErrorViewModelTests.cs

- **Objetivo:** Evaluar el comportamiento de la propiedad ShowRequestId del modelo ErrorViewModel.
- **Cobertura de pruebas:**
 - **Caso Nulo:** Verifica que ShowRequestId devuelve false cuando RequestId es nulo.
 - **Caso Vacío:** Confirma que ShowRequestId retorna false cuando RequestId está vacío.
 - **Caso Válido:** Asegura que ShowRequestId devuelve true cuando RequestId contiene un valor válido.



1. AdminViewTest.cs

- **Objetivo:** Validar el correcto funcionamiento de las vistas relacionadas con la administración en el proyecto.
- **Cobertura de pruebas:**
 - **AgregarEmpleadoModel:** Verifica que la página se carga sin lanzar excepciones.
 - **DashboardModel:** Valida que la vista del dashboard se cargue correctamente.
 - **GestionAsistenciasModel:** Asegura que la lista de asistencias se gestione correctamente y simula datos para verificar la funcionalidad.
 - **GestionEmpleadosModel:** Comprueba la correcta carga de la vista de gestión de empleados.
 - **HistorialCambiosModel:** Valida la funcionalidad de historial de cambios y su manejo cuando no hay datos.
 - **ReportesModel:** Verifica que la vista de reportes se cargue sin errores.
 - **VerEmpleadosModel:** Comprueba que la lista de empleados se muestre correctamente en la vista y su manejo cuando no hay datos.

2. AsistenciaViewTest.cs

- **Objetivo:** Validar las vistas relacionadas con la gestión de asistencias en el proyecto.
- **Cobertura de pruebas:**
 - **AgregarAsistenciaModel:** Comprueba que la vista de agregar asistencia se cargue correctamente y valida la funcionalidad para agregar datos.
 - **FiltrarPorFechaModel:** Valida que la funcionalidad de filtrado por fecha en la vista opere sin errores.
 - **HistorialModel:** Asegura que la vista del historial se carga correctamente.
 - **ConfirmacionModel:** Verifica que la página de confirmación funcione correctamente, incluyendo la simulación de éxito en la operación.

3. EmpleadoViewTest.cs

- **Objetivo:** Garantizar que las vistas relacionadas con los empleados funcionen correctamente.
- **Cobertura de pruebas:**
 - **AgregarEmpleadoModel:** Valida que la vista se cargue y que los empleados puedan ser añadidos correctamente.
 - **DetallesModel:** Comprueba que los detalles de un empleado se carguen correctamente.
 - **IndexModel:** Valida que la lista de empleados se cargue correctamente en la vista de índice y simula datos para verificar el conteo y contenido.

4. HomeViewTest.cs Mock

- **Objetivo:** Validar las funcionalidades principales de las vistas de inicio y privacidad.
- **Cobertura de pruebas:**
 - **Index:** Comprueba que el método Index devuelve un resultado de tipo ViewResult.
 - **Privacy:** Verifica que el método Privacy también devuelve un resultado de tipo ViewResult.

5. ErrorViewModelTests.cs

Objetivo: Validar la funcionalidad del modelo ErrorViewModel en la gestión de identificadores de solicitud (RequestId).

Cobertura de pruebas:

- **ShowRequestId (true):** Comprueba que ShowRequestId retorna verdadero cuando RequestId no es nulo ni vacío.
- **ShowRequestId (false):** Valida que ShowRequestId retorna falso cuando RequestId es nulo o vacío.

6. IndexModelTests.cs

Objetivo: Validar el correcto funcionamiento de la página de inicio (Index) del módulo de asistencia.

Cobertura de pruebas:

- **OnGet:** Verifica que la ejecución del método OnGet no genera excepciones.

7. RegistrarAsistenciaModelTests.cs

Objetivo: Asegurar que el modelo de la vista para registrar asistencias opera correctamente.

Cobertura de pruebas:

- **OnGet:** Valida que el método OnGet no arroje excepciones durante su ejecución.

Reporte de Pruebas mutantes

All files Stryker.NET Report

All files												
65												
24												
12												
File / Directory	Mutation Score		Killed	Survived	Timeout	No.coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
All files	64.36	73.03	65	24	0	12	20	0	14	65	36	135
Controllers	73.02	85.19	46	8	0	9	13	0	14	46	17	90
Models	77.78	77.78	7	2	0	0	0	0	0	7	2	9
Views	62.50	62.50	10	6	0	0	6	0	0	10	6	22
Program.cs	15.38	20.00	2	8	0	3	1	0	0	2	11	14

- Killed (65): Mutantes eliminados por las pruebas. Tus pruebas lograron detectar los errores introducidos en estas 65 mutaciones.
- Survived (24): Mutantes que sobrevivieron. Esto significa que tus pruebas no detectaron los errores introducidos y no fallaron. Estas son áreas donde tus pruebas necesitan mejorar.
- Timeout (0): No hubo casos en los que las pruebas excedieran el tiempo de ejecución permitido.
- No Coverage (12): Mutantes que no fueron alcanzados por ninguna prueba. Esto indica código no cubierto.
- Ignored (20): Mutantes ignorados por filtros internos (por ejemplo, mutantes en bloques ya cubiertos).
- Compile Errors (14): Mutantes que causaron errores de compilación y no pudieron ser probados.

Controllers Stryker.NET Report



Mutants

Tests

All files / Controllers

65						24				12		
File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
Controllers	73.02	85.19	46	8	0	9	13	0	14	46	17	90
AdminController.cs	71.43	81.08	30	7	0	5	9	0	10	30	12	61
AsistenciaController.cs	60.00	100.00	6	0	0	4	1	0	2	6	4	13
EmpleadoController.cs	100.00	100.00	6	0	0	0	2	0	2	6	0	10
HomeController.cs	80.00	80.00	4	1	0	0	1	0	0	4	1	6

Controllers:

- Puntaje total: 73.02%
- De los cubiertos: 85.19%
- Mutantes totales: 90
- 46 Killed: Buen resultado, pero hay margen de mejora.
- 8 Survived: Estos indican áreas donde las pruebas no detectaron mutaciones.
- 9 No Coverage: Hay partes del código en los controladores no alcanzadas por pruebas.
- 14 Compile Errors: Mutaciones en los controladores causaron errores de compilación.

Models Stryker.NET Report



Mutants

Tests

All files / Models

65						24				12		
File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
Models	77.78	77.78	7	2	0	0	0	0	0	7	2	9
Administrador.cs	0.00	0.00	0	2	0	0	0	0	0	0	2	2
Asistencia.cs	N/A	N/A	0	0	0	0	0	0	0	0	0	0
Empleado.cs	100.00	100.00	2	0	0	0	0	0	0	2	0	2
ErrorViewModel.cs	100.00	100.00	3	0	0	0	0	0	0	3	0	3
Historial.cs	100.00	100.00	2	0	0	0	0	0	0	2	0	2

Models:

- Puntaje total y cubierto: 77.78%
- Mutantes totales: 9
- 7 Killed: La mayoría de las pruebas cubrieron correctamente estas mutaciones.
- 2 Survived: Representan áreas donde las pruebas podrían mejorar.

Views Stryker.NET Report



Mutants Tests

All files / Views

65									24		12	
File / Directory	Mutation Score		Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
	Of total	Of covered										
Views	62.50	62.50	10	6	0	0	6	0	0	10	6	22
Admin	N/A	N/A	0	0	0	0	0	0	0	0	0	0
Asistencia	71.43	71.43	5	2	0	0	2	0	0	5	2	9
Empleado	55.56	55.56	5	4	0	0	4	0	0	5	4	13

Views:

- Puntaje total y cubierto: 62.50%
- Mutantes totales: 22
- 10 Killed: Las pruebas en las vistas están funcionando relativamente bien.
- 6 Survived: Algunas mutaciones no detectadas; se necesita mayor cobertura.
- 3 No Coverage: Hay código en las vistas no alcanzado por pruebas.

Reporte de Pruebas de Aceptación

Propósito General:

Como desarrollador, se busca validar los modelos del proyecto para asegurar que los datos son válidos y cumplen con los requisitos definidos.

+ -	
▼	ProyectoAsistencia.Tests
▼	Features
▼	Validar múltiples escenarios en Home y otras páginas
	Acceso a la página principal
	Acceso a una página inexistente
	Acceso a la página de inicio de sesión
	Envío de formulario sin datos
	Acceso a la página de perfil
	Envío de formulario con datos válidos
	Eliminación de un registro inexistente
	Acceso a la página de configuración
	Actualización de un registro
	Consulta de un registro inexistente
	Consulta de un registro válido
	Acceso a una API pública
	Intento de acceso no autorizado
	Descarga de un archivo
	Acceso a una API protegida sin token

Escenario: Acceso a la página principal

Given: Estoy en el cliente HTTP.
Then: Realizo una petición GET a "/".
Resultado: La página principal fue validada exitosamente, asegurando que responde con el código esperado (200).

Feature: Validar múltiples escenarios en Home y otras páginas

Como administrador
Quiero validar diferentes casos de uso en la aplicación
Para asegurar que el sistema funcione correctamente

Scenario: Acceso a la página principal

Given estoy en el cliente HTTP

When realizo una petición GET a "/"

Then obtengo un código de respuesta exitoso

Escenario: Acceso a una página inexistente

Given: Estoy en el cliente HTTP.
Then: Realizo una petición GET a "/non-existing-page".
Resultado: El sistema devolvió correctamente un código de respuesta 404, confirmando que las rutas inexistentes se manejan apropiadamente.

Feature: Validar múltiples escenarios en Home y otras páginas

Como administrador
Quiero validar diferentes casos de uso en la aplicación
Para asegurar que el sistema funcione correctamente

Scenario: Acceso a una página inexistente

Given estoy en el cliente HTTP

When realizo una petición GET a "/non-existing-page"

Then obtengo un código de respuesta 404

Escenario: Acceso a la página de inicio de sesión

Given: Estoy en el cliente HTTP.

Then: Realizo una petición GET a "/login".

Resultado: La página de inicio de sesión fue validada con éxito, asegurando que responde con el código esperado (200).

Feature: Validar múltiples escenarios en Home y otras páginas

Como administrador
Quiero validar diferentes casos de uso en la aplicación
Para asegurar que el sistema funcione correctamente

Scenario: Acceso a la página de inicio de sesión

Given estoy en el cliente HTTP

When realizo una petición GET a "/login"

Then obtengo un código de respuesta exitoso

Escenario: Envío de formulario sin datos

Given: Estoy en el cliente HTTP.

Then: Realizo una petición POST a "/submit-form" con datos vacíos.

Resultado: El formulario fue correctamente rechazado, devolviendo el código de error esperado (400).

Feature: Validar múltiples escenarios en Home y otras páginas

Como administrador
Quiero validar diferentes casos de uso en la aplicación
Para asegurar que el sistema funcione correctamente

Scenario: Envío de formulario sin datos

Given estoy en el cliente HTTP

When realizo una petición POST a "/submit-form" con datos vacíos

Then obtengo un código de respuesta 400

Escenario: Acceso a la página de perfil

Given: Estoy en el cliente HTTP.

Then: Realizo una petición GET a "/profile".

Resultado: La página de perfil fue validada exitosamente, asegurando que responde con el código esperado (200).

Feature: Validar múltiples escenarios en Home y otras páginas

Como administrador
Quiero validar diferentes casos de uso en la aplicación
Para asegurar que el sistema funcione correctamente

Scenario: Acceso a la página de perfil

Given estoy en el cliente HTTP

When realizo una petición GET a "/profile"

Then obtengo un código de respuesta exitoso

Escenario: Envío de formulario con datos válidos

Given: Estoy en el cliente HTTP.

Then: Realizo una petición POST a "/submit-form" con datos válidos.

Resultado: El formulario fue procesado correctamente, devolviendo un código de respuesta exitoso (200).

Feature: Validar múltiples escenarios en Home y otras páginas

Como administrador
Quiero validar diferentes casos de uso en la aplicación
Para asegurar que el sistema funcione correctamente

Scenario: Envío de formulario con datos válidos

Given estoy en el cliente HTTP

When realizo una petición POST a "/submit-form" con datos válidos

Then obtengo un código de respuesta exitoso

Escenario: Eliminación de un registro inexistente

Given: Estoy en el cliente HTTP.

Then: Realizo una petición DELETE a "/delete/999".

Resultado: La eliminación del registro inexistente fue validada correctamente, devolviendo el código esperado (404).

Feature: Validar múltiples escenarios en Home y otras páginas

Como administrador
Quiero validar diferentes casos de uso en la aplicación
Para asegurar que el sistema funcione correctamente

Scenario: Eliminación de un registro inexistente

Given estoy en el cliente HTTP

When realizo una petición DELETE a "/delete/999"

Then obtengo un código de respuesta 404

Escenario: Acceso a la página de configuración

Given: Estoy en el cliente HTTP.

Then: Realizo una petición GET a "/settings".

Resultado: La página de configuración fue validada exitosamente, asegurando que responde con el código esperado (200).

Feature: Validar múltiples escenarios en Home y otras páginas

Como administrador
Quiero validar diferentes casos de uso en la aplicación
Para asegurar que el sistema funcione correctamente

Scenario: Acceso a la página de configuración

Given estoy en el cliente HTTP

When realizo una petición GET a "/settings"

Then obtengo un código de respuesta exitoso

Escenario: Actualización de un registro

Given: Estoy en el cliente HTTP.

Then: Realizo una petición PUT a "/update/1" con datos válidos.

Resultado: La actualización del registro fue procesada correctamente, devolviendo un código de respuesta exitoso (200).

Feature: Validar múltiples escenarios en Home y otras páginas

Como administrador
Quiero validar diferentes casos de uso en la aplicación
Para asegurar que el sistema funcione correctamente

Scenario: Actualización de un registro

Given estoy en el cliente HTTP

When realizo una petición PUT a "/update/1" con datos válidos

Then obtengo un código de respuesta exitoso

Escenario: Consulta de un registro inexistente

Given: Estoy en el cliente HTTP.

Then: Realizo una petición GET a "/record/999".

Resultado: La consulta del registro inexistente fue validada correctamente, devolviendo el código esperado (404).

Feature: Validar múltiples escenarios en Home y otras páginas

Como administrador
Quiero validar diferentes casos de uso en la aplicación
Para asegurar que el sistema funcione correctamente

Scenario: Consulta de un registro inexistente

Given estoy en el cliente HTTP

When realizo una petición GET a "/record/999"

Then obtengo un código de respuesta 404

Escenario: Consulta de un registro válido

Given: Estoy en el cliente HTTP.

When: Realizo una petición GET a "/record/1".

Then: Obtengo un código de respuesta exitoso.

Resultado: La consulta al registro válido fue realizada correctamente, devolviendo un código 200.

Feature: Validar múltiples escenarios en Home y otras páginas

Como administrador
Quiero validar diferentes casos de uso en la aplicación
Para asegurar que el sistema funcione correctamente

Scenario: Consulta de un registro válido

Given estoy en el cliente HTTP

When realizo una petición GET a "/record/1"

Then obtengo un código de respuesta exitoso

Escenario: Acceso a una API pública

Given: Estoy en el cliente HTTP.

When: Realizo una petición GET a "/api/public".

Then: Obtengo un código de respuesta exitoso.

Resultado: El acceso a la API pública fue validado con éxito, retornando un código de respuesta 200.

Feature: Validar múltiples escenarios en Home y otras páginas

Como administrador
Quiero validar diferentes casos de uso en la aplicación
Para asegurar que el sistema funcione correctamente

Scenario: Acceso a una API pública

Given estoy en el cliente HTTP

When realizo una petición GET a "/api/public"

Then obtengo un código de respuesta exitoso

Escenario: Intento de acceso no autorizado

Given: Estoy en el cliente HTTP.

When: Realizo una petición GET a "/admin".

Then: Obtengo un código de respuesta 403.

Resultado: La prueba confirmó que el acceso no autorizado está protegido, devolviendo el código esperado de prohibición (403).

Feature: Validar múltiples escenarios en Home y otras páginas

Como administrador
Quiero validar diferentes casos de uso en la aplicación
Para asegurar que el sistema funcione correctamente

Scenario: Intento de acceso no autorizado

Given estoy en el cliente HTTP

When realizo una petición GET a "/admin"

Then obtengo un código de respuesta 403

Escenario: Descarga de un archivo

Given: Estoy en el cliente HTTP.

When: Realizo una petición GET a "/download/file".

Then: Obtengo un código de respuesta exitoso.

Resultado: La descarga del archivo fue validada correctamente, con un código de respuesta 200.

Feature: Validar múltiples escenarios en Home y otras páginas

Como administrador
Quiero validar diferentes casos de uso en la aplicación
Para asegurar que el sistema funcione correctamente

Scenario: Descarga de un archivo

Given estoy en el cliente HTTP

When realizo una petición GET a "/download/file"

Then obtengo un código de respuesta exitoso

Escenario: Acceso a una API protegida sin token

Given: Estoy en el cliente HTTP.

When: Realizo una petición GET a "/api/private".

Then: Obtengo un código de respuesta 401.

Resultado: El acceso sin token a la API protegida fue bloqueado correctamente, devolviendo el código de error 401 por falta de autenticación.

Feature: Validar múltiples escenarios en Home y otras páginas

Como administrador
Quiero validar diferentes casos de uso en la aplicación
Para asegurar que el sistema funcione correctamente

Scenario: Acceso a una API protegida sin token

- Given estoy en el cliente HTTP
- When realizo una petición GET a "/api/private"
- Then obtengo un código de respuesta 401

SONAR CLOUD -Reporte de cobertura



Summary Issues Security Hotspots **Measures** Code Activity

the last analysis has warnings. [See details](#)

Uncovered Lines	0
Conditions to Cover	0
Uncovered Conditions	0
Overall Code	
Coverage	93.6%
Lines to Cover	223
Uncovered Lines	10
Line Coverage	95.5%
Conditions to Cover	44
Uncovered Conditions	7

ProyectoAsistencia View as Tree 3 files

Coverage 93.6% [See history](#) **New code:** since not provided

	Coverage	Uncovered Lines	Uncovered Conditions
vs/ProyectoAsistencia/v17.	–	–	–
ProyectoAsistencia	93.6%	10	7
ProyectoAsistencia.Tests	–	–	–

3 of 3 shown

La cobertura de código en **SonarCloud** es una métrica clave que evalúa qué porcentaje del código de un proyecto está siendo probado mediante pruebas automatizadas. Este análisis permite identificar las partes del código que no han sido ejecutadas durante los tests, ayudando a mejorar la calidad del software al reducir errores no detectados.

Cobertura General (Coverage):

- Se alcanzó un **93.6%** de cobertura general, lo que indica que la mayoría del código está siendo verificado mediante pruebas automatizadas.
- Existen **10 líneas** y **7 condiciones no cubiertas**, lo que podría representar áreas del código que aún no han sido probadas.

Line Coverage:

- Específicamente, las líneas de código tienen una cobertura del **95.5%**, lo que significa que casi todas las líneas han sido evaluadas en las pruebas.

Reporte de Semgrep

tool	code	severity	confidence	function	file	line	position	message
semgrep	csharp.dotnet.security.mvc-missing-antiforgery.mvc-missing-antiforgery	WARNING	LOW	unknown	ProyectoAsistencia/Controllers/AdminController.cs	57	10	AgregarEmpleado is a state-changing MVC method that does not validate the antiforgery token or do strict content-type checking. State-changing controller methods should either enforce antiforgery tokens or do strict content-type checking to prevent simple HTTP request types from bypassing CORS preflight controls.
semgrep	csharp.dotnet.security.audit.mass-assignment.mass-assignment	WARNING	MEDIUM	unknown	ProyectoAsistencia/Controllers/AdminController.cs	63	24	Mass assignment or Autobinding vulnerability in code allows an attacker to execute over-posting attacks, which could create a new parameter in the binding request and manipulate the underlying object in the application.
semgrep	csharp.dotnet.security.audit.mass-assignment.mass-assignment	WARNING	MEDIUM	unknown	ProyectoAsistencia/Controllers/AdminController.cs	70	24	Mass assignment or Autobinding vulnerability in code allows an attacker to execute over-posting attacks, which could create a new parameter in the binding request and manipulate the underlying object in the application.
semgrep	csharp.dotnet.security.audit.mass-assignment.mass-assignment	WARNING	MEDIUM	unknown	ProyectoAsistencia/Controllers/AdminController.cs	77	24	Mass assignment or Autobinding vulnerability in code allows an attacker to execute over-posting attacks, which could create a new parameter in the binding request and manipulate the underlying object in the application.
semgrep	csharp.dotnet.security.mvc-missing-antiforgery.mvc-missing-antiforgery	WARNING	LOW	unknown	ProyectoAsistencia/Controllers/AsistenciaController.cs	27	10	RegistrarAsistencia is a state-changing MVC method that does not validate the antiforgery token or do strict content-type checking. State-changing controller methods should either enforce antiforgery tokens or do strict content-type checking to prevent simple HTTP request types from bypassing CORS preflight controls.
semgrep	csharp.dotnet.security.mvc-missing-antiforgery.mvc-missing-antiforgery	WARNING	LOW	unknown	ProyectoAsistencia/Controllers/EmpleadoController.cs	18	10	AgregarEmpleado is a state-changing MVC method that does not validate the antiforgery token or do strict content-type checking. State-changing controller methods should either enforce antiforgery tokens or do strict content-type checking to prevent simple HTTP request types from bypassing CORS preflight controls.
semgrep	csharp.dotnet.security.audit.mass-assignment.mass-assignment	WARNING	MEDIUM	unknown	ProyectoAsistencia/Controllers/EmpleadoController.cs	28	20	Mass assignment or Autobinding vulnerability in code allows an attacker to execute over-posting attacks, which could create a new parameter in the binding request and manipulate the underlying object in the application.
semgrep	javascript.lang.security.audit.detect-non-literal-regexp.detect-non-literal-regexp	WARNING	LOW	unknown	ProyectoAsistencia/wwwroot/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js	349	17	RegExp() called with a 'params' function argument, this might allow an attacker to cause a Regular Expression Denial-of-Service (ReDoS) within your application as RegExp blocks the main thread. For this reason, it is recommended to use hardcoded regexes instead. If your regex is run on user-controlled input, consider performing input validation or use a regex checking/sanitization library such as https://www.npmjs.com/package/recheck to verify that the regex does not appear vulnerable to ReDoS.

- Total de Hallazgos: 8
- Avisos de Baja Severidad (LOW): 3
- Avisos de Media Severidad (MEDIUM): 5
- Avisos de Alta Severidad (HIGH): 0
- Errores de Confianza Desconocida: 8

1. Falta de Protección Contra Ataques CSRF (LOW)

Se ha detectado que algunos métodos de la aplicación, como AgregarEmpleado y RegistrarAsistencia, no están validando correctamente que las solicitudes provengan de fuentes legítimas. Esto crea una vulnerabilidad a ataques CSRF (Cross-Site Request Forgery), donde un atacante podría modificar datos o realizar acciones no autorizadas sin ser detectado.

Ubicación del problema:

- AdminController.cs, línea 57
- AsistenciaController.cs, línea 27
- EmpleadoController.cs, línea 18

2. Vulnerabilidad por Exposición de Datos (MEDIUM)

Se ha identificado una vulnerabilidad de over-posting en varios métodos, lo que permite que los parámetros de las solicitudes modifiquen más datos de los que deberían. Esto permite a un atacante agregar o manipular datos internos de la aplicación, lo que podría causar alteraciones no deseadas.

Ubicación del problema:

- AdminController.cs, líneas 63, 70, 77
- EmpleadoController.cs, línea 28

3. Riesgo de Colapso por Expresión Regular (LOW)

Se encontró un uso ineficiente de expresiones regulares en el código JavaScript, específicamente en el archivo `jquery.validate.unobtrusive.js`. Esto podría generar un problema de ReDoS (Regular Expression Denial of Service), donde un atacante puede provocar un uso excesivo de recursos al enviar entradas diseñadas para colapsar la aplicación.

Ubicación del problema:

- `jquery.validate.unobtrusive.js`, línea 349