



**IES HENRI MATISSE**

DAM

**PFG**

**EDUAVÍS**

presenta:

Rafael López Moraleda

Tutor PFG

Chema Parra Mahugo

Valencia 2025

# INDICE

DESCRIPCIÓN DEL PROBLEMA _____	4
SOLUCIÓN PROPORCIONADA _____	6
HERRAMIENTAS UTILIZADAS _____	8
BASE DE DATOS _____	10
Estructura del proyecto _____	12
Organización de la aplicación _____	12
Explicación de las clases _____	13
Interfaz gráfica _____	48
Utilización del programa _____	54
CONCLUSIONES _____	71
Dificultades Encontradas _____	71
Ampliaciones a futuro _____	71
Conclusiones Personales _____	72



## DESCRIPCIÓN DEL PROBLEMA

En muchos centros educativos se gestionan cientos de incidencias relacionadas con el comportamiento de los estudiantes, pero este proceso suele realizarse de forma manual, poco estructurada y con escasa trazabilidad. Los profesores anotan las incidencias en papel o a través de formularios sin un sistema centralizado, lo que provoca pérdida de información, duplicidad de registros y dificultad para hacer un seguimiento real del comportamiento de un alumno a lo largo del curso.

Además, no existe una herramienta visual e intuitiva que permita a los docentes registrar, consultar o modificar incidencias de forma sencilla, ni tampoco una forma automatizada de generar informes, gráficas u obtener estadísticas que ayuden a tomar decisiones.

En este contexto, también se presentan dificultades para coordinar al profesorado implicado, y no siempre es fácil comunicar o escalar las sanciones dentro del equipo docente o hacia la dirección del centro. Todo esto repercute negativamente en la eficacia del sistema de convivencia y disciplina del centro educativo.



## SOLUCIÓN PROPORCIONADA

Para resolver los problemas detectados en la gestión de incidencias escolares, se ha desarrollado EduAvís, una aplicación de escritorio que centraliza todo el proceso de registro, visualización y seguimiento de incidencias en un entorno intuitivo y accesible para el profesorado.

La solución consiste en una plataforma con un diseño pensado para ser utilizado de forma cómoda por usuarios no técnicos. La herramienta permite registrar nuevas incidencias de forma rápida, asociarlas directamente a alumnos y profesores, clasificarlas según su gravedad, y almacenarlas de forma persistente en una base de datos.

Además, el sistema ofrece filtros avanzados para buscar incidencias, gráficas interactivas para visualizar datos estadísticos (como el número de sanciones o avisos por grupo o por fecha), y un sistema de comunicación interna entre módulos. También incorpora una interfaz amigable con botones accesibles.

Con esta solución, se mejora la organización, el control y la eficiencia en la gestión de la convivencia en el aula, permitiendo a los docentes centrarse más en la parte pedagógica que en la burocrática.



## HERRAMIENTAS UTILIZADAS

Para el desarrollo de EduAvis he utilizado una serie de herramientas y tecnologías que me han permitido construir una aplicación robusta, funcional y moderna. A continuación detallo las principales:

- **Visual Studio 2022:** fue el entorno de desarrollo principal que utilicé durante todo el proyecto. Me permitió trabajar cómodamente con XAML, C# y la integración con Entity Framework.



- **.NET (WPF):** empleé Windows Presentation Foundation para diseñar la interfaz gráfica de la aplicación. Esta tecnología me permitió construir ventanas modernas y responsivas adaptadas a distintos tamaños de pantalla.



- **Entity Framework Core:** lo utilicé como ORM para comunicar la aplicación con la base de datos de manera más sencilla y estructurada, sin tener que escribir directamente consultas SQL.
- **MySQL:** usé esta base de datos local para almacenar todos los datos.












- **MahApps.Metro y MaterialDesignInXAML:** estas dos librerías me ayudaron a darle un aspecto más moderno y profesional a la interfaz gráfica, con componentes estilizados como botones, cuadros de texto, pestañas y ventanas.
- **LiveCharts:** esta librería la utilicé para crear gráficas interactivas y mostrar de forma visual la evolución de las incidencias y las estadísticas del centro en el dashboard.
- **Newtonsoft.Json:** la utilicé para manejar los json de manera más eficiente



- **WpfAnimatedGif**: esta librería la utilicé para que la SplashScreen tuviese una animación de un libro haciendo énfasis que es una aplicación destinada al entorno de la educación y le da un toque más moderno que la barra predeterminada de wpf.

Estas herramientas me permitieron cubrir tanto la parte visual como la lógica y la persistencia de datos de forma eficiente, facilitando el desarrollo y mantenimiento del proyecto.

Aquí muestro una foto de mi NuGet:

	<b>MaterialDesignThemes.MahApps</b> por James Willock ResourceDictionary instances containing Material Design templates and styles for WPF controls in the MahApps library.	3.1.0 5.2.1
	<b>Microsoft.EntityFrameworkCore</b> por Microsoft Entity Framework Core is a modern object-database mapper for .NET. It supports LINQ queries, change tracking, updates, and schema migrations. EF Core works with SQL Server, Azure SQL Database, SQLite, Azure Cosmos DB, MySQL, PostgreSQL, and other databases through a provider plugin API.	8.0.11 9.0.5
	<b>Microsoft.EntityFrameworkCore.Design</b> por Microsoft Shared design-time components for Entity Framework Core tools.	8.0.10 9.0.5
	<b>Microsoft.EntityFrameworkCore.Proxies</b> por Microsoft Lazy loading proxies for Entity Framework Core.	8.0.10 9.0.5
	<b>MySQL.EntityFrameworkCore</b> por Oracle Corporation MySQL.EntityFrameworkCore adds support for Microsoft Entity Framework Core.	9.0.0 9.0.3
	<b>Newtonsoft.Json</b> por James Newton-King Json.NET is a popular high-performance JSON framework for .NET	13.0.3
	<b>NLog</b> por Jarek Kowalski, Kim Christensen, Julian Verdurmen NLog is a logging platform for .NET with rich log routing and management capabilities. NLog supports traditional logging, structured logging and the combination of both.	5.4.0 5.5.0
	<b>Pomelo.EntityFrameworkCore.MySql</b> por Laurents Meyer, Caleb Lloyd, Yuko Zheng Pomelo's MySQL database provider for Entity Framework Core.	8.0.0 8.0.3
	<b>WpfAnimatedGif</b> por Thomas Levesque A library to display animated GIF images in WPF	2.0.0 2.0.2

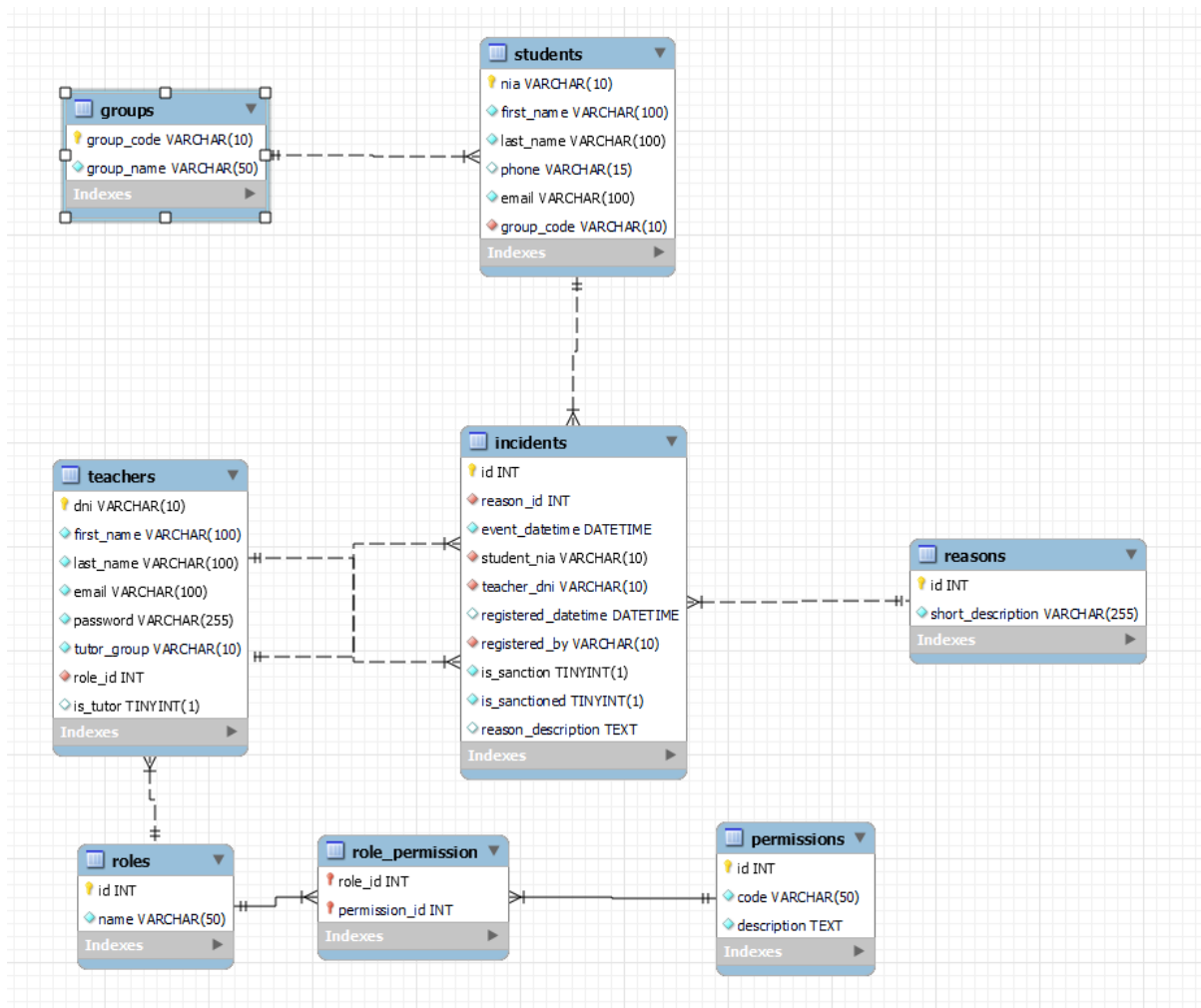
## BASE DE DATOS

La base de datos de EduAvis ha sido diseñada para almacenar y gestionar toda la información necesaria para el funcionamiento del sistema. Se compone de un total de 8 tablas relacionadas entre sí, y su estructura permite un control detallado sobre los usuarios, grupos, incidencias y permisos.

Entre las tablas principales se encuentran:

- **Students:** guarda los datos personales de los alumnos, incluyendo nombre, apellidos, grupo al que pertenecen, email y teléfono.
- **Teachers:** almacena la información de los profesores, con campos como el DNI, nombre, apellidos, email, contraseña, grupo, si son tutores principales o no y rol
- **Groups:** tabla sencilla que contiene los diferentes grupos o clases del centro.
- **Incidents:** es una de las tablas más importantes. Guarda todas las incidencias registradas por los profesores, relacionando cada una con un alumno, un motivo, un profesor responsable, y datos como la fecha, si ha sido informada o sancionada y por el profesor que ha sido escrita que no tiene por qué ser el mismo que está implicado.
- **Reasons:** define los distintos motivos por los cuales se puede registrar una incidencia.
- **Roles:** es una tabla donde almaceno los distintos roles que existen.
- **Permissions:** guardo los distintos permisos existentes
- **Role\_permission:** Es una tabla intermedia entre roles y permissions porque cada role puede tener varios permisos asignados y los permisos se pueden asignar a cualquier rol.

Esta base de datos está normalizada y diseñada para evitar la redundancia de información. Además, gracias a la integración con Entity Framework, es posible trabajar con ella desde el código de forma eficiente y estructurada, mediante modelos y relaciones claramente definidas.



# Estructura del proyecto

## Organización de la aplicación

El proyecto EduAvis ha sido organizado siguiendo una estructura modular y coherente, lo que facilita tanto el mantenimiento como la escalabilidad de la aplicación. Esta estructura se ha diseñado pensando en la separación de responsabilidades, respetando los principios de la arquitectura MVVM (Model-View-ViewModel) y permitiendo una navegación clara y eficiente por los distintos componentes del sistema.

A continuación se describe cada uno de los módulos principales y su función:

### 1. Backend

Contiene la lógica relacionada con la base de datos y los modelos de datos. Aquí se definen las entidades que se corresponden con las tablas de la base de datos, utilizando Entity Framework como ORM.

**Model:** Incluye todas las clases que representan las entidades del sistema, como Student, Teacher, Incident, Group, Permission, etc. Estas clases están decoradas con anotaciones de datos (DataAnnotations) para mapear correctamente los atributos y relaciones.

### 2. Frontend

Incluye todo lo relacionado con la interfaz de usuario (UI).

**Dialog:** Contiene las ventanas (modales) utilizadas.

**UC:** Contiene los distintos controles reutilizables de la interfaz, como UCIncidents, UCDashboard, UCAdministration, entre otros. Cada uno representa una sección funcional específica.

### 3. MVVM

Este módulo contiene toda la lógica relacionada con la implementación del patrón MVVM. Cada componente visual tiene asociado un ViewModel que contiene la lógica de presentación y vinculación de datos.

**Base:** Incluye clases base para los ViewModels, como MVBaseCRUD o MVBase, que proporcionan funcionalidades comunes.

#### 4. Resource

Este módulo se utiliza para agrupar recursos adicionales utilizados por la interfaz de usuario.

**Image:** Carpeta donde se almacenan imágenes y logotipos utilizados dentro del programa.

**Utiles:** Contiene utilidades y funciones utilizadas en el programa como EmailService que se encarga de enviar las notificaciones por correo electrónico o SessionManager que se encarga de almacenar todos los roles que tiene el usuario al iniciar sesión.

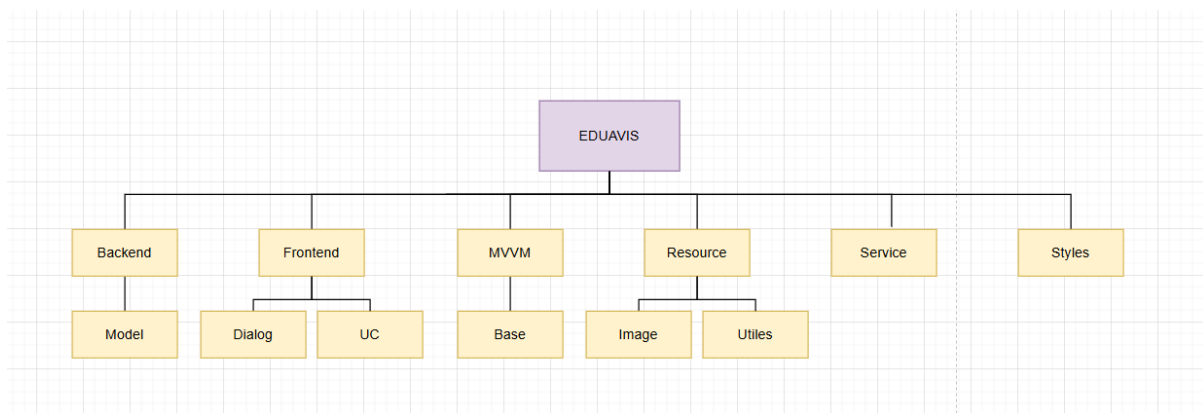
#### 5. Service

Agrupar todas las clases que se encargan de proporcionar servicios reutilizables, teniendo un GenericService que es un servicio genérico que permite heredar las funcionalidades CRUD a los distintos servicios, donde algunos tienen funciones específicas.

#### 6. Styles

Contiene algunos de los estilos que se utilizan en la aplicación, no todos ya que fue implementado a posteriori.

Aquí un esquema visual de la estructura:



Esta organización ha permitido trabajar de forma ordenada, facilitar el testing, y asegurar que cada módulo cumpla una función específica sin acoplamientos innecesarios.

### Explicación de las clases

Ahora voy a explicar todas las clases del proyecto.

Backend

Model

Toda esta carpeta ha sido generada con el Entity Framework, por eso no hago mucho hincapié, tiene algunos ligeros cambios que he hecho yo.

### **BdeduavisContext**

Esta clase es la que se encarga de conectar todo el proyecto con la base de datos. Básicamente, actúa como un puente entre el código y MySQL. Dentro de ella se definen todas las tablas que vamos a usar, como los profesores, alumnos, roles, permisos, grupos, etc.

En resumen, sin esta clase no se podría trabajar con los datos desde el código de forma sencilla, ya que es la que usa Entity Framework

### **Group**

Esta clase representa los grupos que hay en el centro, por ejemplo “1ºA” o “2ºB”. Cada grupo tiene un código y un nombre. Aparte, está relacionada con dos cosas importantes: los alumnos que pertenecen a ese grupo y los profesores que lo tutorizan.

### **Incident**

Esta clase es la que representa los partes o incidencias que se registran en el sistema. Cada vez que un profesor crea un parte, se guarda como un objeto de esta clase.

Está conectada con otras entidades del sistema como Teacher, Student y Reason, para que desde el parte podamos acceder a toda esa información directamente. Además, los campos `isSanction` e `IsSanctioned` nos ayudan a saber si el incidente conlleva sanción y si ya ha sido informado.

### **Permission**

Esta clase representa los permisos que puede tener cada usuario. Cada permiso tiene un código único que sirve para identificarlo en el sistema, y una descripción que explica para qué sirve ese permiso. Además, está relacionada con los roles, ya que un permiso puede estar asignado a varios roles, y así podemos controlar qué puede hacer cada tipo de usuario según el rol que tenga asignado.

### **Reason**

Esta clase guarda los distintos motivos por los que se puede registrar una incidencia. Cada motivo tiene un ID y una descripción corta que lo identifica. Además, está relacionada con las incidencias, ya que un mismo motivo puede estar asociado a varias de ellas. Esto nos permite tener organizado el porqué de cada incidente registrado.

## **Role**

Esta clase representa los distintos roles que pueden tener los profesores dentro del sistema, como por ejemplo administrador o usuario normal. Cada rol tiene un nombre único y puede estar asignado a varios profesores. Además, cada rol puede tener diferentes permisos asociados, lo que nos permite controlar qué acciones puede hacer cada uno según el rol que tenga.

## **RolePermission**

Esta clase sirve como tabla intermedia para unir los roles con los permisos. Gracias a ella podemos asignar varios permisos a un mismo rol y, al mismo tiempo, un permiso puede estar en diferentes roles. Solo tiene dos campos: el ID del rol y el ID del permiso, y se encarga de enlazarlos correctamente.

## **Student**

Esta clase representa a cada alumno del sistema. Cada estudiante tiene un NIA único, nombre, apellidos, teléfono, correo y el grupo al que pertenece.

Además, está conectada con la clase Group, lo que permite saber de qué curso es cada alumno, y también con Incident, ya que un alumno puede tener varios partes registrados.

## **Teacher**

Esta clase representa a un profesor del sistema. Cada uno tiene un DNI único, nombre, apellidos, correo, contraseña, grupo asignado, y el rol que se le ha dado (por ejemplo, administrador o profesor normal).

Además, el profesor puede estar relacionado con varios incidentes, tanto como quien lo registra como si es el implicado.

## **Login**

La pantalla de login es la primera vista que aparece al ejecutar la aplicación. Está pensada para que los profesores accedan a sus funciones dentro de la plataforma. El formulario pide el correo electrónico y la contraseña del profesor.

```

private void btnLogin_Click(object sender, RoutedEventArgs e)
{
    string email = txtEmail.Text;
    string password = passwordBoxPassword.Password;

    if (string.IsNullOrEmpty(email) || string.IsNullOrEmpty(password))
    {
        MessageBox.Show("Please enter both email and password.", "Validation Error", MessageBoxButton.OK, MessageBoxImage.Warning);
        return;
    }

    // Incluir Role y sus RolePermissions y Permission en la misma consulta
    var teacher = _context.Teachers
        .Include(t => t.Role)
        .ThenInclude(r => r.RolePermissions)
        .ThenInclude(rp => rp.Permission)
        .FirstOrDefault(t => t.Email == email);

    if (teacher == null || teacher.Password != password)
    {
        MessageBox.Show("Invalid email or password.", "Login Failed", MessageBoxButton.OK, MessageBoxImage.Error);
        return;
    }

    if (chkRememberMe.IsChecked == true)
        SecureStorage.SaveCredentials(email, password);
    else
        SecureStorage.ClearCredentials();

    SessionManager.CurrentUser = teacher;

    SessionManager.Roles = teacher.Role.RolePermissions
        .Select(rp => rp.Permission.Code)
        .ToList();

    MainWindow mainWindow = new MainWindow(_context, teacher);
    mainWindow.Show();
    this.Close();
}

```

El método btnLogin se encarga de validar los datos introducidos por el usuario y permitir el acceso al sistema si todo es correcto. Esta función se ejecuta cuando el usuario pulsa el botón "SIGN IN".

Primero, recojo el correo y la contraseña que ha escrito el usuario en los campos del formulario. Si alguno de los dos está vacío, se muestra un mensaje de advertencia pidiendo que se rellenen ambos campos antes de continuar. De este modo, evito que se intente hacer una consulta innecesaria a la base de datos.

Después, busco en la base de datos si existe un profesor con ese correo electrónico.

Una vez encontrado el profesor, comparo su contraseña con la que se ha introducido en el formulario. Si no coincide o no existe el usuario, se muestra un mensaje de error. En caso contrario, si todo está bien, se guarda el usuario en la sesión actual mediante SessionManager.CurrentUser y también se cargan los permisos que le corresponden, que luego se usarán para limitar o permitir el acceso a ciertas funcionalidades.

También tengo en cuenta si el usuario ha marcado la opción de "Recordarme". En ese caso, guardo sus credenciales de forma segura con la clase SecureStorage. Si no lo marca, me aseguro de borrar cualquier dato anterior guardado.



Por último, si el login es exitoso, abro la ventana principal (MainWindow) pasándole tanto el contexto de la base de datos como el profesor logueado. Y cierro la ventana de login para que no siga abierta en segundo plano.

## SplashScreen

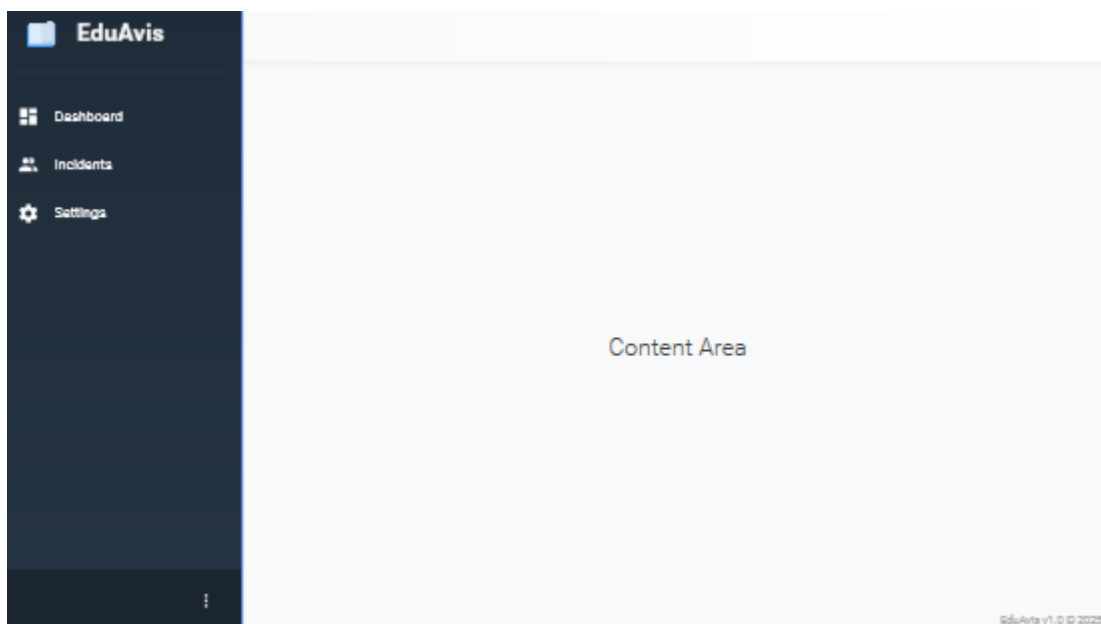
La pantalla *SplashScreen* se muestra al iniciar la aplicación y sirve como presentación visual mientras se comprueba la conexión con la base de datos. Si todo va bien, se revisan posibles credenciales guardadas para iniciar sesión automáticamente. En caso contrario, se abre la ventana de login. Esta lógica permite mejorar la experiencia de usuario y centralizar el acceso desde un único punto de entrada.

Tiene una espera artificial que no hace falta, pero si no la ponía no tenía sentido una *SplashScreen*, y lo más importante, da sensación al usuario de aplicación de más calidad.

Ahora voy a explicar el *MainWindow* aunque no esté en la carpeta Frontend para que tenga más sentido.

## MainWindow

La ventana principal de la aplicación es la encargada de gestionar toda la navegación y el flujo interno entre los distintos módulos del sistema. Su funcionamiento se basa en la carga dinámica de pantallas (UserControls) dentro de un contenedor central, y todo está gestionado mediante un sistema MVVM que permite mantener separada la lógica de cada módulo.



Al iniciarse la ventana, se cargan los tres principales *ViewModels*: el de incidentes (*MVIncident*), el del panel principal (*MVDashboard*) y el de administración

(*MVAdministration*). Cada uno de estos modelos recibe como parámetro el contexto de base de datos y el profesor que ha iniciado sesión. Esto permite que cada módulo tenga acceso a los datos necesarios y que estén personalizados según el usuario actual.

En el evento `MainWindow_Loaded`, estos modelos se inicializan mediante sus métodos `Start()`, que normalmente sirven para preparar la lógica interna o precargar datos si es necesario. Después de esto, se abre automáticamente la pantalla del dashboard como vista inicial.

```
1 referencia
private async void MainWindow_Loaded(object sender, RoutedEventArgs e)
{
    mvIncident = new MVIncident(_context, _teacher);
    await mvIncident.Start();

    mvDashboard = new MVDashboard(_context, _teacher);
    await mvDashboard.Start();

    mvAdministration = new MVAdministration(_context, _teacher);
    await mvAdministration.Start();

    btnDashboard_Click(this, new RoutedEventArgs());

    if (SessionManager.HasPermission(PermissionCodes.ManageRoles))
    {
        btnAdministration.Visibility = Visibility.Visible;
    }
}
```

La navegación entre secciones (como incidentes, administración o ajustes) se realiza mediante botones, y cada uno de ellos llama a un método que instancia el `UserControl` correspondiente. Luego, se llama a un método común llamado `NavigateToAsync`, que se encarga de limpiar el panel central y cargar ahí la nueva pantalla. Si esa pantalla implementa una interfaz de refresco (`IRefreshable`), se llama automáticamente a su método `Refresh()` para que actualice los datos.

```
4 referencias
private async Task NavigateToAsync(UserControl uc)
{
    mainPanel.Children.Clear();
    mainPanel.Children.Add(uc);

    if (uc is IRefreshable refreshable)
    {
        await refreshable.Refresh();
    }
}
```

Otro aspecto importante es la gestión de permisos. Al cargarse la ventana, se comprueba si el usuario tiene permisos especiales (por ejemplo, para ver la parte de administración). Esto se hace a través de un gestor de sesión que guarda los roles del usuario. Si el permiso está presente, se activa la opción correspondiente del menú.

También se incluye la lógica para cerrar sesión. Cuando el usuario pulsa en "Logout", se limpian los datos de sesión, las credenciales guardadas y se redirige automáticamente a la pantalla de login, cerrando la ventana principal.

En resumen, esta clase es la que controla el flujo completo de la aplicación una vez se ha iniciado sesión. Gestiona los módulos disponibles, la navegación entre ellos, la seguridad según los permisos, y el refresco de los datos cuando se entra en cada pantalla.

Ahora voy a hablar de los Diálogos que son todos iguales, solo cambian dependiendo del modelo, teniéndose que adaptar y el de *ModifyRole* que hablaré más en profundidad.

Las clases: *AddNewIncident*, *AddStudent*, *AddTeacher*, *ModifyIncident*, *ModifyStudent* y *ModifyTeacher*, son dialogos que se conectan con su respectivo ViewModel y trabajan con un objeto publico dentro del ViewModel,. Todos usan metodos similares solo cambiando el servicio que usan. Por ejemplo:

```
public async Task<bool> saveStudent()
{
    try
    {
        bool niaExists = await _studentService.NiaExistsAsync(Student.Nia);
        if (niaExists)
        {
            MessageBox.Show("A student with this NIA already exists.", "Duplicate NIA", MessageBoxButton.OK, MessageBoxImage.Warning);
            return false;
        }

        bool correct = await _studentService.AddAsync(Student);

        if (correct)
        {
            Student = new Student();
            await Start();
            SelectedRole = null;
        }

        return correct;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error saving incident: {ex.Message}");
        return false;
    }
}
```

Lo único diferente es que los de *student* hace una comprobación si el Nia existe y los de *teacher* hace una comprobación de DNI y la comprobación de si hay un tutor principal.

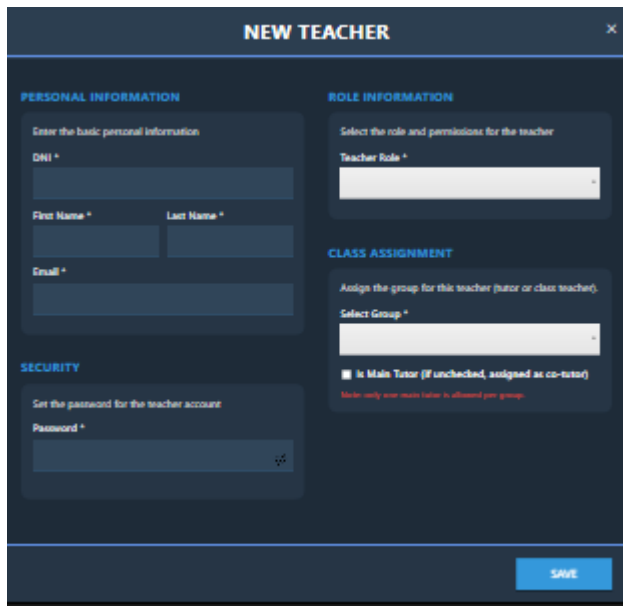
```

bool dniExists = await _teacherService.DniExistsAsync(NewTeacher.Dni);
if (dniExists)
{
    MessageBox.Show("A teacher with this DNI already exists.", "Duplicate DNI", MessageBoxButton.OK, MessageBoxImage.Warning);
    return false;
}

if (IsTutorRoleSelected && NewTeacher.IsTutor)
{
    bool hasMainTutor = await _teacherService.HasMainTutorInGroupAsync(NewTeacher.TutorGroup);
    if (hasMainTutor)
    {
        MessageBox.Show("This group already has a main tutor. Please assign as co-tutor or choose another group.",
            "Main Tutor Exists", MessageBoxButton.OK, MessageBoxImage.Warning);
        return false;
    }
}
}

```

A continuación, pondré una foto de cada formulario:



**NEW TEACHER** [X]

**PERSONAL INFORMATION**  
Enter the basic personal information

DNI \*

First Name \* Last Name \*

Email \*

**SECURITY**  
Set the password for the teacher account

Password \*

**ROLE INFORMATION**  
Select the role and permissions for the teacher

Teacher Role \*

**CLASS ASSIGNMENT**  
Assign the group for this teacher (tutor or class teacher).

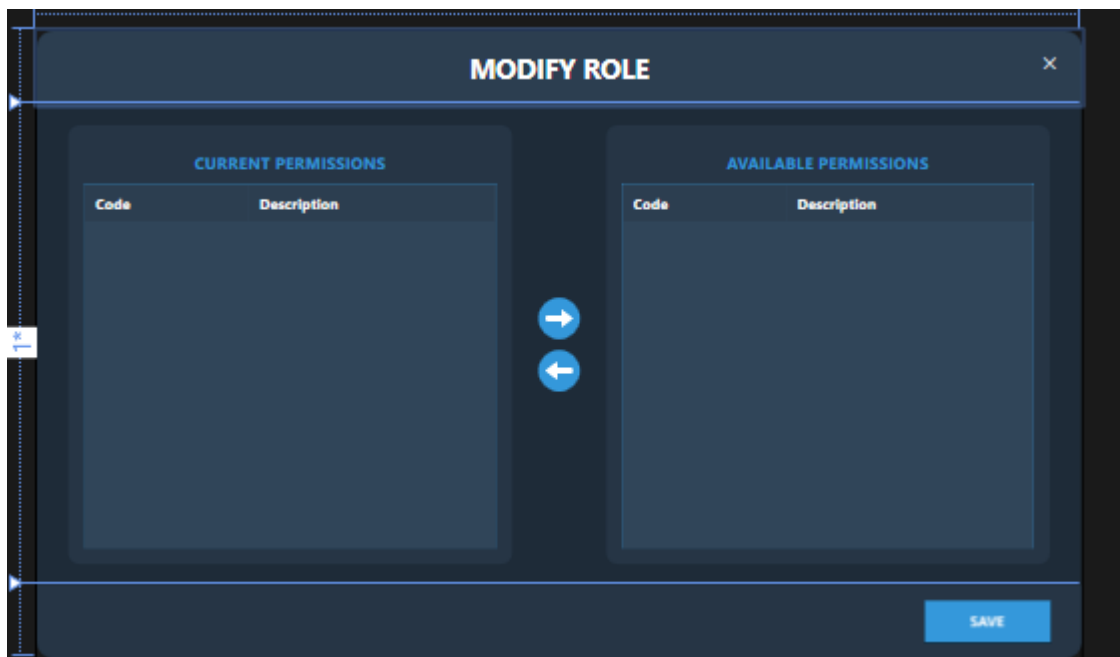
Select Group \*

☐ is Main Tutor (If unchecked, assigned as co-tutor)

Note: only one main tutor is allowed per group.

SAVE

El caso de *ModifyRole* es diferente ya que no hice un CRUD al uso con típico formulario.



**MODIFY ROLE** [X]

**CURRENT PERMISSIONS**

Code	Description

→

←

**AVAILABLE PERMISSIONS**

Code	Description

SAVE

Este dialogo tienes 2 datagrids y dos botones como diferente, el footer y el header son idénticos al resto de diálogos. El botón que apunta a la izquierda se llama *AddPermission* y el de la derecha se llame *RemovePermission*.

```

1 referencia
private void AddPermission_Click(object sender, RoutedEventArgs e)
{
    var selected = _mvAdministration.SelectedAvailablePermission;
    if (selected != null)
    {
        var ownList = _mvAdministration.OwnPermissions.ToList();
        ownList.Add(selected);
        _mvAdministration.OwnPermissions = ownList;

        var available = _mvAdministration.PermissionList.ToList();
        available.Remove(selected);
        _mvAdministration.PermissionList = available;
    }
}

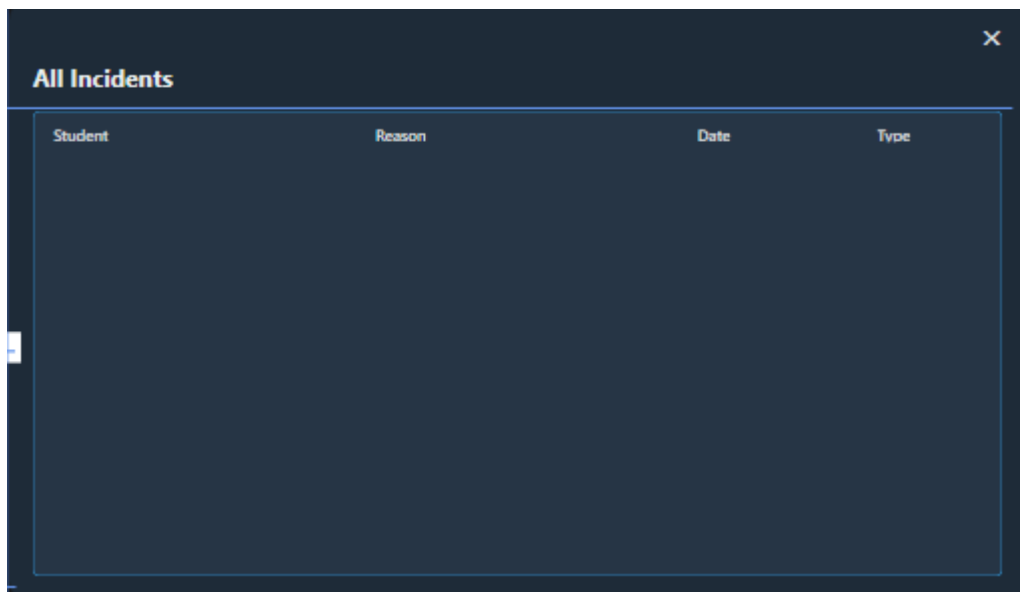
1 referencia
private void RemovePermission_Click(object sender, RoutedEventArgs e)
{
    var selected = _mvAdministration.SelectedOwnPermission;
    if (selected != null)
    {
        var available = _mvAdministration.PermissionList.ToList();
        available.Add(selected);
        _mvAdministration.PermissionList = available;

        var ownList = _mvAdministration.OwnPermissions.ToList();
        ownList.Remove(selected);
        _mvAdministration.OwnPermissions = ownList;
    }
}

```

Lo que hacen estos dos métodos es que, al pinchar en un permiso y darle al botón correspondiente, ese permiso se guarda en una variable del ViewModel llamada *SelectedAvailablePermission*. Luego, se añade a una lista local (temporal) que he creado en el método, y esa lista se iguala a la lista *OwnPermissions* del ViewModel, que es la que realmente se muestra en pantalla. Después, ese mismo permiso se elimina de la lista *PermissionList* porque un permiso no puede estar en las dos listas a la vez. Y en el caso contrario (eliminar), lo que hace es justo lo mismo, pero al revés: lo devuelve a la lista de disponibles y lo quita de la lista de asignados.

*ViewIncidents* es el dialogo que se utiliza en el *Dashboard*, es muy simple, solo tiene un datagrid con la lista de los incidentes



Lo único que hace es recibir por parámetros el ViewModel y una lista con un getAll de todos los incidentes.

## UCIncident

En los UC ya hay mucho más contenido el que comentar.

*UCIncidents*, este es el UC donde más contenido hay, tiene un datagrid que recibe una lista de incidentes dependiendo de los permisos que tenga el usuario, todos los UserControl del proyecto extienden la clase *IRefreshable* que explicaré más adelante, para que todos los UC siempre tengan la lista actualizada para el contexto actual.

```
4 referencias
public async Task FilterIncidentsByPermissions()
{
    var all = await _incidentService.GetAllAsync();
    string dni = _teacher.Dni;
    string? tutorGroup = _teacher.TutorGroup;

    if (SessionManager.HasPermission(PermissionCodes.ModifyOtherIncidents))
    {
        incidentsList = new ObservableCollection<Incident>(all);
        return;
    }

    var filtered = all.Where(i =>
        (SessionManager.HasPermission(PermissionCodes.ModifyRegisteredIncidents) && i.RegisteredBy == dni) ||
        (SessionManager.HasPermission(PermissionCodes.ModifyOwnIncidents) && i.TeacherDni == dni) ||
        (SessionManager.HasPermission(PermissionCodes.ViewTutorGroupIncidents) &&
            tutorGroup != null && i.StudentNiaNavigation?.GroupCode == tutorGroup)
    );

    incidentsList = new ObservableCollection<Incident>(filtered);
}
```

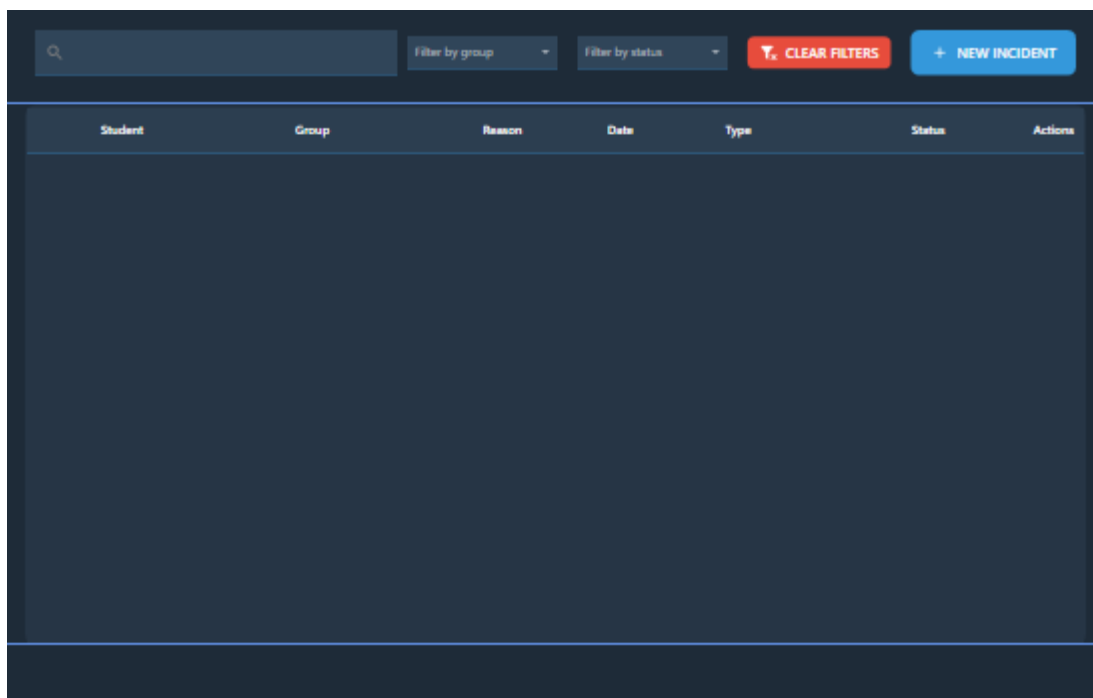
Este método está en el *MVIncident*, y lo que hace es guardar en local una lista completa de incidentes, junto con el DNI del profesor que está conectado y su grupo. El primer if lo que

comprueba es si el usuario tiene el permiso *ModifyOtherIncidents*, que es el más permisivo, ya que permite ver y modificar cualquier incidente. Si lo tiene, entonces se guarda directamente toda la lista completa (*all*) dentro de *incidentsList*, y se termina el método ahí.

En caso de no tener ese permiso, se pasa a una serie de filtros en el *Where*, ordenados desde el permiso más flexible hasta el más restrictivo:

- Si tiene el permiso *ModifyRegisteredIncidents*, solo podrá ver y modificar los incidentes que haya registrado él mismo (*RegisteredBy == dni*).
- Si tiene el permiso *ModifyOwnIncidents*, podrá ver o modificar únicamente los incidentes en los que él esté como profesor implicado (*TeacherDni == dni*).
- Y por último, si tiene el permiso *ViewTutorGroupIncidents* y además tiene asignado un grupo como tutor, podrá acceder solo a los incidentes de alumnos que pertenezcan a ese grupo (*i.StudentNiaNavigation?.GroupCode == tutorGroup*).

Una vez aplicados esos filtros, se guarda el resultado en *incidentsList* para que sea la lista que se muestre en pantalla o se trabaje en la vista.



Ahora voy a explicar la lógica para los botones de acción que utilizan la misma todos los UC.



```

1 referencia
private async void btnAddIncident_Click(object sender, RoutedEventArgs e)
{
    _mvIncident.incident = new Incident();
    _mvIncident.groupTeacherSelected = null;
    _mvIncident.groupSelected = null;
    AddNewIncident addNewIncident = new AddNewIncident(_mvIncident, _teacher);
    addNewIncident.ShowDialog();
}

```

Como en este caso lo que voy a hacer es añadir un incidente nuevo, lo primero que hago es instanciar desde cero el incidente público que está en el *ViewModel*. Esto lo hago por si hubiese algún dato anterior o algo residual, así me aseguro de trabajar siempre con un incidente completamente nuevo, sin arrastrar nada raro de antes. Una vez hecho eso, lanzo el diálogo que corresponde para que el usuario pueda rellenar todos los datos del nuevo incidente.

```

1 referencia
private async void btnEdit_Click(object sender, RoutedEventArgs e)
{
    if (sender is FrameworkElement fe && fe.DataContext is Incident incident)
    {
        _mvIncident.incident = incident;
        _mvIncident.date = incident.EventDatetime.Date;
        _mvIncident.time = DateTime.Today.Add(incident.EventDatetime.TimeOfDay);
        await _mvIncident.LoadCBforModifyIncident();

        ModifyIncident modifyIncident = new ModifyIncident(_mvIncident, _teacher);
        modifyIncident.ShowDialog();
    }
    else
    {
        MessageBox.Show("ERROR 404");
    }
}

```

Este método se lanza al hacer clic en el botón de editar. Lo primero que hago es comprobar que lo que se ha clicado viene de un *FrameworkElement* y que su *DataContext* es un *Incident*. Si es así, le paso ese incidente al *ViewModel* para que lo tenga como el actual y además le paso la fecha y la hora por separado, ya que están en el mismo *DateTime* y luego se usan por separado en la interfaz. Luego simplemente abro el dialogo necesario para usar esta lógica.

```

1 referencia
private async void btnDelete_Click(object sender, RoutedEventArgs e)
{
    if (sender is FrameworkElement fe && fe.DataContext is Incident incident)
    {
        var result = MessageBox.Show("Are you sure you want delete this incident?", "Confirm Delete", MessageBoxButton.YesNo, MessageBoxImage.Warning);

        if (result == MessageBoxResult.Yes)
        {
            await _incidentService.DeleteAsync(incident);
            await _mvIncident.FilterIncidentsByPermissions();
        }
    }
    else
    {
        MessageBox.Show("ERROR 404");
    }
}

```

Este método se activa cuando se pulsa el botón de eliminar. Lo primero que hago es comprobar que el botón pertenece a una fila de la tabla y que tiene un *Incident* como *DataContext*. Si es así, lanzo un mensaje de confirmación para que el usuario confirme si de verdad quiere eliminar ese incidente.

Si pulsa que sí, llamo al servicio para borrarlo de la base de datos y luego vuelvo a cargar la lista de incidentes pero ya filtrada según los permisos que tenga el profesor (esto lo hace el método *FilterIncidentsByPermissions* del *ViewModel*).

En este UC es el único que tiene la funcionalidad de enviar un correo electrónico

```

1 referencia
private async void btnInform_Click(object sender, RoutedEventArgs e)
{
    try
    {
        if (sender is FrameworkElement fe && fe.DataContext is Incident incident)
        {
            _mvIncident.incident = incident;

            EmailService emailService = new EmailService();
            string body = emailService.BuildIncidentEmailBody(incident);
            string toEmail = incident.StudentNiaNavigation?.Email;

            if (string.IsNullOrEmpty(toEmail))
            {
                MessageBox.Show("No email address found for the student.", "Missing Email", MessageBoxButton.OK, MessageBoxImage.Warning);
                return;
            }

            bool success = await emailService.SendEmailAsync(toEmail, body);

            if (success)
            {
                MessageBox.Show("Incident report sent successfully.", "Email Sent", MessageBoxButton.OK, MessageBoxImage.Information);
                _mvIncident.incident.IsSanctioned = true; // Mark the incident as sanctioned after sending the email
                await _mvIncident.updateIncident();
                _mvIncident.UpdateIncidentInList(incident);
            }
            else
            {
                MessageBox.Show("The email could not be sent. Please try again later.", "Sending Failed", MessageBoxButton.OK, MessageBoxImage.Error);
            }
        }
        else
        {
            MessageBox.Show("Incident data not found.", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"An unexpected error occurred:\n{ex.Message}", "Unexpected Error", MessageBoxButton.OK, MessageBoxImage.Error);
    }
}

```

Este método se ejecuta al pulsar el botón de sancionar o informar sobre un incidente. Lo primero que hace es comprobar si el botón que se ha pulsado está asociado a un incidente concreto. Si es así, se guarda ese incidente en el **ViewModel** para trabajar con él. Luego se crea una instancia del servicio de correos (*EmailService*) que se encarga de generar el cuerpo del mensaje a partir de los datos del incidente.

Una vez generado el cuerpo del email, se comprueba si el alumno al que va dirigido tiene correo electrónico registrado. Si no lo tiene, se avisa al usuario mediante un mensaje. Si sí que tiene, se intenta enviar el email. Si el envío se realiza con éxito, se muestra una confirmación y se marca el incidente como sancionado.

## UCDashboard

El UserControl es el panel principal del dashboard para profesores. En la parte superior se muestra el nombre del profesor y la fecha actual. Luego hay dos columnas: una más ancha a la izquierda con una gráfica de incidentes que cambia según el filtro de tiempo que se seleccione (una semana, un mes, un año o todo el histórico), y justo debajo una tarjeta con los últimos 5 incidentes registrados, mostrando el nombre del alumno, el motivo y si es una sanción o solo un aviso.

A la derecha, la columna más estrecha sirve como resumen: muestra los totales de incidentes, avisos y sanciones, con iconos e indicadores visuales. Debajo hay una sección de tareas rápidas donde el profesor puede añadir tareas personales y marcarlas como hechas.



```

2 referencias
public void UpdateChartData(int timeFilterIndex)
{
    if (incidentsList == null) return;

    DateTime startDate = GetStartDateFromFilter(timeFilterIndex);
    var filteredIncidents = incidentsList.Where(i => i.EventDatetime >= startDate);

    // Agrupar por fecha
    var groupedData = filteredIncidents
        .GroupBy(i => i.EventDatetime.Date)
        .OrderBy(g => g.Key)
        .ToList();

    // Limpiar datos anteriores
    ChartSeries.Clear();
    DaysLabels.Clear();

    if (!groupedData.Any()) return;

    // Preparar datos para las series
    var sanctionValues = new ChartValues<int>();
    var warningValues = new ChartValues<int>();

    foreach (var group in groupedData)
    {
        DaysLabels.Add(group.Key.ToString("MM/dd"));
        sanctionValues.Add(group.Count(i => i.isSanction));
        warningValues.Add(group.Count(i => !i.isSanction));
    }

    // Crear las series
    ChartSeries.Add(new LineSeries
    {
        Title = "Sanctions",
        Values = sanctionValues,
        Stroke = Brushes.Red,
        Fill = Brushes.Transparent,
        PointGeometry = DefaultGeometries.Circle,
        PointGeometrySize = 8
    });

    ChartSeries.Add(new LineSeries
    {
        Title = "Warnings",
        Values = warningValues,
        Stroke = Brushes.Orange,
        Fill = Brushes.Transparent,
        PointGeometry = DefaultGeometries.Circle,
        PointGeometrySize = 8
    });

    OnPropertyChanged(nameof(ChartSeries));
    OnPropertyChanged(nameof(DaysLabels));
}

```

Este método actualiza los datos de la gráfica según el filtro de tiempo seleccionado. Para ello:

1. **Filtra** los incidentes a partir de una fecha calculada con *GetStartDateFromFilter*.
2. **Agrupar** los incidentes por día.
3. **Cuenta** cuántos son sanciones y cuántos advertencias por día.
4. **Actualiza** las líneas del gráfico (*ChartSeries*) y las etiquetas del eje X (*DaysLabels*) con esos datos.

Todo esto está en el MVDashboard, ahora todo lo relacionado con las *Quick Task*

#### AddTask

-Añade una nueva tarea con el título indicado (si no está vacío).

#### RemoveTask

-Elimina una tarea de la colección si existe.

### LoadTasks

-Carga desde disco todas las tareas asociadas al Dni del usuario usando *TaskService*.

### SaveTasks

-Guarda todas las tareas actuales en disco mediante *TaskService*.

Todo esto utilizando una clase llamada UserTasks que tiene una lista de tareas y el dni para poder identificar al usuario, ya que cada usuario tiene su propia lista, más adelante se explicara la clase a profundidad.

## UCSettings

En esta pantalla permito al usuario cambiar su contraseña.

A la derecha tengo un formulario con tres campos: contraseña actual, nueva contraseña y repetir la nueva contraseña. Cuando el usuario pulsa el botón de "Actualizar contraseña", hago las siguientes comprobaciones:

1. Verifico que la contraseña actual que ha escrito coincida con la guardada.
2. Si es correcta, reviso que la nueva contraseña y la repetida coincidan.
3. Si todo está bien, actualizo la contraseña del profesor y muestro un mensaje de éxito.
4. Si algo falla (por ejemplo, no coinciden las nuevas o la actual es incorrecta), se muestra un mensaje de error adecuado.
5. Al final, limpiamos todos los campos del formulario por seguridad.

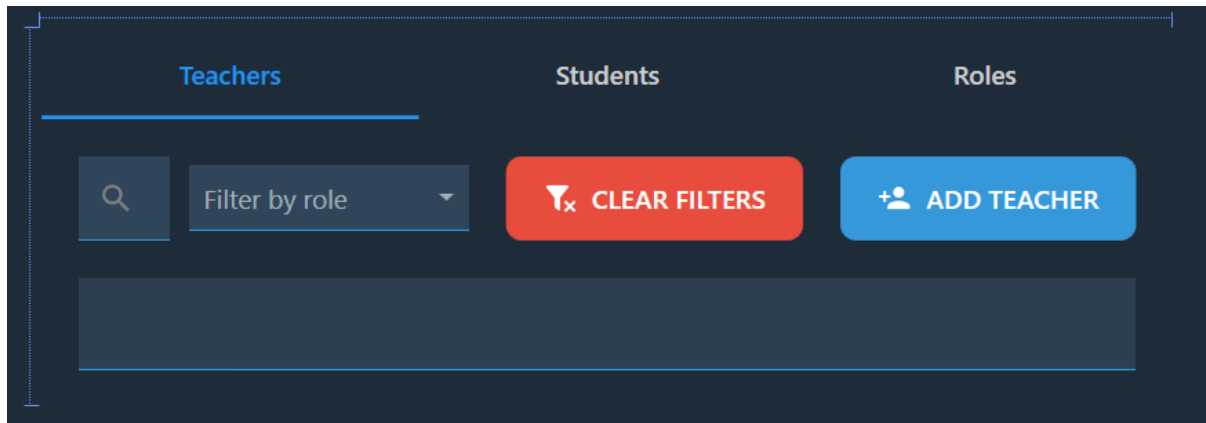
```
1 referencia
private async void Button_Click(object sender, RoutedEventArgs e)
{
    if (_teacher.Password == CurrentPasswordBox.Password)
    {
        if (NewPasswordBox.Password == RepeatPasswordBox.Password)
        {
            _teacher.Password = NewPasswordBox.Password;
            await _mvAdministration.updateTeacherPassword(_teacher);
            MessageBox.Show("Password updated successfully.", "Success", MessageBoxButton.OK, MessageBoxImage.Information);
        }
        else
        {
            MessageBox.Show("New passwords do not match.", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
        }
    }
    else
    {
        MessageBox.Show("Current password is incorrect.", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
    }

    CurrentPasswordBox.Clear();
    NewPasswordBox.Clear();
    RepeatPasswordBox.Clear();
}
```

Todo esto lo hago en el Button\_Click, y uso el ViewModel MVAdministration para llamar a la función que actualiza la contraseña en la base de datos.

## UCAAdminstración

Esta sección de la aplicación está diseñada para que los administradores puedan gestionar fácilmente a los profesores, estudiantes y roles del sistema. Todo se organiza en pestañas y con un diseño limpio.



### Pestaña de Teachers

En esta pestaña permito:

1. Buscar profesores por nombre con una caja de búsqueda vinculada a `SearchTeacherText`.
2. Filtrar por rol usando un ComboBox enlazado a la lista `RoleFilterOptions`.
3. Limpiar los filtros con el botón `Clear Filters`, que resetea el texto de búsqueda y la selección de rol.
4. Añadir un nuevo profesor mediante un Dialog llamado `AddTeacher`, que se abre al pulsar el botón correspondiente.
5. Editar un profesor ya existente: al pulsar el botón de lápiz, se guarda el profesor seleccionado en el `ViewModel` y se abre `ModifyTeacher`.
6. Eliminar un profesor: se pide confirmación, y si el usuario acepta, se borra de la base de datos y se actualiza la lista con `RefreshTeachersAsync`.

### Pestaña de Students

Funciona de manera similar a la de profesores:

1. Búsqueda por nombre con `SearchStudentText`.
2. Filtro por grupo (`SelectedGroupFilter`), usando la lista de grupos cargada en `GroupList`.
3. Los botones permiten añadir (`AddStudent`), editar (`ModifyStudent`) o eliminar estudiantes tras confirmación.
4. Al eliminar, uso `DeleteAsyncPro` del servicio de estudiantes y luego recargo la lista con `RefreshStudentsAsync`.

## Pestaña de Roles

Aquí simplemente muestro una tabla con los roles existentes y un botón para editarlos. Al pulsar "Editar", se abre el diálogo `ModifyRole`, y antes de mostrarlo, cargo los permisos del rol con `LoadOwnPermissionsAsync`.

## MVBase

La clase `MVBase` centraliza la validación de formularios. Controla automáticamente si hay errores en los campos y desactiva el botón de guardar cuando ocurre. Usa un contador interno (`errorCount`) que se actualiza al detectar errores, y expone el método `IsValid()` para comprobar si todo es válido. Así se evita repetir esta lógica en cada pantalla.

## MVBaseCRUD

La clase `MVBaseCRUD<T>` extiende la lógica base con operaciones genéricas de alta, modificación y borrado (`Add`, `Update`, `Delete`). Usa un servicio (`GenericService<T>`) que se encarga del acceso a datos y simplifica la gestión de cualquier tipo de entidad sin repetir código. Todo se ejecuta de forma asíncrona para no bloquear la interfaz.

## PropertyChangedDataError

Esta clase sirve como base para que las propiedades de los modelos se mantengan sincronizadas con la interfaz (gracias a *`INotifyPropertyChanged`*) y además puedan validarse automáticamente usando anotaciones (*`IDataErrorInfo`*). Así, cuando un usuario introduce un dato incorrecto, se muestra el error correspondiente sin necesidad de escribir validaciones a mano.

## MVAdministration

Esta parte de la aplicación es la encargada de gestionar tanto a los profesores como a los alumnos. El *ViewModel* `MVAdministration` es bastante completo y carga los datos, permite filtrarlos, y también guardar y editar los registros de forma directa.

Nada más entrar en esta pantalla, se llama al método `Start()`, que inicializa todos los servicios necesarios y carga todos los datos de la base de datos: profesores, alumnos, grupos, roles y permisos. Esto hace que todo esté actualizado en la interfaz sin tener que recargar a mano.

### Guardar profesor – `saveTeacher()`

Este método se usa para añadir un profesor nuevo. Antes de guardarlo, comprueba si ya existe un profesor con ese DNI. Si el rol es de tutor, también revisa que el grupo no tenga ya

uno asignado. Si todo está bien, guarda el nuevo profesor y se recarga la pantalla para que aparezca en la lista.

```
public async Task<bool> saveTeacher()
{
    try
    {
        bool dniExists = await _teacherService.DniExistsAsync(NewTeacher.Dni);
        if (dniExists)
        {
            MessageBox.Show("A teacher with this DNI already exists.", "Duplicate DNI", MessageBoxButton.OK, MessageBoxImage.Warning);
            return false;
        }

        if (IsTutorRoleSelected && NewTeacher.IsTutor)
        {
            bool hasMainTutor = await _teacherService.HasMainTutorInGroupAsync(NewTeacher.TutorGroup);

            if (hasMainTutor)
            {
                MessageBox.Show("This group already has a main tutor. Please assign as co-tutor or choose another group.",
                    "Main Tutor Exists", MessageBoxButton.OK, MessageBoxImage.Warning);
                return false;
            }
        }

        bool correct = await _teacherService.AddAsync(NewTeacher);

        // Create a new incident for the next entry if successful
        if (correct)
        {
            NewTeacher = new Teacher();
            await Start(); // Reinitialize the view model
            SelectedRole = null;
            OnPropertyChanged(nameof(IsTutorRoleSelected));
        }

        return correct;
    }
    catch (Exception ex)
    {
        // Log error or handle exception
        Console.WriteLine($"Error saving incident: {ex.Message}");
        return false;
    }
}
```

Editar profesor – updateTeacher()

Es parecido al anterior, pero para actualizar los datos de un profesor ya existente. También controla el tema del tutor principal. Si todo va bien, actualiza y vuelve a cargar la lista.



```

1 referencia
public async Task<bool> updateTeacher()
{
    try
    {
        if (IsTutorRoleSelected && Teacher.IsTutor)
        {
            bool hasMainTutor = await _teacherService.HasMainTutorInGroupAsync(Teacher.TutorGroup);

            if (hasMainTutor)
            {
                MessageBox.Show("This group already has a main tutor. Please assign as co-tutor or choose another group.",
                                "Main Tutor Exists", MessageBoxButton.OK, MessageBoxImage.Warning);
                return false;
            }
        }

        bool correct = await _teacherService.UpdateAsync(Teacher);

        if (correct)
        {
            NewTeacher = new Teacher();
            await Start(); // Reinitialize the view model
            SelectedRole = null;
            OnPropertyChanged(nameof(IsTutorRoleSelected));
        }

        return correct;
    }
    catch (Exception ex) {
        Console.WriteLine($"Error saving incident: {ex.Message}");
        return false;
    }
}

```

### Permisos por rol – SaveRolePermissionsAsync()

Este método guarda los permisos que tiene cada rol. Por ejemplo, si a un rol de “Coordinador” le quiero quitar o poner permisos, lo hago desde aquí. Si se guarda bien, sale un mensaje de confirmación.

```

public async Task<bool> SaveRolePermissionsAsync()
{
    try
    {
        if (Role == null || OwnPermissions == null)
            return false;

        await _permissionsService.UpdateRolePermissionsAsync(Role.Id, OwnPermissions.ToList());
        MessageBox.Show("Permissions updated successfully!", "Success", MessageBoxButton.OK, MessageBoxImage.Information);
        return true;
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Error updating permissions: {ex.Message}", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
        return false;
    }
}

```

En resumen, este ViewModel me ha servido para controlar toda la parte de administración desde un único sitio, y hace que todo funcione de forma limpia y sin complicaciones. Además, con los filtros y las validaciones que he metido, se evitan errores comunes como duplicados o tutores repetidos.

## MVDashboard

El *MVDashboard* es el *ViewModel* principal de la pantalla de inicio de la aplicación, que muestra tanto los últimos incidentes como una gráfica con estadísticas y una lista de tareas rápidas. Al iniciarse, se cargan los incidentes según los permisos del usuario y se calcula automáticamente cuántos avisos y sanciones hay.

Una vez cargados los incidentes, el método *SeparateIncidents* los cuenta por tipo (avisos o sanciones) para mostrarlos como resumen. Además, se guardan y muestran los cinco más recientes en la parte inferior del *dashboard*.

```
private void SeparateIncidents()
{
    if (incidentsList == null)
    {
        totalWarnings = "0";
        totalSanctions = "0";
        totalIncidents = "0";
        return;
    }

    totalWarnings = incidentsList.Count(i => !i.isSanction).ToString();
    totalSanctions = incidentsList.Count(i => i.isSanction).ToString();
    totalIncidents = incidentsList.Count().ToString();
}
```

Respecto a las gráficas, se actualizan en función de un filtro de tiempo (última semana, mes, año o todo el tiempo). Esto se hace con el método *UpdateChartData*, que calcula los datos desde una fecha concreta usando *GetStartDateFromFilter*. Según el filtro, agrupa los incidentes por día y los convierte en dos series: una para sanciones (roja) y otra para avisos (naranja).

```

public void UpdateChartData(int timeFilterIndex)
{
    if (incidentsList == null) return;

    DateTime startDate = GetStartDateFromFilter(timeFilterIndex);
    var filteredIncidents = incidentsList.Where(i => i.EventDatetime >= startDate);

    // Agrupar por fecha
    var groupedData = filteredIncidents
        .GroupBy(i => i.EventDatetime.Date)
        .OrderBy(g => g.Key)
        .ToList();

    // Limpiar datos anteriores
    ChartSeries.Clear();
    DaysLabels.Clear();

    if (!groupedData.Any()) return;

    // Preparar datos para las series
    var sanctionValues = new ChartValues<int>();
    var warningValues = new ChartValues<int>();

    foreach (var group in groupedData)
    {
        DaysLabels.Add(group.Key.ToString("MM/dd"));
        sanctionValues.Add(group.Count(i => i.isSanction));
        warningValues.Add(group.Count(i => !i.isSanction));
    }

    // Crear las series
    ChartSeries.Add(new LineSeries
    {
        Title = "Sanctions",
        Values = sanctionValues,
        Stroke = Brushes.Red,
        Fill = Brushes.Transparent,
        PointGeometry = DefaultGeometries.Circle,
        PointGeometrySize = 8
    });

    ChartSeries.Add(new LineSeries
    {
        Title = "Warnings",
        Values = warningValues,
        Stroke = Brushes.Orange,
        Fill = Brushes.Transparent,
        PointGeometry = DefaultGeometries.Circle,
        PointGeometrySize = 8
    });

    OnPropertyChanged(nameof(ChartSeries));
    OnPropertyChanged(nameof(DaysLabels));
}

```

Por último, también incluye un sistema de tareas personales para el usuario. Cada profesor puede tener su lista propia, que se carga automáticamente al iniciar y se guarda cuando se modifica. Estas tareas no dependen de la base de datos, sino que se gestionan desde un archivo local asociado al DNI del usuario. El *ViewModel* permite añadir y eliminar tareas fácilmente desde la interfaz.

```

1 referencia
public void RemoveTask(UserTask task)
{
    if (task != null && UserTasks.Contains(task)) UserTasks.Remove(task);
}

1 referencia
public void LoadTasks()
{
    try
    {
        var tasks = TaskService.LoadTasks(CurrentDni);
        UserTasks.Clear();
        foreach (var task in tasks) UserTasks.Add(task);
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine($"LoadTasks error: {ex.Message}");
    }
}

```

Este ViewModel es el encargado de gestionar toda la lógica relacionada con las incidencias dentro de la aplicación. Se encarga de cargar los datos necesarios (como los grupos, estudiantes, razones y profesores), así como de filtrar, crear y modificar incidencias. Desde esta clase controlo tanto la gestión de los datos como la lógica asociada a la interfaz para que esté siempre sincronizada.

Una de las primeras cosas que hago al inicializar el *ViewModel* es definir los filtros y elementos interactivos que usa el usuario. Por ejemplo, tengo propiedades como *SearchText*, *FilterGroup* o *FilterStatus* que permiten filtrar fácilmente la lista de incidencias por nombre del alumno, grupo o estado (sancionado o pendiente). Todo esto se refleja en una propiedad *FilteredIncidentsList*, que devuelve los resultados ya filtrados sin tener que tocar manualmente la lista principal.

```

public IEnumerable<Incident> FilteredIncidentsList
{
    get
    {
        IEnumerable<Incident> list = incidentsList ?? Enumerable.Empty<Incident>();

        if (!string.IsNullOrEmpty(SearchText))
        {
            list = list.Where(i =>
                $"{i.StudentNiaNavigation?.FirstName} {i.StudentNiaNavigation?.LastName}"
                .ToLower().Contains(SearchText.ToLower()));
        }

        if (FilterGroup != null)
        {
            list = list.Where(i => i.StudentNiaNavigation?.GroupCode == FilterGroup.GroupCode);
        }

        if (FilterStatus == "Sanctioned")
        {
            list = list.Where(i => i.IsSanctioned);
        }
        else if (FilterStatus == "Pending")
        {
            list = list.Where(i => !i.IsSanctioned);
        }

        return list;
    }
}

```

Al iniciar el *ViewModel* (con el método *Start*), se instancian los servicios necesarios y se cargan todos los datos relevantes desde la base de datos: estudiantes, grupos, razones y profesores. También filtro las incidencias según los permisos del profesor que ha iniciado sesión. Esto se hace con el método *FilterIncidentsByPermissions*, que tiene en cuenta si el profesor puede ver solo las incidencias que ha registrado, las de su grupo o todas si tiene permisos especiales.

```

public async Task FilterIncidentsByPermissions()
{
    var all = await _incidentService.GetAllAsync();
    string dni = _teacher.Dni;
    string? tutorGroup = _teacher.TutorGroup;

    if (SessionManager.HasPermission(PermissionCodes.ModifyOtherIncidents))
    {
        incidentsList = new ObservableCollection<Incident>(all);
        return;
    }

    var filtered = all.Where(i =>
        (SessionManager.HasPermission(PermissionCodes.ModifyRegisteredIncidents) && i.RegisteredBy == dni) ||
        (SessionManager.HasPermission(PermissionCodes.ModifyOwnIncidents) && i.TeacherDni == dni) ||
        (SessionManager.HasPermission(PermissionCodes.ViewTutorGroupIncidents) &&
            tutorGroup != null && i.StudentNiaNavigation?.GroupCode == tutorGroup)
    );

    incidentsList = new ObservableCollection<Incident>(filtered);
}

```

En cuanto a la creación y edición de incidencias, tengo los métodos *saveIncident* y *updateIncident*, que son bastante similares. En ambos casos, se actualizan los datos necesarios (como la fecha, el DNI del profesor que la registra, etc.) y se guarda la información usando el servicio de incidencias. Si todo va bien, se vuelve a cargar la lista y se limpia el formulario para permitir seguir trabajando con una nueva incidencia.

También tengo propiedades y métodos pensados para facilitar el trabajo con los ComboBox. Por ejemplo, al seleccionar un grupo, se actualiza automáticamente la lista de estudiantes o profesores correspondientes, filtrando solo los que pertenecen a ese grupo. Esto se hace con propiedades como *FilteredStudents* o *FilteredTeachers*, que trabajan a partir de la selección del usuario.

En general, este ViewModel me permite tener el control completo sobre la gestión de incidencias dentro de la aplicación, desde la carga de datos hasta la creación y modificación, pasando por filtros y sincronización con la interfaz de usuario. Todo está centralizado y preparado para reaccionar a los cambios en tiempo real.

### **EmailService**

Una de las funcionalidades clave que quise implementar en el sistema EduAvis fue el envío automático de correos cuando se registra una incidencia disciplinaria. Para ello, creé una clase llamada *EmailService* que se encarga de componer y enviar notificaciones por correo electrónico a la dirección del alumno o del responsable, con todos los detalles del incidente.

En cuanto a la parte técnica, el correo se envía desde una cuenta de Gmail configurada específicamente para el sistema, utilizando su servidor SMTP con autenticación mediante contraseña de aplicación (ya que Gmail no permite usar la contraseña normal por seguridad). Se usa una librería estándar (*SmtpClient*).

```

public class EmailService
{
    private readonly string _fromEmail = "eduavis.system@gmail.com";
    private readonly string _appPassword = [REDACTED];

    1 referencia
    public async Task<bool> SendEmailAsync(string toEmail, string body)
    {
        try
        {
            var smtpClient = new SmtpClient("smtp.gmail.com")
            {
                Port = 587,
                Credentials = new NetworkCredential(_fromEmail, _appPassword),
                EnableSsl = true,
            };

            var mailMessage = new MailMessage
            {
                From = new MailAddress(_fromEmail, "EduAvis System"),
                Subject = "[EduAvis] Incident Notification",
                Body = body,
                IsBodyHtml = true
            };

            mailMessage.To.Add(toEmail);
            await smtpClient.SendMailAsync(mailMessage);

            return true;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error sending email: {ex.Message}");
            return false;
        }
    }
}

```

Luego el *body* es el otro método que está automatizado para que dependiendo de la incidencia envíe un correo electrónico.

## IRefreshable

Para algunos componentes del sistema que necesitaban actualizar sus datos desde la base de datos, decidí crear una interfaz sencilla llamada *IRefreshable*. Básicamente, lo único que exige es que la clase que la implemente tenga un método *Refresh* que se encargue de recargar su contenido.

Esta interfaz me ha servido para unificar y estandarizar el comportamiento de refresco en varias partes de la aplicación, especialmente en los controles visuales que muestran listas dinámicas. De este modo, cuando necesito actualizar la información desde fuera, solo tengo que llamar a *Refresh* sin preocuparme por cómo lo hace internamente cada clase.

## PermissionCodes

En la clase *PermissionCodes* guardo todos los permisos disponibles como constantes. Así evito errores al usarlos en el código y tengo todo más organizado y fácil de mantener.

## SecureStorage

La clase `SecureStorage` me permite guardar las credenciales del usuario (email y contraseña) de forma segura en el equipo local. Lo que hace es coger el email y la contraseña.

Con `ProtectedData.Protect` que es una función de Windows que cifra datos para el usuario actual. El resultado lo guardo en un archivo llamado `credentials.dat`.

```
var data = $"{email}|{password}";  
var encrypted = ProtectedData.Protect(  
    Encoding.UTF8.GetBytes(data),  
    null,  
    DataProtectionScope.CurrentUser);  
File.WriteAllBytes(FilePath, encrypted);
```

También tiene un método `ClearCredentials` para eliminar el archivo, por ejemplo, al cerrar sesión. De esta forma, el login se recuerda de forma segura y no tengo que pedirlo cada vez, mejorando la experiencia del usuario sin comprometer la seguridad.

## SessionManager

Esta clase guarda quién ha iniciado sesión (`CurrentUser`) y qué permisos tiene (`Roles`). Luego, con el método `HasPermission`, puedo comprobar fácilmente si ese usuario tiene cierto permiso pasando el código correspondiente. Esta clase la hice statica para poder revisar los permisos del usuario en cualquier parte de la aplicación de una forma rápida.

## UserTask

Esta clase representa una tarea personal asociada a un usuario concreto mediante su DNI. Incluye un título, una descripción y un booleano `IsCompleted` para saber si está completada. Es simple y funcional, pensada para gestionar tareas dentro del panel personal de la aplicación.

## IGenericService

Simplemente es una interfaz que va a usar el servicio genérico que tiene las funcionalidades

```
1 referencia  
public interface IGenericService<T> where T : class  
{  
    5 referencias  
    Task<bool> AddAsync(T entity);  
  
    12 referencias  
    Task<IEnumerable<T>> GetAllAsync();  
    1 referencia  
    Task<T> GetByIdAsync(int id);  
    6 referencias  
    Task<bool> UpdateAsync(T entity);  
    3 referencias  
    Task<bool> DeleteAsync(T entity);  
    1 referencia  
    Task<IEnumerable<T>> FindAsync(Expression<Func<T, bool>> predicate);  
}
```



## GenericService

Este servicio es probablemente el más importante de todo el backend, ya que encapsula la lógica CRUD de forma genérica. Gracias a esta clase, se pueden reutilizar las operaciones básicas (crear, leer, actualizar y borrar) en cualquier entidad del sistema. Posteriormente, otras clases del backend heredan de esta base para aplicar esa funcionalidad directamente sobre sus modelos concretos, lo que evita repetir código y facilita el mantenimiento.

## GroupService

No tiene nada, solo está creada para poder usar el CRUD de los grupos.

## IncidentService

En este servicio solo volví a implementar el método para obtener todos los incidentes porque necesitaba incluir la navegación entre entidades relacionadas, y eso el servicio genérico no me lo permitía. Por eso, aunque aprovecho la base del *GenericService*, en este caso concreto tuve que hacer esa excepción.

```
2 referencias
public async Task<IEnumerable<Incident>> GetAllAsync()
{
    var query = _context.Incidents.AsQueryable();

    // Obtén todas las propiedades de navegación
    var navigationProperties = _context.Model.FindEntityType(typeof(Incident))
        .GetNavigations()
        .Select(e => e.Name);

    // Incluye cada propiedad de navegación
    foreach (var navigationProperty in navigationProperties)
    {
        query = query.Include(navigationProperty);
    }

    return await query.ToListAsync();
}
```

## PermissionsService

Este servicio es una extensión del servicio genérico, pero enfocado en la gestión de permisos asociados a los roles. En concreto, tiene dos métodos que me interesaban especialmente:

*GetPermissionsByRoleIdAsync*: este método me sirve para sacar todos los permisos que tiene asignado un rol concreto. Va a la tabla intermedia *RolePermissions* y me devuelve los permisos relacionados con ese *roleId*.

```
1 referencia
public async Task<IEnumerable<Permission>> GetPermissionsByRoleIdAsync(int roleId)
{
    return await _context.RolePermissions
        .Where(rp => rp.RoleId == roleId)
        .Select(rp => rp.Permission)
        .ToListAsync();
}
```

*UpdateRolePermissionsAsync*: con este lo que hago es actualizar los permisos de un rol. Primero borra todos los permisos anteriores de ese rol, y luego añade los nuevos que yo le paso como lista. Es decir, hace una especie de “reseteo” y vuelve a guardar lo nuevo, este metodo lo hice exclusivamente para el Dialogo de *ModifyRole*.

```
1 referencia
public async Task UpdateRolePermissionsAsync(int roleId, List<Permission> newPermissions)
{
    var existing = _context.RolePermissions.Where(rp => rp.RoleId == roleId);
    _context.RolePermissions.RemoveRange(existing);

    foreach (var permission in newPermissions)
    {
        _context.RolePermissions.Add(new RolePermission
        {
            RoleId = roleId,
            PermissionId = permission.Id
        });
    }

    await _context.SaveChangesAsync();
}
```

## ReasonService

No tiene nada, solo está creada para poder usar el CRUD de los *reason*.

## RoIService

Este servicio está pensado para trabajar con los roles dentro del sistema:

- **GetRoleByTeacherIdAsync**: este método me sirve para obtener el rol que tiene asignado un profesor. Primero busca al profesor por su ID. Si el profesor no existe o no tiene rol, devuelve null.

```
0 referencias
public async Task<Role?> GetRoleByTeacherIdAsync(int teacherId)
{
    var teacher = await _context.Teachers.FindAsync(teacherId);
    if (teacher == null || teacher.RoleId == null)
        return null;

    return await _context.Roles
        .Include(r => r.RolePermissions)
        .ThenInclude(rp => rp.Permission)
        .FirstOrDefaultAsync(r => r.Id == teacher.RoleId);
}
```

## StudentService

Este servicio es el encargado de gestionar todo lo relacionado con los estudiantes.

- FindStudentsByGroupAsync: me sirve para buscar todos los alumnos que pertenecen a un grupo concreto.

```
1 referencia
public async Task<IEnumerable<Student>> FindStudentsByGroupAsync(string groupName)
{
    try
    {
        var students = await _context.Students
            .Where(s => s.GroupCodeNavigation.GroupName == groupName)
            .ToListAsync();

        logger.Info($"Students found in group: {groupName}");
        return students;
    }
    catch (Exception ex)
    {
        LogError("Error while fetching students by group", ex);
        return Enumerable.Empty<Student>();
    }
}
```

- FindGroupByStudentAsync: este método lo uso cuando tengo un alumno y necesito saber a qué grupo pertenece.

```
1 referencia
public async Task<Group?> FindGroupByStudentAsync(Student student)
{
    if (student == null || string.IsNullOrEmpty(student.GroupCode))
        return null;

    return await _context.Groups
        .FirstOrDefaultAsync(g => g.GroupCode == student.GroupCode);
}
```

- NiaExistsAsync: simplemente comprueba si ya existe un estudiante con ese NIA. Lo uso al registrar nuevos alumnos para evitar duplicados.

```
1 referencia
public async Task<bool> NiaExistsAsync(string nia)
{
    return await _context.Students.AnyAsync(s => s.Nia == nia);
}
1 referencia
```

- DeleteAsyncPro: esta es una versión personalizada del borrado. Antes de eliminar al alumno, revisa si tiene incidencias asociadas. Si las tiene, muestra un mensaje de confirmación para advertir que se van a eliminar también esas incidencias. Así evito borrados accidentales.

```

public async Task<bool> DeleteAsyncPro(Student student)
{
    var loadedStudent = await _context.Students
        .Include(s => s.Incidents)
        .FirstOrDefaultAsync(s => s.Nia == student.Nia);

    if (loadedStudent == null)
        return false;

    int totalIncidents = loadedStudent.Incidents.Count;

    if (totalIncidents > 0)
    {
        var confirm = MessageBox.Show(
            $"This student is linked to {totalIncidents} incident(s).\nIf you continue, all related incidents will also be deleted.\n\nDo you want to proceed?",
            "Confirm Deletion",
            MessageBoxButton.YesNo,
            MessageBoxImage.Warning);

        if (confirm != MessageBoxResult.Yes)
            return false;
    }

    _context.Students.Remove(loadedStudent);
    await _context.SaveChangesAsync();
    return true;
}

```

## TaskService

Este servicio lo hice para gestionar las tareas personales de cada usuario. Lo que hace es guardar y cargar las tareas desde un archivo local (formato JSON), que está separado por DNI para que cada usuario tenga sus propias tareas.

- *LoadTasks*: se encarga de leer el archivo del usuario y cargar las tareas guardadas. Si no hay archivo o está vacío, simplemente devuelve una lista vacía. También me aseguro de que todas las tareas tengan el DNI del usuario por si acaso.

```

public static List<UserTask> LoadTasks(string dni)
{
    try
    {
        string path = GetFilePath(dni);

        string directory = Path.GetDirectoryName(path);
        if (!Directory.Exists(directory))
        {
            Directory.CreateDirectory(directory);
        }

        if (!File.Exists(path))
            return new List<UserTask>();

        var json = File.ReadAllText(path);
        if (string.IsNullOrWhiteSpace(json))
            return new List<UserTask>();

        var tasks = JsonConvert.DeserializeObject<List<UserTask>>(json) ?? new List<UserTask>();

        foreach (var task in tasks)
        {
            if (string.IsNullOrEmpty(task.Dni))
                task.Dni = dni;
        }

        return tasks;
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine($"Error loading tasks: {ex.Message}");
        return new List<UserTask>();
    }
}

```

- *SaveTasks*: guarda la lista actual de tareas en el archivo correspondiente. Antes de hacerlo, me aseguro de que el directorio exista y de que todas las tareas tengan asignado el DNI correctamente.

```

public static void SaveTasks(string dni, List<UserTask> tasks)
{
    try
    {
        string path = GetFilePath(dni);

        string directory = Path.GetDirectoryName(path);
        if (!Directory.Exists(directory))
        {
            Directory.CreateDirectory(directory);
        }

        foreach (var task in tasks)
        {
            task.Dni = dni;
        }

        string json = JsonConvert.SerializeObject(tasks, Formatting.Indented);
        File.WriteAllText(path, json);
    }
    catch (Exception ex)
    {
        System.Diagnostics.Debug.WriteLine($"Error saving tasks: {ex.Message}");
    }
}

```

En resumen, este servicio me permite tener una lista de tareas rápida y sencilla para cada usuario sin necesidad de usar la base de datos.

## TeacherService

Este servicio se encarga de todo lo relacionado con los profesores. Hereda del GenericService, así que ya tiene el CRUD básico, pero aquí le añadí métodos propios porque necesitaba más funcionalidades específicas.

- FindTeacherByGroupAsync: busca los profesores que son tutores del grupo que le paso.

```
public async Task<IEnumerable<Teacher>> FindTeacherByGroupAsync(string groupName)
{
    try
    {
        var teachers = await _context.Teachers
            .Where(t => t.TutorGroupNavigation.GroupName == groupName)
            .ToListAsync();

        return teachers;
    }
    catch (Exception ex)
    {
        LogError("Error while fetching teachers by group", ex);
        return Enumerable.Empty<Teacher>();
    }
}
```

- GetGroupByTeacherAsync: devuelve el grupo del que un profesor es tutor, si tiene.

```
1 referencia
public async Task<Group?> GetGroupByTeacherAsync(Teacher teacher)
{
    if (teacher == null || string.IsNullOrEmpty(teacher.TutorGroup))
        return null;

    return await _context.Groups
        .FirstOrDefaultAsync(g => g.GroupCode == teacher.TutorGroup);
}
```

- getAllAsycAndRoles: saca todos los profesores incluyendo también su rol (me viene bien para listados).

```
1 referencia
public async Task<IEnumerable<Teacher>> getAllAsycAndRoles()
{
    try
    {
        return await _context.Teachers
            .Include(t => t.Role)
            .ToListAsync();
    }
    catch (Exception ex)
    {
        LogError("Error while fetching all teachers with roles", ex);
        return Enumerable.Empty<Teacher>();
    }
}
// Fin TeacherService.cs
```

- HasMainTutorInGroupAsync: comprueba si ya hay un tutor principal asignado a un grupo.

```

public async Task<bool> HasMainTutorInGroupAsync(string groupCode)
{
    try
    {
        return await _context.Teachers
            .AnyAsync(t => t.TutorGroup == groupCode && t.IsTutor);
    }
    catch (Exception ex)
    {
        LogError("Error checking for existing main tutor", ex);
        return false;
    }
}

```

- DniExistsAsync: sirve para ver si ya existe un profesor con ese DNI (evita duplicados).

```

1 referencia
public async Task<bool> DniExistsAsync(string dni)
{
    return await _context.Teachers.AnyAsync(t => t.Dni == dni);
}

```

- DeleteAsyncPro: este es como el delete normal pero mejorado. Si el profesor está relacionado con incidentes, avisa antes de borrarlo, para que el usuario sepa que se perderán también esos datos.

```

1 referencia
public async Task<bool> DeleteAsyncPro(Teacher teacher)
{
    // Cargar relaciones si no vienen incluidas
    var loadedTeacher = await _context.Teachers
        .Include(t => t.IncidentRegisteredByNavigations)
        .Include(t => t.IncidentTeacherDniNavigations)
        .FirstOrDefaultAsync(t => t.Dni == teacher.Dni);

    if (loadedTeacher == null)
        return false;

    int totalIncidents = loadedTeacher.IncidentRegisteredByNavigations.Count + loadedTeacher.IncidentTeacherDniNavigations.Count;

    if (totalIncidents > 0)
    {
        var confirm = MessageBox.Show(
            $"This teacher is linked to {totalIncidents} incident(s).\nIf you continue, all related incidents will also be deleted.\n\nDo you want to proceed?",
            "Confirm Deletion",
            MessageBoxButton.YesNo,
            MessageBoxImage.Warning);

        if (confirm != MessageBoxResult.Yes)
            return false;
    }

    _context.Teachers.Remove(loadedTeacher);
    await _context.SaveChangesAsync();
    return true;
}

```

## Interfaz gráfica

La interfaz gráfica de EduAvis la he diseñado intentando que sea lo más clara y accesible posible para cualquier tipo de usuario, incluso para aquellos que no tienen muchos conocimientos de informática. Desde el principio me marqué como objetivo que cualquier persona, con solo abrir la aplicación, entendiera fácilmente qué tenía que hacer.

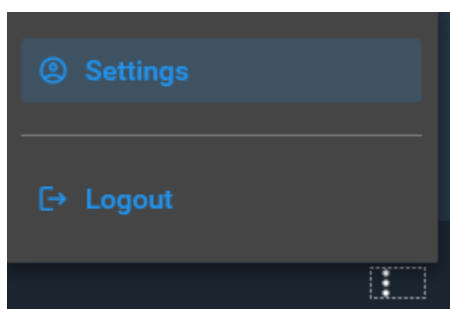
Para conseguirlo, me he centrado en destacar bien los botones más importantes, tanto con colores como con iconos, para que el usuario no tenga que pensar demasiado ni perder tiempo buscando las funciones principales. De esta forma, acciones como crear una incidencia, guardar o cerrar sesión se pueden hacer de forma rápida e intuitiva.



Este es el Dashboard donde a la izquierda hay un menú con iconos y un texto descriptivo de lo que hace cada botón.



Abajo hay 3 puntos, es un menú desplegable donde podemos cerrar sesión, esto es algo que muchas empresas utilizan, como por ejemplo Google





En la interfaz podemos observar que los botones que hacen algún tipo de acción tienen un color que destaca sobre el resto de la interfaz como ViewAll ,el + de QuickTask (añadir una tarea personalizada) y el tic verde para marcar como que ya has completado la tarea.

The screenshot shows the EduAvis application interface. On the left is a dark sidebar menu with options: Dashboard, Incidents, Administration, and Settings. The main area displays a table of incidents with columns: Student, Group, Reason, Date, Type, Status, and Actions. The table contains 16 rows of incident data. At the top right of the main area, there are filters and buttons: 'Filter by group' (set to 'All'), 'CLEAR FILTERS', and '+ NEW INCIDENT'.

Student	Group	Reason	Date	Type	Status	Actions
Wilson, Jack	1ESO-A	Disruptive behavior in class	01/04/2025 03:09	SANCTION	PENDING	
Taylor, Sophie	1ESO-A	Late arrival to class	01/04/2025 08:35	SANCTION	PENDING	
Cruz, Alexander	2ESO-A	Disrespect to teacher	02/04/2025 11:25	SANCTION	SANCTIONED	
Vargas, Jackson	4ESO-A	Use of mobile phone during class	02/04/2025 13:15	SANCTION	PENDING	
Contreras, Christopher	1BACH-A	Physical aggression against classmate	03/04/2025 10:30	SANCTION	SANCTIONED	
Carrillo, Anthony	2BACH-A	Verbal aggression against classmate	03/04/2025 09:45	SANCTION	SANCTIONED	
Cardenas, Thomas	1DAM	Skiping class	04/04/2025 08:00	SANCTION	PENDING	
Guzman, Lucy	1DAM	Damaging school property	04/04/2025 12:30	SANCTION	SANCTIONED	
Escobar, Eli	2DAM	Smoking in school premises	05/04/2025 11:00	SANCTION	SANCTIONED	
Velasquez, Madelyn	2DAM	Leaving school without permission	05/04/2025 14:15	SANCTION	PENDING	
Castaneda, Miles	2DAM	Copying during exam	06/04/2025 10:00	SANCTION	SANCTIONED	
Meza, Audrey	2DAM	Not completing assignments	06/04/2025 09:00	SANCTION	PENDING	
Brown, Lucas	1ESO-A	Illness during school hours	07/04/2025 10:15	WARNING	PENDING	
Johnson, Olivia	1ESO-A	Minor accident during break time	07/04/2025 12:30	WARNING	PENDING	
David, Noah	1ESO-A	Student fell ill and parents were notified	08/04/2025 05:10	WARNING	PENDING	

Showing 74 results

Este es el menú de incidentes. Como se puede observar, el panel lateral izquierdo es estático y permite navegar entre las distintas partes de la aplicación en cualquier momento.

El diseño mantiene en todo momento una gama de colores coherente. Los botones más importantes están destacados visualmente para guiar al usuario sin dificultad. Por ejemplo, el botón "New Incident" utiliza el mismo color que el botón "View All" de otras pantallas, lo cual refuerza la consistencia visual de la aplicación.

Además, he asociado el color rojo a las acciones de eliminación, una práctica habitual en interfaces de usuario. En las acciones disponibles para cada incidente (editar, borrar, etc.), se respeta este patrón de colores, lo que facilita la comprensión de las funcionalidades sin necesidad de explicación adicional.

<div>EduAvis</div> <div> <div>Dashboard</div> <div>Incidents</div> <div>Administration</div> <div>Settings</div> </div>	Teachers				Students				Roles			
	<input type="text"/> <div>Filter by role</div> <div>CLEAR FILTERS</div> <div>ADD TEACHER</div>											
	Full Name				Email				Role			
	admin, admin				admin				Administrator			
	Young, Laura				laura.young@school.edu				Administrator			
	Hall, Sharon				sharon.hall@school.edu				Director			
	rafa, rafa				123131@fmaad				Director			
	Allen, Mark				mark.allen@school.edu				Director			
	raad, Rafa				deandadas@dadadada				Director			
	cofete, cofete				pryba@preb				Teacher			
	Martin, Joseph				joseph.martin@school.edu				Teacher			
	Lee, Susan				susan.lee@school.edu				Teacher			
	Walker, Paul				paul.walker@school.edu				Teacher			
	Wilson, Patricia				patricia.wilson@school.edu				Tutor			
	rafa, rafa				rafa@rafa.com				Tutor			
	Navarro, Miguel				miguelnavarro@edu				Tutor			
	Smith, John				john.smith@school.edu				Tutor			

Aquí se muestra el **menú de administración**, el cual está restringido únicamente a los usuarios que cuentan con los permisos necesarios.

La estética de esta pantalla sigue la misma línea visual que la del menú de incidentes, manteniendo así una coherencia en el diseño de toda la aplicación. Sin embargo, hay una diferencia notable: en la parte superior se encuentra un menú con tres opciones bien diferenciadas. Gracias al trabajo de diseño, se ha conseguido que cada opción sea clara y comprensible antes incluso de hacer clic, permitiendo al usuario saber exactamente qué contenido verá con solo una mirada.

### PERSONAL INFORMATION

Enter the basic personal information

DNI \*

First Name \*Last Name \*

Email \*

### ROLE INFORMATION

Select the role and permissions for the teacher

Teacher Role \*

### CLASS ASSIGNMENT

Assign the group for this teacher (tutor or class teacher).

Select Group \*

### SECURITY

Set the password for the teacher account

Password \*

Aquí podemos ver que los campos que salen en rojo están mal y debe de modificarlos para que puedan ser insertados, igualmente en otros dialogos donde no hay esa interfaz de color

rojo para decirte lo que debes de cambiar, saltará un messagebox diciendo error:

The screenshot shows a web application window titled "NEW TEACHER" with a close button (X) in the top right corner. The window is divided into two main sections: "PERSONAL INFORMATION" and "ROLE INFORMATION".

**PERSONAL INFORMATION**

Enter the basic personal information

DNI \*  
dasdasdas

First Name \*  
Last Name \*

Email \*

**SECURITY**

Set the password for the teacher account

Password \*

**ROLE INFORMATION**

Select the role and permissions for the teacher

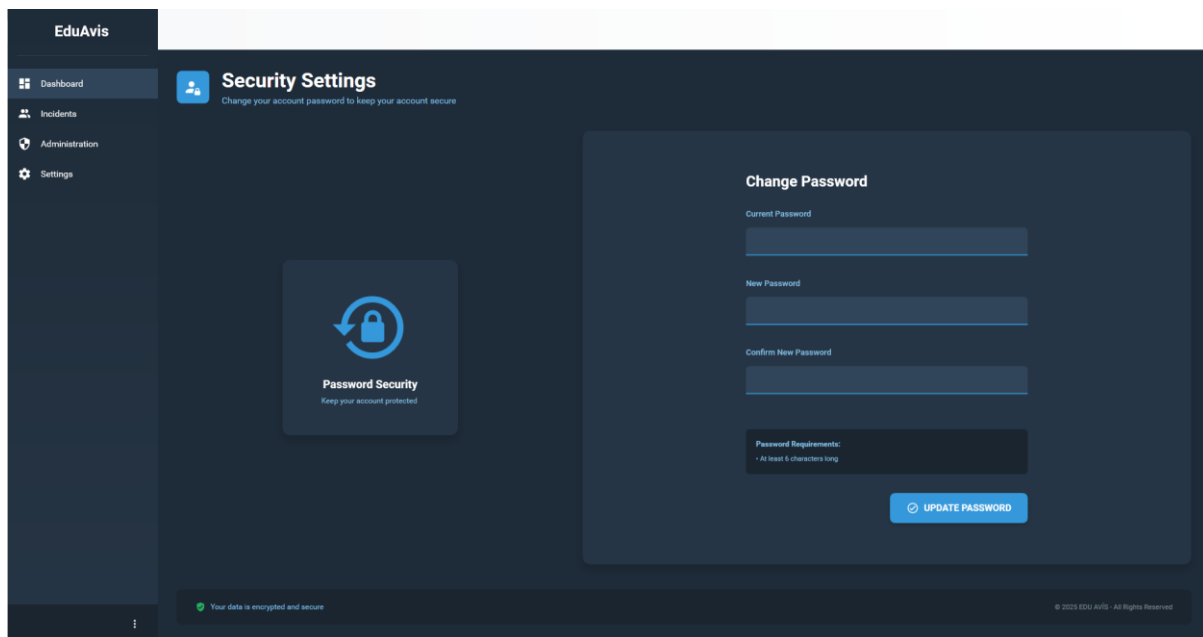
Teacher Role \*

Validation errors:  
DNI must be alphanumeric (no symbols).  
First name is required.  
Last name is required.  
Email is required.  
Password is required.  
Tutor group is required.

Acceptar

SAVE

En caso de guardar algo correctamente también se notificará



## Pantalla de configuración

En esta sección se permite al usuario cambiar su contraseña para mejorar la seguridad de su cuenta. Visualmente, se mantiene la misma línea de diseño que el resto de la aplicación, con una interfaz limpia y clara.

La disposición de los elementos está pensada para que el usuario entienda rápidamente qué debe hacer. El formulario para cambiar la contraseña está bien separado en bloques, y se acompaña de un pequeño aviso con los requisitos mínimos. Además, se incluye un mensaje que indica que los datos están cifrados, lo que transmite una mayor sensación de confianza y seguridad.

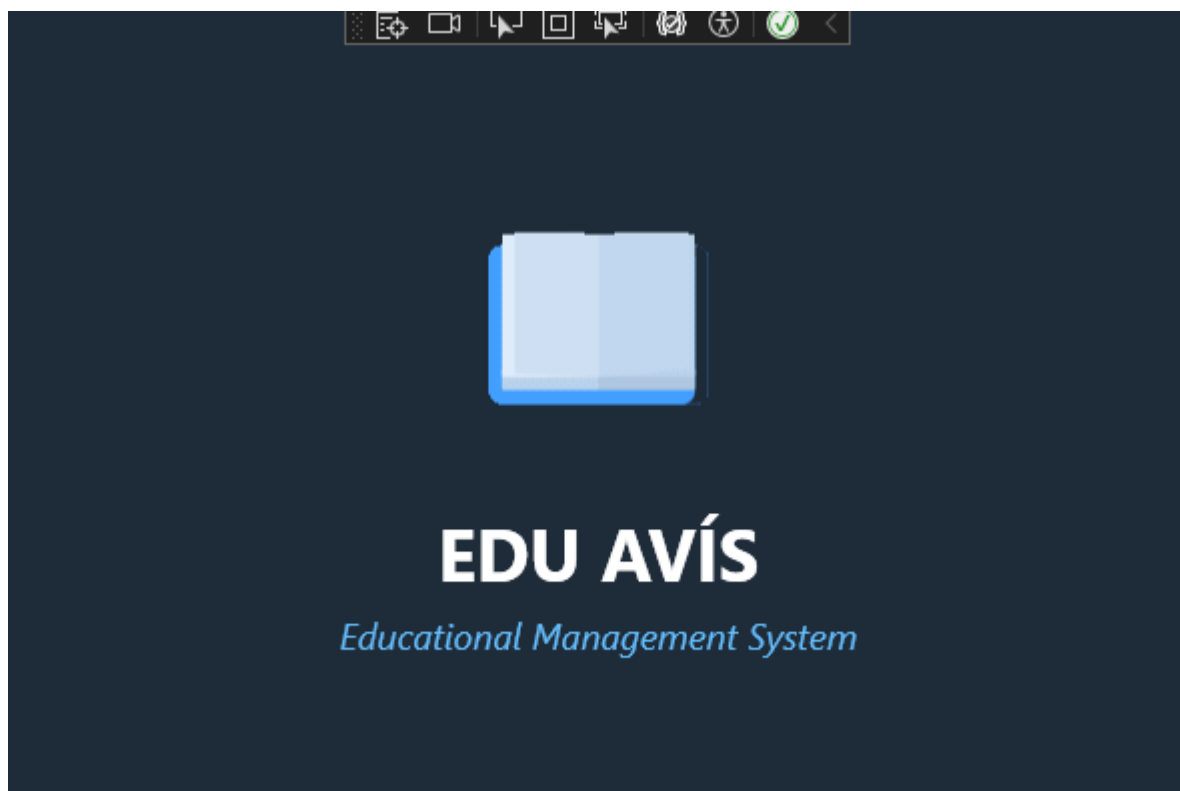
Este tipo de pantallas refuerzan la idea de que EduAvis ha sido diseñada teniendo en cuenta tanto la funcionalidad como la experiencia del usuario.

En resumen, el diseño de la interfaz de EduAvis está orientado a la simplicidad, la claridad visual y la eficiencia en el uso diario. El objetivo ha sido construir una herramienta intuitiva, moderna y funcional, adaptable a distintos perfiles de usuario

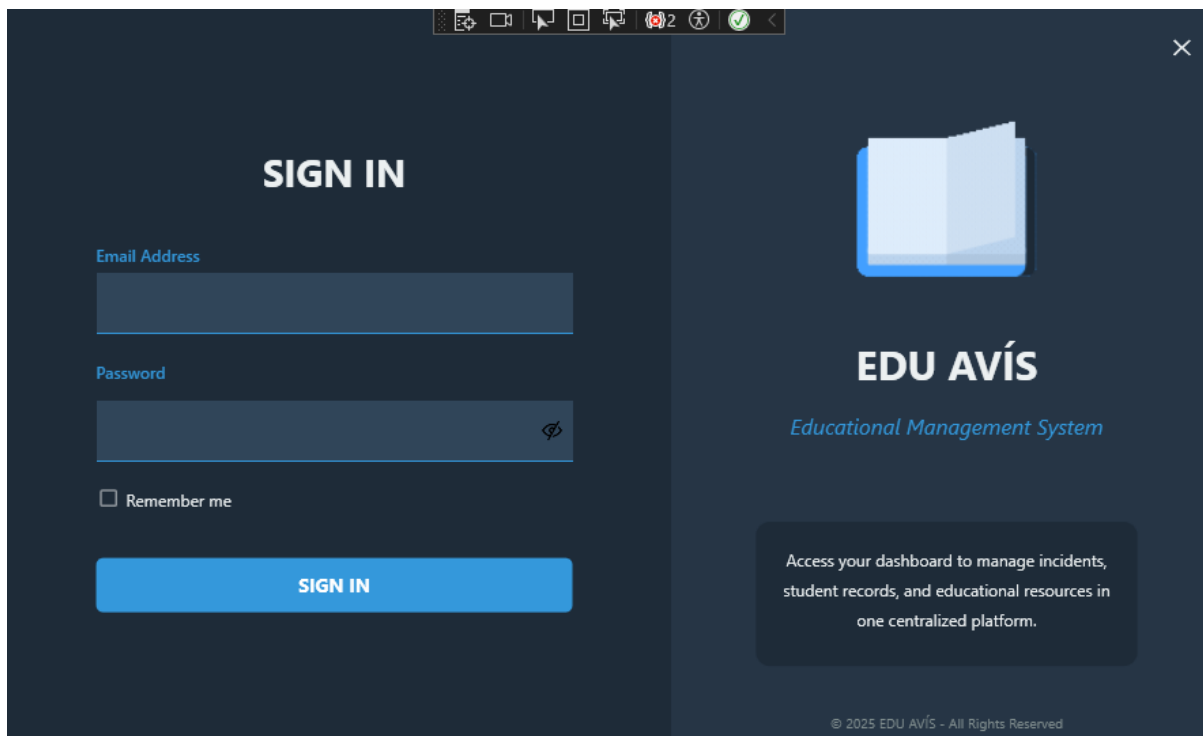
## Utilización del programa

Ahora voy a explicar detalladamente todo el programa.

La primera ventana que veremos será el SplashScreen donde hace la validación de que la base de datos está conectada correctamente ya sea que este la base de datos correctamente como el usuario y contraseña para entrar a MYSQL.



Luego cuando haya cargado correctamente nos aparecerá el login, donde deberemos iniciar sesión con nuestro correo electrónico y contraseña, el botón de “remember me” hará que encripte tanto el correo como la contraseña para evitar iniciar sesión cuando vuelvas a abrir la aplicación, podrás cerrar sesión dentro de la aplicación en caso de que no quieras se inicie automáticamente al abrirla.

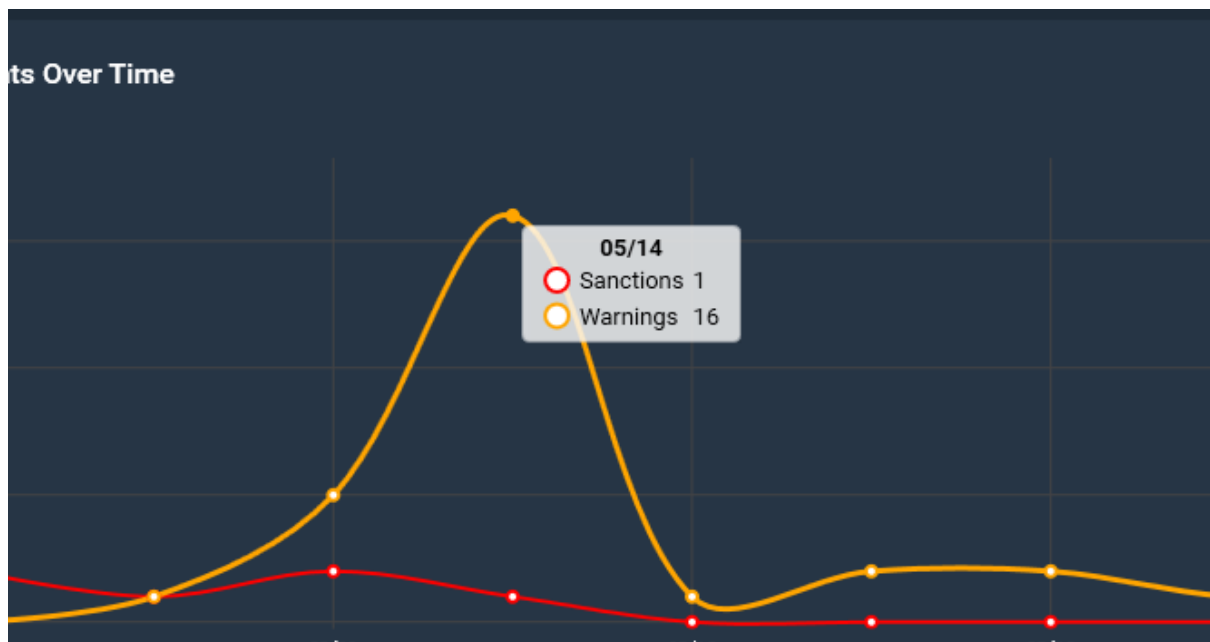


Luego nos abrió el Mainmenu con el UCDashboard abierto por predeterminado donde podremos ver información relevante. \*En este caso estoy en la vida del administrador entonces nos aparece el botón de Administration y podemos ver el Summary global, en caso de abrir con otro usuario se nos acotará dependiendo de los permisos que tengamos\*.

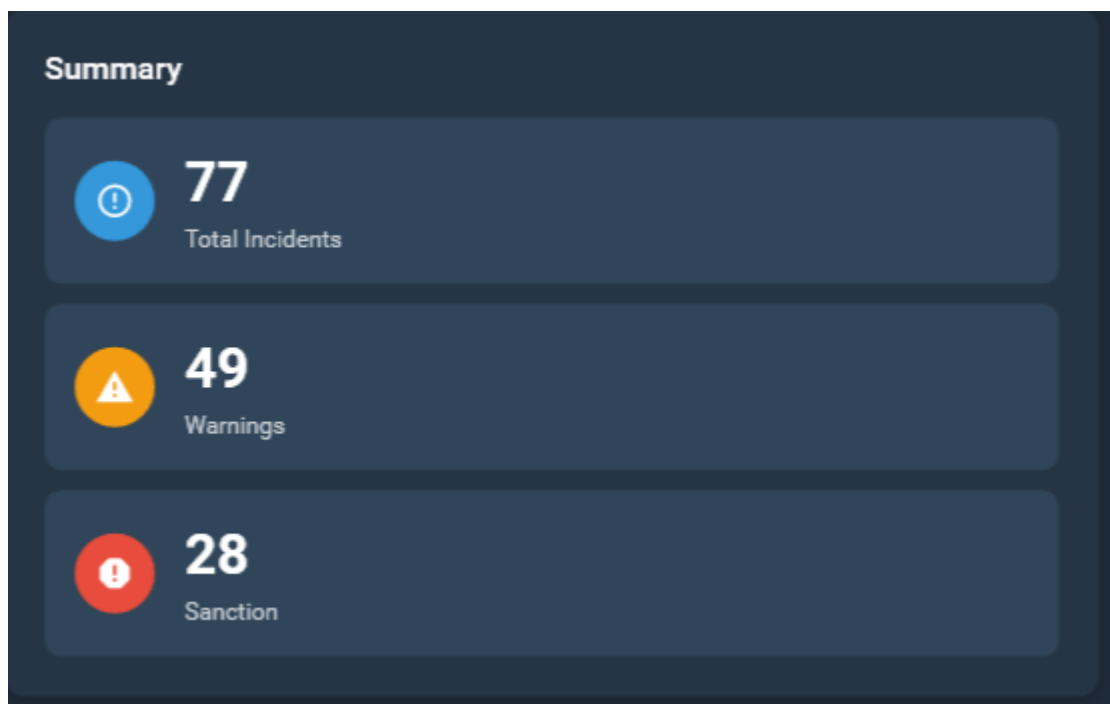


En el Dashboard podremos ver una gráfica de la cantidad de incidentes durante el tiempo, arriba a la derecha del gráfico podremos filtrar.

Si ponemos el cursor encima de algún punto podremos ver información detallada



El Summary tiene la información de los incidentes puestos, si amonestación o si es sanción, dependiendo del rol que sea el usuario verá más o menos, en mi caso como es la vista de administración veo todos, pero si fuese por ejemplo un profesor, solo vería los que ha puesto ese profesor o esté implicado.



Aquí veremos los 5 incidentes más recientes, también dependiendo del rol veremos más o menos.



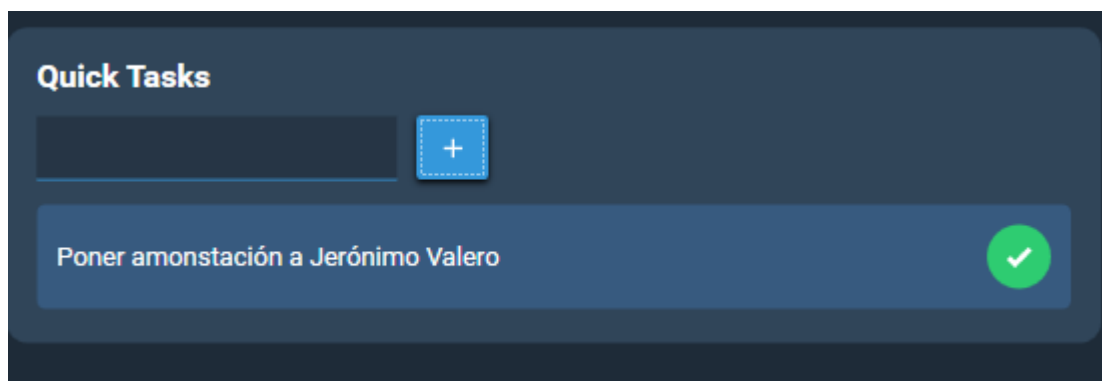
Recent Incidents			<a href="#">VIEW ALL</a>
Campos, Isaac	Disrespect to teacher	<b>WARNING</b>	
alias, hugo	Smoking in school premises	<b>WARNING</b>	
Feo, Samuel	Use of mobile phone during class	<b>WARNING</b>	
Alumno, Damian	Physical aggression against classmate	<b>WARNING</b>	
Acevedo, Savannah	Skipping class	<b>WARNING</b>	

Arriba a la derecha veremos un botón con el texto VIEW ALL, que con eso podremos ver la lista completa sin necesidad de entrar al apartado de incidentes.

All Incidents				
Student	Reason	Date	Type	
Wilson, Jack	Disruptive behavior in class	01/04/2025 03:09	<b>SANCTION</b>	
Taylor, Sophie	Late arrival to class	01/04/2025 08:30	<b>SANCTION</b>	
Cruz, Alexander	Disrespect to teacher	02/04/2025 11:20	<b>SANCTION</b>	
Vargas, Jackson	Use of mobile phone during class	02/04/2025 13:10	<b>SANCTION</b>	
Contreras, Christopher	Physical aggression against classmate	03/04/2025 10:30	<b>SANCTION</b>	
Carrillo, Anthony	Verbal aggression against classmate	03/04/2025 09:45	<b>SANCTION</b>	
Cardenas, Thomas	Skipping class	04/04/2025 08:00	<b>SANCTION</b>	
Guzman, Lucy	Damaging school property	04/04/2025 12:30	<b>SANCTION</b>	
Escobar, Eli	Smoking in school premises	05/04/2025 11:00	<b>SANCTION</b>	
Velasquez, Madelyn	Leaving school without permission	05/04/2025 14:15	<b>SANCTION</b>	

Aquí podremos ver todos los incidentes que ha habido, esto es la vista de administrador, si fuese un profesor solo los que estuviese implicado.

Y, por último, tenemos un apartado de quick tasks donde podremos apuntar tareas y que no se nos olviden. Si por ejemplo quieres poner una amonestación a un alumno, pero no tienes tiempo ahí lo puedes apuntar y con el check verde que aparece podrás eliminarlo.



Ahora vamos a ver el aparte de incidentes y explicaré todas las opciones que hay.

EduAvis - Educational Management System

Dashboard Incidents Administration Settings

Filter by group: All CLEAR FILTERS + NEW INCIDENT

Student	Group	Reason	Date	Type	Status	Actions
Wilson, Jack	1ESO-A	Disruptive behavior in class	01/04/2025 03:09	SANCTION	SANCTIONED	
Taylor, Sophie	1ESO-A	Late arrival to class	01/04/2025 08:30	SANCTION	SANCTIONED	
Cruz, Alexander	3ESO-A	Disrespect to teacher	02/04/2025 11:20	SANCTION	SANCTIONED	
Vargas, Jackson	4ESO-A	Use of mobile phone during class	02/04/2025 13:10	SANCTION	PENDING	
Contreras, Christopher	1BACH-A	Physical aggression against classmate	03/04/2025 10:30	SANCTION	SANCTIONED	
Carrillo, Anthony	2BACH-A	Verbal aggression against classmate	03/04/2025 09:45	SANCTION	SANCTIONED	
Cardenas, Thomas	1DAM	Skipping class	04/04/2025 08:00	SANCTION	PENDING	
Guzman, Lucy	1DAM	Damaging school property	04/04/2025 12:30	SANCTION	SANCTIONED	
Escobar, Eli	2DAM	Smoking in school premises	05/04/2025 11:00	SANCTION	SANCTIONED	
Velasquez, Madelyn	2DAM	Leaving school without permission	05/04/2025 14:15	SANCTION	PENDING	
Castaneda, Miles	2DAM	Copying during exam	06/04/2025 10:00	SANCTION	SANCTIONED	
Meza, Audrey	2DAM	Not completing assignments	06/04/2025 09:20	SANCTION	PENDING	
Brown, Lucas	1ESO-A	Illness during school hours	07/04/2025 10:15	WARNING	PENDING	
Johnson, Olivia	1ESO-A	Minor accident during break time	07/04/2025 12:30	WARNING	PENDING	
Davis, Noah	1ESO-A	Student fell ill and parents were notified	08/04/2025 09:10	WARNING	PENDING	

Showing 77 results

Como podemos observar aquí están todos los incidentes que hay en el centro debido a que tengo el permiso de administrador, si fueses otro rol solo verías los pertinentes.

Primero vamos con los filtros, hay 3 filtros, uno por búsqueda de nombre, otro por grupo y otro por estado, para ver los que están en pendiente de informar o los que ya han sido informados.

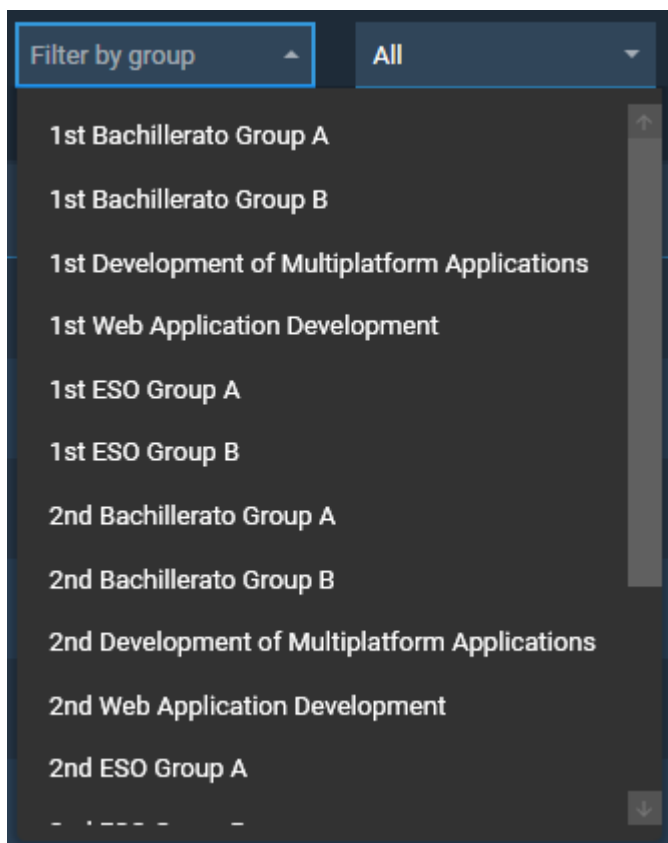
Q Eli

Student	Group	Reason	Date
Escobar, Eli	2DAM	Smoking in school premises	05/04/2025 11:00
Flores, Elijah	2ESO-B	Minor accident during break time	18/04/2025 09:45

Aquí podemos ver que funciona correctamente y no tenemos porque buscar por su nombre, si pusiésemos su apellido o parte del también saldría.

<input type="text" value="Esc"/>			
Student	Group	Reason	Date
Escobar, Eli	2DAM	Smoking in school premises	05/04/2025 11:00

Luego más a la derecha podremos ver el filtro de grupo, que nos saldrá una lista de todos los grupos existentes y podremos filtrar por ella.

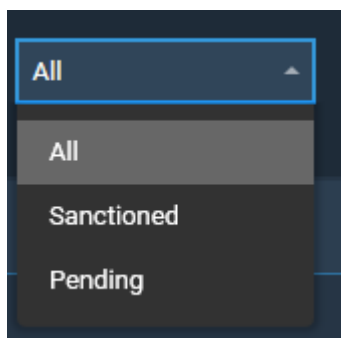


<input type="text"/> <div>1st ESO Group A</div> <div>All</div>				
Student	Group	Reason	Date	Type
Wilson, Jack	1ESO-A	Disruptive behavior in class	01/04/2025 03:09	SANCTION
Taylor, Sophie	1ESO-A	Late arrival to class	01/04/2025 08:30	SANCTION
Brown, Lucas	1ESO-A	Illness during school hours	07/04/2025 10:15	WARNING
Johnson, Olivia	1ESO-A	Minor accident during break time	07/04/2025 12:30	WARNING
Davis, Noah	1ESO-A	Student fell ill and parents were notified	08/04/2025 09:10	WARNING
Wilson, Jack	1ESO-A	Disruptive behavior in class	11/04/2025 11:55	SANCTION
Wilson, Jack	1ESO-A	Disrespect to teacher	13/04/2025 09:00	SANCTION
Davis, Noah	1ESO-A	Student fell ill and parents were notified	14/04/2025 11:30	WARNING
Davis, Noah	1ESO-A	Disrespect to teacher	14/05/2025 02:10	SANCTION
Johnson, Olivia	1ESO-A	Physical aggression against classmate	14/05/2025 23:57	WARNING
Davis, Noah	1ESO-A	Disrespect to teacher	14/05/2025 00:05	WARNING
Davis, Noah	1ESO-A	Physical aggression against classmate	14/05/2025 01:05	WARNING
Brown, Lucas	1ESO-A	Damaging school property	29/05/2025 12:00	WARNING
dasdas, asdasd	1ESO-A	Damaging school property	01/01/0001 00:00	WARNING

Aquí un ejemplo de su uso.

Por último, podremos filtrar por su estado.

Que existen All (Todos), Pending (Pendiente) y Sancionted (Sancionado).



Obviamente se puede juntar filtros, por si quisieses ver los pendientes de una clase en específico.

1st ESO Group A

Sanctioned

CLEAR FILTERS

NEW INCIDENT

Student	Group	Reason	Date	Type	Status	Actions
Wilson, Jack	1ESO-A	Disruptive behavior in class	01/04/2025 03:09	SANCTION	SANCTIONED	<div><div></div><div></div><div></div></div>
Taylor, Sophie	1ESO-A	Late arrival to class	01/04/2025 08:30	SANCTION	SANCTIONED	<div><div></div><div></div><div></div></div>
Wilson, Jack	1ESO-A	Disrespect to teacher	13/04/2025 09:00	SANCTION	SANCTIONED	<div><div></div><div></div><div></div></div>
Brown, Lucas	1ESO-A	Damaging school property	29/05/2025 12:00	WARNING	SANCTIONED	<div><div></div><div></div><div></div></div>

Luego tenemos la opción de Clear Filters que nos permitirá eliminar todos los filtros de una vez.

Por último, tenemos el New Incident que es el más importante, donde el usuario podrá poner incidencias que se guardarán en la base de datos.

**NEW INCIDENT**

**TEACHER INFORMATION**

Select the teacher related to the incident

☐ Is the teacher writing the report?

**STUDENT INFORMATION**

Select the student related to the incident

**REASON**

Select the reason for the incident

**DATE and TIME**

Please select the date and time when the incident occurred

6/6/2025

22:16

**DESCRIPTION**

**CLASSIFICATION**

Set the classification of the incident

☐ Mark as sanction

**SAVE**

A continuación, explico brevemente el funcionamiento de cada uno de los campos que se deben completar al registrar una incidencia en la aplicación:

En el apartado *Teacher Information* se debe indicar qué profesor está implicado en la incidencia. Existe un botón que permite asignarse a uno mismo automáticamente (es decir, el usuario que está registrando la incidencia).

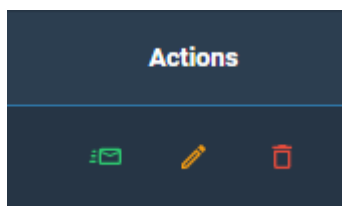
En *Student Information*, se debe seleccionar al alumno implicado. Para ello, primero se elige el curso al que pertenece y, a continuación, se selecciona el alumno concreto dentro de ese curso.

El campo *Reason* permite seleccionar el motivo que mejor se ajusta a la situación. Si no hay ninguno que encaje, siempre se puede seleccionar la opción “Other” para especificar otro tipo de motivo.

En *Date and Time* se debe indicar la hora exacta en la que ocurrió el incidente.

En el campo *Description* podemos escribir libremente una descripción personalizada del incidente. Esta información es importante, ya que más adelante se incluirá en el correo electrónico que se enviará con el resumen de la incidencia.

Por último, se puede marcar la opción *Sanctioned* en caso de que el incidente conlleve una sanción. Si esta casilla no se marca, la incidencia se registrará por defecto como una simple amonestación.



Dentro de la tabla donde aparecen listados los estudiantes implicados en incidencias, encontraremos tres iconos que nos permiten interactuar con cada registro:

- El primer icono es una carta de color verde. Al pulsarla, se enviará el parte directamente al estudiante correspondiente. Además, si el incidente aún no había sido informado, el sistema cambiará automáticamente su status de *Pending* a *Sanctioned*.



**EduAvis System** <eduavis.system@gmail.com>

para mí ▾



Traducir al español



## Disciplinary Incident Notification

Dear student,

We would like to inform you of a disciplinary incident involving the following student:

<b>Student:</b>	López, Rafae
<b>Group:</b>	2DAM
<b>Date of Incident:</b>	06/06/2025
<b>Reason:</b>	Copying during exam
<b>Type:</b>	Sanction
<b>Description:</b>	Ha copiado en el examen utilizando el móvil y al retirarle el examen se ha puesto a gritar.

...

This notice is part of the disciplinary protocol followed at EduAvis. Please review the incident accordingly.

Sincerely,

**EduAvis Disciplinary System**

- El segundo icono es un lápiz de color amarillo. Esta opción permite modificar la incidencia con total libertad, ya sea para corregir algún dato o actualizar la información registrada.

MODIFY INCIDENT

TEACHER INFORMATION

Select the teacher related to the incident

1st Web Application De

Hernandez, Charles

☐ Is the teacher writing the report?

REASON

Select the reason for the incident

Copying during exam

DESCRIPTION

Ha copiado en el examen utilizando el móvil y al retirarle el examen se ha puesto a gritar.

STUDENT INFORMATION

Select the student related to the incident

2nd Development of Mi

López, Rafae

DATE and TIME

Please select the date and time when the incident occurred

6/6/2025

22:22

CLASSIFICATION


Set the classification of the incident

☒ Mark as sanction
 ☒ Mark as sanctioned

UPDATE

- Por último, encontramos el icono de una papelera de color rojo. Esta opción permite eliminar completamente la sanción en caso de que haya sido registrada por error o ya no sea necesaria.

Confirm Delete



¿Are you sure you want delete this incident?

Sí

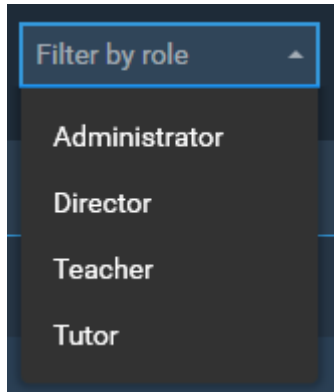
No

En la sección de Administración, tenemos acceso completo a la gestión tanto de estudiantes como de profesores. Desde esta vista, podemos crear, modificar y eliminar cualquier usuario de tipo alumno o profesor según sea necesario.



También tenemos la parte de roles donde podremos modificar los permisos de cada rol, pero no podremos crear ninguno nuevo.

En el apartado de profesores tenemos el mismo filtro de búsqueda y un filtro dependiendo del rol.



Tendremos también como en incidentes la opción de limpiar filtros y el *Add Teachers*, para añadir profesores.

## NEW TEACHER

### PERSONAL INFORMATION

Enter the basic personal information

DNI \*

First Name \* Last Name \*

Email \*

### ROLE INFORMATION

Select the role and permissions for the teacher

Teacher Role \*

### CLASS ASSIGNMENT

Assign the group for this teacher (tutor or class teacher).

Select Group \*

### SECURITY

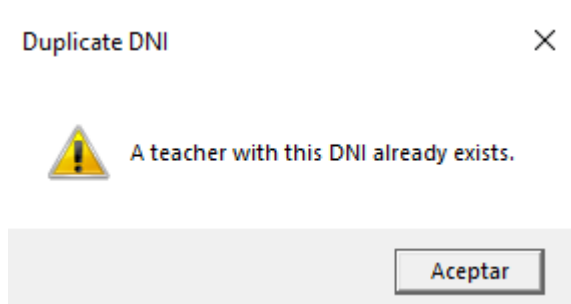
Set the password for the teacher account

Password \*

SAVE

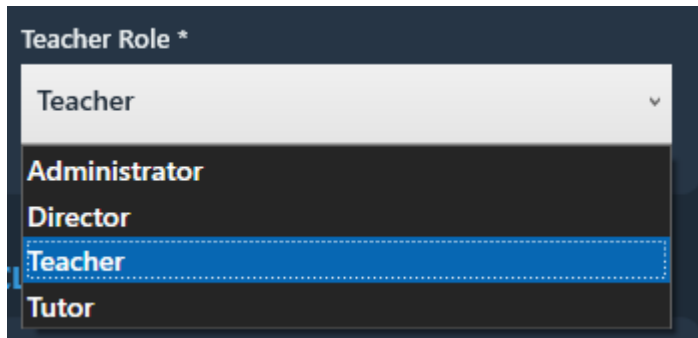
Aquí podremos deberemos de rellenar este formulario para añadir un profesor.

En *Personal Information* deberemos de poner el DNI, nombre, apellidos y DNI, están en rojo hasta que el usuario escriba algo apto para la inserción, en caso de poner un DNI que ya existe la aplicación nos avisará.

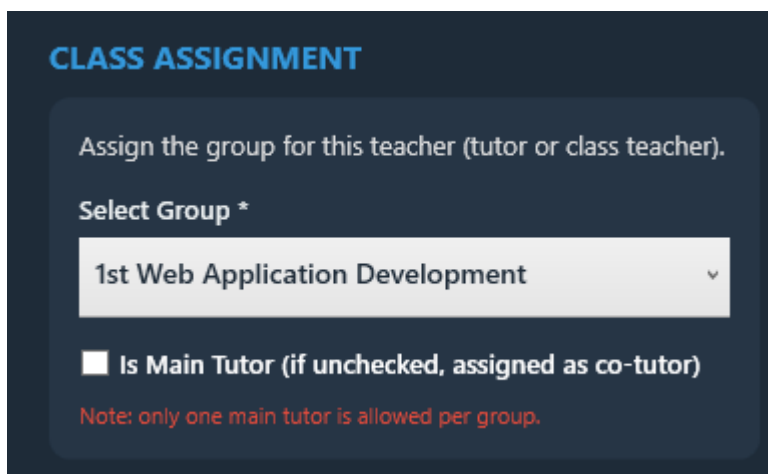


En *security* deberemos de poner la contraseña con el que el usuario iniciará sesión, debe de tener al menos 6 caracteres.

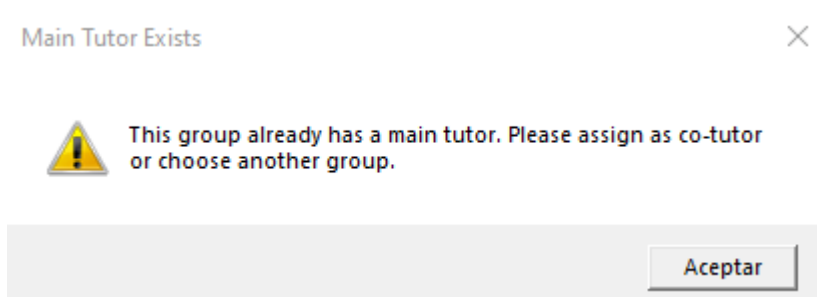
En *Role Information* nos dará los roles existentes el cual dependiendo del rol tendrá unos permisos u otros.



Quiero hacer énfasis en si eliges la opción de Tutor, ya que nos aparecerá una opción extra.




Aquí deberemos de poner a la clase que pertenece el usuario, en caso de elegir el rol de tutor deberá tener en cuenta que solo puede existir un tutor principal por clase, por lo cual si dejamos en blanco esa opción será co-tutor de la clase elegida. En caso de marcarla hará una comprobación de si existe ya un tutor principal en esa clase.



Luego, en esta parte tenemos las mismas opciones que antes, tanto para editar como para eliminar profesores. Si intentamos borrar uno que esté relacionado con algún incidente ya sea porque lo ha registrado o porque es el afectado, la aplicación nos avisará indicando en cuántos incidentes está implicado, y nos preguntará si queremos borrarlo igualmente. Si confirmamos, se eliminarán también todas las incidencias en las que dicho profesor esté involucrado.

Confirm Deletion

✕



This teacher is linked to 39 incident(s).  
If you continue, all related incidents will also be deleted.

Do you want to proceed?

Sí

No

En el apartado de estudiantes es un simil al de profesores, lo único que cambia es el formulario.

NEW STUDENT

✕

PERSONAL INFORMATION

Enter the basic personal information

NIA \*

First Name \*

Last Name \*

Email \*

Phone (Optional)

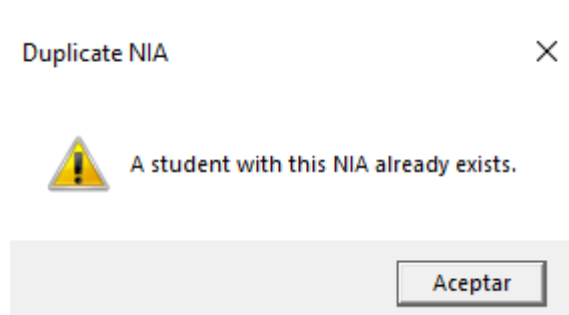
GROUP ASSIGNMENT

Assign the student to a group

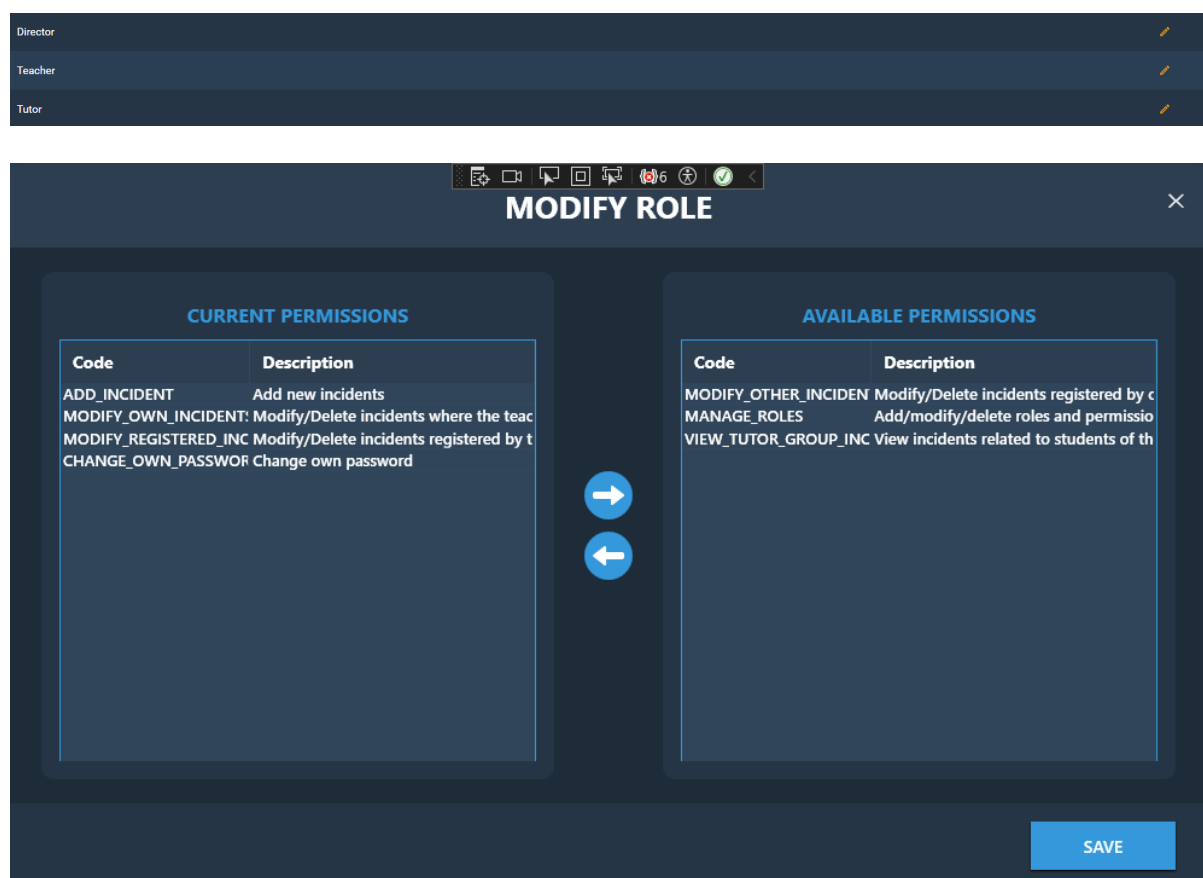
Select Group \*

SAVE

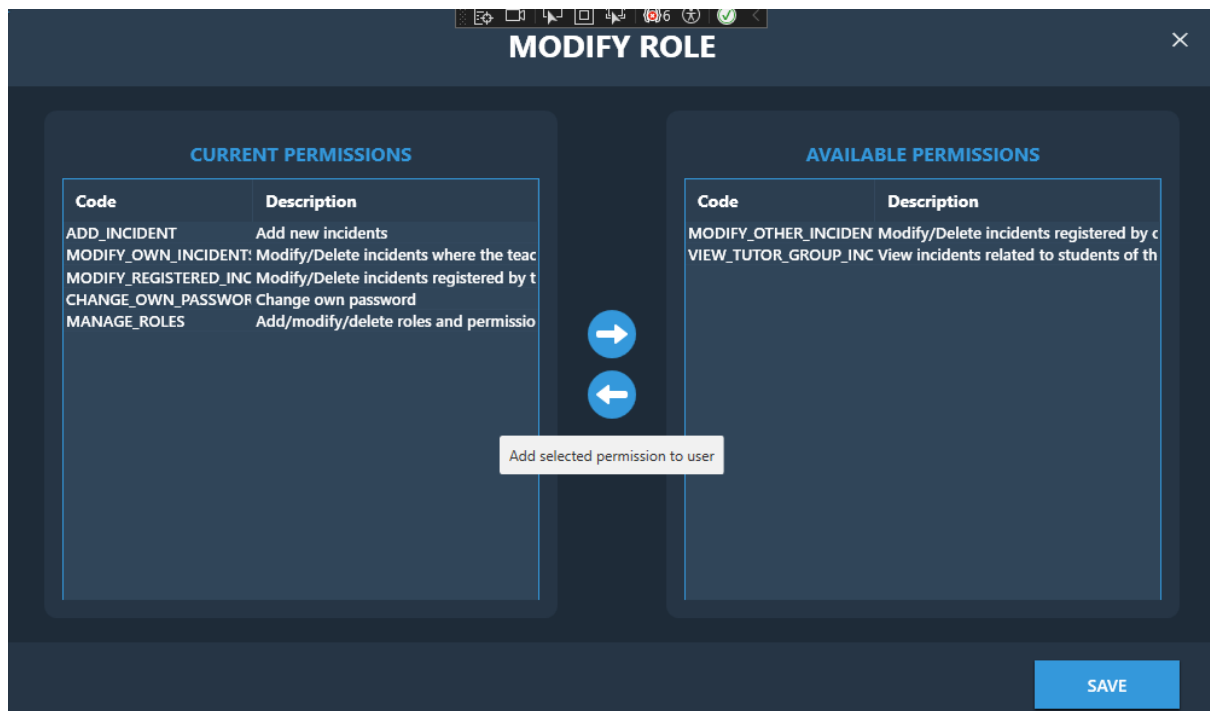
Aquí igual que en el de los profesores, en caso de poner un NIA que exista nos avisará de que no podemos crear el estudiante debido a eso.



En el último tab tendremos la opción de roles donde podremos modificar los roles existentes a nuestro placer utilizando el icono del lapiz amarillo.



Este es el dialogo de modificar role, donde a la izquierda tendremos los actuales permisos de rol, a la izquierda saldrán los roles restantes que existen y podremos añadir. Para añadir o eliminar permiso solo deberemos dar click al permiso a quitar/añadir y a la flecha donde queremos enviarlo. Por ejemplo, si quisiese poner el role `MANAGE_ROLES`, solo debería de clicarle al permiso y darle a la flecha de la izquierda



## CONCLUSIONES

### Dificultades Encontradas

A lo largo del desarrollo me he encontrado con bastantes dificultades, algunas más recientes y frustrantes que otras, pero en general han estado presentes desde el principio.

Una de las primeras fue con los ComboBox que utilizaba para filtrar alumnos. No conseguía que se sincronizasen correctamente y me tiré varios días dando vueltas hasta que di con el fallo, que al final resultó ser una tontería.

También tuve líos con la parte gráfica de los *DateTime* y *TimeSpan*. No había manera de que se mostrase bien el *TimeSpan*, y tras mucho pelearme descubrí que lo que tenía que usar era *DateTime* directamente.

Con las gráficas también me he encontrado problemas, sobre todo una excepción de *null reference* que saltaba sin motivo aparente. Hablándolo con Jero, me comentó que era un fallo de la propia librería, así que no tenía mucho que hacer.

Otra cosa que me trajo de cabeza fue mantener el contexto siempre actualizado. Cada vez que creaba, editaba o borraba algo, tenía que asegurarme de que el resto de la aplicación (sobre todo los *UserControls*) lo reflejara bien, si no perdía el sentido del programa.

También tuve un caso raro donde no podía borrar algunos alumnos o profesores, mientras que otros sí. Después de investigar bastante, descubrí que los que no se podían eliminar tenían incidentes asociados, y como Entity Framework no pone el Cascade por defecto, simplemente no los borraba aunque la aplicación decía que sí.

Y, sin duda, lo peor ha sido que de vez en cuando la aplicación deja de hacer las operaciones CRUD sin ninguna razón lógica. Funciona casi siempre, pero a veces simplemente deja de hacerlo. Este error aleatorio ha sido el más desesperante de todos.

### Ampliaciones a futuro

De cara al futuro, hay varias cosas que me gustaría mejorar o añadir. Lo primero sería pulir todos los MessageBox o directamente eliminarlos, ya que no terminan de encajar con un diseño moderno.

También me gustaría que se pudiera importar la base de datos de un colegio completo con solo un clic y un archivo, facilitando así la puesta en marcha del sistema.

Otra idea que tengo en mente es incorporar algún tipo de inteligencia artificial que permita hacer consultas rápidas o interpretar mejor lo que busca el usuario, especialmente en apartados como las estadísticas o la gestión de incidentes.

Aunque el enfoque del proyecto está pensado para profesorado, me gustaría que en un futuro también pudieran acceder los alumnos, con funciones adaptadas a ellos. Incluso me planteo añadir un chat tipo WhatsApp para que puedan comunicarse de forma directa con los profesores.

Por último, me gustaría que tanto profesores como alumnos pudieran tener una foto de perfil y que su perfil se pareciera un poco más a una red social.

## Conclusiones Personales

Aunque pueda sonar al típico texto, la verdad es que con este proyecto he aprendido muchísimo de .NET, sobre todo porque era mi primer proyecto “grande” y real. Si hoy tuviera que empezarlo desde cero, cambiaría bastantes cosas, tanto a nivel de organización como en el diseño de la base de datos. Pero claro, el tiempo era limitado y no podía permitirme tirar todo atrás solo por dejarlo más limpio o eficiente.

También he pensado que quizás me hubiese gustado hacer un proyecto que me motivara aún más, algo más personal o con una idea propia. Pero soy consciente de que tengo tendencia algo ambiciosa, y seguramente si hubiera hecho eso me hubiese quedado grande para mí.

Aun así, me quedo con todo lo que he aprendido: desde cómo organizar un proyecto real, hasta cómo gestionar problemas técnicos del día a día, pasando por el trato con herramientas y librerías que nunca había usado. Siento que he dado un salto muy grande como programador.

Y la verdad que he tenido muchos altibajos también, sobre todo por tener que programar después de las prácticas y haber empezado todo un poco tarde en general. Solo con pensar que el examen de .NET lo suspendí y hoy he sido capaz de sacar adelante algo así, me ha motivado mucho para seguir aprendiendo y mejorando.