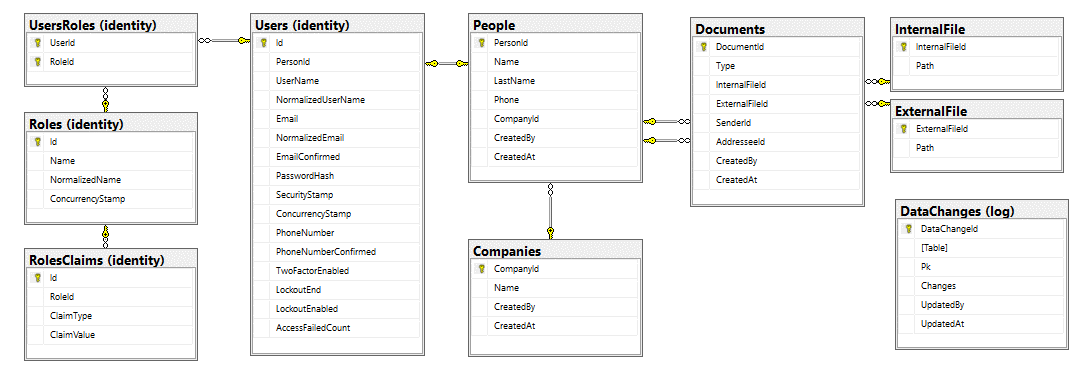
**MODELO DE DATOS**



Para el manejo de la seguridad de la aplicación se optó por usar el identity server, este framework nos ofrece un modelo de datos ya definido. Sobre esta implementación cabe mencionar que se configuró en un esquema independiente (identity), buscando una mejor organización en nuestra base de datos.

Debido a que se definió securizar nuestra api granularmente, en la tabla RolesClaim se guardarán las claims correspondientes a los permisos que tiene cada role sobre el acceso a los recursos de nuestra api. Para tratar de darle más claridad a esta característica se muestra la siguiente imagen.

Interfaz de usuario gráfica

Descripción generada automáticamente

Sobre cada recurso de datos, en este caso las tablas Documents, People, Companies y Users, se configuraron 3 tipos de acceso Búsquedas (query), Agregar información (add) o Actualizar información (edit), dependiendo del rol se le asignarán los permisos correspondientes para poder acceder a los métodos de la API relacionadas con estas acciones. En el caso del rol UserManager, este tendrá permisos para realizar cualquier acción sobre la tabla Document (búsquedas, agregar o editar información); sobre la tabla People sólo tendrá permisos para realizar búsquedas y en las demás tablas este rol no tiene ningún permiso de acceso. El rol User, sólo tiene acceso para realizar búsquedas sobre la tabla Documentos.

Los permisos dados a cada rol se muestran en la siguiente tabla.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Otro claim que se creó es el view, la idea con este es poder determinar en el frontend de la aplicación que recursos el usuario puede visualizar. Por tiempo esta característica no se implementó en el frontend, para los accesos en el front se utilizó el rol.

Tabla

Descripción generada automáticamente

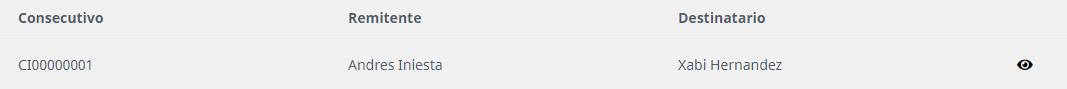
El modelo de datos diseñado para dar solución a los problemas planteados se muestra a continuación:

Diagrama

Descripción generada automáticamente

En el diseño del modelo de datos se decidió utilizar una tabla People independiente de la Users, en la cual se almacenará la información de las personas. Se pensó en esta tabla ya que no todas las personas que se registran en la plataforma son usuarios de ella, por ejemplo, los remitentes de una comunicación externa. Con esta tabla entonces, se centraliza la información de las personas que interactúan en la comunicación solventando el problema que no todas las personas son usuarios evitando romper la 3NF que se dio como prerrequisito para el diseño del modelo de datos. Con esta misma idea, se creó la tabla Company, con la cual se registrará la información de las empresas en las que las personas involucradas en una comunicación trabajan.

La tabla Document es el centro de la solución. Esta tabla se pensó para almacenar la información de la comunicación, en esta se registra el remitente, el destinatario, el tipo de comunicación (interna o externa). Para solucionar el requisito de un consecutivo para las comunicaciones internas y externas, se optó por crear una tabla independiente para cada una de ellas, y aprovechar el índice autoincremental de sus llaves primarias para esta tarea. Con esto se evita tener que calcular este consecutivo cada vez que se genere un nuevo documento, ya que en mi experiencia esto suele generar problemas cuando se presenta alta concurrencia. En este caso el consecutivo en el formato que lo requiere el negocio (CI000000001 o CI000000002) se genera a partir de la información almacenada en la tabla. El CI o CE se saca dependiendo del campo Type, y el consecutivo es el número de la llave primaría de la tabla correspondiente a cada tipo de documentación. En nuestro caso en el frontend implementado se dio solución a esto usando un pipe.



Tabla

Descripción generada automáticamente

Por último, tenemos el log de cambios de datos. El diseño de la tabla es simple: en el campo table registramos el nombre de la tabla donde se efectuó la modificación; en campo pk, se registra la llave primaria del registro modificado; en el campo changes se registran todos los cambios que ocurrieron en la entidad; por último, los campos de la persona que hizo la modificación y en qué fecha y hora se realizó.

Interfaz de usuario gráfica, Texto, Aplicación, Tabla

Descripción generada automáticamente con confianza media

Las clases que requieran esta funcionalidad deben heredar de la clase Auditable.

Cabe aclarar que cuando se cree un nuevo registro este no generará registros en la tabla de logs, para saber el usuario que creó el registro y la hora de creación las tablas que heredan de la clase Auditable ya cuentan con campos para almacenar esta información. En esta tabla se registrarán únicamente los cambios que se hagan en datos previamente registrados. Esto se hizo por una decisión de diseño, si se llegara a requerir se puede cambiar este comportamiento, para que tanto la creación como la actualización registren información en la tabla de logs.

En el frontend los cambios realizados sobre las tablas se pueden ver de la siguiente forma.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico, Teams

Descripción generada automáticamente

Deudas Técnicas:

* No se crearon servicios con paginación
* Los claims view asignados a los roles no se usaron en el frontend para permitir o no el ingreso a los recursos por parte de los usuarios.
* El diseño del frontend puede mejorar. Se hizo algo básico.
* El trigger asociado a la tabla donde se almacenan las comunicaciones no lo realicé, no se me ocurrió cómo sacar la información de los datos que cambiaban usando el trigger.
* Algunas más que estos momentos no recuerdo.