Universidad de Oviedo

Memoria del Trabajo Fin de Máster realizado por

Carlos Otero Verdejo

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

Desarrollo de un servicio de notificación de averías en la instalación

FEBRERO 2017

INDICE

INDICE DE FIGURAS

1.- INTRODUCCIÓN

1.1.- Identificación del proyecto

Título:	Desarrollo de un servicio de notificación de averías en la instalación
Tutor Empresa:	Javier Piñera
Tutor académico:	José Antonio Cancelas Caso
Autor:	Carlos Otero Verdejo
Fecha:	Febrero de 2017
Financiación:	Mecalux Software Solutions

1.2.- Marco contextual

Desde los últimos años el uso de dispositivos móviles, ya sean Smartphone o tabletas, ha aumentado drásticamente entre la población. Desde el año 2000, cuando se puso en el mercado el primer dispositivo con cámara, el número de usuarios de móviles se quintuplicado, y para el año 2020 se prevé que llegara a los 5500 millones. Esto representaría aproximadamente el 70% de la población mundial, según dio a conocer el Cisco en su informe *Visual Networking Index Global Mobile Data Traffic (2015 a 2020).*

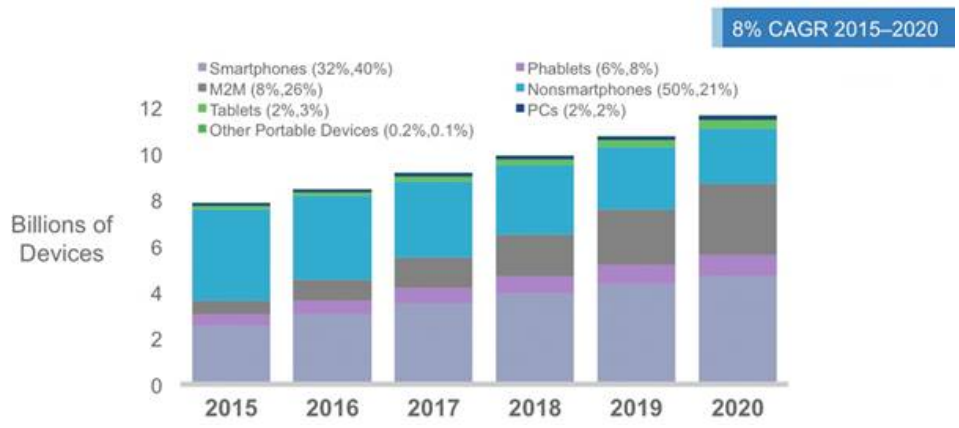

Figura 1.1.- Tráfico global de datos móviles

La creciente mejora de las prestaciones de estos dispositivos hace que cada vez sea posible realizar muchas más tareas. Las empresas han visto el potencial de estos dispositivos y por ello muchas de ellas han decidido emplearlos en su ámbito de trabajo. El uso de dispositivos móviles en las empresas facilita la circulación de la información, haciendo que esta llegue más rápido a su destino. A su vez también se reduce el uso de recursos, tema muy importen en las grandes industrias.

En la figura 1.2 se observa los resultados del informe realizado por Cisco. Como se puede apreciar entorno al año 2007 había más dispositivos conectados que personas en el mundo. Si nos fijamos en el 2015 esa cifra ya se había triplicado, lo que significa que hoy en día el número de dispositivos móviles por persona rondara los 4.

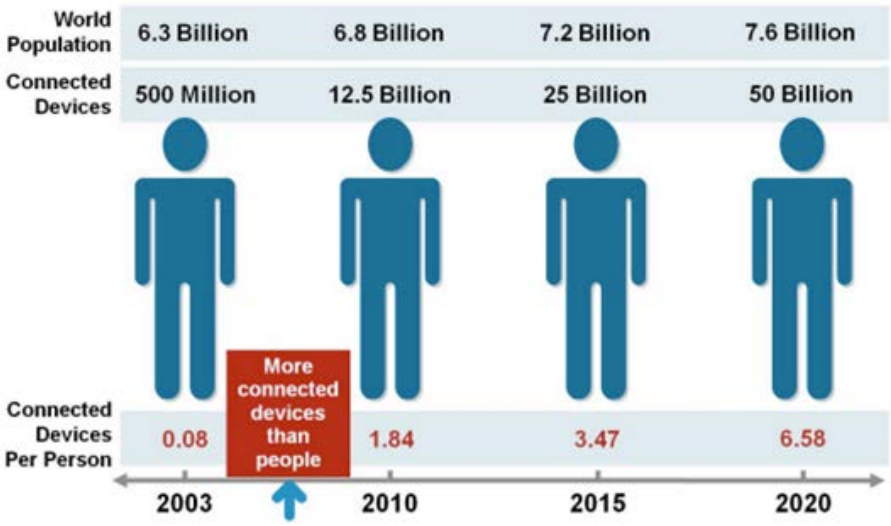

Figura 1.2.- Dispositivos conectados en la población

2.- DESCRIPCIÓN DEL SISTEMA

2.1.- Mecalux Software Solutions

El sistema que se expone en el presente documento ha sido desarrollado en la empresa Mecalux Software Solutions. Dicha empresa está situada en el polígono industrial de los Campones en Gijón, Asturias.

Mecalux Software Solutions es una sub-empresa de Mecalux S.A cuya principal ocupación son los sistemas de almacenaje. Su actividad consiste en el diseño, fabricación, comercialización y prestación de servicios relacionados con las estanterías metálicas, almacenes automáticos y otras soluciones de almacenamiento.

En concreto Mecalux Software Solutions se encarga de realizar software para la gestión de almacenes automáticos.

2.2.- Almacenes automáticos

Los almacenes automáticos permiten gestionar y optimizar los procesos derivados del almacenaje, preparación y expedición de mercancías. De este modo se controla en todo momento la entra de productos al almacén y se gestiona de forma óptima su correcta colocación y su posterior expedición.

Se trata de un sistema de almacenamiento que permite aumentar al máximo las ratios de productividad, reducir el espacio de almacén necesario y reducir los movimientos necesarios para trasladar las mercancías hasta su lugar de recepción o expedición.

En la Figura 2.1 se observa un ejemplo de almacén automático de Mecalux donde se aprecian dos almacenes (Izquierda) para palets y (Derecha) para cajas. Los palets y cajas son guiados de forma automática desde las estanterías a las zonas de expedición. Del mismo modo también se colocan los palets y cajas desde las salidas de fábrica a su correspondiente ocupación en las estanterías.

Figura 2.1.- Almacén automático Mecalux

Para realizar todas estas operaciones un almacén automático está formado por dos sistemas autónomos que se comunican entre ellos, el Software de gestión de almacenes (SGA) y el Software de control (Galileo).

2.2.1.- Software de Gestión de Almacenes

El software de gestión de almacenes es un sistema de información y gestión que tiene como finalidad optimizar la gestión física y documental del flujo de mercancías, desde su entrada en el almacén hasta su salida final. Todo este proceso se fundamenta en una planificación continua que proporciona el seguimiento global de las actividades y el control de existencias en tiempo real, multiplicando la rentabilidad de todas las áreas.

Mecalux comercializa distintos SGA según las necesidades y no es necesario que trabaje con el software de control propio de la empresa (Galileo), sino que se puede adaptar a otros sistemas de control.

El SGA optimiza todos los movimientos, procesos y operativas dentro del almacén, y permite controlar sus operaciones fundamentales:

- Recepción de mercancía
- Almacenaje e inventario
- Preparación de pedidos
- Expediciones

Cuando un producto terminado y empaquetado, ya sea en cajas o en palets entra en el almacén, el SGA se encarga de destinarle un espacio en las estanterías. Una vez calculado el lugar de destino se comunica con el Software de control y le da las ordenes.

Del mismo modo cuando se genera una orden de salida de algún producto el SGA determina que producto tiene que salir del almacén y donde tiene que ser llevado y se lo comunica al software de control que se encarga de llevarlo hasta su lugar de recogida.

2.2.2.- Software de control - Galileo

Galileo es el software de control que se encarga de gestionar todas las maquinas presentes en la instalación. Para ello controla sus componentes hardware, registra las averías o eventos que se puedan producir y se comunica con el software de gestión de almacenes (SGA). Recibe las órdenes del SGA y se encarga de controlar las distintas máquinas para mover los productos según esas órdenes. Galileo se ejecuta en ordenadores industriales situados en el propio almacén.

Para realizar el programa de control que ejecutara Galileo, en Mecalux disponen de una herramienta llamara Designer. Mediante esta aplicación, se programan las instalaciones al completo. Ha sido desarrollada para poder programar las instalaciones por bloques ya que un almacén está formado por una gran cantidad de máquinas con una funcionalidad determinada. Por ello cada máquina tiene su propio código que nos es necesario modificar para cada instalación. De esta forma se realizan programas con una arquitectura orientada a objetos.

3.- OBJETO

El objetivo del proyecto es realizar un sistema que notifique las averías activas de los almacenes automáticos en los dispositivos móviles de los operarios. Estos podrán ver en tiempo real que averías y en que máquinas están bloqueando la planta y así poder actuar con mayor rapidez.

El proyecto está divido en dos partes, por un lado, la obtención y clasificación de las averías en las instalaciones, y por otro lado la notificación y visualización de las mismas en dispositivos móviles.

En el presente documento se expone únicamente la resolución de la obtención y clasificación de las averías.

Figura 3.1.- Visión general de proyecto

En la Figura 3.1 se muestra un diagrama del sistema de notificación de averías. En un principio el sistema de control de las máquinas detecta y notifica las averías al sistema de gestión de almacenes, y este las almacena en la base de datos. Un servidor web se encargará de gestionar las averías de la base de datos y notificarlas a los dispositivos móviles.

El sistema debe diseñarse para funcionar tanto con el software de control y gestión de almacenes de Mecalux como el de otras compañías.

3.1.- Obtención y clasificación de averías

Esta es la parte del proyecto que se expone en el presente documento. Su objetivo es obtener las averías que se generan en las distintas máquinas del almacén y clasificarlas según unos criterios de:

- Asignación a usuarios o grupos de usuarios.
- Reglas de escalado según rangos de personal.
- Restricciones temporales (tiempo de avería).
- Restricciones según origen y procedencia de las averías.

Una vez que las averías han sido clasificadas el mismo sistema se encargara de:

- Almacenarlas para mantener un histórico.
- Identificar a que usuario/grupo debe notificar.
- Identificar si han cumplido los límites de tiempo u otras reglas de escalado de manera que la avería se deba notificar a más personas.
- Generar las notificaciones hacia esas personas.

A su vez el sistema de clasificación permitirá a los dispositivos móviles obtener todas las averías activas en la instalación que correspondan al usuario de ese terminal. Una vez que el usuario visualiza las averías puede identificar de forma rápida a qué hora y en que máquina se originó dicha avería. Junto con esta información se añade una descripción de la avería y una sugerencia que permita al usuario intentar solventarla de forma rápida.

Para que no haya varios usuarios encargándose de las mismas averías el usuario podrá acusar una avería de forma que los demás usuarios asignados a esa avería podrán dedicarse a otras.

Uno de los criterios de clasificación es la regla de escalado. Esta permitirá notificar las averías a jefes o superiores si el tiempo de vida de la avería supera unos límites programados.

El sistema de clasificación está alojado en un servidor web desde el que los distintos dispositivos móviles podrán acceder para obtener los datos de las averías.

Por otro lado, es necesario crear un sistema que se encargue de obtener las averías activas en las máquinas de los almacenes y las actualice en la base de datos. Para ello se tendrá que crear una aplicación que se comunique con el sistema de control de la instalación y a su vez modifique la base de datos. Esta es una parte importante ya que la base de datos debe estar actualizada con las averías activas de la instalación de forma continua y sin retardos. De esta forma los usuarios de la aplicación podrán recibir las notificaciones de las averías en tiempo real y realizar las tareas de reparación rápidamente.

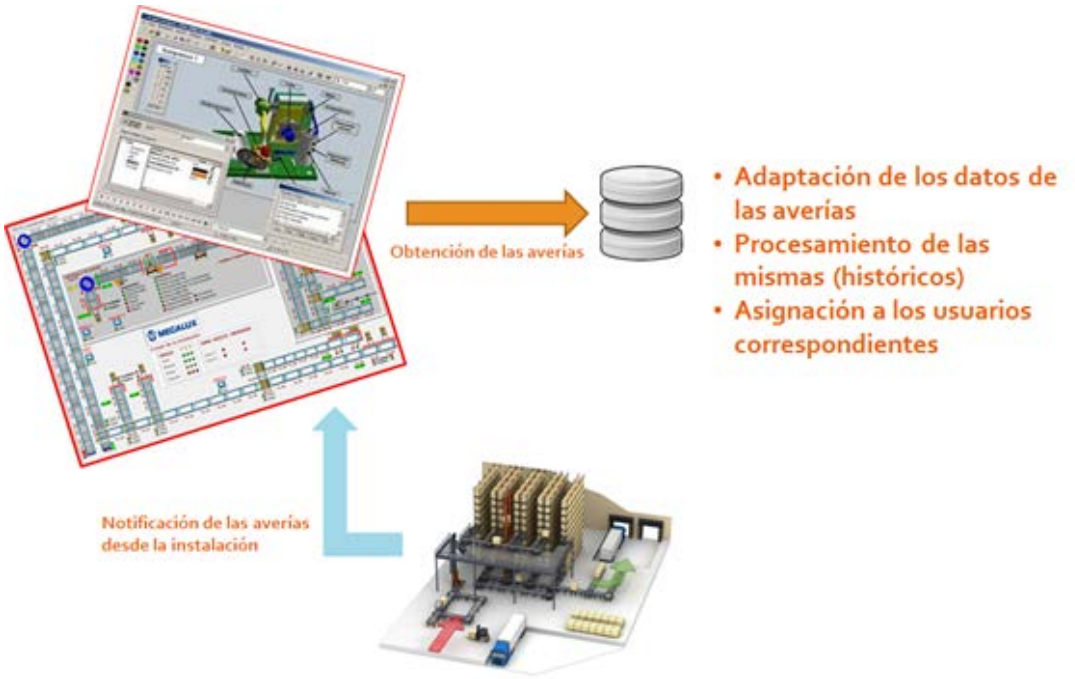

Figura 3.2.- Esquema clasificación de averías

3.2.- Notificación y visualización de averías

Lo que se expone en este apartado no corresponde a este proyecto. Se trata de una posible mejora o aplicación del proyecto.

El objetivo de esta ampliación es permitir la visualización de las averías en un dispositivo móvil, ya sea Android, Windows Phone o iOs. Para ello estos dispositivos móviles se deben conectar al sistema de clasificación de manera que puedan obtener los datos de las averías a mostrar.

Obviamente, solo deben obtener las notificaciones y datos de las averías los usuarios que se pertenezcan a dicha empresa y estén inscritos en dicha aplicación, aparte de cumplir las restricciones anteriormente comentadas.

Figura 3.3.- Esquema de visualización de averías

La conexión desde el dispositivo móvil con el sistema de clasificación puede estar en distintas redes. Para ello se utiliza un sistema de servicio web al que se puede acceder desde cualquier parte.

4.- ANÁLISIS DE ALTERNATIVAS

Antes empezar el desarrollo del proyecto se valoraron las distintas tecnologías para abordar cada apartado del proyecto.

Para realizar todo el código de las aplicaciones se valoraron distintos lenguajes de programación. Entre ellos destacamos C, C++ y C#.

En primer lugar, se disponían de diferentes lenguajes de programación como pueden ser C, C++ o C#. En el departamento se utilizaban C y C#. El lenguaje C no es un lenguaje orientado a objetos y para trabajar con las bases de datos esto es un gran impedimento por lo que se rechazó. Por otro lado, C# se trata de un lenguaje de programación orientada a objetos que permite a los desarrolladores complicar aplicaciones sólidas y seguras que se ejecutan en .NET Framework. Es un lenguaje sencillo y fácil de utilizar y que proporciona funciones de consulta (LINQ) para hacer búsquedas en bases de datos, por lo que desde el departamento se asignó este para trabajar.

Para realizar la base de datos en la empresa se utilizan principalmente Oracle y MySQL. Oracle es una de las herramientas más potentes de bases de datos y necesitan una aplicación muy grande para utilizarlas aparte de que las licencias de uso son demasiado caras. Por ello se optó por utilizar MySQL ya que se trata de un software libre y que también se impuso desde el departamento.

5.- DISEÑO Y DESARROLLO

Para la realización de las diferentes aplicaciones del proyecto se ha utilizado un lenguaje orientado a objetos C# y SQL para trabajas con bases de datos (MySQL).

El proyecto se ha divido en tres partes:

- Base de datos donde se almacenará todos los usuarios, averías y definiciones, reglas de asignación, compañías, etc.
- Servidor Web donde se procesarán todos los datos y tendrá los métodos de comunicación con los distintos dispositivos. Estos métodos podrán ser POST si es el usuario quien envía la información al servidor y GET si el usuario solicita información al servidor y este se la envía.
- Servicio de notificaciones que será el encargado de notificar a los usuarios que disponen de averías activas sin resolver.
- Servicio de Windows encargado de comunicarse con el sistema de control y crear y modificar las averías en la base de datos.

A continuación, se detallan cada uno de los aparatados anteriores.

5.1.- Base de Datos

La base de datos es el pilar principal de la aplicación donde están almacenados todos los datos de las empresas. A continuación, se detallan los grupos de datos que se almacenaran:

- Averías: Este grupo almacena las averías de cada instalación. A su vez consta de otras dos tablas con las que se podrán definir las averías en distintos idiomas.
- Usuarios: Aquí se almacenan los usuarios de cada instalación y por otro lado se podrán crear grupos de usuarios.
- Reglas: Para notificar las averías se deben cumplir una serie de reglas, estas reglas podrán ser de tiempo y de escalado. Las reglas de tiempo se cumplen cuando la avería esta activa a una hora en un determinado día. Las reglas de escalado se cumplen cuando las averías están activas durante un periodo de tiempo.
- Permisos: Los permisos se asignan a grupos de usuarios y permitirán dar o no acceso a los usuarios a determinadas operaciones.
- Móvil: Almacena los móviles de los usuarios registrados.
- Peso de las maquinas: Se registra el peso de cada máquina en la instalación.

En la Figura 5.1 está representado el modelo de datos con todas las relaciones entre las diferentes tablas.

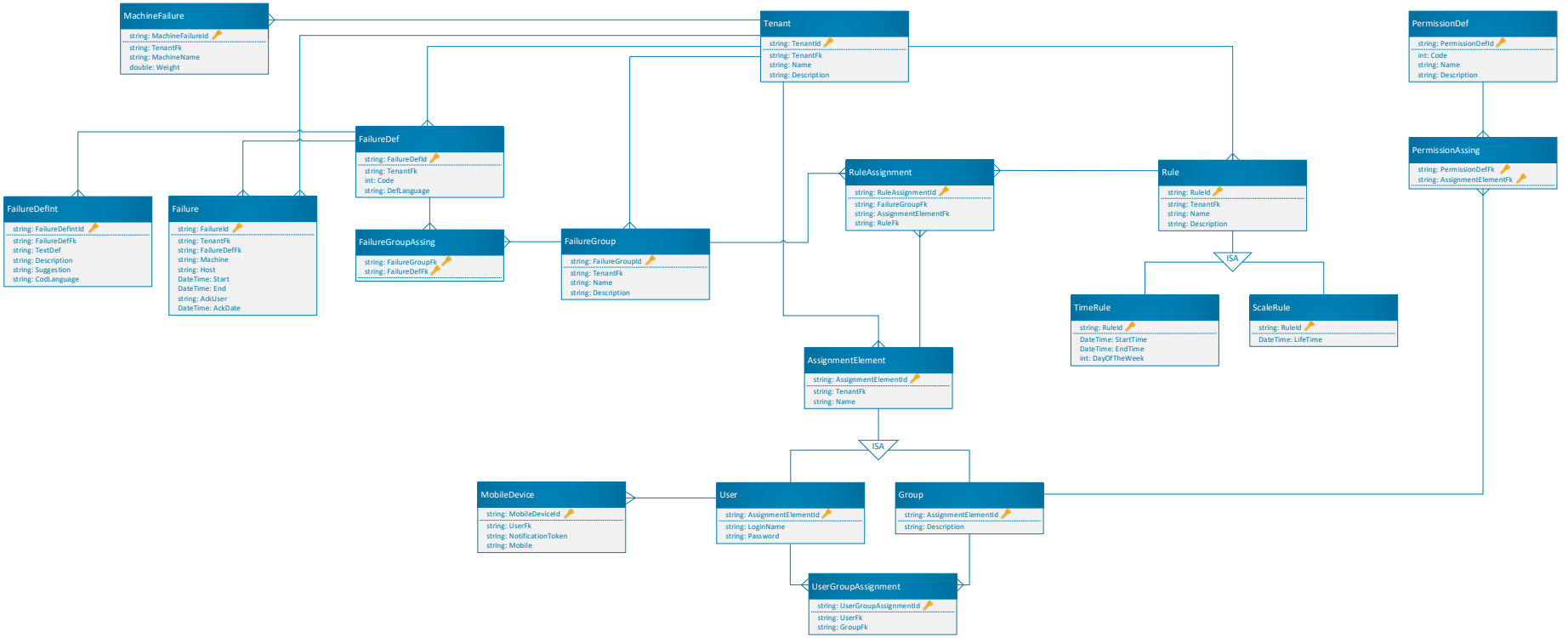

Figura 5.1.- Esquema de la base de datos.

Sobre la base de datos se realizan las búsquedas necesarias para enviar y notificar las averías a los usuarios. Como se puede apreciar hay una tabla central que contiene las compañías (Tenant) y de la que colgara el resto de tablas. Estas compañías podrán tener como padre otra compañía con el fin de poder representar y organizar los grupos de empresas.

Cada una de las compañías tiene su conjunto de usuarios y grupos, averías, reglas, permisos, máquinas y grupos de averías que a continuación se detallan.

5.1.1.- Compañías – Tenant

Esta tabla almacena todas las compañías que están asociadas al sistema de notificación de averías. Sus campos son:

- TenantId [string]: Identificador único para cada compañía que actúa como clave principal de la tabla.
- TenantFk [string]: Identificador de la compañía (Tenant) cliente, que actúa como clave ajena hacia la tabla Tenant.
- Name [string]: Nombre completo de la compañía.
- Description [string]: Breve descripción de la compañía.

Figura 5.2.- Tenant

5.1.2.- AssignmentElement

Esta tabla presenta una relación ISA, donde se genera una tabla de orden superior con los atributos comunes y otras tablas para cada una de las entidades subordinadas, con la misma clave que la entidad superior y que implementen los atributos comunes y otros propios de cada sub-entidad. De este modo se crea una herencia hacia las tablas User y Group. De forma que cuando se creen usuarios o grupos de usuarios se creara un elemento en esta tabla para cada uno de ellos. De esta forma podremos asignar reglas de averías a usuarios o grupos de usuarios. Sus campos son:

- AssignmentElementId [string]: Identificador único para cada usuario o grupo de usuarios que actúa como clave principal de la tabla.

- TenantFk [string]: Identificador de la compañía (Tenant) cliente que actúa como clave ajena hacia la tabla Tenant.
- Name [string]: Nombre del usuario o grupo de usuarios.

Figura 5.3.- AssignmentElement

5.1.3.- User

En esta tabla están guardados todos los usuarios de una compañía que tengan acceso al servicio de notificaciones. Sus campos son:

- AssignmentElementId [string]: Identificador único para cada usuario (User) que actúa como clave principal de la tabla. Este identificador relaciona a cada usuario con un Assignement Element por lo que hereda también los campos de este (TenantFk y Name).
- LoginName [string]: Identificador de cada usuario para acceder al servicio de notificaciones. Este campo no se podrá repetir dentro de una misma compañía.
- Password [string]: Contraseña de cada usuario para acceder al servicio de notificaciones.

Figura 5.4.- User

5.1.4.- Group

Esta tabla contiene los grupos de usuarios. Sus campos son:

- AssignmentElementId [string]: Identificador único para cada grupo de usuarios (Group) que actúa como clave principal de la tabla. Este identificador relaciona a cada usuario con un Assignement Element por lo que hereda también los campos de este (TenantFk y Name).
- Description [string]: Breve descripción del grupo de usuarios.

Figura 5.5.- Group

5.1.5.- UserGroupAssignment

Esta tabla tiene como objetivo romper la relación N-N que se genera al relacionar usuarios con grupos de usuarios. Dado que un usuario puede pertenecer a varios grupos y un grupo puede tener distintos usuarios se genera una relación N-N. Estas relaciones no son recomendables y debemos tratar de evitarlas mediante la utilización de tablas intermedias como esta que utilicen relaciones de uno a muchos. Sus campos son:

- UserGroupAssignmentId [string]: Identificador único de para cada entrada de la tabla que actúa como clave principal.
- UserFk [string]: Identificador del usuario cliente que actúa como calve ajena hacia la tabla User.
- GroupFk [string]: Identificador del grupo de usuarios que actúa como calve ajena hacia la tabla Group.

De esta forma cada usuario tendrá un campo en esta tabla que lo relacionará con un grupo en concreto, y de la misma forma cada grupo tendrá un campo que lo relacionará con un usuario.

Figura 5.6.- UserGroupAssignment

5.1.6.- MobileDevice

En esta tabla se irán almacenado los toquen de notificaciones para cada usuario que acceda a la aplicación de notificaciones y tenga los permisos necesarios. Para cada usuario que acceda a la aplicación desde un dispositivo distinto se le asignara un toquen que permitirá a la aplicación enviarle las notificaciones. Cuando este usuario cierre sesión en la aplicación se tendrá que borrar el toquen para no notificarle averías. Sus campos son:

- MobileDeviceId [string]: Identificador único de para cada entrada de la tabla que actúa como clave principal.

- UserFk [string]: Identificador del usuario cliente que actúa como calve ajena hacia la tabla User.
- NotificationToken [string]: Representa un identificador único que permitirá a la aplicación enviar notificaciones a un dispositivo móvil.
- MobileId [string]: Representa un identificador único de cada dispositivo móvil.

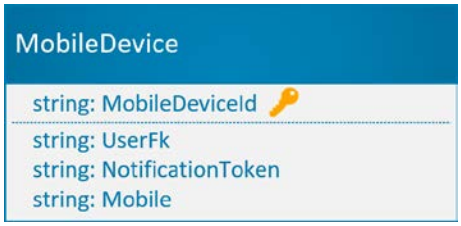

Figura 5.7.- MobileDevice

5.1.7.- Rule

Esta tabla presenta una relación ISA de herencia hacia las tablas TimeRule y ScaleRule. De forma que cuando se creen reglas de tiempo (TimeRule) o reglas de escalado (ScaleRule) se creará un elemento en esta tabla para cada uno de ellos. De esta forma podremos asignar reglas de averías a reglas de tiempo o escalado. Sus campos son:

- RuleId [string]: Identificador único para cada regla de tiempo (TimeRule) o regla de escalado (ScaleRule) que actúa como clave principal de la tabla.
- TenantFk [string]: Identificador de la compañía (Tenant) cliente que actúa como clave ajena hacia la tabla Tenant.
- Name [string]: Nombre de la regla.
- Description [string]: Breve descripción de la regla.

Figura 5.8.- Rule

5.1.8.- TimeRule

En esta tabla están guardadas todas las reglas de tiempo para cada compañía. Esta regla permite decidir que averías podrán ser notificadas según la hora y día de la semana a la que se asignen los grupos de averías. Sus campos son:

- RuleId [string]: Identificador único para cada regla de tiempo (TimeRule) que actúa como clave principal de la tabla. Este identificador relaciona a cada regla de tiempo con una regla por lo que hereda también los campos de este (TenantFk, Name y Description).
- StartTime [DateTime]: Representa la hora del día a la que empieza en intervalo de notificaciones.
- EndTime [DateTime]: Representa la hora del día en la que termina el intervalo de notificaciones.
- DayOfTheWeek [Int]: Representa los días de la semana en las que se podrá notificar averías.

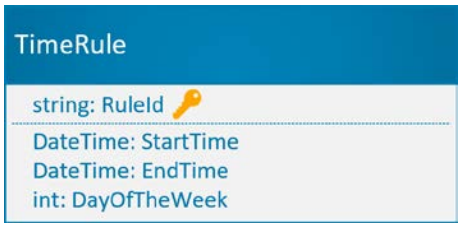

Figura 5.9.- TimeRule

5.1.9.- ScaleRule

En esta tabla están guardadas todas las reglas de escalado para cada compañía. Esta regla permitirá decidir las averías que se podrán notificar si se supera un tiempo máximo de vida en cada avería.

- RuleId [string]: Identificador único para cada regla de tiempo (TimeRule) que actúa como clave principal de la tabla. Este identificador relaciona a cada regla de tiempo con una regla por lo que hereda también los campos de este (TenantFk, Name y Description).
- LifeTime [DateTime]: representa el tiempo de vida que puede permanecer un grupo de averías antes de ser notificado.

Figura 5.10.- ScaleRule

5.1.10.- PermissionDef

En esta tabla están definidos todos los permisos que se podrán asignar a los grupos de usuarios. Esto permitirá restringir determinadas funciones a los usuarios. Es la única tabla que no depende de la compañía. Sus campos son:

- PermisionDefId [string]: Identificador único para cada permiso que actúa como clave principal de la tabla.
- Code [Int]: Representa el código numérico de cada permiso.
- Name [string]: Nombre de cada permiso.
- Description [string]: Breve descripción de cada permiso.

Figura 5.11.- PermissionDef

5.1.11.- PermissionAssing

Esta tabla tiene como objetivo romper la relación N-N que se genera al relacionar permisos con grupos de usuarios. Dado que un permiso puede estar asignado a varios grupos y un grupo puede tener distintos permisos se genera una relación N-N. Estas relaciones no son recomendables y debemos tratar de evitarlas mediante la utilización de tablas intermedias como esta que utilicen relaciones de uno a muchos. Sus campos son:

- PermissionDefFk [string]: Identificador del permiso cliente que actúa como calve ajena hacia la tabla PermissionDef.
- AssignmentElement [string]: Identificador del grupo de usuarios cliente que actúa como calve ajena.

La combinación de ambos campos actúa como clave primaria de la tabla.

Figura 5.12.- PermissionAssing

5.1.12.- FailureDef

Esta tabla tendrá todos los tipos de averías que pueden ocurrir en una compañía. Sus campos son:

- FailureDefId [string]: Identificador único de cada avería que actúa como calve principal de la tabla.
- TenantFk [string]: Identificador de la compañía (Tenant) cliente que actúa como clave ajena hacia la tabla Tenant.
- Code [Int]: Representa el código numérico de cada avería.

Figura 5.13.- FailureDef

5.1.13.- FailureGroup

En esta tabla se guardan los grupos de averías a los que se asignarán las reglas de notificación. Cada grupo tendrá asignado un número de averías. Sus campos son:

- FailureGroupId [string]: Identificador único de cada grupo de averías que actúa como calve principal de la tabla.
- TenantFk [string]: Identificador de la compañía (Tenant) cliente que actúa como clave ajena hacia la tabla Tenant.
- Name [string]: Nombre del grupo de averías.
- Description [string]: Breve descripción del grupo de averías.

Figura 5.14.- FailureGroup

5.1.14.- FailureGroupAssing

Esta tabla tiene como objetivo romper la relación N-N que se genera al relacionar averías con grupos de averías. Dado que una avería puede estar asignada a varios grupos y

un grupo puede tener distintas averías se genera una relación N-N. Estas relaciones no son recomendables y debemos tratar de evitarlas mediante la utilización de tablas intermedias como esta que utilicen relaciones de uno a muchos. Sus campos son:

- PermisionDefFk [string]: Identificador de la avería cliente que actúa como clave ajena hacia la tabla PermissionDef.
- FailureGoupFk [string]: Identificador del grupo de averías cliente que actúa como calve ajena hacia la tabla FailureGroup.

La combinación de ambos campos actúa como calve primaria de la tabla.

Figura 5.15.- FailureGroupAssing

5.1.15.- FailureDefInt

Esta tabla almacenará la descripción detallada de cada avería en distintos idiomas. Sus campos son:

- FailureDefIntId [string]: Identificador único de cada definición de avería que actúa como calve principal de la tabla.
- FailureDefFk [string]: Identificador de la avería cliente que actúa como clave ajena hacia la tabla FailureDef
- TextDef [string]: Breve descripción de la avería.
- Description [string]: Descripción detallada de la avería.
- Suggestion [string]: Sugerencia de actuación para resolver la avería.
- CodLaguage [string]: Código que identifica el idioma de la descripción de la avería.

Figura 5.16.- FailureDefInt

5.1.16.- Failure

Esta será la tabla principal de la base de datos, ya que será donde se almacenarán las averías que se vayan generando en la instalación. Sus campos son:

- FailureId [string]: Identificador único de cada avería, que actúa como clave primaria de la tabla.
- FailureDefFk [string]: Identificador de la avería cliente que actúa como clave ajena hacia la tabla FailureDef.
- TenantFK [string]: Identificador de la avería cliente que actúa como clave ajena hacia la tabla Tenant.
- Machine [string]: Representa la máquina en la que se ha generado la avería.
- Host [string]: Representa el host al que pertenece la máquina.
- Start [DateTime]: Representa la hora y fecha en la que se originó la avería.
- End [DateTime]: Representa la hora y fecha en la que se soluciona la avería.
- AckUser [string]: Representa el usuario que se hizo cargo de reparar la avería.
- AckDate [DateTime]: Representa la hora y fecha en la que el usuario se encargó de reparar la avería.

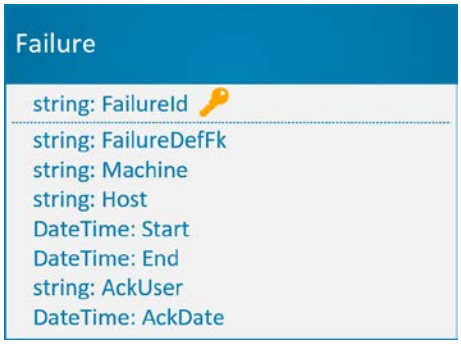

Figura 5.17.- Failure

5.1.17.- MachineFailure

Esta tabla almacenará el peso que tiene una maquina en la aplicación. Es decir, el porcentaje de afectación que tiene una máquina sobre la instalación por estar en avería. Cada vez que se genere una avería en la instalación se comprueba si existe en la tabla y se crea con peso uno. Un usuario con permisos podrá ajustar los pesos cuando crea conveniente. Sus campos son:

- MachineFailureId [string]: Identificador único de cada máquina, que actúa como clave primaria de la tabla.
- TenantFk [string]: Identificador de la compañía (Tenant) cliente que actúa como clave ajena hacia la tabla Tenant.
- MachineName [string]: Nombre de la máquina.

- Weight [double]: Valor numérico entre 0 -1 que representa el peso de la maquina en la compañía.

Figura 5.18.- MachineFailure

5.1.18.- RuleAssignment

En esta tabla se guardan las relaciones entre los grupos de averías, los usuarios o grupos de usuarios y las reglas. De este modo se podrá identificar de forma rápida las averías que corresponden a cada usuario según las reglas. Sus campos son:

- RuleAssignmentId [string]: Identificador único de cada relación que actúa como clave primaria de la tabla.
- FailureGroupFk [string]: Identificador del grupo de averías (FailureGroup) cliente que actúa como clave ajena hacia la tabla FailureGroup.
- AssignmentElementFk [string]: Identificador al usuario o grupo de usuarios (AssignmentElement) cliente que actúa como clave ajena hacia la tabla AssignmentElement.
- RuleFk [string]: Identificador de la regla (Rule) cliente que actúa como clave ajena hacia la tabla Rule.

Figura 5.19.- RuleAssignment

5.2.- Acceso a datos

Desde el servidor web se accede a la base de datos para realizar búsquedas, modificar y obtener datos. Para facilitar todas estas acciones se utilizará un framework de Ado.Net llamado EntityFramework.

5.2.1.- Entity Framework

Entity Framework es un conjunto de tecnologías de ADO.NET que permiten el desarrollo de aplicaciones de software orientadas a datos, en nuestro caso la base de datos vista en el apartado anterior. Esto nos permite trabajar con la base de datos en forma de objetos y propiedades especificas del dominio, como clientes y direcciones de cliente, sin tener que preocuparnos por las tablas y columnas de la base de datos. Se basa en modelos ER (Entidad - Relación) donde el modelo esta mapeado con la base de datos permitiendo que las consultas y modificaciones hechas en el este se reflejen en la base de datos.

La principal ventaja es que podremos trabajar sobre el modelo de objetos, no sobre las tablas o vistas de la base de datos utilizando el mismo lenguaje para las diferentes bases de datos.

Para poder trabajar correctamente se necesita crear los modelos con herencia si fuera necesario y especificar las relaciones entre las tablas con las claves ajenas. Por otro lado, será necesario establecer cuál es el proveedor ADO.NET, en nuestro caso MySQL.

En la Figura 5.20 se muestra la arquitectura de Entity Framework para el acceso a datos.

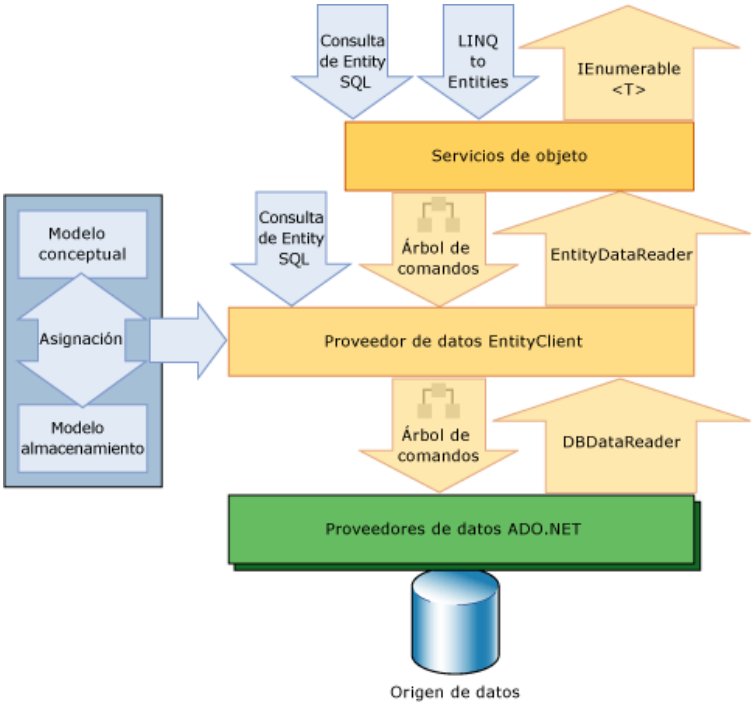

Figura 5.20.- Arquitectura Entity Framework

5.3.- Servidor Web

Un servidor web es un programa que ejecuta una aplicación del lado del servidor, realizando conexiones bidireccionales síncronas o asíncronas con el cliente. Es un tipo de software que suministra servicios a los usuarios o terminales que los solicitan. En nuestro caso será la aplicación la que realice las peticiones, tanto para solicitar información de la base de datos como para acusar las averías. El servidor se encargará de procesar la información que corresponda a cada usuario desde la base de datos y entregarla. En la Figura 5.21 se muestra un esquema general del funcionamiento.

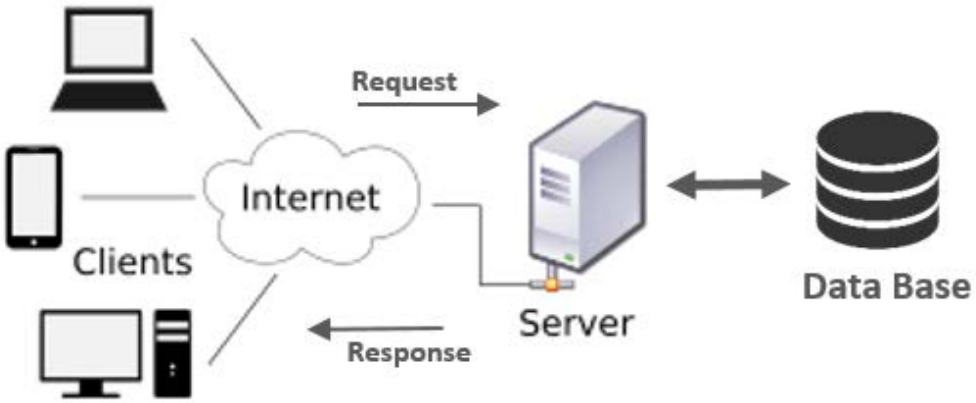

Figura 5.21.- Esquema servidor web - Base de datos

La comunicación entre los clientes y el servidor se establece por medio de un protocolo, concretamente el protocolo HTTP de la capa de aplicación del modelo OSI. El servidor se mantiene a la espera de peticiones HTTP. Estas peticiones serán básicamente de dos tipos:

- **GET**: Se trata de obtener información del servidor. Enviar datos desde el servidor a la aplicación, que ese nuestro caso ha obtenido de la base de datos. Para solicitar la información en cada caso es necesario enviar al servidor (request) algunos datos que este pueda interpretar para filtrar la información.
- **POST**: Mediante estos métodos se envía información desde los clientes al servidor para que este la procese y almacene en la base de datos. Cuando enviamos los datos, el servidor los procesa y almacena en la base de datos y luego a través de una redirección este devuelve (response) cierta información del procesamiento.

Como vemos ambos métodos solicitan una respuesta al servidor, en el caso de los métodos GET ira la información que el cliente ha solicitado junto con un mensaje de error que detallara el estado de la solicitud. Los métodos POST únicamente devolverán el mensaje de error.

Todos los métodos que implementa el servidor devuelven un objeto de la clase "ErrorInfo" que se muestra en la Figura 5.22. Mediante el ErrorCode se detalla el estado de la solicitud. Si por alguna razón el programa del servidor no puede realizar la tarea se indica eUnknownError y se detalla el error mediante una excepción.

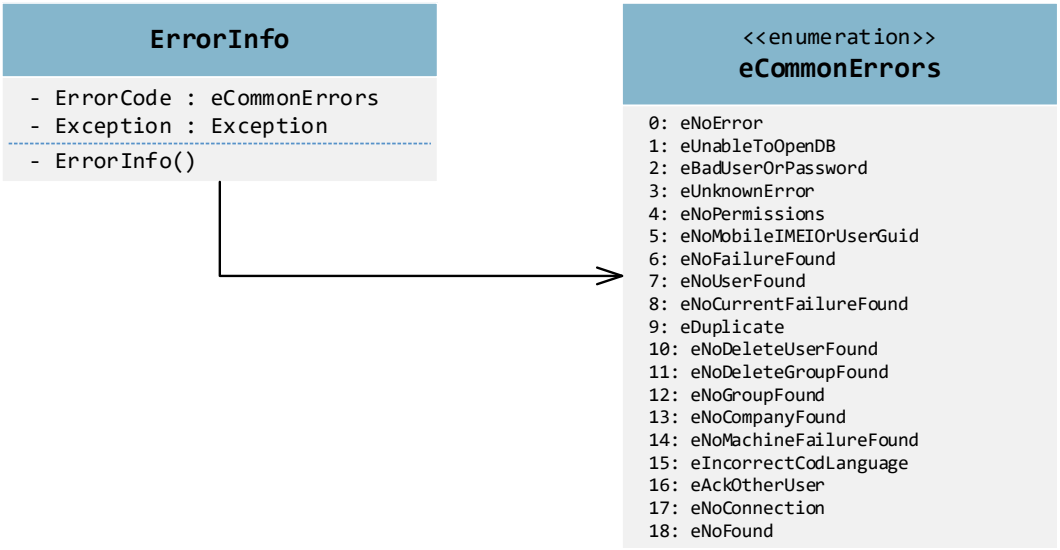

Figura 5.22.- Clase ErrorInfo

Por otro lado, el servidor también es el encargado de comprobar si se han creado nuevas averías activas en la base de datos y notificárselas a los usuarios. Esto se desarrolla a parte en el punto 5.4 del presente documento.

El servidor cuenta con muchos métodos según sean para trabajar desde la aplicación o desde designer. Muchos de ellos son para obtener datos de la base de datos y otros para introducir datos en ella.

El servidor cuenta con muchos métodos diferentes para realizar distintas funciones, estos los agruparemos en tres grupos: Aplicación, Designer y Cliente.

5.3.1.- Aplicación

La aplicación móvil utiliza siete métodos de comunicación con la base de datos, todos ellos a través del servidor. En la figura 5.23 se muestra un esquema con los casos de uso de funcionamiento. Tenemos dos actores, por un lado, la aplicación móvil de cada usuario y por otro lado la base de datos con toda la información.

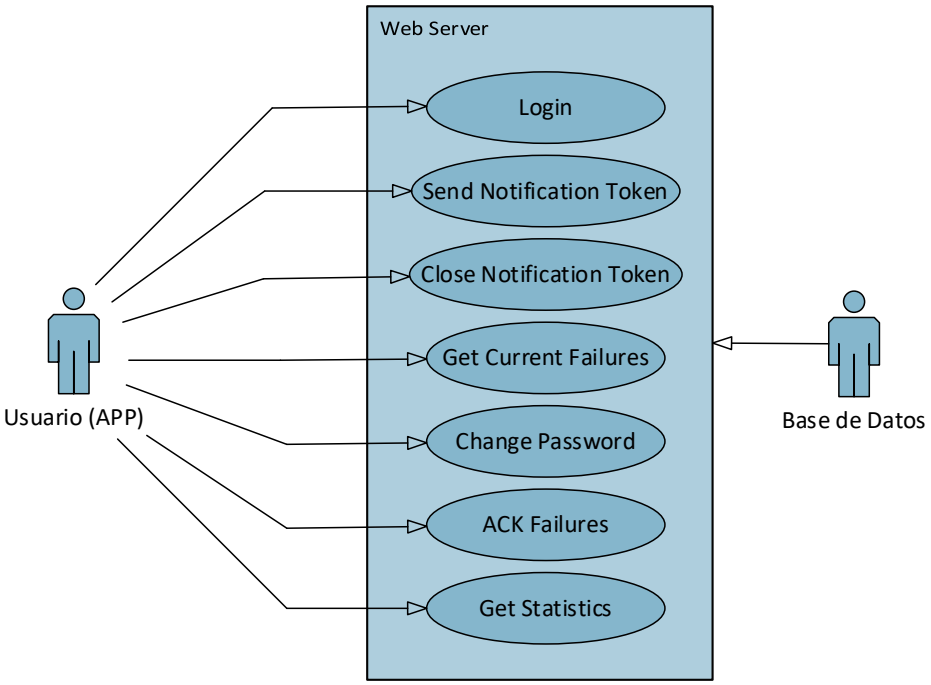

Figura 5.23.- Casos de uso Aplicación

A continuación, se detalla cada uno de los escenarios de caso de uso anteriores.

5.3.1.1.- Login

Se trata de un método con el que el usuario podrá acceder a la aplicación para ver las averías activas que le correspondan. El caso de uso comienza cuando el usuario accede a la aplicación, introduce sus datos de registro y pulsa el botón de conectar. Una vez que el usuario introduce sus datos se hace la petición de login al servidor, este busca en la base de datos si el usuario es correcto, obtiene todos los datos necesarios del usuario y genera la respuesta. En la Figura 5.24 se muestra el diagrama de secuencia.

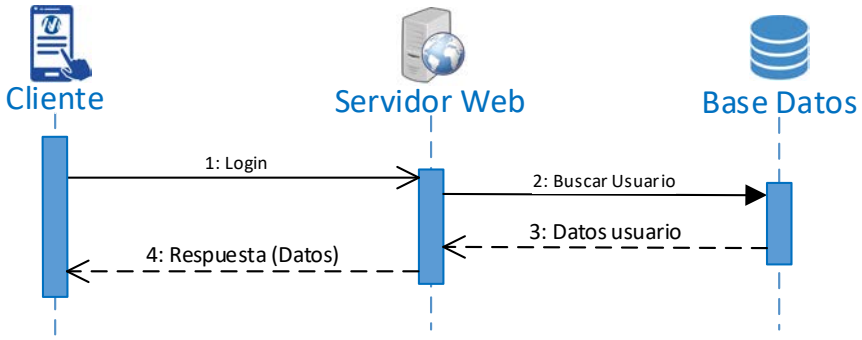

Figura 5.24.- Diagrama de secuencia Login

Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET que incluya la siguiente información:

DTULoginResult **Login**

- userName [string]: Nombre de usuario.
- password [string]: Clave de acceso personal del usuario.
- company [string]: Empresa en la que está registrado el usuario.

El servidor devuelve un objeto de la clase DTULoginResult, en la Figura 5.25 se muestra un diagrama de la clase y sus asociaciones. El objeto está compuesto por un ErrorInfo que detalla el estado de la petición, el identificador de usuario, que será necesario en las sucesivas peticiones, un enumerable de DTUTenantPermission que especifica los permisos que el usuario tiene en cada una de las compañías, un enumerable de los idiomas en los que dispone las averías y las compañías a las que puede acceder.

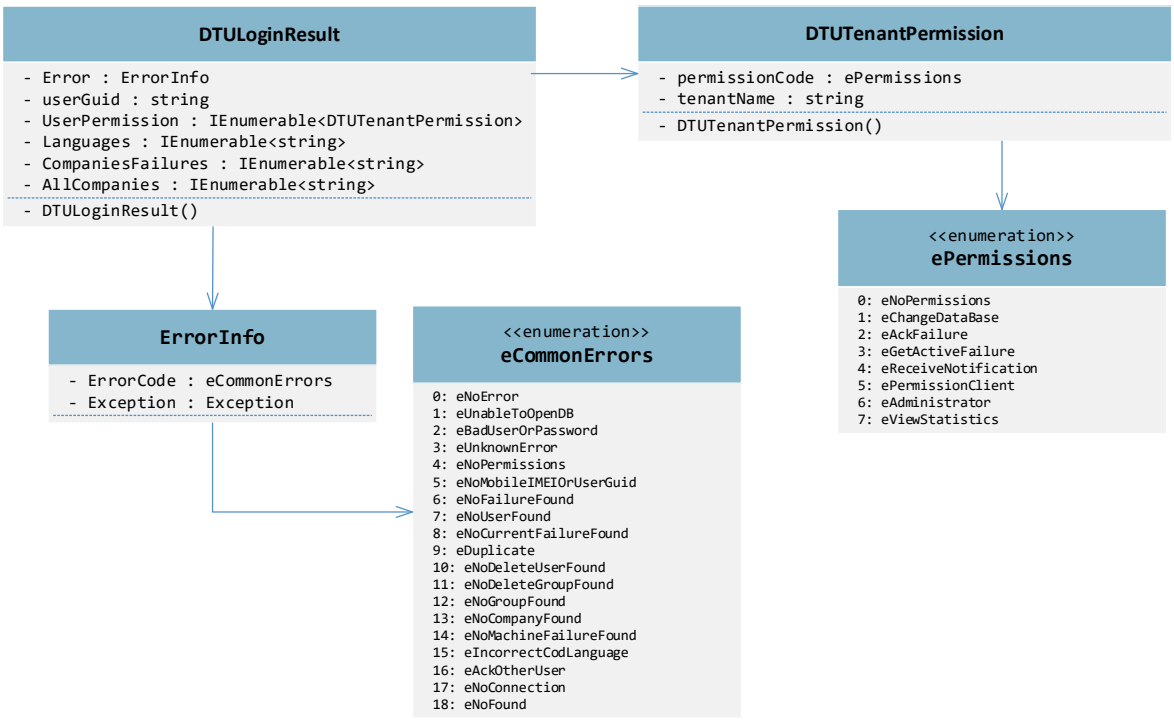

Figura 5.25.- Diagrama de clases DTULoginResult

En la parte del servidor se llevan a cabo una serie de acciones hasta enviar los datos. En la Figura 5.26 se muestra un diagrama de actividad con los pasos resumidos.

1. Comprobar que la compañía existe en la base de datos.

   No. Retornar error de compañía no encontrada (eNoCompanyFound).

   Si. Continuar.

2. Comprobar que el usuario es correcto

No. Retornar error de usuario o contraseña incorrecta (eBadUserOrPassword).

Si. Continuar

3. Obtener los datos del usuario.
4. Retornar datos.

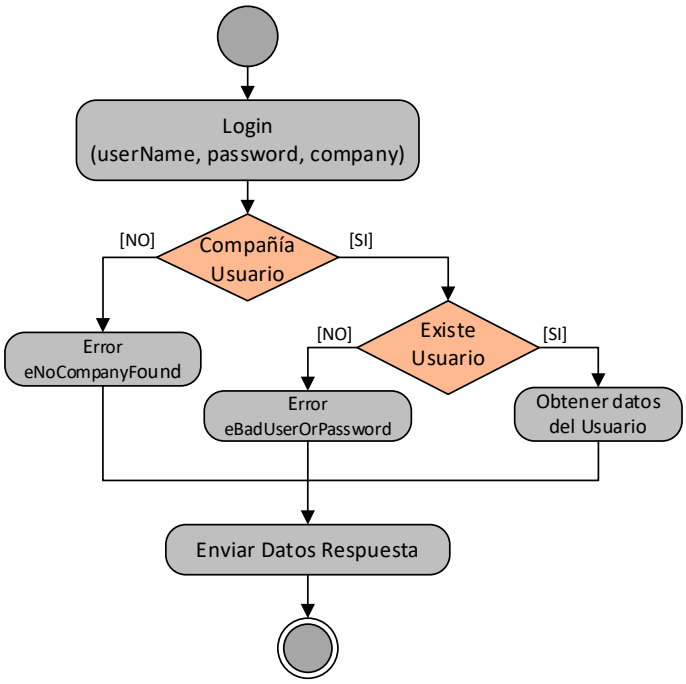

Figura 5.26.- Diagrama de actividad Login

5.3.1.2.- Send Notification Token

Mediante este método GET la aplicación envía el toquen de notificaciones que ha obtenido. Una vez que la aplicación recibe la respuesta del servidor de que el usuario se ha registrado correctamente, solicita un toquen de notificaciones al servicio de notificaciones y lo envía junto con un identificador único para cada dispositivo. De esta forma cuando el servidor detecte nuevas averías activas para el usuario podrá notificárselas. Cada usuario podrá estar registrado en varias aplicaciones, por ello se utiliza el identificador único del dispositivo.

El diagrama de secuencia se muestra en la Figura 5.27. En primer lugar, la aplicación solicita un toquen al servicio de notificaciones, una vez que esta lo obtiene se lo envía al servidor que lo almacena en la base de datos.

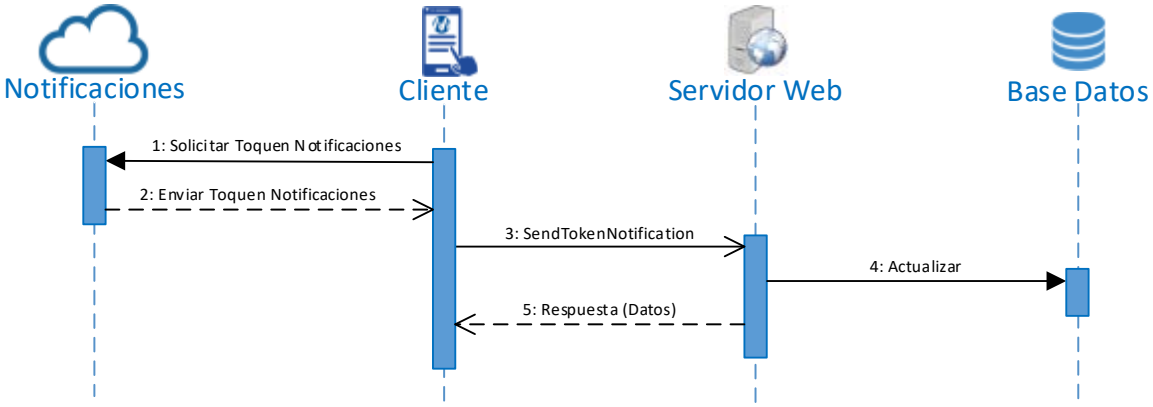

Figura 5.27.- Diagrama de secuencia de SendTokenNotification

Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET que incluya la siguiente información:

ErrorInfo **SendTokenNotification**

- userIdentificator [string]: Identificador del usuario
- deviceId [string]: Identificador del móvil que registra el toquen de notificaciones.
- tokenNotification [string]: Toquen de registro del móvil en el servicio de notificaciones. Mediante este número se podrá enviar notificaciones personales al usuario.

El servidor devuelve un objeto de la clase ErrorInfo visto en la Figura 5.28 que detalla el estado de la solicitud.

Los pasos seguidos en el servidor para guardar el toquen son los siguientes:

1. Comprobar que el usuario existe
   No. Retornar error de usuario no encontrado.
   Si. Continuar
2. Comprobar que existen la compañía.
   No. Retornar error de compañía no encontrada.
   Si. Continuar.
3. Comprobar que el usuario tiene permisos.
   No. Retornar error de usuario sin permisos.
   Si. Continuar
4. Comprobar si existe el identificador de usuario.
   No. Crear nueva entrada en MobileDevice para el usuario con los datos.
   Si. Actualizar el registro de MobileDevice del usuario.
5. Retornar datos.

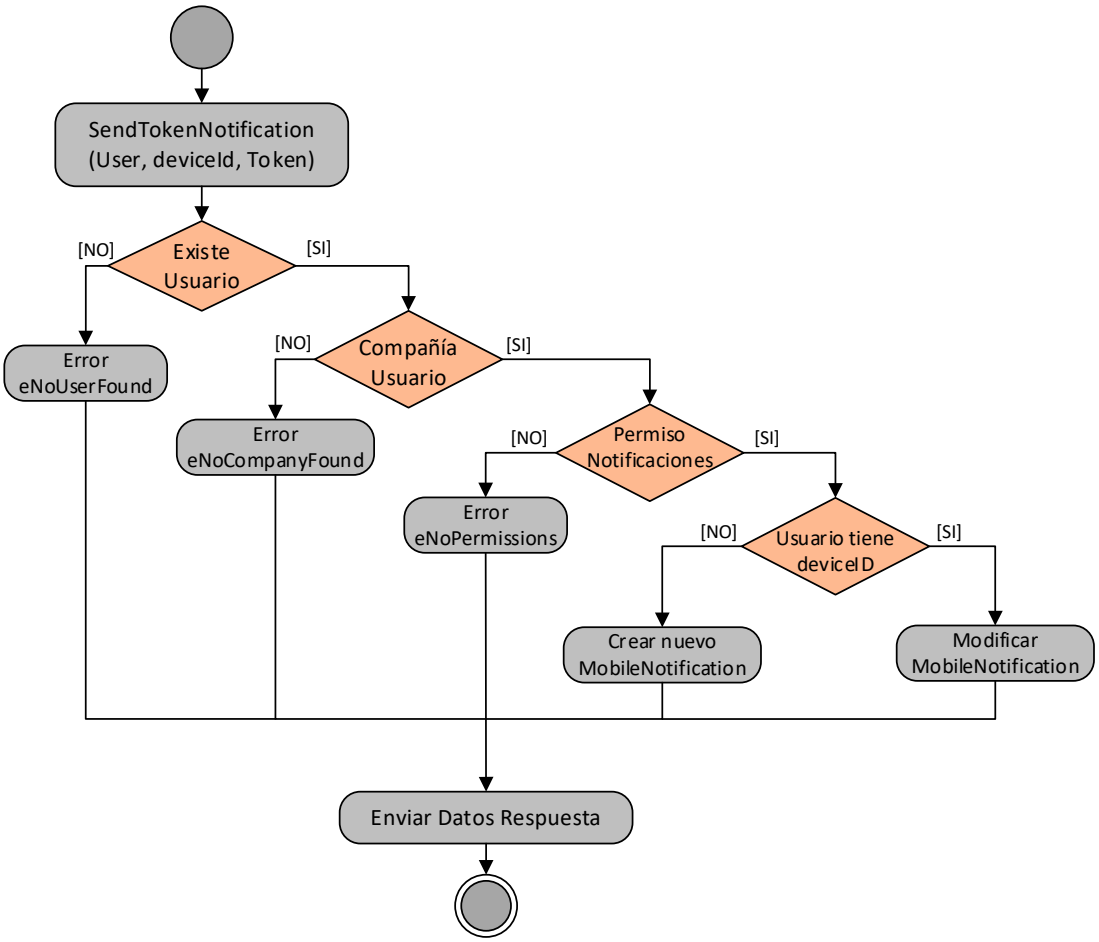

Figura 5.28.- Diagrama de actividad del SendTokeNotification

5.3.1.3.- Close Token Notification

Cuando el usuario cierra sesión en la aplicación, se debe borrar el toquen de notificaciones de la base de datos para no seguir recibiendo alertas de averías activas. Mediante este método el dispositivo móvil indica al servidor que debe borrarlo siguiendo la siguiente secuencia.

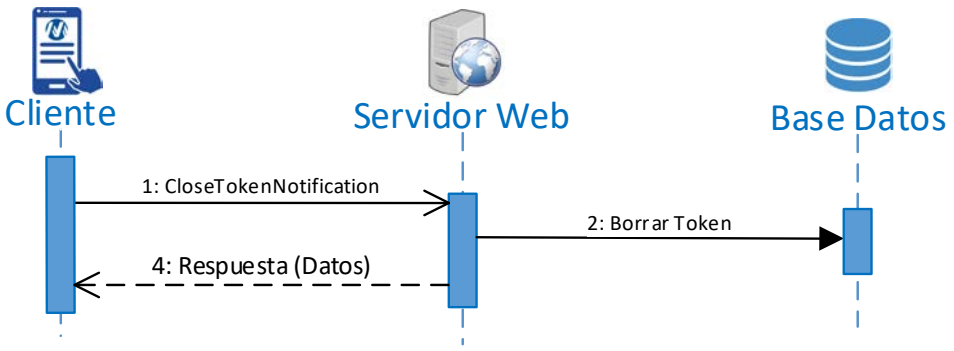

Figura 5.29.- Diagrama secuencia de CloseTokenNotification

Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET que incluya la siguiente información:

ErrorInfo **CloseTokenNotification**

- userIdentificator [string]: Identificador del usuario
- deviceId [string]: Identificador del móvil que registra el toquen de notificaciones.

En el servidor se seguirá el siguiente diagrama de actividad:

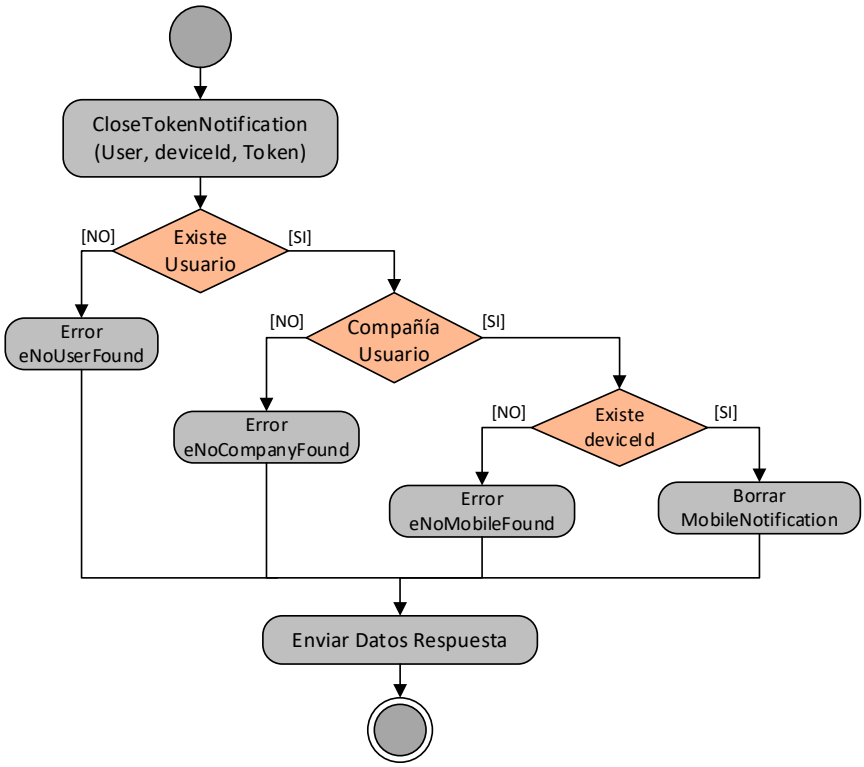

Figura 5.30.- Diagrama de actividad de CloseTokenNotification

5.3.1.4.- Get Current Failures

Este método será el encargado de buscar todas las averías activas que correspondan al usuario. En primer lugar, obtendrá todas las averías activas para las compañías solicitadas y a continuación se filtran para las reglas del usuario. Las reglas a tener en cuenta serán las propias del usuario y las de los grupos a los que pertenecen. Una vez obtenidas todas las averías del usuario se agrupan junto con un ErrorInfo y se envían.

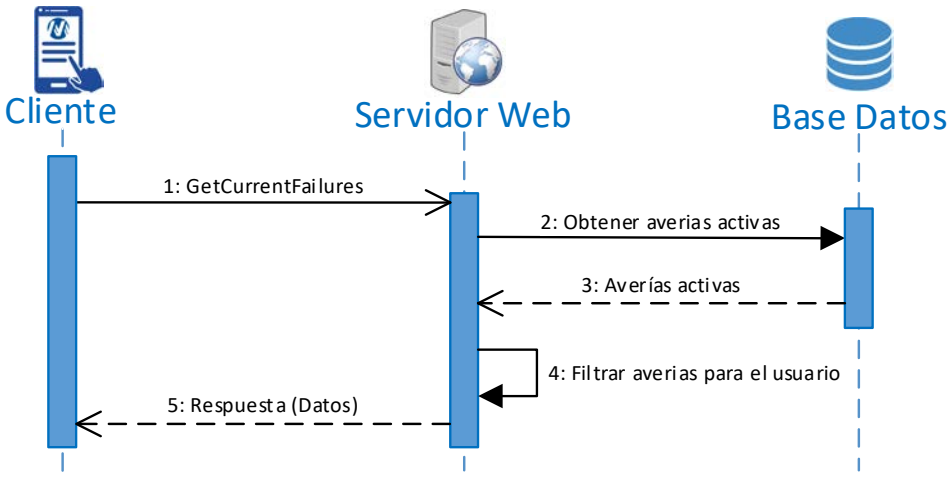

Figura 5.31.- Diagrama de secuencia GetCurrentFailures

Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET que incluya la siguiente información:

DTUCurrentFailures **GetCurrentFailures**

- userIdentificator [string]: Identificador del usuario
- failureCompanyName [string]: Compañía de la que se van a recibir las averías.
- codLanguage [string]: Lenguaje en el que se quiere recibir las averías. El código tiene que ser estándar.

El servidor devuelve un objeto de la clase DTUCurrentFailures, en la Figura 5.32 se muestra un diagrama de la clase y sus asociaciones. El objeto está compuesto por un ErrorInfo que detalla el estado de la petición y un enumerable de averías.

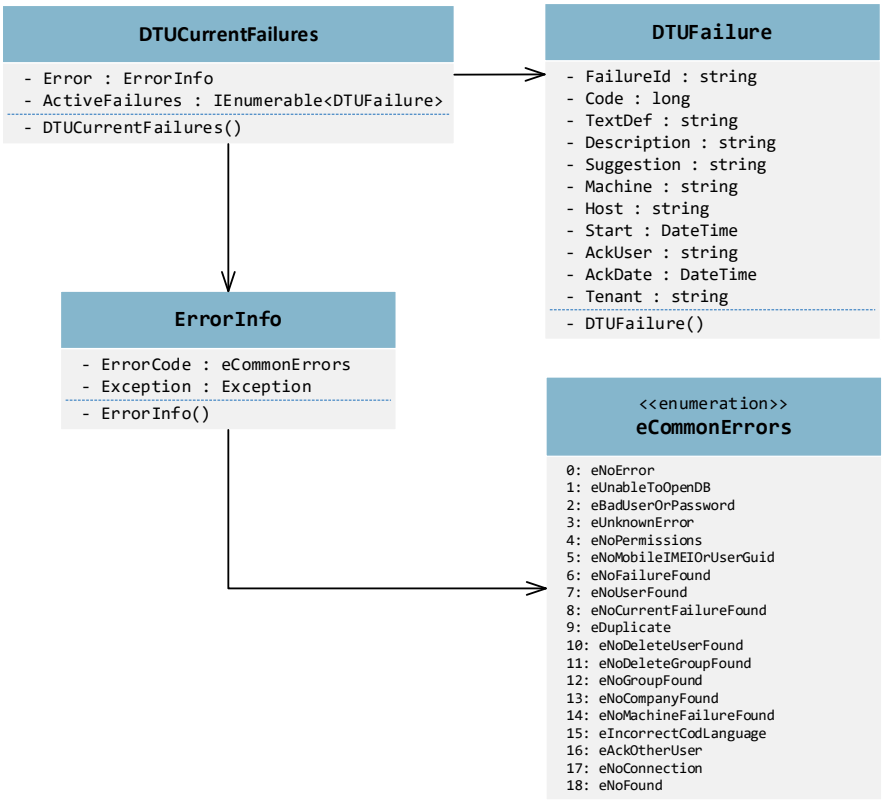

Figura 5.32.- Diagrama de clases DTUCurrentFailures

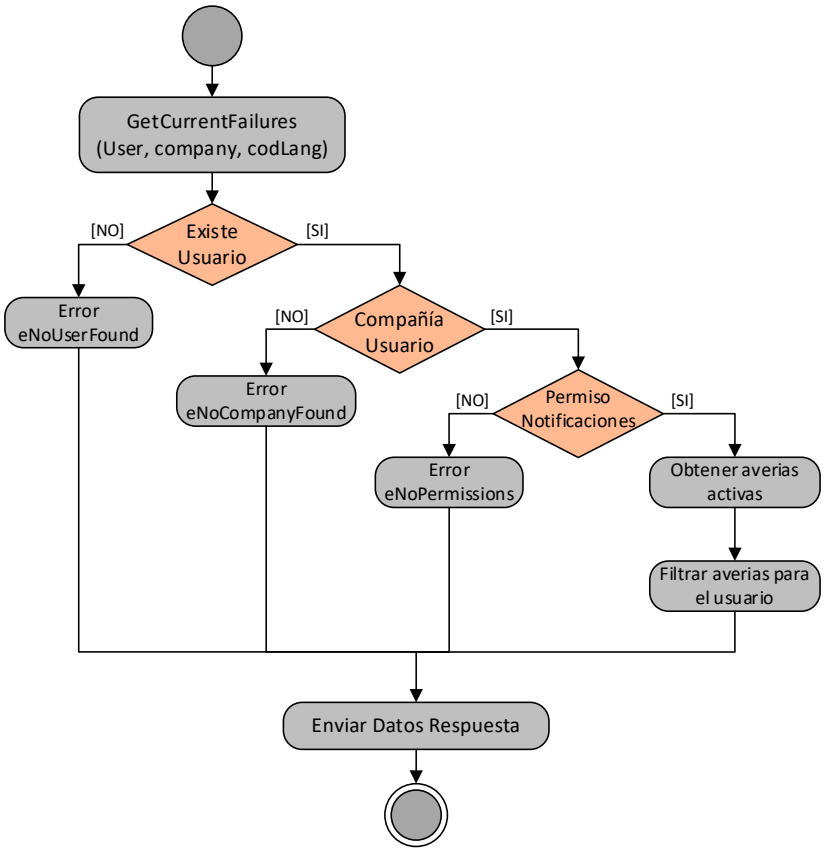

Figura 5.33.- Diagrama de actividad GetCurrentFailures

5.3.1.5.- Change Password

Este método permite al usuario cambiar su contraseña de acceso a la aplicación. Desde la configuración de la aplicación podrá introducir una contraseña nueva indicando previamente la contraseña antigua. El diagrama de secuencia seguido se muestra en la Figura 5.34.

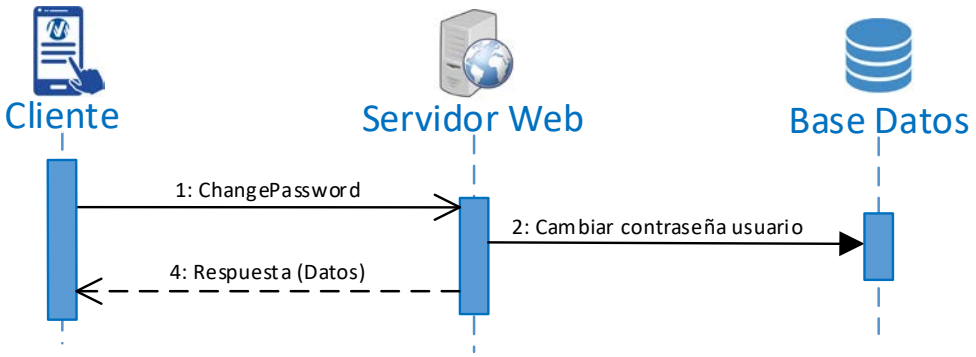

Figura 5.34.- Diagrama de secuencia ChangePassword

Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET que incluya la siguiente información:

ErrorInfo **ChangePassword**

- userIdentificator [string]: Identificador del usuario
- oldPassword [string]: Antigua contraseña del usuario.
- newPassword [string]: Nueva contraseña del usuario.

El servidor devuelve un objeto de la clase ErrorInfo visto en la Figura 5.35 que detalla el estado de la solicitud.

Los pasos seguidos en el servidor para cambiar la contraseña son los siguientes:

1. Comprobar que existe el usuario
   - No. Retornar error de usuario o contraseña incorrecta
   - Si. Actualizar la contraseña del usuario.

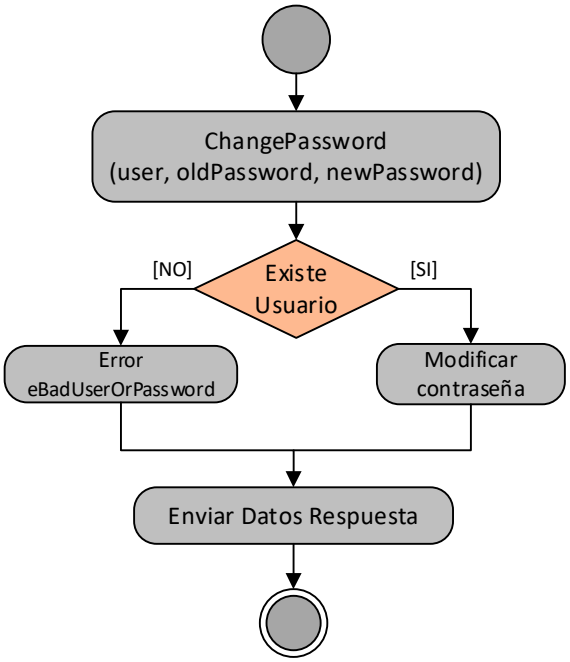

Figura 5.35.- Diagrama de actividad ChangePassword

5.3.1.6.- ACK Failures

Este método permite a los usuarios indicar que se hacen cargo de ciertas averías activas. Para cada avería que el usuario indique se almacena en la base de datos el nombre del usuario y la hora a la que se hizo cargo. El diagrama de secuencia seguido se muestra en la Figura 5.36.

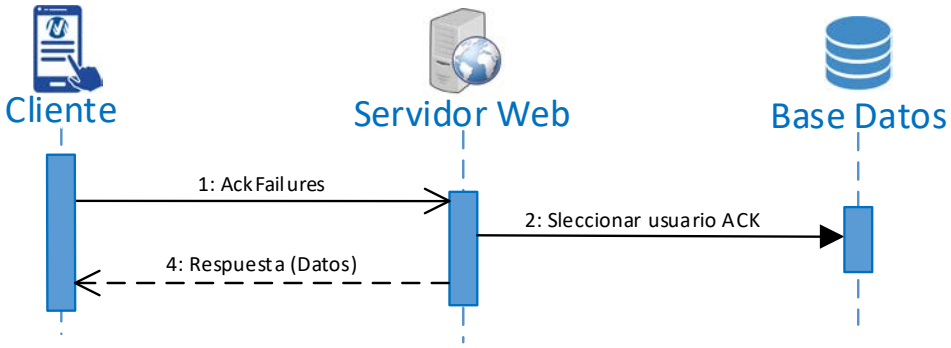

Figura 5.36.- Diagrama de secuencia ACKFailures

Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET que incluya la siguiente información:

ErrorInfo **ACKFailures**

- userIdentificator [string]: Identificador del usuario.

- AckFailuresId [Ienumerable<string>]: Identificadores de las averías.

El servidor devuelve un objeto de la clase ErrorInfo visto en la Figura 5.37 que detalla el estado de la solicitud.

Los pasos seguidos en el servidor para añadir a cada avería la información del usuario son:

1. Comprobar que el usuario existe
   No. Retornar error de usuario no encontrado (eNoUserFound).
   Si. Continuar
2. Comprobar que existen la compañía.
   No. Retornar error de compañía no encontrada (eNoCompanyFound).
   Si. Continuar.
3. Comprobar que el usuario tiene permisos.
   No. Retornar error de usuario sin permisos (eNoPermissions).
   Si. Continuar
4. Comprobar si existe la avería.
   No. Retornar error de avería no encontrada (eNoFailureFound).
   Si. Continuar.
5. Comprobar que no tiene usuario ACK.
   No. Retornar error de que ya tiene usuario ACK (eAckOtherUser).
   Si. Modificar usuario y hora de ACK.
6. Retornar datos.

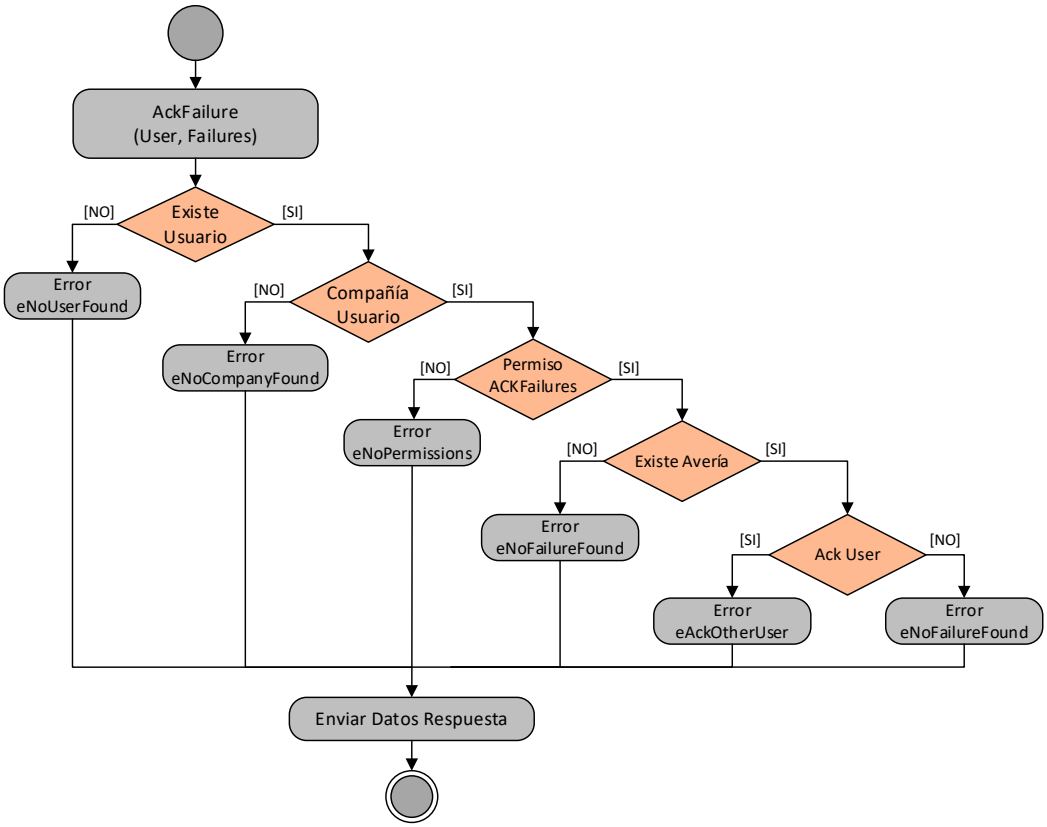

Figura 5.37.- Diagrama de actividad AckFailure

5.3.1.7.- Get Statistics

Mediante este método la aplicación móvil obtendrá las estadísticas de disponibilidad de la instalación en un periodo determinado. También recibirá la disponibilidad y la afectación de cada una de las máquinas con averías en ese periodo. La disponibilidad de una instalación en un periodo se calcula obteniendo el tiempo total en el que la instalación tuvo alguna máquina en avería. Primero se calcula el tiempo de avería de cada máquina y con ello la disponibilidad total. Mediante el peso de cada máquina, que indica el porcentaje que afecta a la instalación que la maquina este en avería, se calcula la afectación real de la maquina a la instalación. Con el tiempo total de avería de todas las máquinas en ese periodo ya se obtiene la disponibilidad de la instalación. El diagrama de secuencia seguido se muestra en la Figura 5.38.

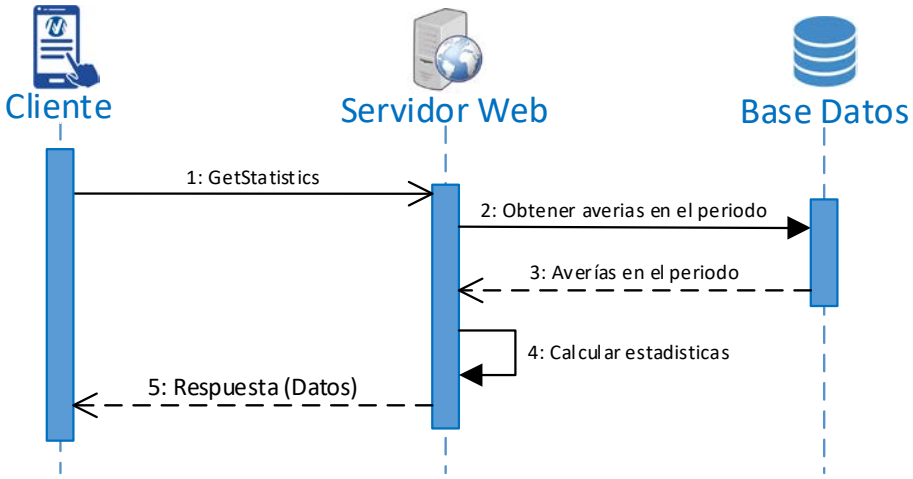

Figura 5.38.- Diagrama de secuencia de GetStatistics

Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET que incluya la siguiente información:

DTUStatistics **GetStatistics**

- userIdentificator [string]: Identificador del usuario.
- company [string]: Compañía de la que se desea conocer las averías.
- nTimeIntervals [IEnumerable<TimeIntervals>]: Periodos de tiempo en los que se desea conocer las estadísticas.

El servidor devuelve un objeto de la clase DTUStatistics, en la Figura 5.39 se muestra un diagrama de la clase y sus asociaciones. El objeto está compuesto por un ErrorInfo que contiene la información acerca de la solicitud, las estadísticas de la compañía y un enumerable de estadísticas de todas las máquinas afectadas en los periodos de tiempo. A parte está la clase de TimeIntervals que representa los intervalos de tiempo, está formada por una fecha de inicio y una fecha de fin.

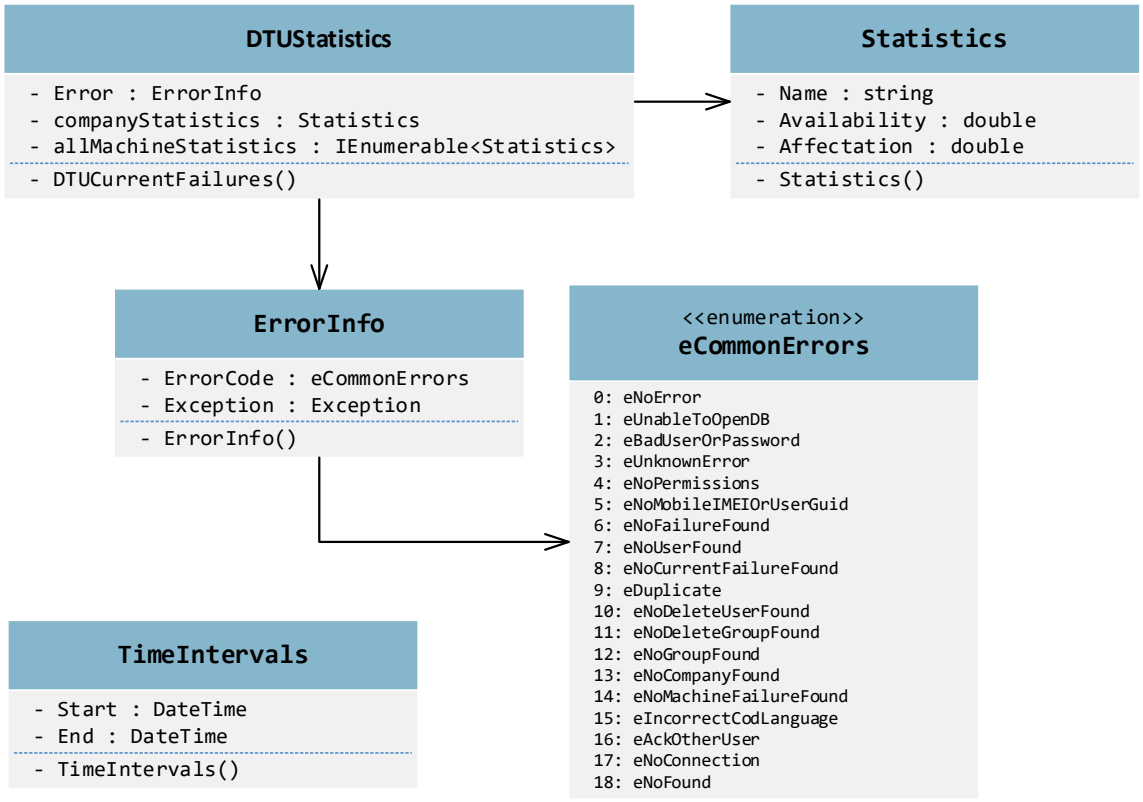

Figura 5.39.- Diagrama de clases DTUStatistics

Los pasos seguidos en el servidor para obtener y enviar las estadísticas son los siguientes:

1. Comprobar que el usuario existe
   - No. Retornar error de usuario no encontrado (eNoUserFound).
   - Si. Continuar
2. Comprobar que existen la compañía.
   - No. Retornar error de compañía no encontrada (eNoCompanyFound).
   - Si. Continuar.
3. Comprobar que el usuario tiene permisos para ver las estadísticas.
   - No. Retornar error de usuario sin permisos (eNoPermissions).
   - Si. Continuar
4. Obtener las averías en los pedidos seleccionados.
5. Calcular las estadísticas.
6. Retornar datos.

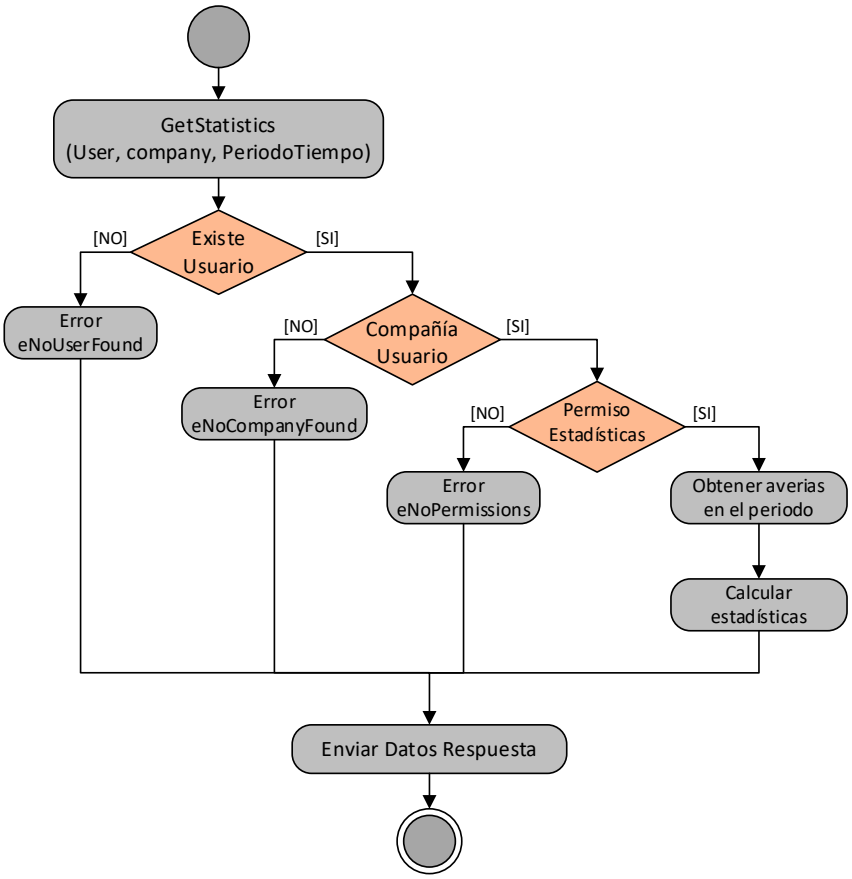

Figura 5.40.- Diagrama de actividad de GetStatistics

5.3.2.- Designer

En este grupo están los métodos que permitirán desde la aplicación de escritorio designer modificar la base de datos. En la aplicación designer se ha añadido una sección desde donde se podrán crear usuarios, grupos de usuarios, reglas, definiciones de averías, grupos de averías, permisos y las relaciones entre ellos.

Para poder crear, modificar y eliminar todos estos elementos se han creado una serie de métodos. Por un lado, se han creado los métodos que permitirán a la aplicación obtener los datos ya existentes. De este modo podrá mostrarlos para que el usuario pueda eliminarlos o editarlos. Por otro lado, para crear, modificar o eliminar se han creado otros métodos que incorporan un campo “action” donde se indica la acción a realizar (eAddItem, eEditItem y eDeleteItem).

A continuación, se comenta cada uno de los casos de uso.

5.3.2.1.- UsersAction

Mediante este método se podrán crear, modificar y eliminar usuarios de una compañía. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición POST que incluya la siguiente información:

ErrorInfo **Users**Action

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía en la que se quieren modificar los usuarios.
- aUsers [IEnumerable<ModelUser>]: Usuarios para crear, modificar o eliminar.

Para informar al método si los usuarios son para crear, modificar o eliminar cada objeto de ModelUser incorpora un parámetro eAction donde se especifica la acción que se desea llevar a cabo. En la Figura 5.41 se observa la estructura de la clase ModelUser.

Para informar a Designer del resultado de la petición el método retorna un objeto ErrorInfo donde se detalla el resultado.

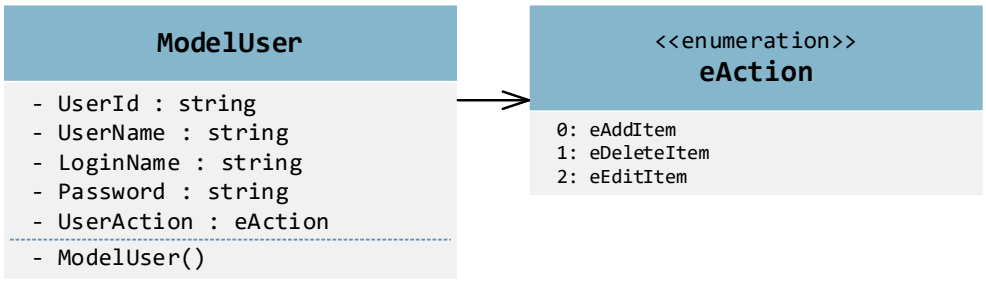

Figura 5.41.- Diagrama de clases ModelUser

5.3.2.2.- UsersGroupAction

Mediante este método se podrán crear, modificar y eliminar grupos de usuarios de una compañía. Además de crear o modificar el nombre y descripción del grupo se podrán añadir usuarios y permisos. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición POST que incluya la siguiente información:

ErrorInfo **UsersGroupAction**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía en la se quieren modificar los grupos de usuarios.
- aUsersGroup [IEnumerable<ModelGroup>]: Grupos para crear, modificar o eliminar.

En la Figura 5.42 se observa la clase ModelGroup. Para informar a Designer del resultado de la petición el método retorna un objeto ErrorInfo donde se detalla el resultado.

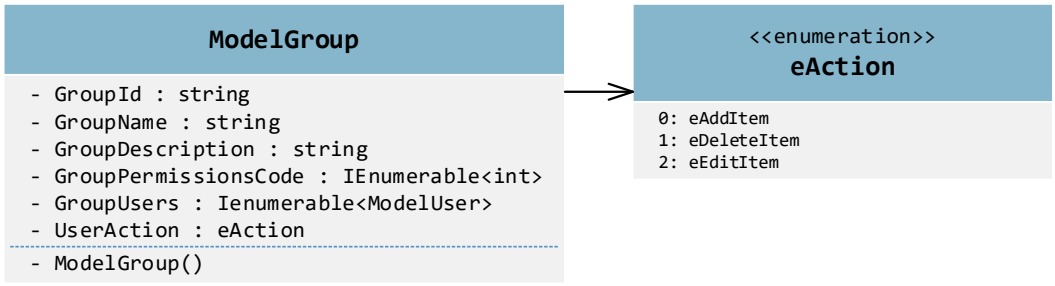

Figura 5.42.- Diagrama de clases de ModelGroup

5.3.2.3.- FailuresGroupAction

Mediante este método se podrán crear, modificar y eliminar grupos de averías de una compañía. Además de poder crear o modificar el grupo con su nombre y descripción se podrán añadir averías. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición POST que incluya la siguiente información.

ErrorInfo **FailuresGroupAction**

- userIdentificator [string]: Identificador del usuario.
- companyName [string]: Compañía en la se quieren modificar los grupos de averías.
- failuresGroup [IEnumerable<ModelFailuresGroup>]: Grupos de averías para crear, modificar o eliminar.

En la Figura 5.43 se observa la clase ModelFailuresGroup. Para informar a Designer del resultado de la petición el método retorna un objeto ErrorInfo donde se detalla el resultado.

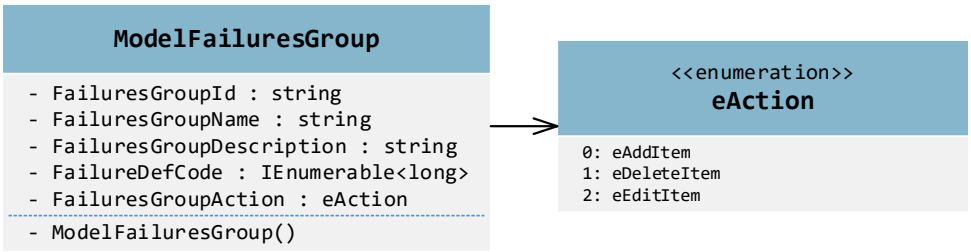

Figura 5.43.- Diagrama de clases de ModelFailuresGroup

5.3.2.4.- TimeRuleAction

Mediante este método se podrán crear, modificar y eliminar reglas de tiempo para una compañía. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición POST que incluya la siguiente información:

ErrorInfo **TimeRuleAction**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía en la que se quiere modificar las reglas de tiempo.

- aTimeRules [IEnumerable<ModelTimeRule>]: Reglas de tiempo para crear, modificar o eliminar.

En la Figura 5.44 se observa la clase ModelTimeRule. Para informar a Designer del resultado de la petición el método retorna un objeto ErrorInfo donde se detalla el resultado.

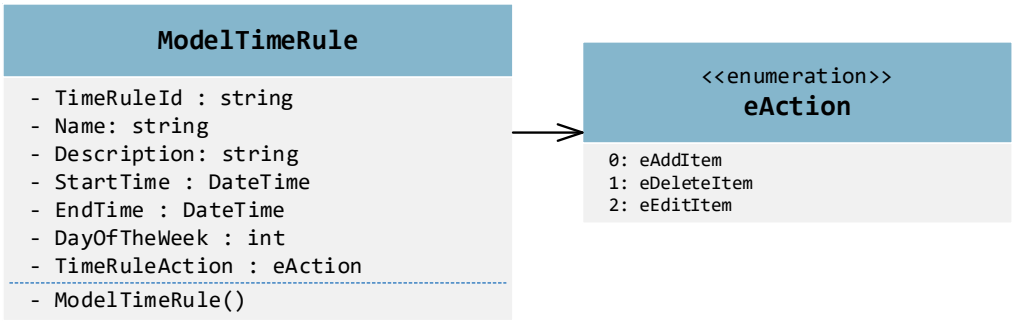

Figura 5.44.- Diagrama de clases de ModelTimeRule

5.3.2.5.- ScaleRuleAction

Mediante este método se podrán crear, modificar y eliminar reglas de escaldo para una compañía. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición POST que incluya la siguiente información.

ErrorInfo **ScaleRuleAction**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía en la que se quiere modificar las reglas de escalado.
- aScaleRules [IEnumerable<ModelScaleRule>]: Reglas de escalado para crear, modificar o eliminar.

En la Figura 5.45 se observa la clase ModelScaleRule. Para informar a Designer del resultado de la petición el método retorna un objeto ErrorInfo donde se detalla el resultado.

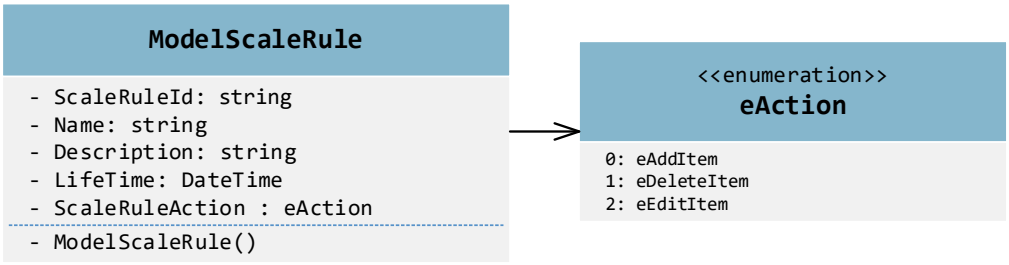

Figura 5.45.- Diagrama de clases de ModelScaleRule

5.3.2.6.- TenantAction

Mediante este método se podrán crear, modificar o eliminar compañías (tenant). Para realizar la petición al servidor los dispositivos tendrán que hacer una petición POST con la siguiente información.

ErrorInfo **TenantAction**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía a la que pertenece el usuario que quiere realizar la modificación.
- aTenant [IEnumerable<ModelTenant>]: Reglas de escalado para crear, modificar o eliminar.

En la Figura 5.46 se observa la clase ModelTenant. Para informar a Designer del resultado de la petición el método retorna un objeto ErrorInfo donde se detalla el resultado.

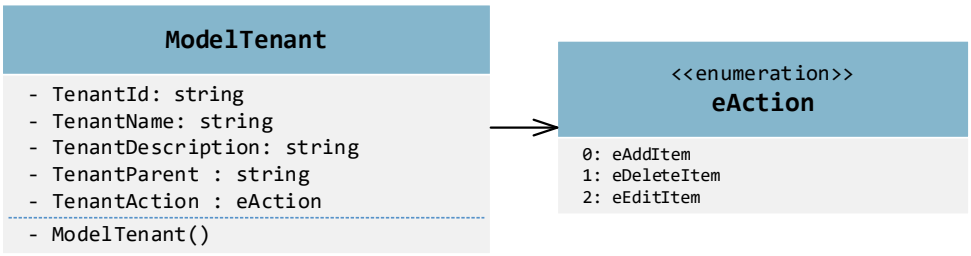

Figura 5.46.- Diagrama de clases de ModelTenant

5.3.2.7.- MachineWeightAction

Mediante este método los dispositivos podrán únicamente modificar el peso de las máquinas de la instalación de una compañía. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición POST que incluya la siguiente información.

ErrorInfo **TenantAction**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [Ienumerable<string>]: Compañía a la que pertenece el usuario que quiere realizar la modificación.
- aTenant [IEnumerable<ModelUpdateMachineFailure>]: Máquinas a las que se quiere modificar el peso.

En la Figura 5.47 se observa la clase ModelTenant. Para informar a Designer del resultado de la petición el método retorna un objeto ErrorInfo donde se detalla el resultado.

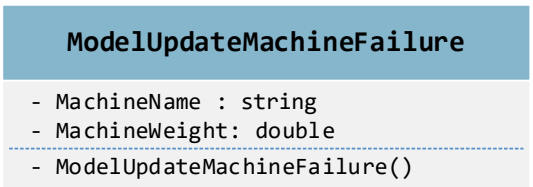

Figura 5.47.- Diagrama de clases de ModelUpdateMachineFailure

5.3.2.8.- RuleAssignmentAction

Mediante este método los dispositivos podrán crear, modificar o eliminar reglas de asignación entre los usuarios o grupos, grupos de averías y las reglas de tiempo o escalado. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición POST con la siguiente información.

ErrorInfo **RuleAssignment**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía a la que pertenece el usuario que quiere realizar la modificación.
- aRuleAssignment [IEnumerable<ModelRuleAssignment>]: Reglas de asignación que se quieren modificar.

En la Figura 5.48 se observa la clase ModelRuleAssignment. Para pasar la información de los usuarios, grupos de usuarios, reglas de tiempo, reglas de escalado y grupos de averías se utilizarán los modelos de cada uno de ellos visto anteriormente. Para informar a Designer del resultado de la petición el método retorna un objeto ErrorInfo donde se detalla el resultado.

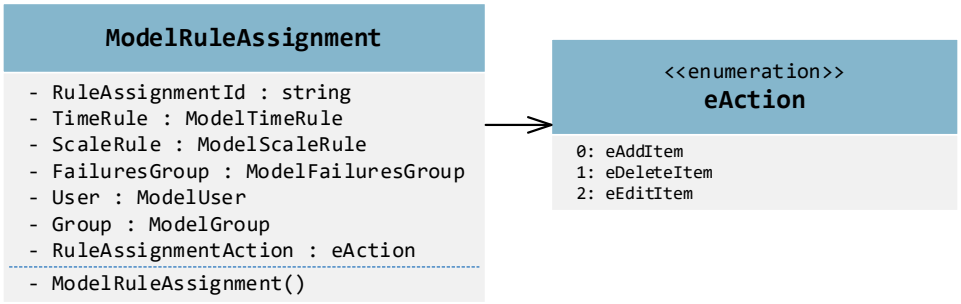

Figura 5.48.- Diagrama de clases de ModelRuleAssignment

5.3.2.9.- GetCompanyUser

Este método retorna todos los usuarios de una compañia. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET con la siguiente información.

DTUUser **GetCompanyUser**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía de la que se quieren obtener los usuarios.

En la Figura 5.49 se observa la clase DTUUser que contiene a su vez una lista de objetos de ModelUser y un objeto de ErrorInfo para informar del resultado de la operación.

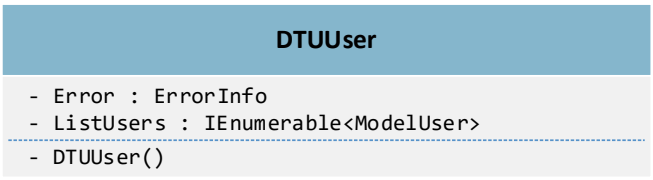

Figura 5.49.- Diagrama de clases de DTUUser

5.3.2.1.- GetCompanyUsersGroup

Este método retorna todos los grupos de usuarios de una compañía. Para cada grupo también de indicaran los usuarios y permisos. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET con la siguiente información.

DTUGroup **GetCompanyUseresGroup**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía de la que se quieren obtener los grupos de usuarios.

En la Figura 5.50 se observa la clase DTUGroup que contiene a su vez una lista de objetos de ModelGroup y un objeto de ErrorInfo para informar del resultado de la operación.

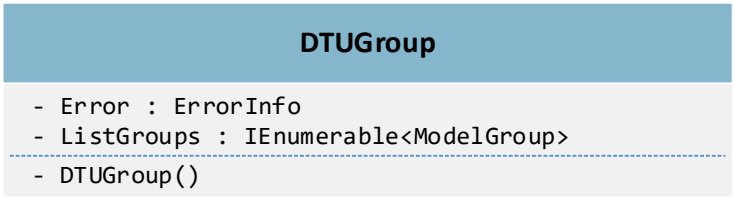

Figura 5.50.- Diagrama de clases de DTUGroup

5.3.2.1.- GetFailureDef

Este método retorna todos los códigos de averías y sus correspondientes definiciones para una compañía. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET con la siguiente información.

DTUFailureDefinition **GetFailureDef**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía de la que se quieren obtener todos los códigos de averías y sus definiciones.

En la Figura 5.51 se observa la clase DTUFailureDefinition que contiene a su vez una lista de objetos de ModelFailureDefinition y un objeto de ErrorInfo para informar del resultado de la operación.

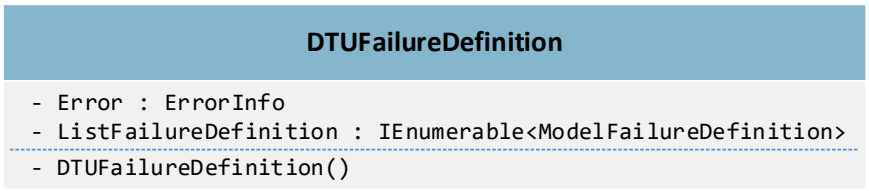

Figura 5.51.- Diagrama de clases de DTUFailureDefinition

5.3.2.2.- GetFailureGroup

Este método retorna todos los grupos de averías de una compañía y las averías que este grupo tienen asignadas. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET con la siguiente información.

DTUFailuresGroup **GetFailuresGroup**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía de la que se quieren obtener todos los grupos de averías.

En la Figura 5.52 se observa la clase DTUFailuresGroup que contiene a su vez una lista de objetos de ModelFailuresGroup y un objeto de ErrorInfo para informar del resultado de la operación.

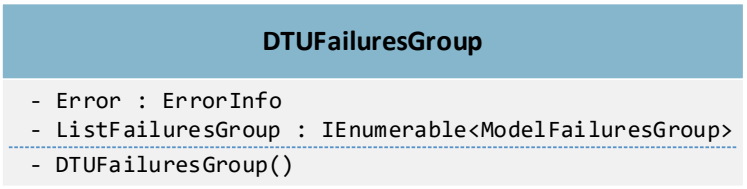

Figura 5.52.- Diagrama de clases de DTUFailuresGroup

5.3.2.3.- GetTimeRule

Este método retorna todas las reglas de tiempo de una compañía. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET con la siguiente información.

DTUTimeRule **GetTimeRule**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía de la que se quieren obtener todas las reglas de tiempo.

En la Figura 5.53 se observa la clase DTUTimeRule que contiene a su vez una lista de objetos de ModelTimeRule y un objeto de ErrorInfo para informar del resultado de la operación.

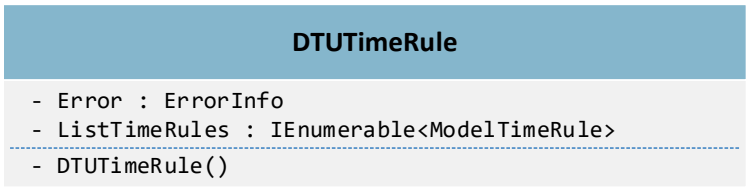

Figura 5.53.- Diagrama de clases de DTUTimeRule

5.3.2.4.- GetScaleRule

Este método retorna todas las reglas de escalado de una compañía. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET con la siguiente información.

DTUScaleRule **GetScaleRule**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía de la que se quieren obtener todas las reglas de escalado.

En la Figura 5.54 se observa la clase DTUScaleRule que contiene a su vez una lista de objetos de ModelScaleRule y un objeto de ErrorInfo para informar del resultado de la operación.

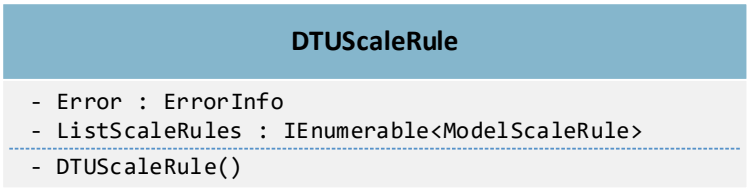

Figura 5.54.- Diagrama de clases de DTUScaleRule

5.3.2.5.- GetMachineFailure

Este método retorna todas las maquinas en las que se ha generado una avería en algún momento de la vida de la instalación. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET con la siguiente información.

DTUUpdateMachineFailure **GetMachineFailure**

- aCompany [string]: Compañía de las que se quieren obtener todas las máquinas de averías.

En la Figura 5.55 se observa la clase DTUUpdateMachineFailure que contiene a su vez una lista de objetos de ModelUpdateMachineFailure y un objeto de ErrorInfo para informar del resultado de la operación.

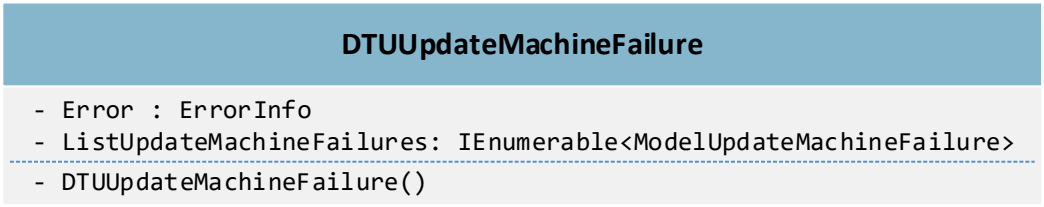

Figura 5.55.- Diagrama de clases de DTUUpdateMachineFailure

5.3.2.6.- GetTenant

Este método retorna todas las compañías definidas en la base de datos con el nombre de la empresa padre a la que pertenecen si la hubiera. Si el nombre de la compañía se define como NULL devuelve todas las compañías de la base de datos. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET con la siguiente información.

DTUTenant **GetTenant**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía de las que se quieren obtener todas sus compañías hijos. Si este valor es NULL se devolverán todas las compañías de la base de datos.

En la Figura 5.56 se observa la clase DTUTenant que contiene a su vez una lista de objetos de MoldelTenant y un objeto de ErrorInfo para informar del resultado de la operación.

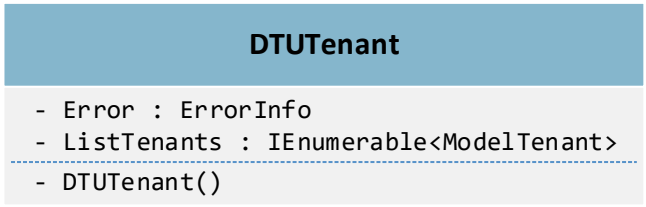

Figura 5.56.- Diagrama de clases de DTUTenant

5.3.2.7.- GetRuleAssignment

Este método retorna todas las reglas de asignación para un usuario o grupo de usuarios perteneciente a una compañía. Para realizar la petición al servidor los dispositivos tendrán que hacer una petición GET con la siguiente información.

DTURuleAssignment **GetRuleAssignment**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía de las que se quieren obtener las reglas de asignación del usuario o grupo de usuarios.
- assignmentElementId [string]: Identificador del usuario o grupo de usuarios del que se quieren obtener las reglas d asignación.

En la Figura 5.57 se observa la clase DTURuleAssignment que contiene a su vez una lista de objetos de MoldelTenant y un objeto de ErrorInfo para informar del resultado de la operación.

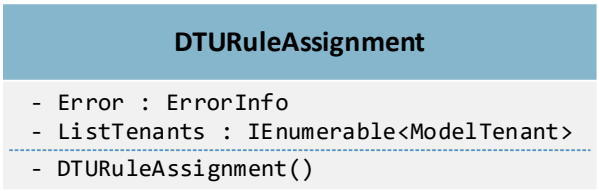

Figura 5.57.- Diagrama de clases de DTURuleAssignment

5.3.3.- Cliente

Los métodos del cliente permitirán a los sistemas de control, ya sean propios de Mecalux o externos actualizar la lista de definición de averías, así como crear o modificar averías activas en la base de datos.

Para permitir a los distintos sistemas de control que se pueden instalar en un almacén acceder a estos métodos se creó una puerta de enlace (Gateway) que enlaza las peticiones del sistema de control con la base de datos. En el apartado 5.5 se detalla el funcionamiento del mismo.

5.3.3.1.- SetFailureDef

Mediante este método se podrá actualizar las listas de averías disponibles para una instalación, así como detallar los distintos idiomas en los que se encuentran. Para realizar la petición al servidor los clientes tendrán que hacer una petición POST con la siguiente información.

ErrorInfo **SetFailureDef**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía a la que pertenece el usuario que quiere realizar la modificación.
- allNewFailureDef [IEnumerable<DTUNewFailureDef>]: Conjunto de averías con sus correspondientes definiciones.

En la clase DTUNewFailureDef se detalla el código de la avería y las distintas definiciones que tiene.

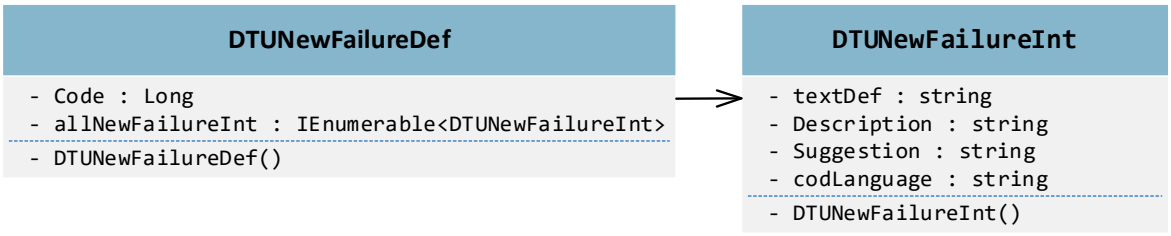

Figura 5.58.- Diagrama de clases DTUNewFailureDef

5.3.3.2.- SetNotifyFailure

Mediante este método los sistemas de control podrán guardar las averías en la base de datos. Para realizar la petición al servidor los sistemas cliente tendrán que hacer una petición POST con la siguiente información.

ErrorInfo **SetNotifyFailure**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía a la que pertenece el usuario que quiere realizar la modificación.
- allNewNotifyFailure [IEnumerable<DTUNotifyFailure>]: Conjunto de averías a notificar.

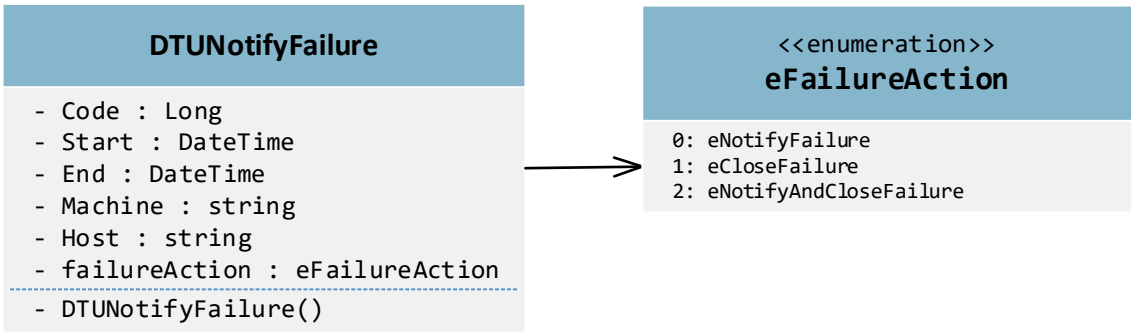

Figura 5.59.- Diagrama de clases DTUNotifyFailure

Se podrán notificar tres tipos de averías según se indique en failureaction:

- NotifyFailure: Avería nueva que se encuentra activa, es decir no tiene fecha de fin, solo fecha de inicio.
- CloseFailure: Se utiliza para notificar que una avería activa que a esta en la base de datos debe cerrarse, es decir ponerle fecha de fin.
- NotifyAndCloseFailure: Avería nueva que se crea en la base de datos con fecha de inicio y fin.

5.3.3.3.- EndComputerFailures

Mediante este método se podrán cerrar todas las averías activas en la base de datos para una lista de host (ordenadores). Para realizar la petición al servidor los sistemas cliente tendrán que hacer una petición POST con la siguiente información.

ErrorInfo **EndComputerFailures**

- userIdentificator [string]: Identificador del usuario.
- nCompanyName [string]: Compañía a la que pertenece el usuario que quiere realizar la modificación.
- allEndComputers [IEnumerable<string>]: Conjunto de ordenadores (Host) de los que se quiere cerrar las averías.

El servidor busca todas las averías activas para cada host y les pone como fecha fin la actual en el momento de la llamada al método.

5.3.3.4.- EndTenantFailures

Mediante este método se podrán cerrar todas las averías activas en la base de datos de una lista de compañías. Para realizar la petición al servidor los sistemas cliente tendrán que hacer una petición POST con la siguiente información.

ErrorInfo **EndTenantFailures**

- userIdentificator [string]: Identificador del usuario.
- allEndTenant [IEnumerable<string>]: Conjunto de compañías (Tenant) de las que se quiere cerrar las averías.

El servidor busca todas las averías activas para cada compañía y les pone fecha de fin la actual en el momento de la llamada al método.

5.4.- Servicio de notificaciones

Como ya vimos en el apartado 5.3 el servidor es el encargado de gestionar las peticiones de los clientes tanto para enviar como para recibir información hacia la base de datos. Esta no será su única funcionalidad, también será el encargado de gestionar la notificación de nuevas averías activas a los usuarios registrados en el servicio de notificaciones.

Para manejar el servicio de notificaciones el servidor contará con distintos hilos de procesamiento trabajando en paralelo. Cada uno de estos hilos se encarga de gestionar las averías de una única empresa. Este proceso se repetirá de forma cíclica cada treinta segundos.

En primer lugar, el hilo comprueba qué usuarios están registrados en el servicio de notificaciones y tienen permisos para recibirlas. A continuación, obtiene todas las averías activas para cada una de las empresas a las que tiene acceso ese cliente y comprueba si cumple las reglas de asignación. Si hay alguna avería activa asignada a ese usuario se comprueba en una tabla si esa avería ya se le fue notificada y si no es así se procede a enviar una notificación.

Para que los dispositivos móviles reciban las notificaciones se utilizará un servicio de notificadores. Para el caso de Android e iOS se utilizará la plataforma Firebase Cloud Messaging (FCM) y para Windows el servicio de notificaciones de inserción de Windows (WNS).

5.4.1.- Firebase Cloud Messaging (FCM)

Firebase Cloud Messaging (FCM) es una solución multiplataforma que nos permitirá enviar, de forma gratuita y segura, notificaciones a los dispositivos móviles. Mediante las notificaciones push podremos enviar paquetes de datos, en los que incluiremos la información de las averías activas, desde el servidor hacia los usuarios gracias a la intervención del servicio de mensajería del servidor Firebase.

Para hacer funcionar este sistema necesitamos de la interacción de tres elementos:

- El servidor web.
- La app Android o iOS (Failures Manager).
- El servidor de Firebase.

Figura 5.60.- Envio de notificaciones push en Firebase

Como se muestra en el diagrama anterior, nuestro servidor envía una petición al servidor de Firebase. Luego la plataforma notifica a los dispositivos que estén registrados al proyecto.

Para realizar él envió de notificaciones correctamente y poder enviar mensajes aislados a cada usuario independientemente, en la parte de la aplicación móvil el desarrollador debe incluir Firebase. En primer lugar, se tiene que registrar la aplicación en Firebase con lo que se obtendrá una clave para el servidor y la aplicación que se debe incluir en el proyecto. Esta clave será necesaria en la parte del servidor para enviar los mensajes a nuestra aplicación y no otras.

Para que cada usuario puede recibir únicamente las notificaciones de las averías que le corresponden es necesario obtener un identificador único por usuario. En la Figura 5.61 mostrada a continuación se representa un esquema del proceso llevado a cabo para enviar notificaciones a cada usuario.

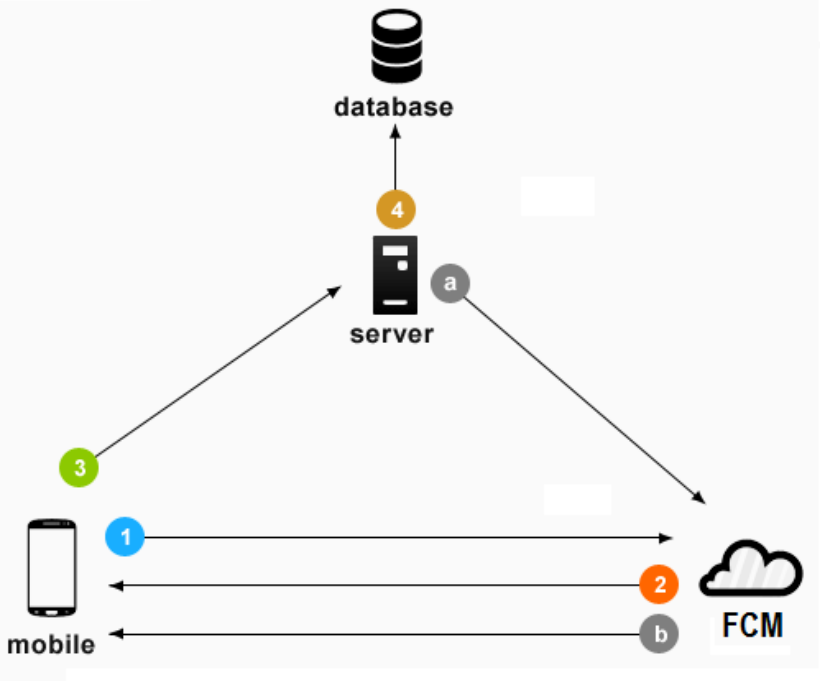

Figura 5.61.- Esquema detallado de las notificaciones push Firebase

Lo primero que se hace es obtener el toquen único de cada usuario que se registra en la aplicación y se almacena en la base de datos como muestran los siguientes pasos:

1. El usuario se registra en la aplicación con su usuario y contraseña. Si este tiene permisos de recibir notificaciones la aplicación solicita a Firebase un toquen de notificaciones que será único para ese usuario.
2. Firebase genera un toquen y se lo devuelve a la aplicación.
3. La aplicación envía el toquen de ese usuario al servidor junto con un identificador único del dispositivo móvil.
4. El servidor almacena el toquen y el identificador en la tabla de MobilDevice comentada en el punto 5.1.6 de este documento.

Una vez el toquen está almacenado en la base de datos el hilo de notificaciones anteriormente comentado empieza a buscar averías activas para el usuario. Cuando este usuario tiene averías activas que notificar se siguen los siguientes pasos:

a. El servidor envía un mensaje HTTP al servidor Firebase con la información de las averías activas.

b. Firebase se encarga de enviar la notificación al usuario registrado en esa aplicación con la información.

En el mensaje HTTP que se envía desde el servidor se incluye únicamente la información acerca del número de averías activas y algunos parámetros de configuración que se explican a continuación:

- **Collapse_key:** Este parámetro identifica un grupo de mensajes que se pueden contraer para que solo el último mensaje se envíe cuando se reanude la entrega. Esto es para evitar que se envíe demasiada cantidad de los mismos mensajes cuando el dispositivo se vuelve a conectar o se activa.
- **Time_To_Live:** Este parámetro especifica el tiempo (en segundos) que el mensaje se guarda en el almacenamiento de FCM si el dispositivo está desconectado.
- **Notification:** Este parámetro especifica los pares de clave/valor predefinidos visibles para el usuario de la carga de notificación.
  - **Title:** Indica título de notificación.
  - **Body:** Indica texto del cuerpo de la notificación, donde se pondrá el número de averías activas para este usuario.
  - **Icon:** Indica el ícono de notificación.
  - **Tag:** Indica si cada mensaje de notificación genera una nueva entrada en el panel lateral de notificaciones en Android. Si se configura y ya hay una notificación con la misma etiqueta, la nueva notificación reemplaza la existente en el panel lateral de notificaciones.
  - **Sound:** Indica un sonido para reproducir cuando el dispositivo recibe la notificación.
- **To:** Este parámetro especifica el destinatario del mensaje, corresponde al toquen de notificación único de cada usuario.

Cuando la app está en segundo plano, Android dirige los mensajes de notificación a la bandeja del sistema. Cuando el usuario hace clic en una notificación abre el lanzador de la app de forma predeterminada y obtiene todas las averías activas agrupadas por máquinas.

5.4.2.- Servicio de notificación de Windows (WNS)

Con este servicio se podrá enviar notificación de las averías activas a los usuarios de dispositivos de Windows de manera segura y bajo consumo.

El sistema es similar al utilizado en Firebase, en primer lugar, la aplicación ha de estar registrada en el panel de la tienda. Esto proporcionara las credenciales de la aplicación que nuestro servicio en la nube usara en la autenticación con WNS. Estas credenciales son un identificador de seguridad de paquete (SID) y una clave secreta.

Una vez registrada la aplicación cada usuario que acceda a la aplicación desde un dispositivo de Windows seguirá los siguientes pasos como se muestra en la Figura 5.62.

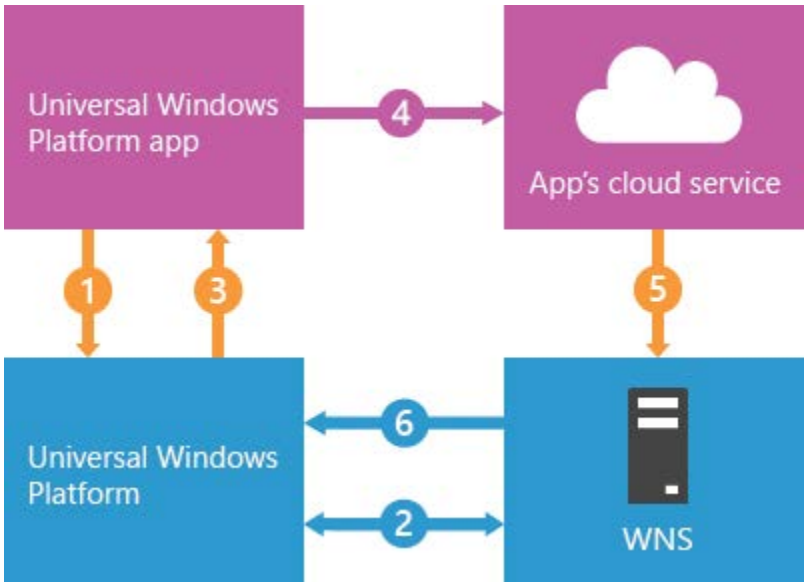

Figura 5.62.- Esquema detallado de las notificaciones push Windows

Los cuatro primeros pasos solo se realizan cuando el usuario se registra en la aplicación. Si no cierra sesión no será necesario realizarlos más. A continuación, se detalla cada uno de los pasos:

1. La aplicación solicita un canal de notificación de inserción de la plataforma universal de Windows.
2. Windows solicita a WNS que cree un canal de notificación. que será devuelto al dispositivo de llamada en forma de identificador uniforme de recursos (URI).
3. Windows devuelve el URI del canal de notificación a tu aplicación.
4. Tu aplicación envía el URI a tu propio servicio de nube. A continuación, debes almacenar el URI en tu propio servicio de nube para poder acceder al URI al enviar notificaciones. El URI es una interfaz entre tu propia aplicación y tu propio servicio; es responsabilidad tuya implementar esta interfaz con estándares web seguros y fiables.
5. Cuando tu servicio de nube tenga una actualización por enviar, WNS recibe una notificación mediante el URI del canal. Para ello, se emite una solicitud HTTPPOST, incluida la carga de notificación, a través de la Capa de sockets seguros (SSL). Este paso requiere autenticación.
6. WNS recibe la solicitud y enruta la notificación hacia el dispositivo pertinente.

5.5.- Servicio Gateway de averías

El Gateway es un servicio de Windows que hace de puente entre la base de datos y el sistema de control del almacén. Un servicio es un programa que funciona en el computador en segundo plano. El servicio se inicia automáticamente cuando arranca el equipo y se

puede parar, pausar y reiniciar en cualquier momento sin mostrar ninguna interfaz de usuario.

Esta puerta de enlace está pensada específicamente para que cualquier sistema de control, ya sea propio de Mecalux (Galileo) o externo que se esté ejecutando en un ordenador industrial, autómata programable, etc. pueda utilizar la aplicación de notificación de averías. El sistema de control deberá implementar una serie de funciones con las que se comunicará con la puerta de enlace. Sera la puerta de enlace la que se comunique con el servidor permitiendo así modificar las averías en la base de datos. El Gateway de averías estará continuamente escuchando en el puerto 3000, que es el puerto en el que los sistemas de control gestionan las averías.

El Gateway se comunicará con el servidor para modificar la base de datos, utilizando los métodos cliente, vistos en el apartado 5.3.3. Se necesitará un usuario en cada compañía que tenga los permisos de Gateway para identificar cada llamada al servidor y una tabla para almacenar cada identificador de compañía con su nombre que se guardará en un archivo de configuración.

Al instalar el servicio se instala a parte una pequeña aplicación de configuración con la que se podrá modificar el archivo y definir cuál es el puerto y el host de conexión, el usuario del Gateway y una tabla para almacenar distintas compañías con su identificador. El programa también permitirá hacer un test de conexión para probar que los datos de conexión y el usuario son correctos antes de guardar la configuración. Para hacer efectiva la configuración en el servicio este debe ser reiniciado, por ello la aplicación de configuración dispone de una pestaña para modificar el estado del servicio. Una vez se ha introducido la configuración se podrá guardar para que el servicio pueda acceder a ella cuando sea necesario. Para ver en detalle el funcionamiento de la aplicación de configuración consulte el manual de usuario en el apartado 6.2 del presente documento.

Para gestionar las averías activas en la base de datos el Gateway implementara tres métodos que se detallan a continuación.

5.5.1.1.- EndCurrentFailures

Este método permite a los sistemas de control cerrar todas las averías activas en la base de datos para una compañía. El método únicamente recibe el identificador de la compañía. En primer lugar, el método comprueba que hay cargada una configuración de conexión, es decir que el puerto y el host de enlace con el servidor no son nulos y a continuación comprueba que el usuario del Gateway no es nulo. Si no se hay ninguna configuración de conexión o un usuario de Gateway se procede a cargar el archivo de configuración. Esto solo se hará cuando se arranque o reinicie el servicio.

Si los datos de conexión y el usuario son correctos se realiza una petición al método del servidor visto en el apartado 5.3.3.4 EndTenantFailures especificando el usuario y

únicamente una compañía. El nombre de la compañía tiene que buscarlo en la tabla de configuración.

5.5.1.2.- EndComputerFailures

Este método permite a los sistemas de control cerrar todas las averías activas en la base de datos para una lista de ordenadores (host). Para ello el método recibe el identificador de la compañía y una lista con los nombres de los ordenadores de los que quiere cerrar las averías activas. Del mismo modo que el apartado anterior se comprueba la configuración de conexión y el usuario y a continuación se hace una petición al método visto en el apartado 5.3.3.3. EndComputerFailures del servidor con el usuario y la lista de host.

5.5.1.3.- GetCurrentFailure

Este método permite a los sistemas de control crear y modificar averías activas en la base de datos. El método recibe un objeto de la clase WCSFaultSet que se muestra en la Figura 5.63.

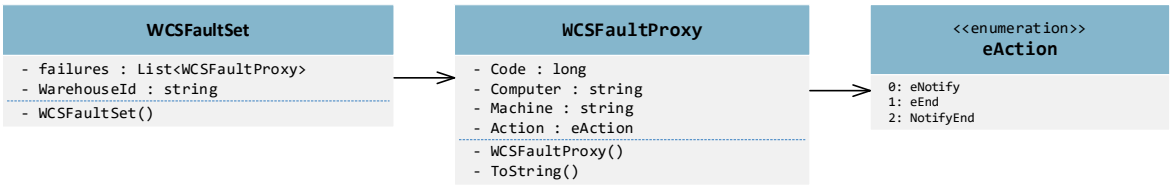

Figura 5.63.- Diagrama de clases de WCSFaultSet

La clase está compuesta por un identificador de la compañía y una lista de averías. Cada una de las averías dispone de un campo Action donde se identifica para qué es la avería, pudiendo ser una de las siguientes opciones:

- **eNotify:** Especifica que se trata de una avería nueva en el sistema de control a la que se pondrá únicamente fecha de inicio en el momento de su creación.
- **eEnd:** Especifica que se quiere cerrar una avería activa que ya se encuentra notificada en la base de datos.
- **eNotifyEnd:** Especifica que se quiere almacenar en la base de datos una avería nueva con fecha de inicio y de fin.

Una vez recibida la petición se comprueba la configuración y ser crean los objetos de la clase DTUNotifyFailure con las averías recibidas para a continuación hacer la petición al método del apartado 5.3.3.2 SetNotifyFailure del servidor.

6.- MANUAL DE USUARIO

6.1.- Aplicación escritorio

Para modificar la base de datos de forma cómoda se realizó una aplicación en Windows form que nos permitirá crear elementos y relaciones entre ellos para prácticamente todas las tablas. Para acceder a la aplicación hace falta tener un usuario y contraseña en alguna de las compañías registradas en la base de datos, además, de tener los permisos necesarios.

En la Figura 6.1 se muestra la pantalla inicial para acceder a la aplicación, donde el usuario debe indicar sus datos de usuario, contraseña y compañía a la que pertenece. Si los datos son correctos se accede a la aplicación mostrada en la Figura 6.5. En caso de que los datos sean incorrectos se mostrará un mensaje de error:

- "This user does not have permissions to modify the database"
- "User or password are incorrect"

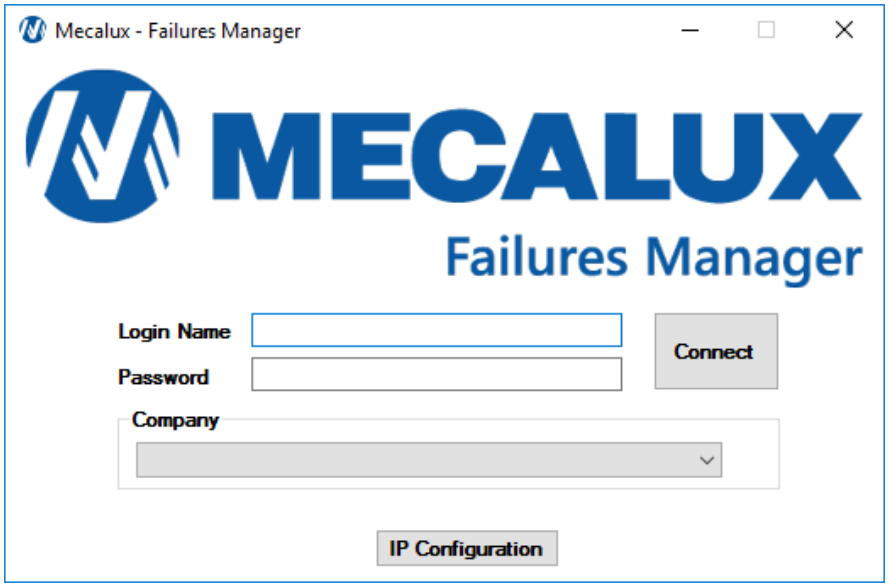

Figura 6.1.- Pantalla Login

Mediante el botón de "IP configuration" el usuario podrá acceder a una pestaña donde ajustar el puerto y el host al que se conecta el servicio de obtención de datos.

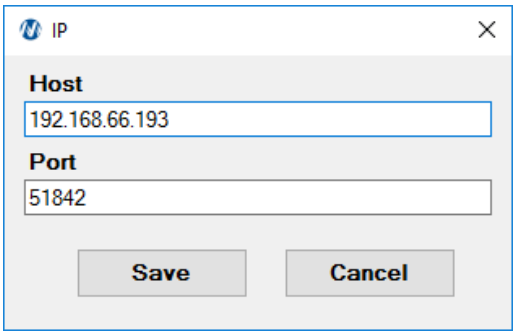

Figura 6.2.- Pantalla de configuracion de la conexión

En todas las ventanas de la aplicación donde el usuario desee realizar una operación y alguno de los campos están vacíos se mostrará un mensaje de advertencia. De esta manera el usuario podrá corregir y volver a realizar la petición.

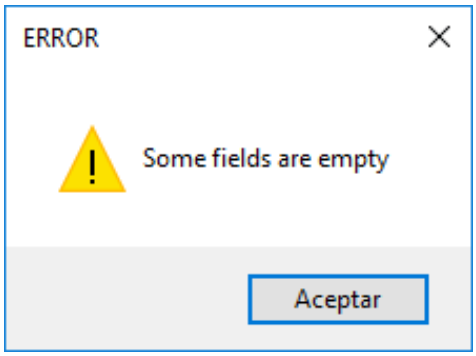

Figura 6.3.- Mensaje de error

Por otro lado, también se mostrará un mensaje de error cuando el programa intente realizar una operación que de una excepción. Por ejemplo, cuando la aplicación haga una llamada al servidor, pero este no esté conectado. En este caso en la ventana emergente se mostrará el mensaje de la excepción.

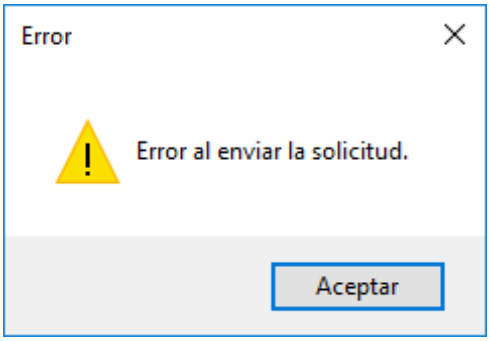

Figura 6.4.- Mensaje de error

A continuación, se muestra la pantalla principal de la aplicación.

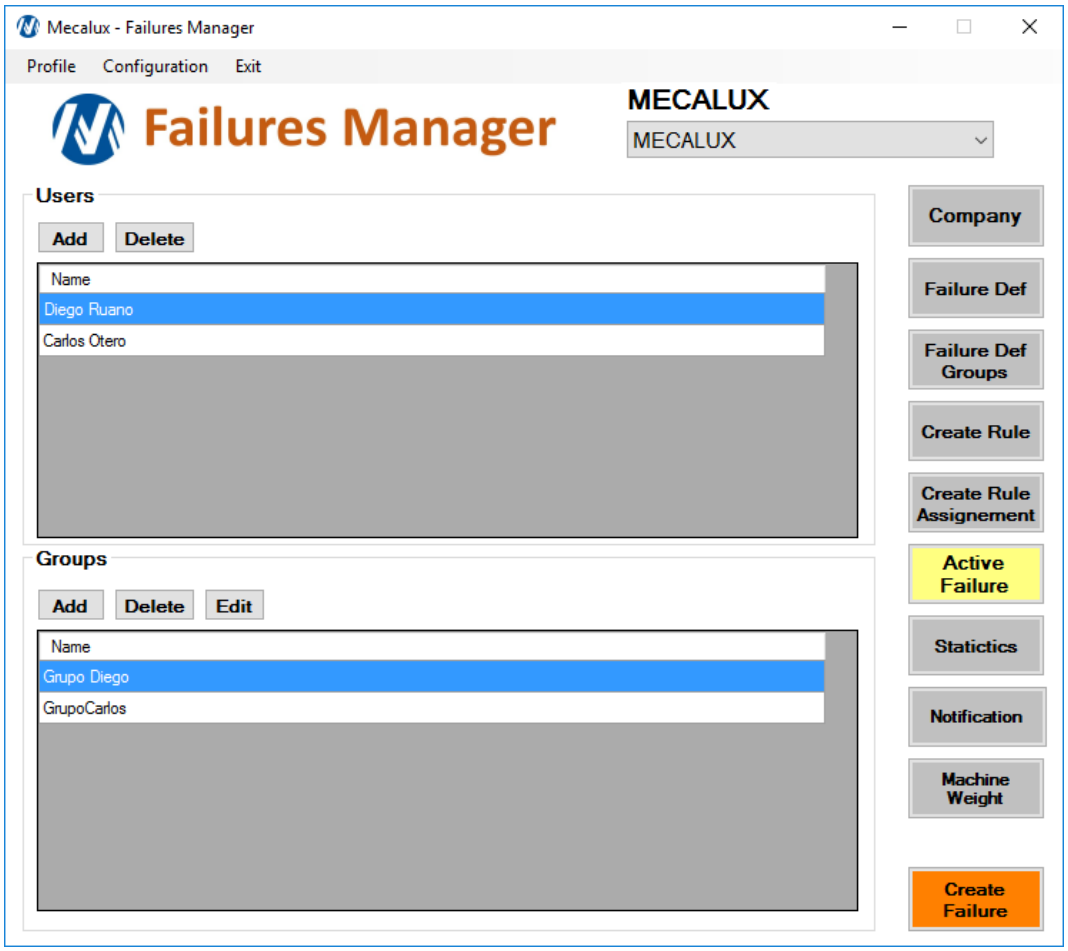

Figura 6.5.- Pantalla principal

En la pantalla principal el usuario puede ver arriba a la derecha la empresa a la que pertenece y elegir la empresa que desea modificar si esta pertenece a la suya. Una vez que se elige una empresa se muestran en la parte central (Users) todos los usuarios de la empresa. Mediante el botón de “Add” se podrán crear usuarios para la empresa seleccionada y mediante el botón de “Delete” se eliminan los usuarios seleccionados. Debajo de los usuarios se muestran los grupos de usuarios, mediante el botón de “Edit” o haciendo doble clic sobre el grupo se accede a la pantalla de configuración de grupos donde podremos seleccionar los permisos que tiene el grupo y los usuarios. Del mismo modo que los usuarios con el botón “Delete” se podrán borrar los grupos seleccionados.

En zona superior de la pantalla hay tres submenús que permitirán cambiar la contraseña, seleccionar el idioma, cerrar sesión y salir de la aplicación

En la parte derecha de la pantalla se encuentran una serie de botones que permitirán al usuario según sus permisos una serie de funciones que se detallan a continuación.

6.1.1.- Crear usuarios

Mediante el botón “Add” se accde a una ventana donde se podrán crear usuarios para la empresa seleccionada. En la Figura 6.6 se muestra como es la pantalla de creación de usuarios.

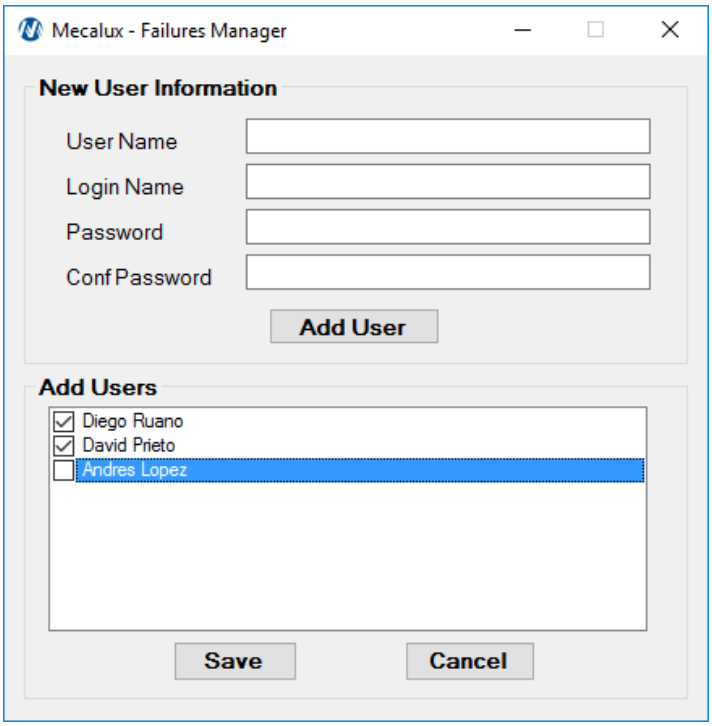

Figura 6.6.- Pantalla de creacion de usuarios

De forma muy sencilla se pueden crear múltiples usuarios sin más que asignar un nombre, identificador de usuario y contraseña. A medida que se van creando usuarios se pueden ver en la lista. Los cambios no se guardan hasta pulsar el botón “Save”.

6.1.2.- Crear y editar grupos

Para crear o editar grupos de usuarios se accede a una pantalla pulsando el botón de “Add” o “Edit” situados en la sección de grupos o haciendo doble clic sobre algún grupo. En esta pantalla se podrá asignar un nombre, una descripción, permisos y usuarios a cada grupo.

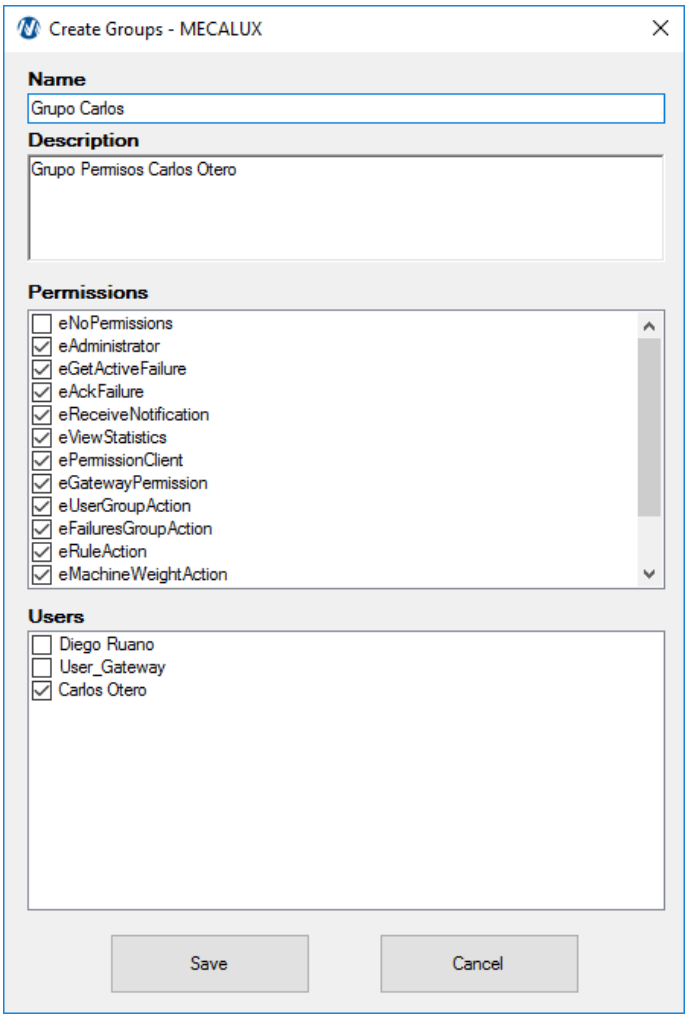

Figura 6.7.- Pantalla de creación de grupos

6.1.3.- Compañías

El primero de los botones de la derecha en la ventana principal, “Company”, abrirá una ventana donde se podrán crear y eliminar grupos.

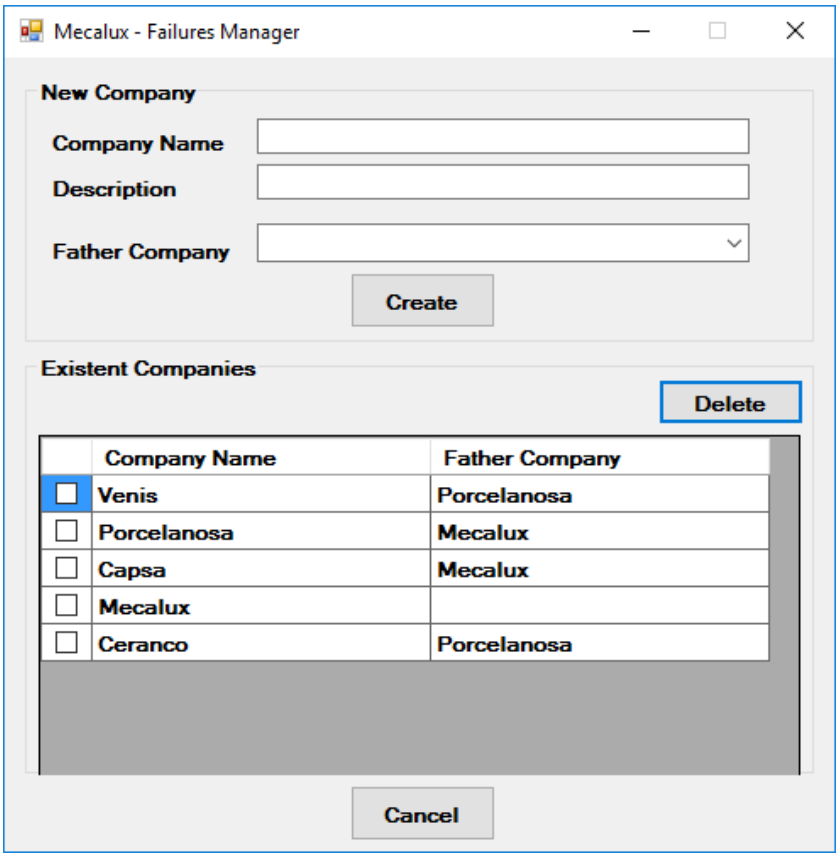

Figura 6.8.-Pantalla de creación de compañias

En la parte de arriba de la ventana se podrán crear grupos indicando el nombre de la empresa y su descripción. También se puede asignar una empresa padre si la tuviera. En la parte de abajo se pueden ver todas las empresas que hay en la base de datos y la empresa de la que cuelgan. Seleccionado las empresas y pulsando “Delete” se podrán eliminar.

6.1.4.- Definiciones de averías

En esta ventana se podrán incluir definiciones de averías para la empresa seleccionada en los idiomas que se quiera. Se podrán introducir los idiomas a mano o mediante la opción de importar de una hoja Excel. Una vez añadido o modificado las averías será necesario guardar los cambios.

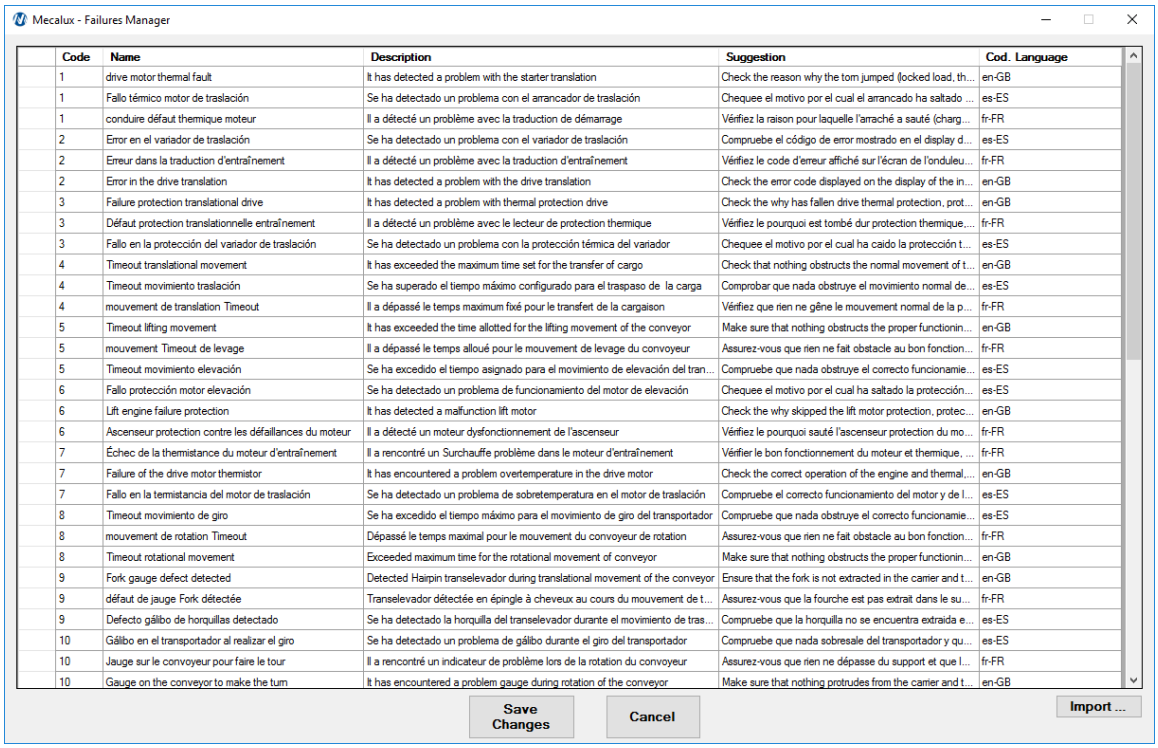

Figura 6.9.- Pantalla de definicion de averías

Para que se puedan guardar los cambios será necesario que todos los campos tengan texto y que al campo del código de lenguaje este dentro de la lista de códigos internacionales.

6.1.5.- Grupos de averías

Esta ventana permite al usuario crear grupos de averías, que son los que irán asociados a los usuarios o grupos de usuarios y a las reglas. En la pantalla se muestran todos los grupos de averías de la compañía seleccionada. Encima de los grupos hay tres botones con los que se podrá crear, modificar y eliminar los grupos.

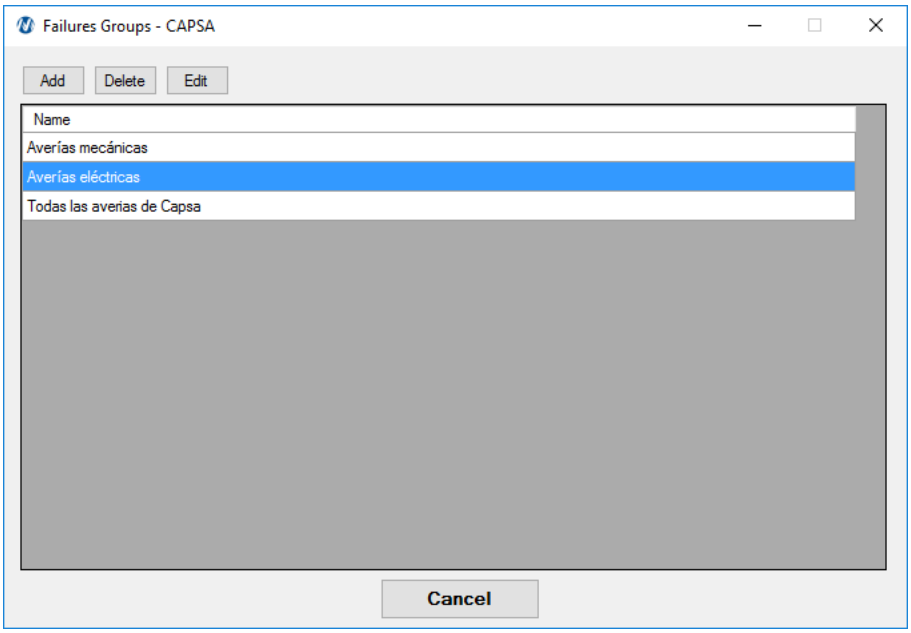

Figura 6.10.- Pantalla de visualizacion de grupos de averías

Para crear y modificar grupos de averías se abre otra ventana en blanco para crear uno nuevo y con los datos del grupo si se desea modificar. Cada grupo de averías permite configurar el nombre una descripción y las averías que tiene asignadas.

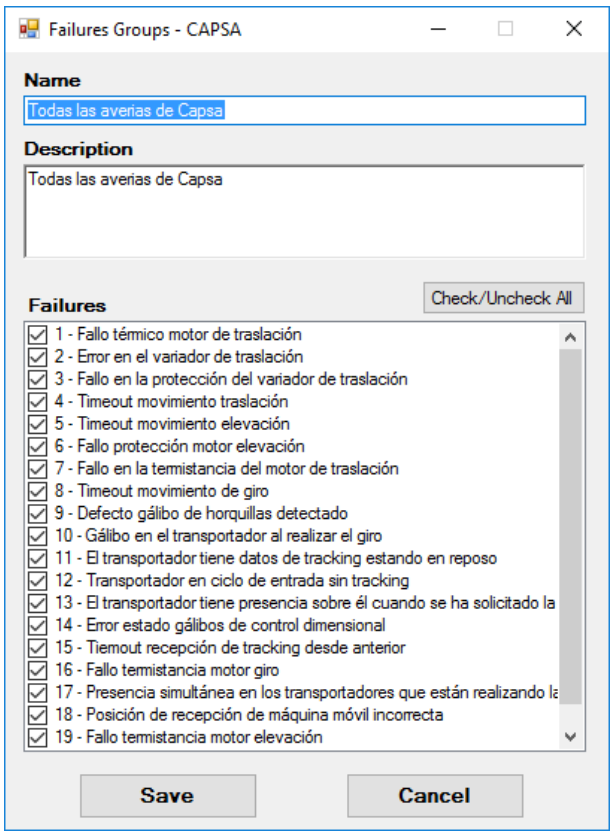

Figura 6.11.- Pantalla de creacion de grupos de averías

6.1.6.- Crear Reglas

En esta ventana se podrán crear y eliminar reglas de tiempo y escalado. De la misma forma que para los grupos de averías se muestran de forma independiente dos tablas, una para las reglas de tiempo y la otra para las reglas de escalado.

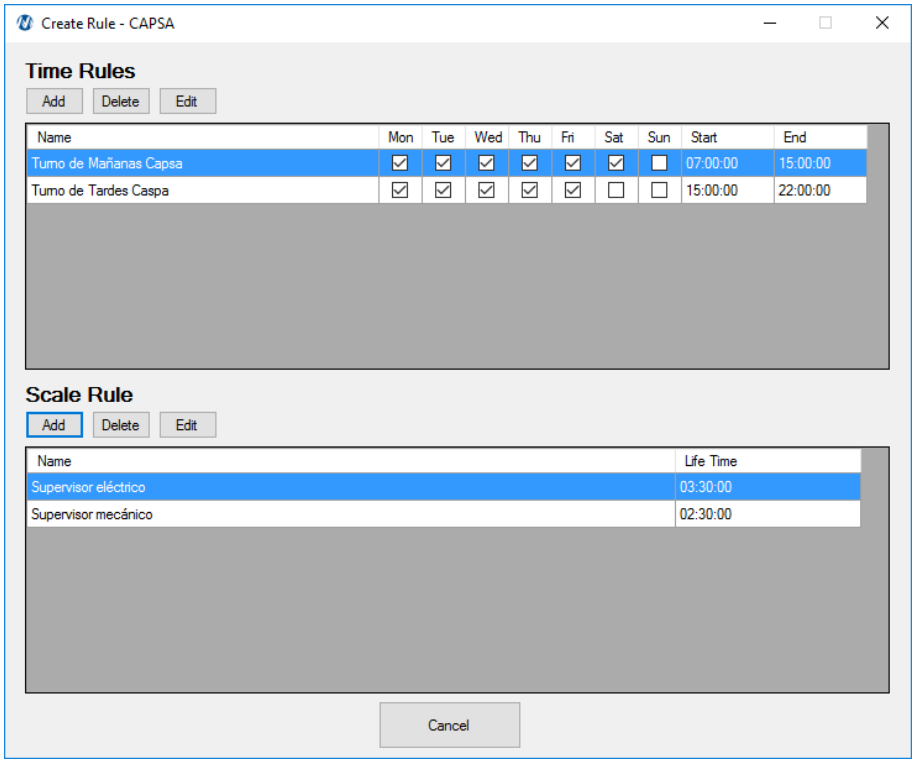

Figura 6.12.- Pantalla de visualización de reglas

Igual que para los casos anteriores disponemos de los botones de añadir, editar y borrar para cada una de las reglas, que abrirá una pantalla especifica con la que poder crear o modificar las reglas.

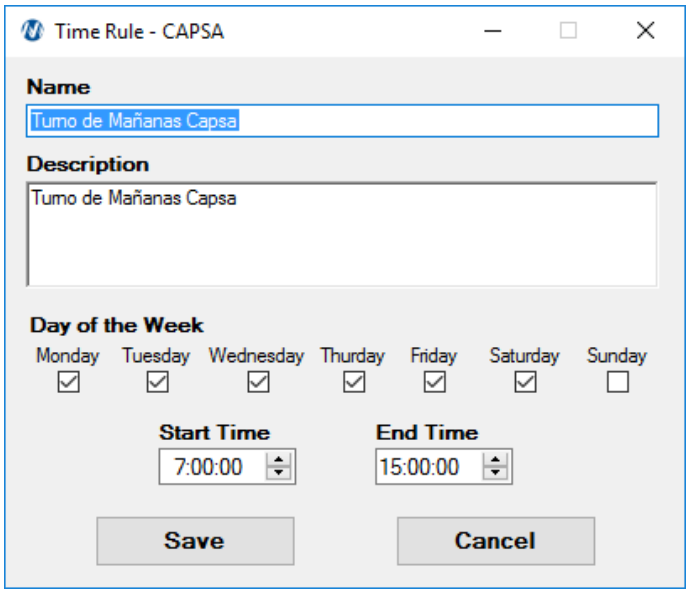

Figura 6.13.- Pantalla de creación de reglas de tiempo

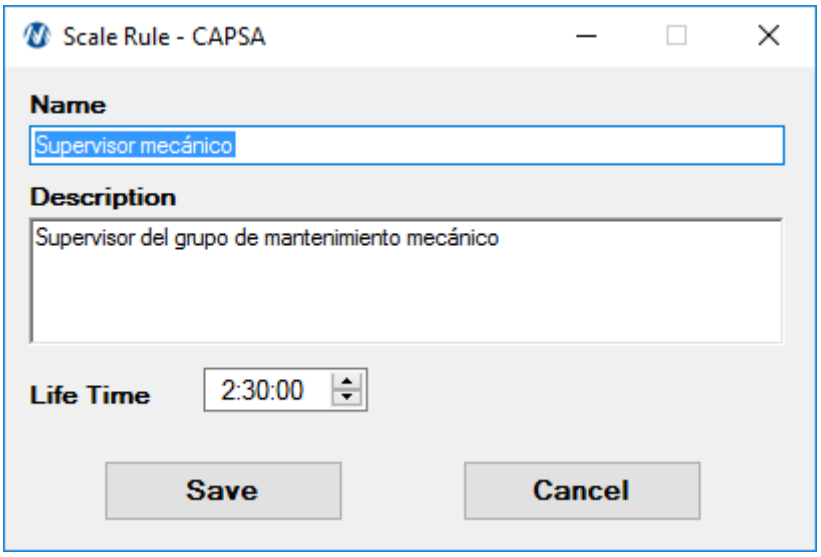

Figura 6.14.- Pantalla de creación de reglas de tiempo

6.1.7.- Reglas de asignación

En esta ventana se podrán realizar las reglas de asignación entre usuarios o grupos de usuarios, grupos de averías y las reglas.

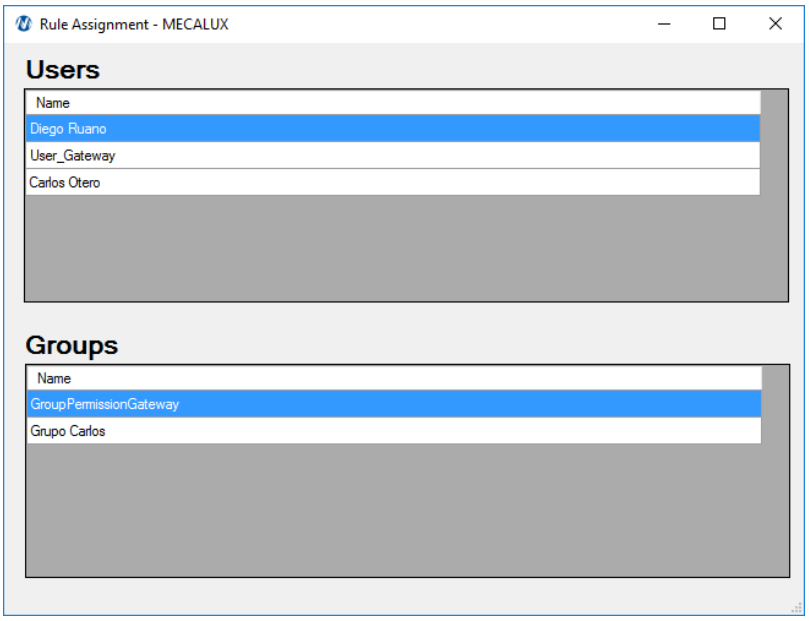

Figura 6.15.- Pantalla de visualización de reglas de asignación

En primer lugar, se muestran todos los usuarios y grupos de usuarios de la compañía. Si se selecciona alguno de ellos se podrá ver en una ventana emergente las reglas que tienen asignadas.

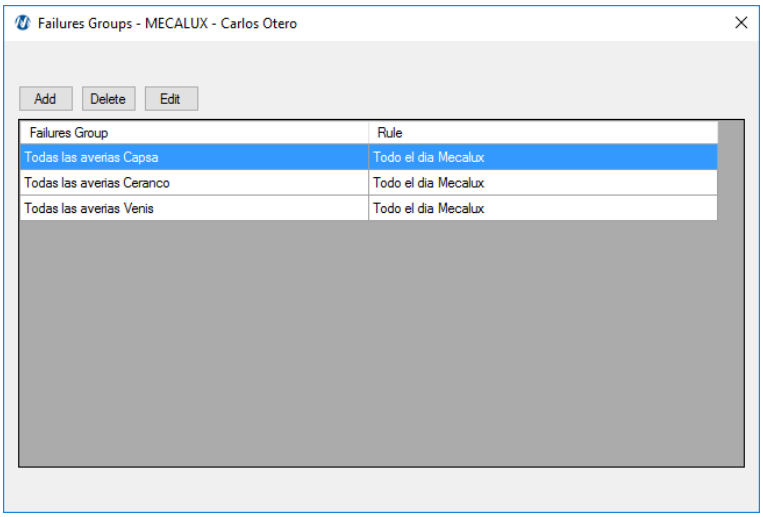

Figura 6.16.- Pantalla de visualización de reglas de asignación

En esta venta se podrán eliminar reglas asignadas, modificarlas y crear reglas nuevas para el usuario o grupo seleccionado hacia todas las compañías que cuelgan de la propia.

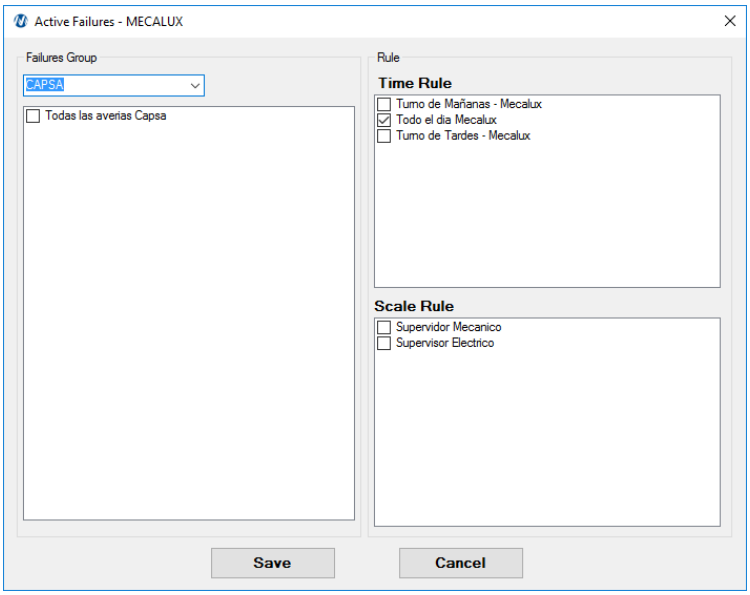

Figura 6.17.- Pantalla de creación de reglas de asignación

6.1.8.- Averías activas

En esta venta únicamente se podrán ver las averías activas que tiene el usuario asignadas en la instalación.

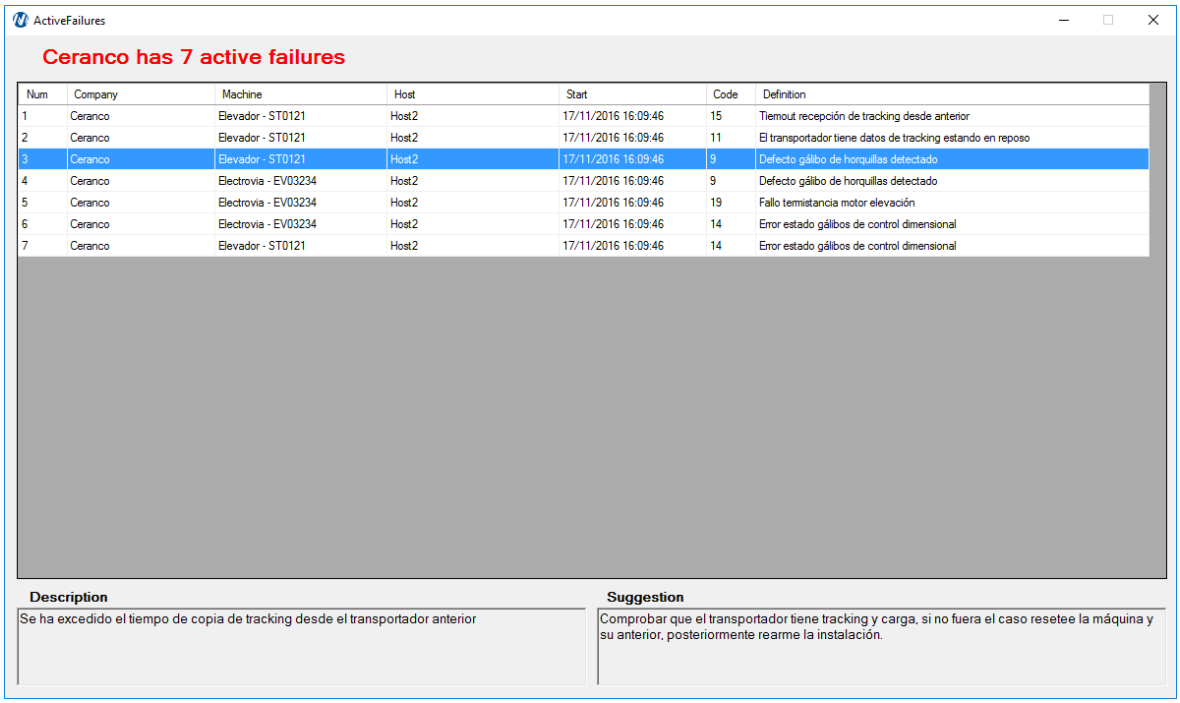

Figura 6.18.- Pantalla de averías activas

De cada avería se muestra la compañía, la maquina afectada, al host al que pertenece, la hora a la que se originó, el código y la definición. Cuando se selecciona una avería, en la parte de abajo se muestra la descripción y la sugerencia para resolver dicha avería.

6.1.9.- Estadísticas

En esta ventana se podrán ver las estadísticas globales de la aplicación dentro del intervalo de tiempo seleccionado y las averías parciales de cada máquina, así como lo que afectan en la instalación.

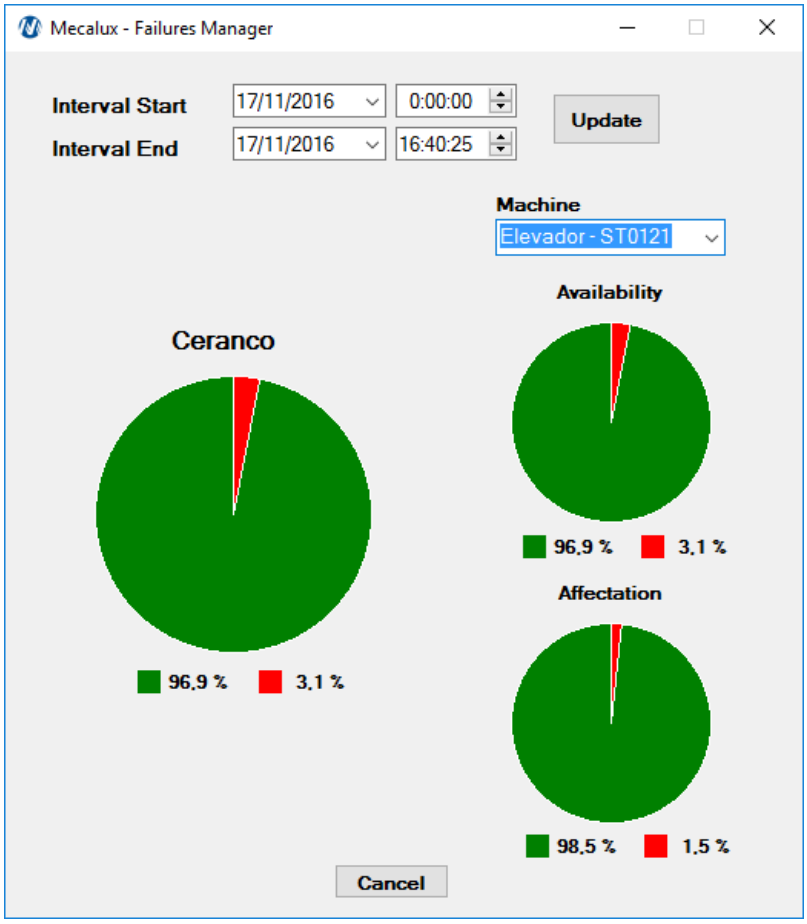

Figura 6.19.- Pantalla de visualización de estadísticas

6.1.10.- Peso de las máquinas

En esta ventana se podrá configurar el peso de cada máquina de la instalación. Simplemente se modifican los valores y se guardan los cambios.

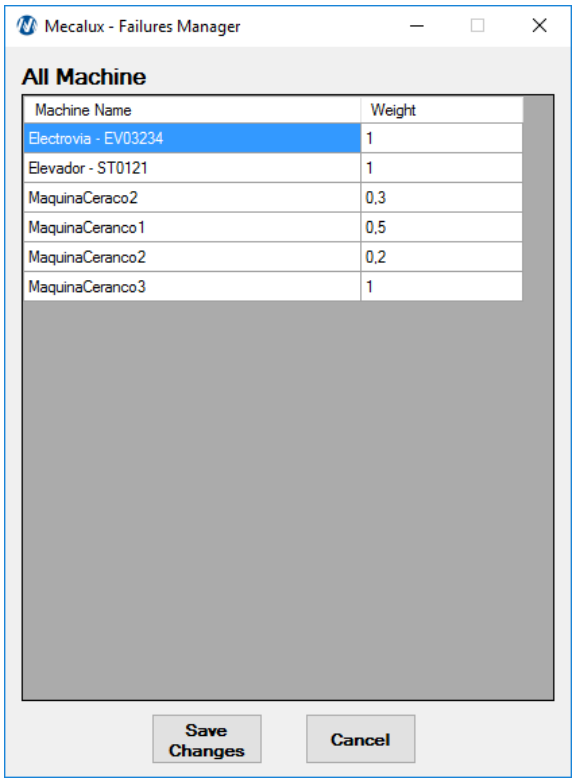

Figura 6.20.- Pantalla de modificación del peso de las maquinas

6.1.11.- Crear averías

Esta ventana únicamente se hizo para hacer pruebas y simulaciones con la aplicación móvil, ya que solo podrá crear averías activas el sistema de control que gestione la instalación.

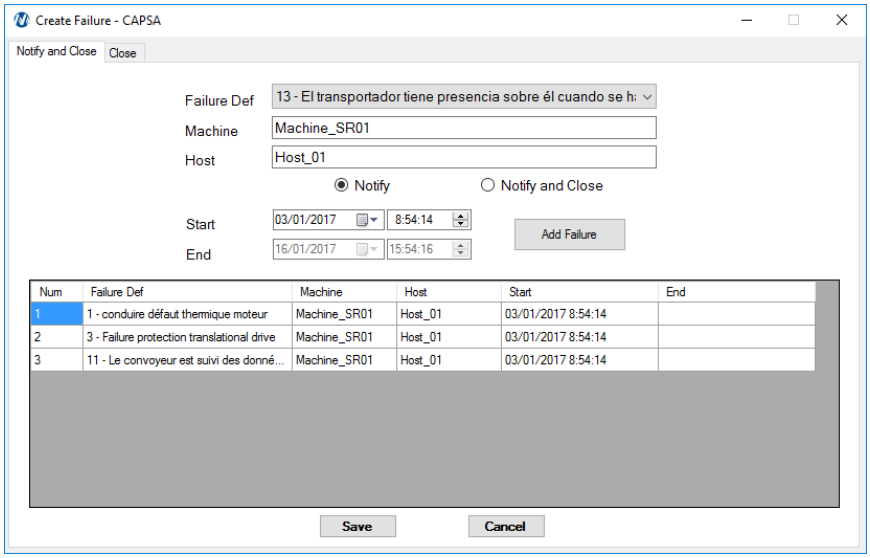

Figura 6.21.- Pantalla de creación de averías

También incorporar una pestaña en la que se podrán cerrar las averías activas, simplemente se les pone fecha de fin la hora en la que cierra.

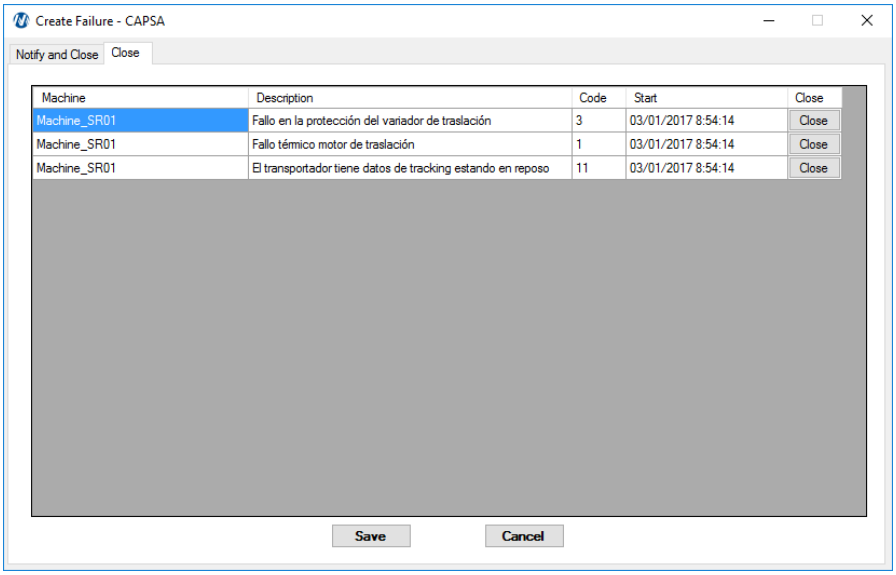

Figura 6.22.- Pantalla para cerrar averías activas

6.2.- Configuración del Gateway de averías

Esta aplicación de escritorio se instala junto al servicio de Windows GatewayFailures y sirve para modificar el archivo de configuración que utiliza el servicio para realizar las comunicaciones con el servidor web.

Al abrir la aplicación esta intenta leer la configuración guardada y la muestra en la pantalla. Si no encontrara una configuración guardada, mostraría todos los campos en blanco para que el usuario pudiera añadir una nueva configuración. En la Figura 6.23 se muestra la pantalla principal de la aplicación donde se muestra la dirección de conexión (Host y Port), el usuario, la contraseña y la compañía. Para comprobar que la información es correcta se dispone de un botón de test que realiza una petición de login informando al usuario si la operación ha sido correcta. A continuación, se muestran los mensajes que pueden ocurrir:

1. Connection Error – Problema de conexión, por favor compruebe el host y el puerto y pruebe de nuevo.
2. Connection Error – Usuario o contraseña incorrecta.
3. Connection Error – Compañía no encontrada.
4. Successful Connection

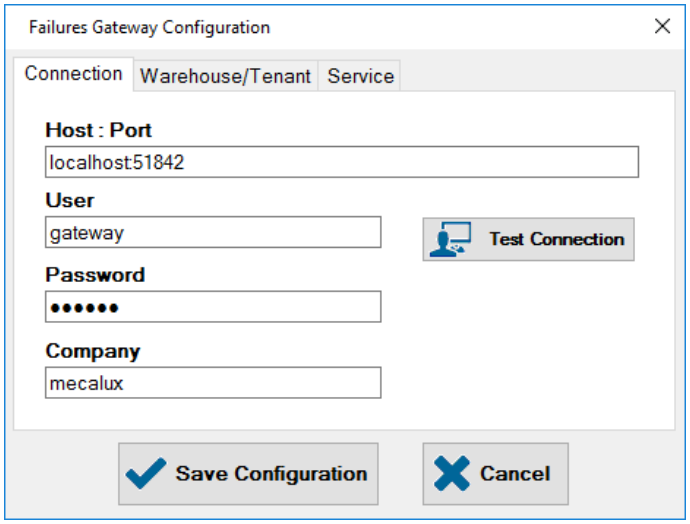

Figura 6.23.- Configuración Gateway – Conexión

En la siguiente pestaña se podrán crear relaciones entre identificadores de almacén (warehouseId) y los nombres de las compañías, ya que servicio de control trabaja con identificadores de almacén con formato numérico y el servidor web los identifica por el nombre. Si al guardar, alguno de los identificadores no es numérico se muestra un error (Algunos identificadores no son números), del mismo modo que si alguna entrada tiene un campo vacío se muestra otro error (Algunos campos están vacíos).

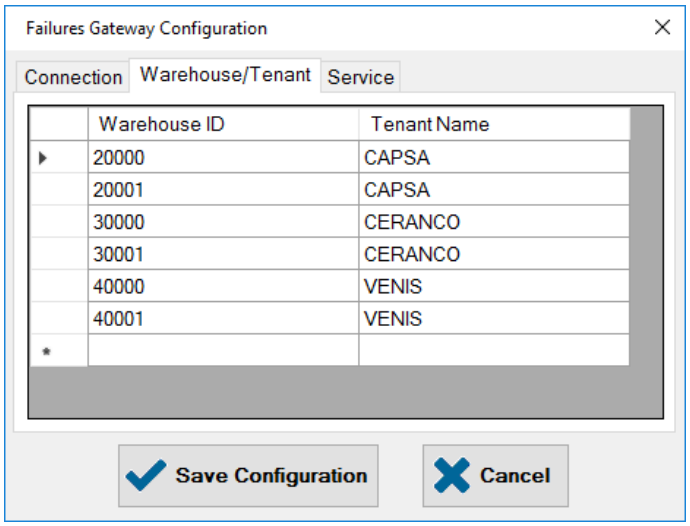

Figura 6.24.- Configuración Gateway - Warehouse Tenant

Para que la configuración guardada sea utilizada por el servicio, este ha de ser reiniciado. Por ello, en la última pestaña se permite ver el estado del servicio y modificarlo. En la parte superior se muestra el nombre y el estado en que se encuentra el servicio y mediante tres botones se permite al usuario arrancar, parar o reiniciar el servicio de forma muy sencilla.

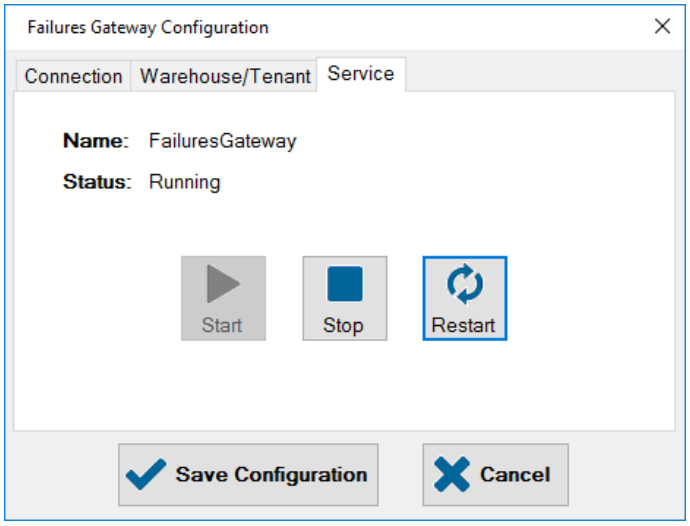

Figura 6.25.- Configuración Gateway - Servicio

7.- RESULTADOS

Primeramente, se realizaron pruebas individuales a cada una de las partes de las que se compone el proyecto.

Una vez desarrollado las distintas partes de las que se componen el proyecto se realizaron pruebas con todo el sistema completo.

Desde el programa de control que se ejecuta en las instalaciones (Galileo) se crearon y eliminaron averías en distintas maquinas. Cada vez que se creaba una avería esta se actualizaba correctamente en la base de datos y por consiguiente se notificaba en los móviles de los usuarios que cumplieran las reglas establecidas.

8.- CONCLUSIONES

Se ha conseguido realizar una aplicación de móvil (otro proyecto) que es capaz de notificar y mostrar las averías activas que hay en las instalaciones. No solo se podría montar en instalaciones propias de Mecalux con su sistema de control, sino que se podrá implantar en otras instalaciones que utilicen sus propios sistemas, simplemente implementando unas funciones específicas. Del mismo modo podrá utilizarse independientemente del sistema de control utilizado (PLC, PC industrial, etc.).

A falta de probar el sistema de notificaciones en una instalación real. El sistema ha funcionado correctamente a todas las pruebas sometidas por los cual las conclusiones acercad el proyecto son positivas.

9.- BIBLIOGRAFÍA

Documentación de logística y almacenes automáticos de Mecalux.

Manual del sistema de control Galileo (Mecalux).

Manual del programa Designer (Mecalux).

Firebase Cloud Messaging Protocolo HTTP - https://firebase.google.com/docs/cloud-messaging/http-server-ref?hl=es-419

Tutoriales de C# - https://msdn.microsoft.com/es-es/library/aa288436(v=vs.71).aspx

Servicios de notificaciones de inserción de Windows (WNS) - https://msdn.microsoft.com/es-es/windows/uwp/controls-and-patterns/tiles-and-notifications-windows-push-notification-services--wns--overview

Acceso a datos y modelado Entity Framework - https://msdn.microsoft.com/es-es/library/bb399572(v=vs.100).aspx

Ejemplos de escenario de Windows Communication Foundation (WCF) - http://msdn.microsoft.com/es-es/library/vstudio/ms751537(v=vs.90).aspx

Crear conexiones a bases de datos SQL Server - http://msdn.microsoft.com/es-es/library/s4yys16a.aspx