Universidad de Oviedo

Memoria del Trabajo Fin de Máster realizado por

ALBERTO ÁLVAREZ GARCÍA

para la obtención del título de

Máster en Ingeniería de Automatización e Informática Industrial

Mejora en el sistema de gestión y visualización de averías de Galileo (Mecalux)

FEBRERO 2015

INDICE DE ILUSTRACIONES

INDICE DE TABLAS

1. OBJETIVOS DEL PROYECTO

1.1. Finalidad

La finalidad del presente trabajo será el desarrollo de un sistema que permita gestionar y visualizar las averías disparadas por el sistema Galileo de forma independiente. Por un lado se dispondrá de un sistema de visualización de averías, implementado mediante una aplicación para terminales móviles, que con un simple registro y logeo, nos permita comprobar las averías producidas y que tiene asignadas el usuario. Por otro lado, se dispondrá de un sistema de gestión de información de usuario. Este sistema nos permitirá añadir usuarios y grupos de usuarios, y asignar a estos las diferentes averías que en caso de activarse tendrán que subsanar. Todo ello bajo unas condiciones temporales y de prioridad.

1.2. Alcance

Lo que se pretende en este proyecto es ampliar las funcionalidades del sistema Galileo – Designer IV desarrollado por Mecalux. A continuación se realizará una descripción explicativa de ambos sistemas y que carencias se pretenden subsanar con el desarrollo de este proyecto.

- **GALILEO IV**

Podemos considerar Galileo como una solución integral que se presenta como alternativa a los PLCs tradicionales. La principal característica de este sistema es que no se trata de un terminal físico concreto, sino que se trata de una herramienta software que corre bajo Windows y que realiza las labores de estos PLCs tradicionales. Razones de eficiencia, rentabilidad, versatilidad y hasta sofisticación hace que algunas empresas confíen en este softPLC desarrollado por Mecalux.

Este sistema de control funciona como un coordinador general. Se encarga del manejo de las máquinas, del control de los componentes hardware, el registro de posibles problemas o eventos, comunicaciones con el Sistema de Gestión del Almacén (SGA), etc. Engloba todas las tareas necesarias para recibir órdenes desde el SGA, enviar dichas órdenes a las diferentes máquinas, y así, mover los contenedores de la instalación.

❖ DESIGNER IV

Designer IV es la herramienta que se utiliza para desarrollar aplicaciones sobre Galileo IV. Esta herramienta cumple dos roles principales:

Servir de entorno integrado de desarrollo (IDE), de manera que desde la misma aplicación, el programador puede definir la lógica del programa, las variables a utilizar, crear las visualizaciones que los operarios utilizarán y además, utilizar las herramientas de depuración previstas, de manera que se pueda depurar y encontrar los errores del sistema.

Actuar como sistema SCADA, que se comunica con el sistema de control, y muestra la visualización creada por el programador, interactúa con los operarios y envía y recibe los datos hacia y desde el sistema de control.

Una característica importante para el desarrollo de este proyecto es la posibilidad que ofrece Designer para visualizar las averías. Para consultar esta característica tenemos que conectarnos remotamente al terminal que ejecuta Galileo y seleccionar la opción indicada.

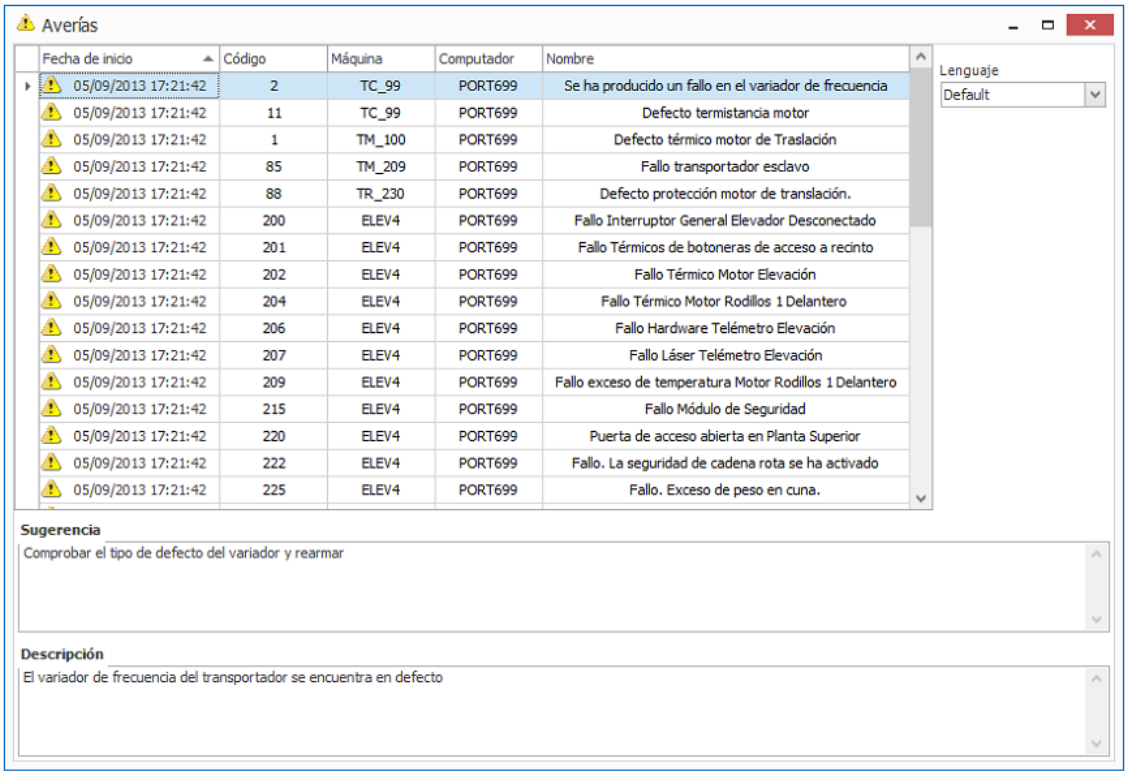


Ilustración 1: Panel de visualización de averías en Designer IV

Se puede ver en la imagen anterior el aspecto actual del visor de averías que incorpora Designer. La ventana mostrada puede funcionar como emergente cuando se active cualquiera de las averías definidas en la pantalla y también puede disparar una alarma sonora. La información que se muestra de las averías es variada. Los datos principales son la fecha de inicio de avería, el código de avería, la máquina y el computador donde se producen y el nombre de la avería. Como datos adicionales, se muestra una descripción detallada de la avería y una sugerencia para solucionar dicha incidencia. También es posible indicar el lenguaje en que se muestra la información.

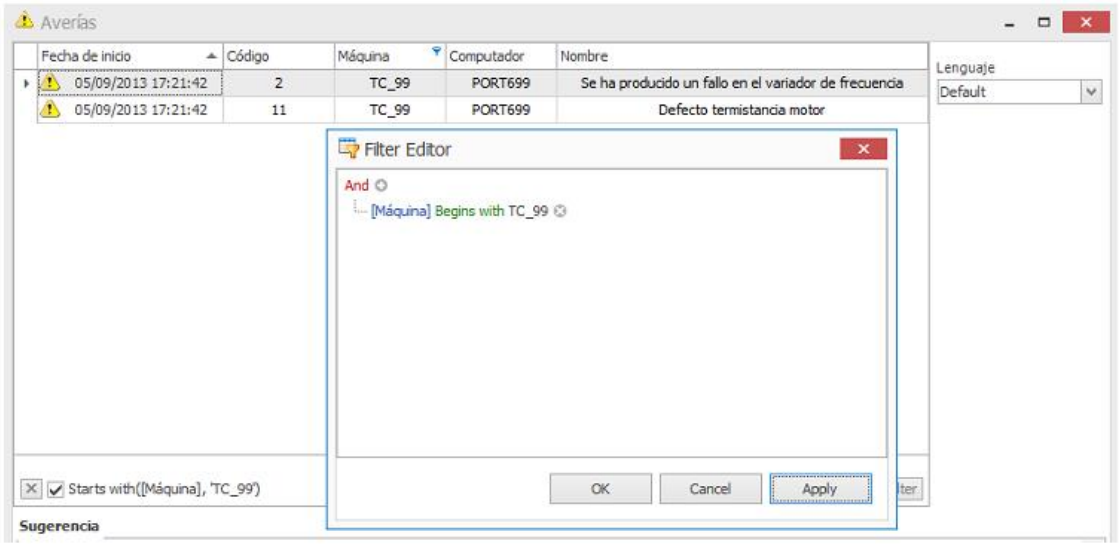


Ilustración 2: Filtrado de averías en Designer IV

Otra de las opciones que nos presenta el visor es la posibilidad de aplicar filtros a la tabla de averías y ver por ejemplo solamente las averías de una máquina o computador.

A parte del visor de averías otro punto importante de Designer+Galileo, es la gestión de usuarios y grupos.

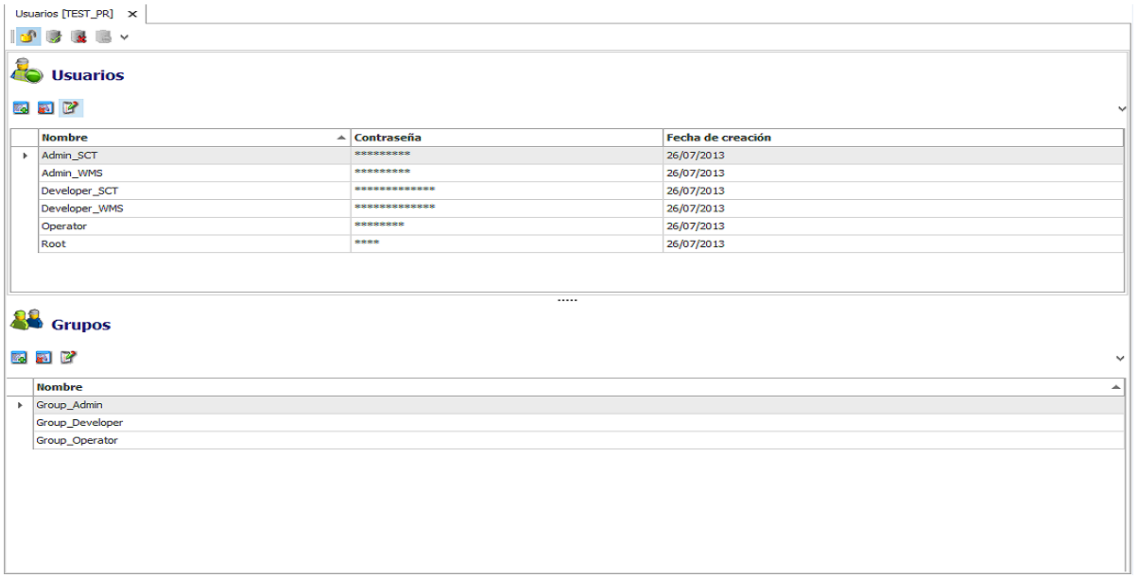


Ilustración 3: Herramienta de gestión de usuarios y grupos en Designer IV

El objetivo de distinguir entre usuarios y grupos dentro del sistema es el que establecer una serie de funcionalidades que, para cada grupo creado, le permita limitar el uso de ciertas acciones sobre el sistema de control de Galileo IV a través del entorno de desarrollo. Estas limitaciones pueden servir para evitar que una manipulación accidental del sistema produzca daños de importancia en la instalación por parte de un usuario inexperto.

Este tipo de medidas será fundamental en el desarrollo del proyecto, pues se pretende exportar dicha arquitectura a nuestro sistema, es decir, en vez de limitar el uso de funciones, se pretenderá restringir la notificación de averías a según qué tipo de usuario.

1.3. Alternativas de programación

Para el desarrollo del proyecto se han planteado una serie de tecnologías válidas todas ellas para la implantación del sistema. A continuación se explicarán las características de cada uno y se facilitará un resumen explicativo al final del apartado en el que se detallará el porqué del uso de cada una y las ventajas que ofrecen.

Debido a que la arquitectura del proyecto se corresponde con la clásica cliente-servidor o con un RFC(Remote Procedure Call), lo primero que se ha planteado introducir son las soluciones para el desarrollo de este tipo de arquitectura. Como la plataforma en la que se desarrollará el proyecto es Windows, debido a su gran implantación en el sector industrial, se buscarán aquellas tecnologías compatibles con dicho sistema.

Se han agrupado las tecnologías planteadas en función de su objetivo, por lo que se indicará en los siguientes apartados sus principales características.

1.3.1. Comunicaciones cliente – servidor

- **WINDOWS COMMUNICATION FOUNDATION**

Windows Communication Foundation (WCF) es un marco de trabajo desarrollado por Microsoft orientado a la creación de aplicaciones orientadas a servicios. Esta tecnología permite enviar datos como mensajes asincrónicos de un extremo del servicio a otro. Uno de estos extremos puede formar parte de un servicio disponible continuamente hospedado por IIS, o puede ser un servicio hospedado en una aplicación.

Está orientado a trabajar con estructuras cliente-servidor, de modo que el cliente pueda solicitar datos al servidor de forma muy sencilla. Estos datos o mensajes pueden ser tan simples como un carácter o una palabra que se envía como un XML, o tan complejos como una secuencia de datos binarios.

Entre los diferentes escenarios en los que es factible su implementación destacan los siguientes:

- Un servicio seguro para procesar transacciones comerciales.
- Un servicio que proporciona datos actualizados a otras personas, como un informe sobre tráfico u otro servicio de supervisión.
- Un servicio de chat que permite a dos personas comunicarte o intercambiar datos en tiempo real.
- Una aplicación panel que sondea los datos de uno o varios servicios y los muestra en una presentación lógica.
- Una aplicación de Silverlight para sondear un servicio en busca de las fuentes de datos más recientes.

Es cierto que el desarrollo de este tipo de aplicaciones ya era posible antes de la aparición de WCF, pero con WCF el desarrollo de extremos resulta más sencillo que nunca ya que proporciona un enfoque muy manejable para la creación de servicios web y clientes de servicios web.

- **Características:**

- **Orientación a servicios**: La arquitectura orientada a servicios es el uso de servicios web para enviar y recibir datos. Los servicios tienen la ventaja general de estar débilmente acoplados entre una aplicación y otra en lugar de incluidos en el código, de forma que cualquier cliente creado en cualquier plataforma pueda conectar con cualquier servicio siempre y cuando se cumplan los contratos esenciales.

- **Interoperabilidad**: implementa los últimos estándares del sector para la interoperabilidad de servicios web.

- **Varios modelos de mensajes**: El más común es el de solicitud/respuesta, en que un extremo solicita datos de otro extremo, y el otro extremo responde. Existen otros modelos como el mensaje unidireccional, en que un único extremo envía un mensaje sin esperar ninguna respuesta. Un modelo más complejo es el modelo de intercambio dúplex donde dos extremos establecen una conexión y envían datos hacia delante y hacia atrás, similar a un programa de mensajería instantánea.

- **Contratos de datos**: Consiste en definir clases que representan un tipo de datos. WCF genera automáticamente los metadatos que permiten a los clientes ajustarse a los tipos de datos que se han diseñado.

- **Seguridad**: Es posible cifrar los mensajes para proteger la privacidad, así como obligar a los usuarios a que se autentiquen antes de recibir mensajes. La seguridad puede implementarse utilizando estándares conocidos como SSL.

- Varios **transportes y codificaciones**: Los mensajes entre cliente y servicio pueden enviarse con cualquiera de los protocolos y codificaciones integrados. La combinación más frecuente consiste en el envío de mensajes SOAP utilizando el protocolo HTTP. WCF también permite enviar mensajes sobre TCP.

- **Mensajes confiables y en cola**: WCF permite en envío de mensajes confiables usando las sesiones confiables implementadas.

- **Mensajes duraderos**.

- **Integración** con otras **tecnologías de Microsoft**.

- **Arquitectura de WCF:**

El gráfico siguiente muestra las capas principales de la arquitectura de WCF.

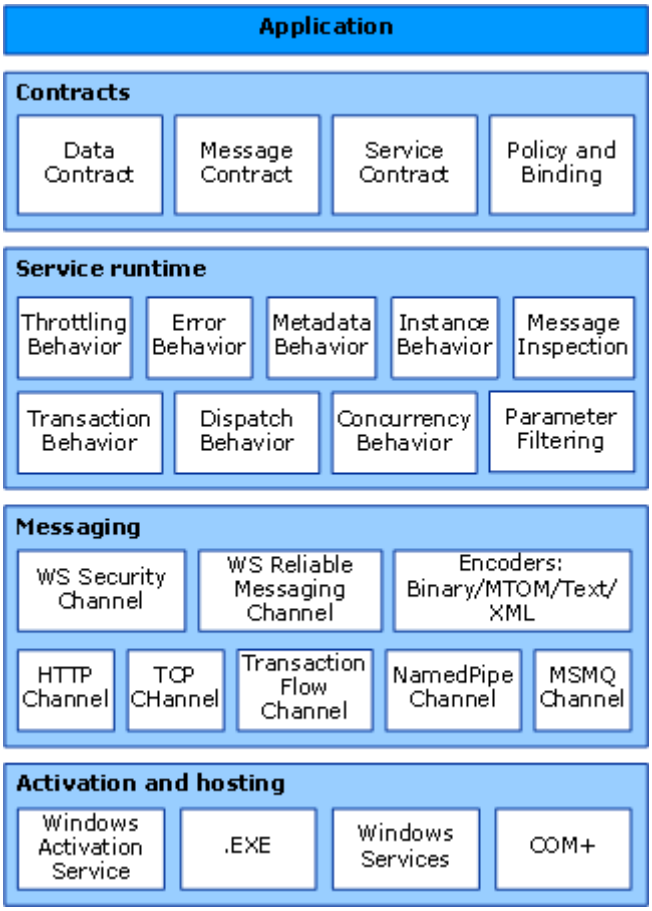


Ilustración 4: Arquitectura WCF

- **SIGNALR**

SignalR es una librería para los desarrolladores de ASP.NET que simplifica el proceso de adición de funcionalidades de tiempo real a nuestras aplicaciones web. Esta funcionalidad de tiempo real consiste en dotar a nuestra aplicación servidora de la habilidad de enviar datos a los clientes conectados, tan pronto como estén disponibles. Esto permite que los clientes no estén a la espera cada vez que quieran nuevos datos.

Con SignalR nuestra aplicación cliente no necesitará refrescar la página para mostrar los nuevos datos, sino que estos se mostraran automáticamente.

SignalR proporciona una sencilla API para realizar llamadas a los clientes desde el servidor (Remote Procedure Call) que llaman a funciones de Javascript implementadas en dicho cliente (también es posible en clientes de otras plataformas). También incluye una API para el manejo y administración de las conexiones entre cliente y servidor (eventos de conexión, desconexión y reconexión) y entre grupos y servidor.

El proceso de llamadas entre cliente y servidor y viceversa se describe en la imagen siguiente.

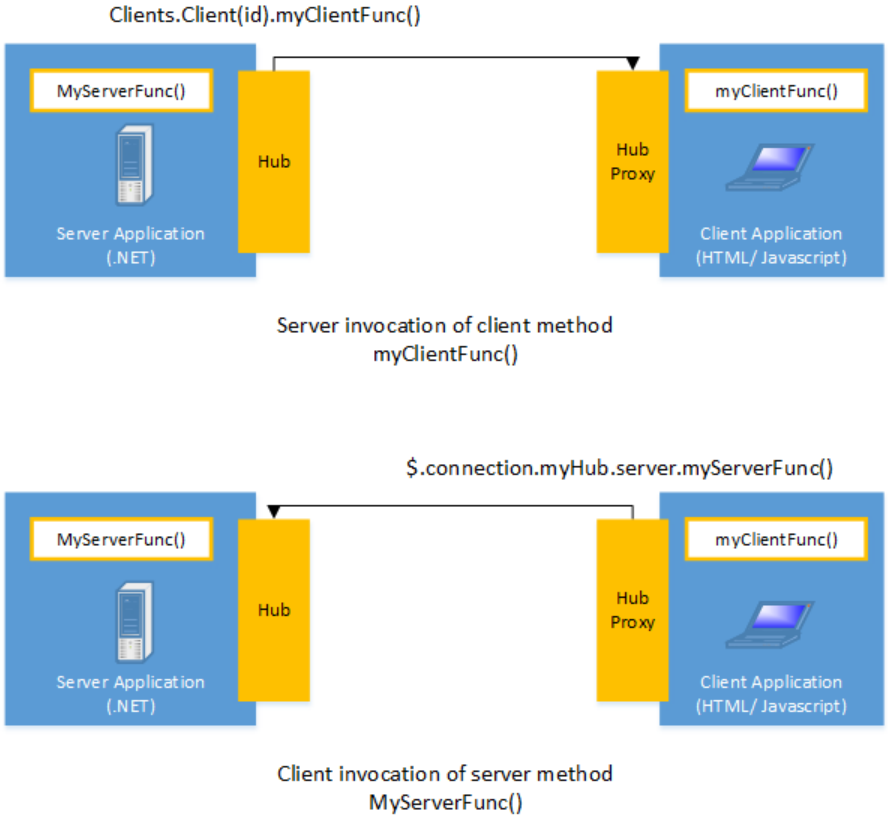


Ilustración 5: Esquema funcionamiento SignalR

SignalR permite manejar de forma automática la conexión y permite enviar mensajes broadcast a todos los clientes conectados. También a clientes específicos.

Diagrama de la arquitectura de SignalR:

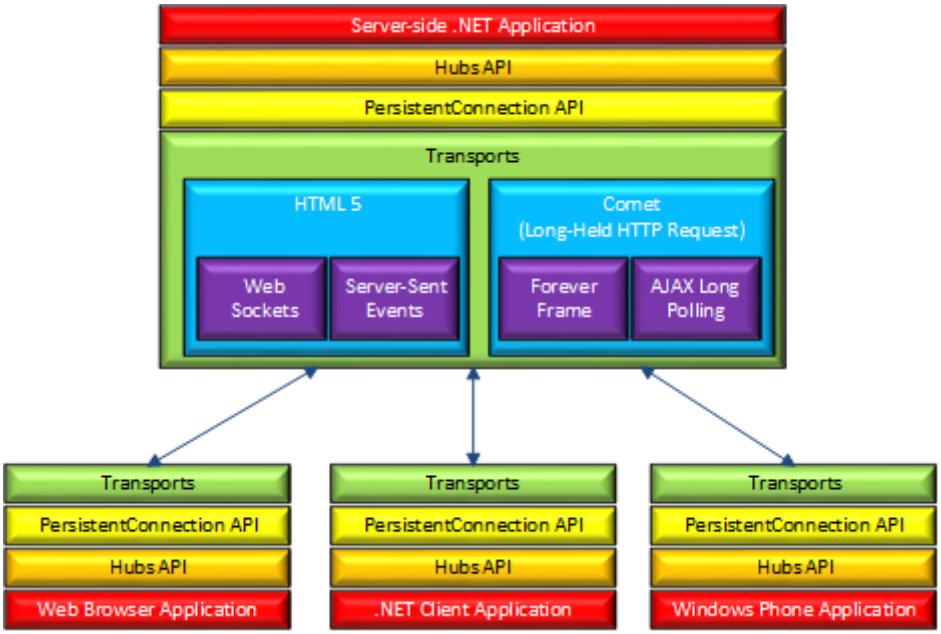


Ilustración 6: Arquitectura SignalR

1.3.1.1.Resumen de características

En este apartado se resumirán las características principales de las técnologías presentadas en el apartado anterior y el porqué de la elección final.

	SIGNALR
VENTAJAS	- Características orientadas a servicios de tiempo real
	- Menor número de peticiones entre cliente y servidor
	- Arquitectura similar a publicación-suscripción
	- Posibilidad de conectar clientes basados en web
INCONVENIENTES	- Tecnología reciente
	- Cierta complejidad

Tabla 1: Resumen de caracteristicas SignalR

	Windows Communication Foundation
VENTAJAS	- Ampliamente implantado
	- Sencillez de uso
	- Compatibilidad con productos Microsoft
	- Posibilidades de configuración
INCONVENIENTES	- Aunque es posible, el envío de eventos es una tarea complicada.
	- Conexión con clientes Web complicada.
	- Elevado número de peticiones para el intercambio de información.

Tabla 2: Resumen de caracteristicas WCF

Dado que se pretende centralizar en un servicio el uso de dos clientes diferentes (uno Windows y otro web) se tendrá que apostar por una tecnología que disponga de herramientas para desarrollarlos, con sencillez si es posible. Además, debido a que uno de los clientes del servicio será una App móvil, será indispensable que el número de peticiones entre cliente y servidor (sobrecarga de la red) sea lo menor posible, ya que lo más normal es que estas se realicen bajo conexiones móviles como el 3G, siendo este tipo de conexión limitado en ancho de banda y prestaciones.

Debido a lo expuesto en este apartado, SignalR será la opción elegida ya que aúna las principales características deseadas. Por un lado es capaz de enviar datos desde el servidor al cliente sin peticiones por parte de este último, básico si se trabaja con conexiones de limitado ancho de banda, y si se desean realizar peticiones puntuales también existe esa posibilidad sin muchas complicaciones. Además está diseñada para trabajar con clientes web utilizando librerías de Jquery y Javascript, optimizando el transporte de los datos siempre que sea posible.

1.3.2. Acceso a datos

Para el acceso a datos solo se ha considerado un framework en concreto. En este caso se trata de Entity Framework (Microsoft), un framework dedicado al acceso a datos y que nos permite trabajar tanto con WCF como con SignalR. Su principal característica es que aísla al programador de las particularidades de cada una de las bases de datos que está utilizando, permitiéndole centrarse en las entidades (tablas) que maneja y convierte dichos datos en un objeto más del sistema. A continuación se amplían las características y se muestra un diagrama a modo explicativo de la arquitectura del framework.

❖ **ENTITY FRAMEWORK**:

Entity Framework es un conjunto de tecnologías o framework de ADO.NET que permiten el desarrollo de aplicaciones de acceso a datos. Los arquitectos y programadores de aplicaciones orientadas a datos se han enfrentado a la necesidad de lograr dos objetivos muy diferentes. Deben modelar las entidades, las relaciones y la lógica de los problemas que resuelven y también trabajar con los motores de datos que se usan para almacenar y recuperar los datos. Los datos pueden abarcar varios sistemas de almacenamiento, cada uno con sus propios protocolos.

Entity Framework permite a los desarrolladores trabajar con datos en forma de objetos y propiedades específicos del dominio, como clientes y direcciones de cliente, sin tener que preocuparse por las tablas y columnas de la base de datos donde se almacenan. Con Entity Framework, los desarrolladores pueden trabajar en un nivel mayor de abstracción cuando tratan con datos y pueden crear y mantener aplicaciones orientadas a datos con menos código que las aplicaciones tradicionales.

Al ser Entity Framework un componente de .NET Framework, las aplicaciones de Entity Framework se pueden ejecutar en cualquier equipo en el que esté instalado .net 3.5 SP1.

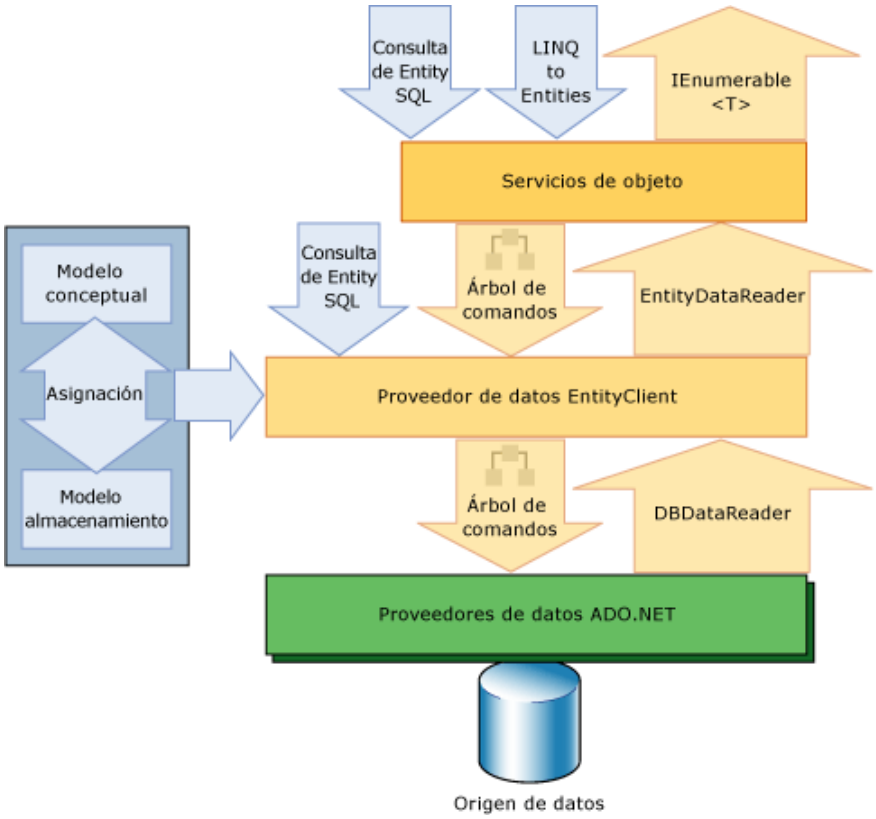


Ilustración 7: Arquitectura Entity Framework

1.3.3. Aplicaciones Web multiplataforma

Para el desarrollo del cliente móvil es necesario emplear una herramienta o framework que permita el acceso a alguna de las principales características que ofrecen los terminales (smartphones) actuales como son las notificaciones, alarmas, avisos, etc., y que posibiliten que el usuario pueda recibir en todo momento avisos sin necesidad de tener la aplicación abierta. Otra de las características deseadas para nuestro cliente móvil es la universalidad, es decir, que esté disponible para el mayor número de plataformas. Esto conlleva un gran problema, y es el número de lenguajes de programación que deben ser conocidos para el desarrollo en cada plataforma. Debido a esto se ha planteado el desarrollo de una aplicación híbrida, desarrollada con tecnologías web como HTML, Javascript y CSS.

Atendiendo a lo anterior y realizando una búsqueda dentro del abanico de opciones que existen, el framework que reúne todas estas características es Phonegap o Apache Cordova. A continuación se mostrará una descripción de las opciones y características que ofrece.

- **PHONEGAP**

PhoneGap es un *framework* para el desarrollo de aplicaciones móviles. Principalmente, PhoneGap permite a los programadores desarrollar aplicaciones para dispositivos móviles utilizando herramientas genéricas tales como JavaScript, HTML5 y CSS3, es decir, permite partir de una aplicación web estándar y adaptarla a los distintos dispositivos móviles existentes. Las aplicaciones resultantes son híbridas, es decir que no son realmente aplicaciones nativas al dispositivo (ya que el renderizado se realiza mediante vistas web y no con interfaces gráficas específicas de cada sistema), pero no se tratan tampoco de aplicaciones web (teniendo en cuenta que son aplicaciones que son empaquetadas para poder ser desplegadas en el dispositivo incluso trabajando con el API del sistema nativo).

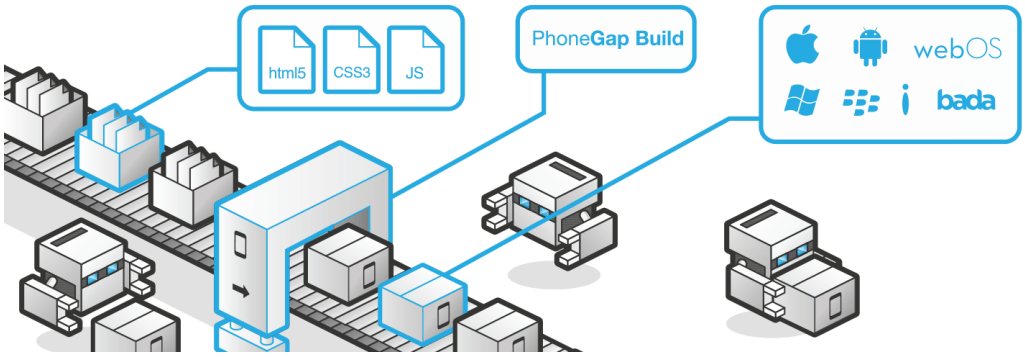


Ilustración 8: Esquema funcionamiento Phonegap

En la tercera versión de PhoneGap se incorpora el uso de una interfaz de comandos a través de consola, una nueva arquitectura de complementos descentralizados y la posibilidad de utilizar un código web unificado para crear múltiples proyectos.

PhoneGap maneja API que permiten tener acceso a elementos como el acelerómetro, la cámara, los contactos en el dispositivo, la red, el almacenamiento, las notificaciones, etc. Estas API se conectan al sistema operativo usando el código nativo del sistema huésped a través de una Interfaz de funciones foráneas en Javascript.

PhoneGap permite el desarrollo ya sea ejecutando las aplicaciones en nuestro navegador web, sin tener que utilizar un simulador dedicado a esta tarea, y brinda la posibilidad de soportar funciones sobre frameworks como Sencha Touch o Jquery Mobile.

PhoneGap es una distribución de **Apache Cordova**. La aplicación se llamó en un principio "PhoneGap", y posteriormente "Apache Callback". Ambos sistemas tienen funciones casi idénticas, la diferencia principal entre Apache Cordova y Phonegap es que el segundo tiene acceso a servicios de compilación en la nube proporcionados por Adobe Creative Cloud.

Apache Cordova es un software de código abierto y tanto este como PhoneGap pueden ser utilizados libremente en cualquier aplicación sin necesidad de atribución o licencias de ningún tipo.

1.4. Descripción del sistema

Este producto pretende cubrir una carencia con la que cuenta el sistema Galileo en su versión actual. Se trata de permitir la monitorización del sistema desde un terminal móvil, debido a que actualmente la visualización de las incidencias se encuentra integrada en la planta y no existen posibilidades de supervisión fuera del almacén.

Podemos decir entonces, que el sistema se compone de dos partes diferenciadas. Por un lado, el terminal móvil de visualización y por otro, un gestor de usuarios y grupos que permitirá asignar determinadas incidencias y prioridades a estos usuarios y grupos.

Una de las similitudes de las partes del sistema es que ambas se encuadran dentro de una arquitectura cliente – servidor. Se tendrá una aplicación servidora que periódicamente comprobará las actualizaciones en la base de datos de Galileo y con ello se actuará en consecuencia, es decir, se enviarán los datos a la aplicación cliente correspondiente.

En los apartados siguientes se describirá con mayor detalle el funcionamiento del sistema.

1.4.1. Sistema de visualización de incidencias:

Una vez conocidas las bondades de las plataformas móviles, se pretende aprovechar dichas características para diseñar un sistema en el que los usuarios encargados de supervisar y gestionar las incidencias del almacén puedan recibir alertas en todo momento con información exhaustiva de la avería. Para ello se ha ideado un sistema como el que se muestra en la ilustración siguiente.

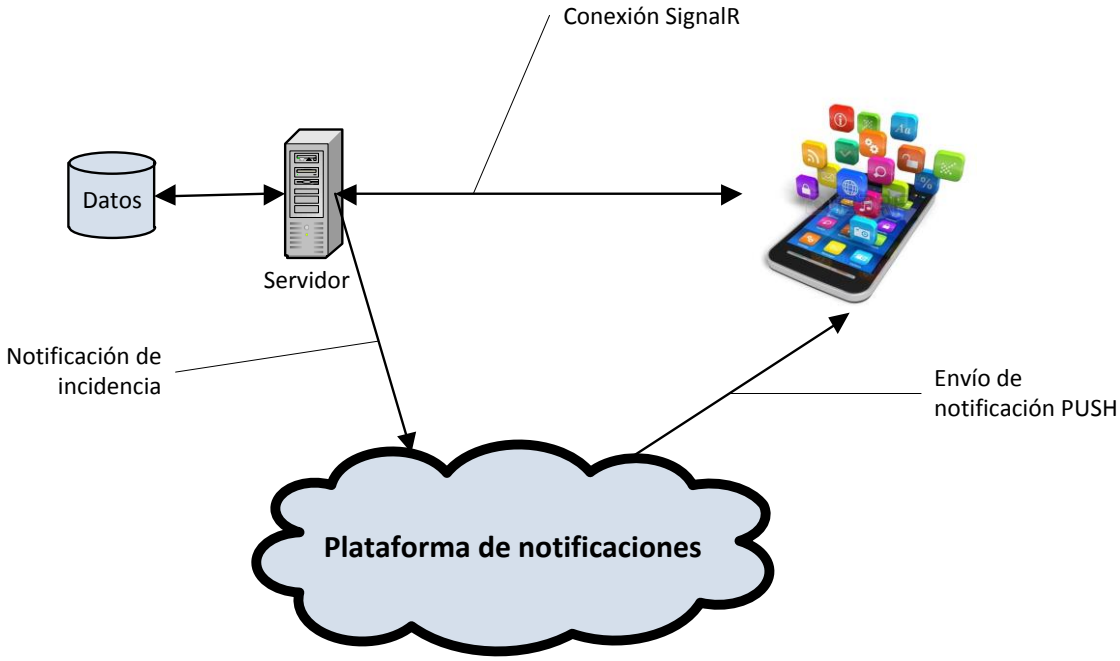


Ilustración 9: Sistema de visualización de incidencias

El eje del sistema será una aplicación web que gestionará la base de datos donde se almacenan las averías disparadas por el softPLC Galileo. Dicha aplicación tendrá que comprobar periódicamente la base de datos, para detectar la aparición de nuevas incidencias. Si se producen nuevas averías, el servicio web consultará qué usuarios o grupos tienen asignada ese tipo de avería. Una vez llegado a este punto, se consultará las reglas temporales y de prioridad, para notificar la avería al usuario o grupo indicado.

Existen dos formas de notificar la avería al usuario. Dependerá del uso que se esté haciendo de la aplicación. Si el usuario está ejecutando la aplicación, se actualizará en tiempo real el número de averías. Esta característica es debida a las posibilidades que nos ofrece SignalR. El usuario no tendrá que realizar continuas peticiones al servicio para comprobar nuevas averías, sino que el servicio será el que las notifique sin petición alguna.

La otra posibilidad de notificación se produce cuando la aplicación no esté activa. En este caso, se aprovechará una de las características que nos ofrecen las nuevas plataformas móviles, las notificaciones PUSH. Con ello, se podrán enviar alertas a los dispositivos sin que estos estén conectados directamente al servicio web. Se utilizarán las diferentes plataformas que nos ofrecen los desarrolladores (Microsoft, Apple, Google, etc). La forma de funcionamiento de las principales plataformas de notificaciones (Windows Phone, Android, iOS) es similar, por lo que la descripción de una será válida para el resto.

- **Descripción del sistema de notificaciones de Microsoft (Microsoft Push Notification Service):**

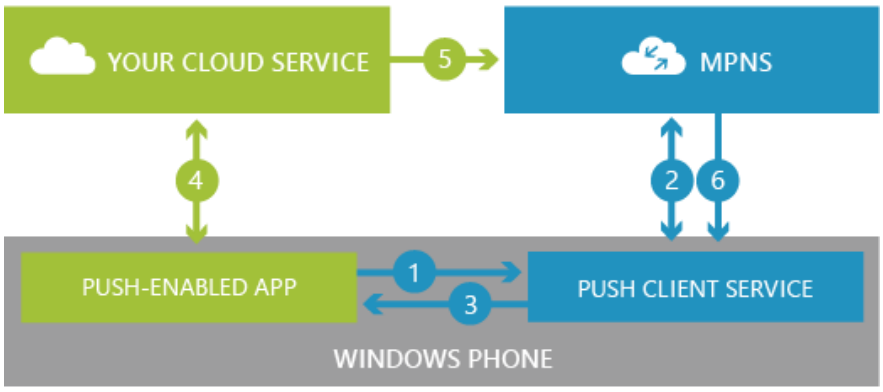


Ilustración 10: Plataforma de notificaciones Windows Phone 8

En el caso de trabajar sobre una plataforma de Windows Phone 8 el funcionamiento del sistema de notificaciones es el siguiente:

1. La aplicación (Mecalux failures) solicita una URI para las notificaciones push al cliente de notificaciones del sistema operativo.

2. El cliente de notificaciones negocia con el Servicio de Notificaciones Push de Microsoft (MPNS) y el MPNS retorna una URI para las notificaciones a nuestro cliente.

3. El cliente de notificaciones envía a la aplicación (Mecalux Failures) la URI obtenida.

4. La aplicación deberá enviar al servicio Web SignalR dicha URI recibida. Esta se almacenará en una base de datos.

5. Cuando el servicio Web tiene información que enviar a la aplicación (Mecalux Failures), tendrá que usar la URI para enviar un mensaje con la codificación pertinente y la información que se quiera notificar al servicio MPNS.

6. Una vez el MPNS haya recibido la notificación procederá a enviarla a la aplicación.

Dependiendo del formato de la notificación en el terminal se mostrará un tipo de notificación u otro. Se ha planteado el uso de notificaciones "toast" (mensaje de alerta). Además, una vez la notificación haya sido recibida por la aplicación, MPNS enviará un código de respuesta al servicio web indicando si la notificación ha sido recibida o no.

1.4.2. Sistema de gestión de usuarios y grupos

El sistema de gestión de usuarios y grupos, consistirá en una herramienta que permita al usuario administrador o gestor del personal del almacén, tener la capacidad de distribuir la asignación de averías entre los diferentes trabajadores de la plantilla y también la posibilidad de agrupar dichos trabajadores en grupos más especializados. Se consigue así tener cierto grado de aislamiento entre trabajadores con el objetivo de asignar averías acordes a su cualificación. Además se proporcionarán herramientas para añadir trabajadores al sistema (con sus respectivos datos), modificarlos y hasta eliminarlos.

El objetivo de la asignación de averías es que se notifiquen dichas averías a usuarios con conocimientos reales y con experiencia previa en el arreglo de las máquinas dañadas.

Para realizar esta distribución de averías se aplicarán una serie de reglas de asignación es decir, se vinculará a un determinado usuario las averías que correspondan y además se indicará el periodo temporal en el que estará disponible para solventar dichas incidencias. Otro parámetro que se tendrá en cuenta es el de prioridad, pues no se notificarán a todos los usuarios a la vez, sino que unos tendrán mayor prioridad que otros a la hora de recibir las notificaciones, con lo que se asegura que si un usuario no puede atender una incidencia el sistema la comunique al de siguiente prioridad y sucesivas.

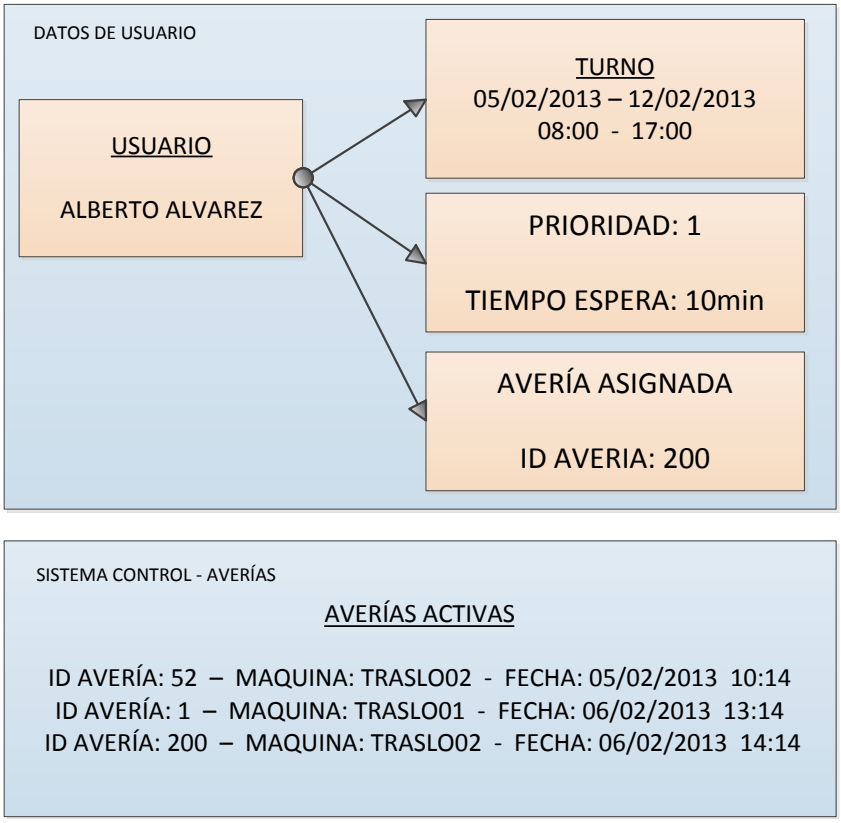


Ilustración 11: Ejemplo de datos de usuario y averías activas

Si se disponen los datos de la ilustración anterior, se observa que el usuario tiene una única avería asignada, en este caso la de ID 200. Al tener asignado el turno de 08:00 a 17:00 todas las averías con identificador 200 que se disparen en ese horario la semana del 5 al 12 de Febrero se notificarán a dicho usuario. La prioridad 1 indica que tiene preferencia sobre otros usuarios, por lo que será el primero en recibir la notificación de incidencia. En caso de que no confirme la recepción del mensaje, la incidencia pasará a notificarse a los usuarios con prioridad 2, y así sucesivamente hasta la confirmación por parte de alguno de que la avería va a proceder a subsanarse.

En cuanto al tiempo de espera, indicará el tiempo que ha de esperar el sistema para notificar la avería a un usuario, con el objetivo de evitar que se envíen notificaciones a varios usuarios a la vez y de que se sobrecargue el sistema.

La arquitectura del sistema descrito anteriormente será la siguiente. Tendrá como parte central una aplicación Web basada en la API de ASP.NET SignalR. Se trata de una tecnología orientada a la llamada remota de procesos o RPC, cuya principal característica, diferenciadora del resto de tecnologías RPC, es que ofrece una serie de herramientas para el envío de datos en tiempo real, sin necesidad de peticiones por parte del cliente. Esta aplicación será la encargada de gestionar y consultar la base de datos de todos nuestros usuarios y grupos. A través de la aplicación cliente, ejecutada en cualquier entorno Windows, el usuario podrá realizar cambios en la

base de datos, crear o modificar usuarios/grupos, asignarles averías y establecer una serie de reglas para su notificación en caso de activarse.

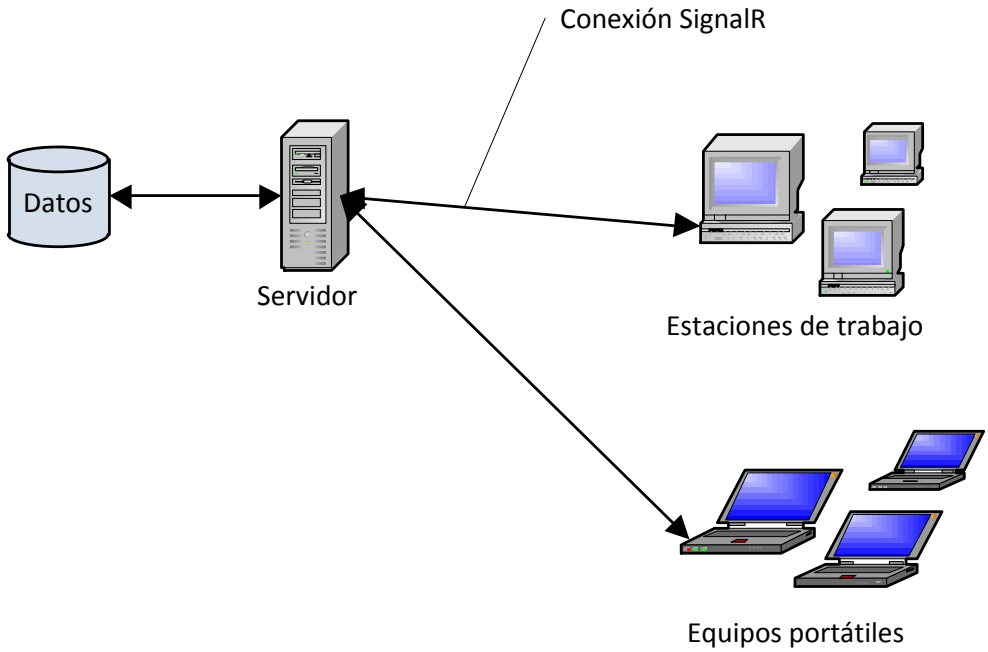


Ilustración 12: Diagrama arquitectura sistema gestión usuarios/grupos

1.4.3. Usuarios del producto

Los posibles usuarios del sistema Mecalux Failures Manager se pueden dividir, a grandes rasgos en dos categorías: el personal administrador o encargado de gestionar la asignación de averías, y el personal de mantenimiento o el encargado de gestionar las incidencias del almacén.

- En el caso del **personal administrador** y gestor de averías, la herramienta les proporciona un medio de fácil utilización para gestionar los usuarios que utilizarán el sistema y asignarles en pocos pasos aquellas averías que tendrán que supervisar, así como los grupos a los que pertenecerán estos usuarios, y el periodo de supervisión que tendrán asignado. Todo ello sin necesidad de utilizar el programa de control del almacén.

- El otro tipo de usuarios, serían los pertenecientes al **personal mantenimiento/incidencias**, es decir, los operarios o encargados de almacén. Será aquel personal cualificado para subsanar o gestionar las diferentes averías o incidencias que puedan producirse en el almacén.

1.4.3.1.Necesidades de los usuarios

Analizadas las carencias del sistema Galileo, actualmente existen varias necesidades que este sistema pretende resolver, a saber:

- Proporcionar una **herramienta liviana**, portable e **independiente del programa de control/diseño** (Designer 4). Esto facilita que al usuario acceder al sistema de gestión de usuarios desde cualquier ordenador sin necesidad de instalar el complejo programa de control.

- Un **sistema de visualización** de incidencias **independiente** del programa de control/diseño. Dado que el sistema de visualización se desarrollará en un terminal móvil, ampliamente implantado en la sociedad, permitirá que no se tenga que hacer uso de ningún computador o programa de cierta complejidad.

- El **sistema de visualización** debe tener **disponibilidad** permanente para recibir las **notificaciones de averías**. Si el sistema de visualización está implementado dentro del programa de control, los usuarios no podrán conocer las incidencias del sistema hasta que se conecten a él. Este sistema soluciona esta carencia.

- Se debe establecer un **sistema de prioridades en la asignación de averías**. Si un usuario no tiene disponibilidad en un momento determinado, se debe comunicar la incidencia a otro usuario disponible.

1.4.3.2. Resumen de capacidades:

En la siguiente tabla se resumen las principales características del sistema en términos de beneficio y características del mismo.

Beneficios para el usuario	Característica que lo proporciona
Independencia del programa de control	Se dispone de una herramienta liviana capaz de ejecutarse en cualquier ordenador.
Sistema de visualización independiente del programa de control	Sistema desarrollado sobre plataforma móvil.
El sistema de visualización tiene disponibilidad permanente para recibir notificaciones de incidencias.	Los sistemas de notificaciones que implementan las plataformas móviles.
Se establece un sistema de prioridades en la asignación de averías.	El sistema es capaz de reasignar una avería a otro usuario cuando el actual se encuentra realizando otra tarea.

Tabla 3: Necesidades de los usuarios

1.4.4. Entorno de implantación

- **Entorno hardware**

Debido a que el producto consta de varias aplicaciones, resumiremos en una tabla las características hardware mínimas de las que se debe disponer en cada caso:

Servicio Web	Administrador usuarios	App Móvil
Procesador Intel o equivalente con velocidad recomendada superior a 1GHz	Procesador Intel o equivalente con velocidad recomendada superior a 1GHz	Terminal Android con WiFi y conexión al menos 3G
Monitor color al menos 15", teclado y ratón.	Monitor color al menos 15", teclado y ratón.	Terminal iPhone con Wifi y conexión al menos 3G
128 MB de memoria RAM	128 MB de memoria RAM	Terminal Windows Phone con Wifi y conexión al menos 3G
50 MB de espacio libre en el disco duro	50 MB de espacio libre en el disco duro	10 MB de espacio libre en la memoria del dispositivo.

Tabla 4: Entorno hardware de implantación

- **Entorno software**

Debido a que el software desarrollado en este proyecto consta de varias partes ejecutadas sobre diferentes plataformas, se indicará en una tabla las aquellas sobre las que se pueden ejecutar dichas aplicaciones:

Servicio Web	**Administrador usuarios**	**App Móvil**
Microsoft Windows Xp	Microsoft Windows Xp	Android 4.xx y superiores
Microsoft Windows 7	Microsoft Windows 7	iOS 4 y superiores
Microsoft Windows 8	Microsoft Windows 8	Windows Phone 8 y 8.1

Tabla 5: Entorno software de implantación

ANÁLISIS
Y
DISEÑO

2. ESPECIFICACIÓN DE LOS CASOS DE USO

2.1. Actores

Un actor define un conjunto coherente de roles que los usuarios del sistema interpretan cuando interactúan con el mismo. Un usuario puede ser un individuo o un sistema externo.

A continuación se enumeran los actores que participan en el sistema así como una breve descripción de ellos.

2.1.1. Usuario

Representa a la persona que interacciona directamente con el sistema. Concretamente se trata de la persona que visualiza las incidencias del almacén y las acusa.

2.1.2 Usuario administrador

Representa a la persona que interactúa con el sistema gestor de usuarios y grupos. Es la persona que asigna las averías a los diferentes usuarios y la que planifica las reglas de asignación y prioridades.

2.1.3. Base de datos

Representa el conjunto de datos de los que se servirá el sistema para su funcionamiento.

3. CASOS DE USO

A continuación se enumeran los casos de uso que se han identificado en el sistema:

- Gestión de usuarios
- Gestión de grupos
- Asignación de averías
- Gestión de reglas
- Visualizar averías activas
- Confirmar averías

- Filtrar averías

Cada uno de los casos de uso enumerados anteriormente se divide en las siguientes partes que se indican a continuación:

- Descripción
- Flujo de eventos
- Precondiciones
- Postcondiciones

3.1. Diagrama de Casos de Uso

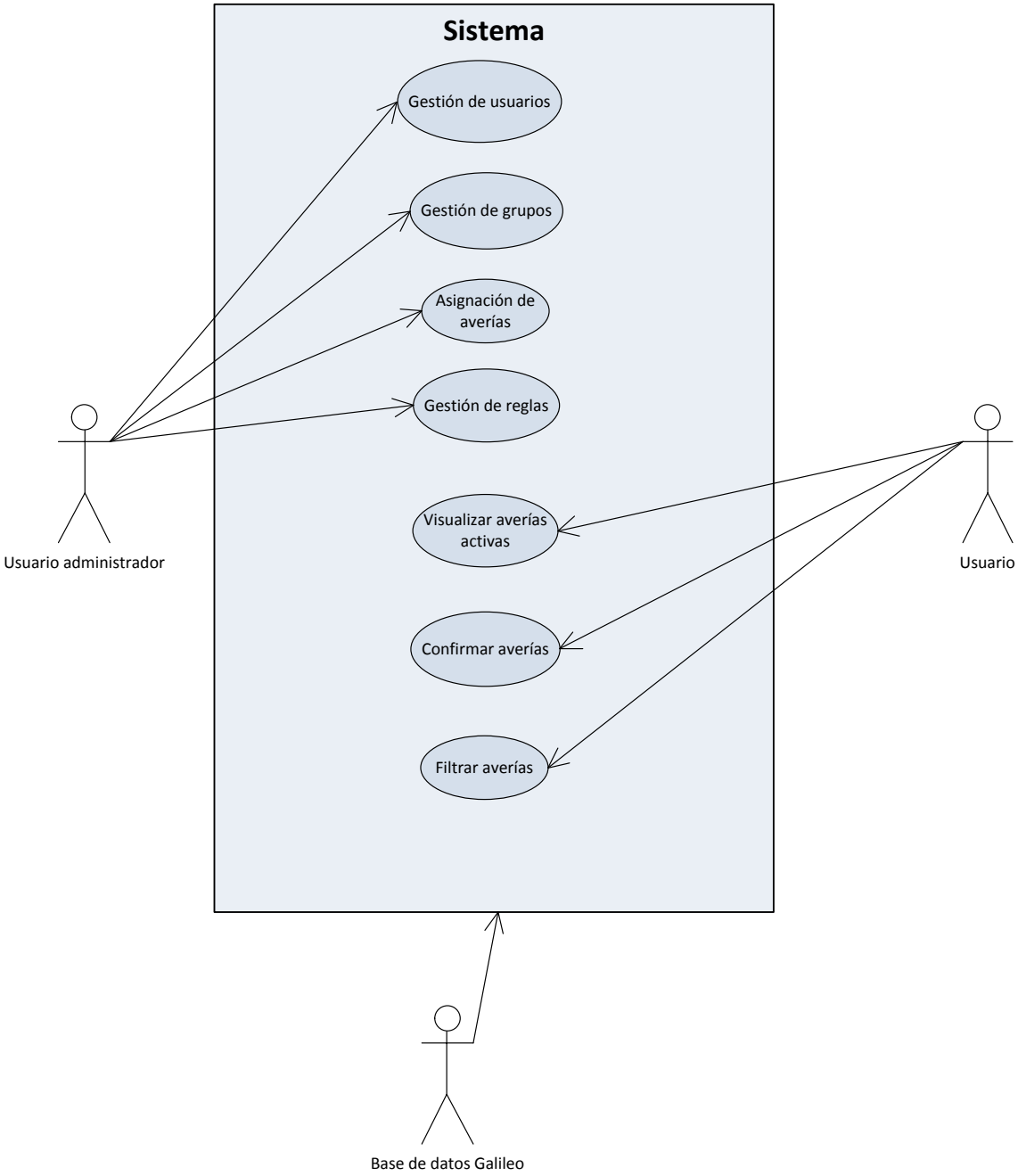


Ilustración 13: Diagrama de casos de uso

3.2. Caso de Uso “Gestión de usuarios”

3.2.1. Descripción

Este caso de uso representa la interacción entre el usuario administrador y el sistema cuando el primero desea realizar las tareas habituales de gestión de usuarios, es decir: añadir usuario, borrar usuario y modificar usuarios existentes.

3.2.2. Flujo de Eventos

El caso de uso comienza cuando el usuario se dispone a realizar la gestión de un usuario, es decir, cuando selecciona alguna de las siguientes opciones: “+”, “-“, “Save”, o selecciona el checkbox asociado a cada usuario en el menú principal.

3.2.2.1. Flujo básico – Gestión usuarios

1. Si el usuario administrador selecciona “+” se muestra un diálogo en el que tendrá que introducir la información de usuario solicitada.

2. Si el usuario administrador selecciona “-“ se borrará aquellos usuarios que estén seleccionados.

3. Si el usuario administrador selecciona a un usuario (selección del checkbox asociado), se carga la información de dicho usuario en la pantalla principal, pudiendo modificar todos los campos activos.

4. Si el usuario administrador selecciona “Save” guardará todos aquellos cambios que se hayan realizado sobre la información del usuario seleccionado.

3.2.2.2. Flujos alternativos

Será necesario añadir flujos alternativos para controlar los siguientes sucesos:

- Errores en la lectura de la base de datos.
- Errores de escritura en la base de datos.
- Errores de formato en los campos de información de usuario.

3.2.3. Precondiciones

No se han determinado.

3.2.4. Postcondiciones

No se han determinado.

3.2.5. Diagrama de actividad

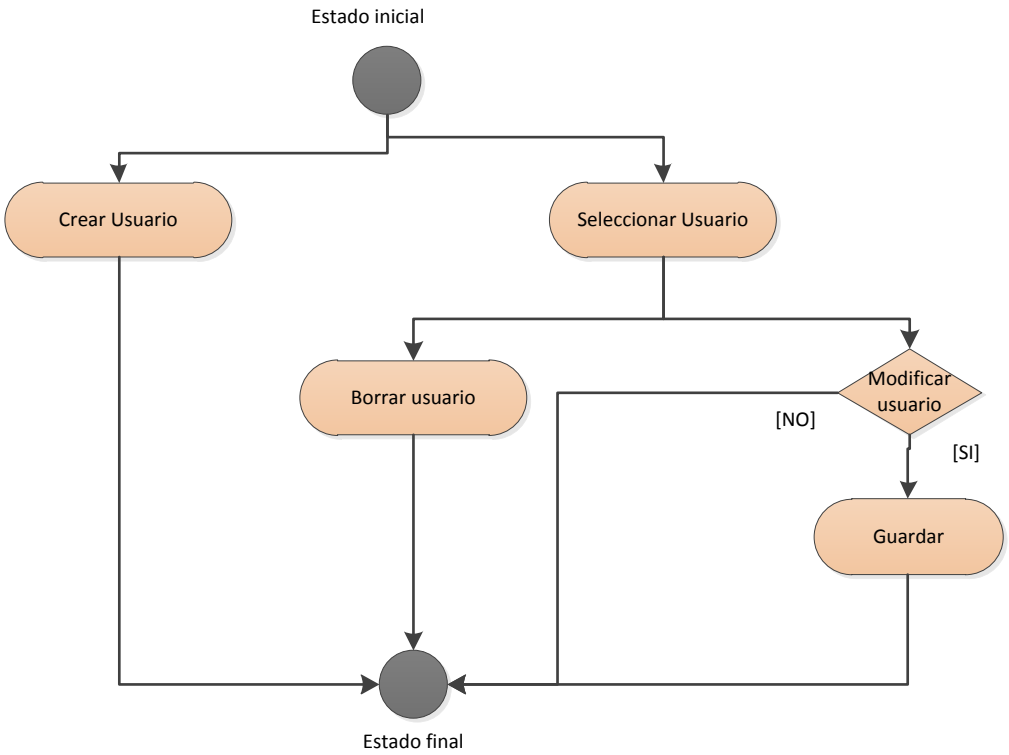


Ilustración 14: Diagrama de actividad para "Gestionar Usuarios"

3.3. Caso de uso “Gestión de Grupos”

3.3.1. Descripción del escenario

Este caso de uso representa la interacción del usuario administrador y el sistema cuando el primero desea gestionar los grupos del sistema, es decir: crear grupo/s, borrar un grupo o grupos y vincular usuarios a grupos.

3.3.2. Flujo de eventos

El caso de uso comienza cuando el usuario administrador se dispone a gestionar los grupos, es decir, cuando selecciona alguna de las opciones: “+”, “-“ o haga clic sobre el checkbox asociado a cada grupo de la lista.

3.3.2.1.Flujo básico – Gestión de grupos

1. Si el usuario administrador selecciona “+” se mostrará un diálogo en el que tendrá que introducir el nombre y la descripción del grupo que desee crear. Se podrán crear uno o varios a la vez.

2. Si el usuario administrador selecciona “-“ se eliminarán aquellos grupos de la lista que estén marcados.

3. Si el usuario administrador selecciona un usuario y hace “check” en aquellos grupos que considere oportunos, asignará al usuario a dichos grupos.

3.3.2.2.Flujos alternativos – Excepciones:

Será necesario añadir flujos alternativos para controlar los siguientes sucesos:

- Errores en la lectura de la base de datos.

- Errores en la escritura en la base de datos.

3.3.3. Precondiciones

Para el caso de vincular un usuario a un grupo, tendrá que crearse con anterioridad el grupo.

3.3.4. Postcondiciones

No se han determinado.

3.3.5. Diagrama de actividad

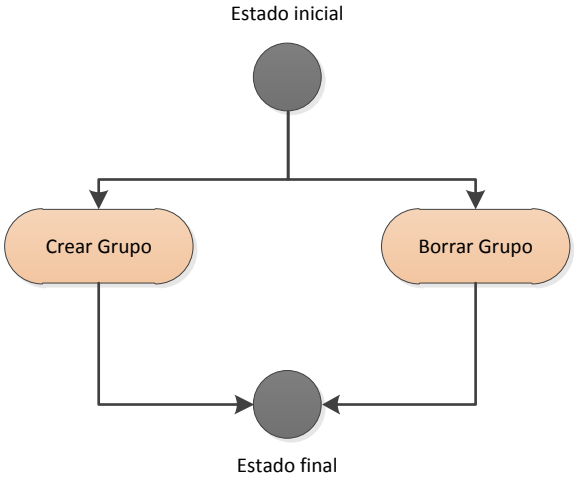


Ilustración 15: Diagrama de actividad para "Gestión de Grupos"

3.4. Caso de Uso "Asignación de Averías"

3.4.1. Descripción del escenario

Este caso de uso representa la vinculación de las averías del sistema con los usuarios o grupos creados. De esta asignación dependerá de que un tipo de avería se notifique a un usuario u otro.

3.4.2. Flujo de eventos

El caso de uso comienza cuando el usuario administrador selecciona en el menú principal la opción "Assign Faults".

3.4.2.1. Flujo básico – Asignación de averías

1. El usuario selecciona una de las opciones que se presentan: *Users* o *Groups.*
2. Una vez seleccionado *Users* o *Groups*, se cargarán en el listbox correspondiente la lista con los usuarios/grupos creados. Solo se permitirá seleccionar una opción de la lista.
3. El usuario administrador seleccionará una opción de la lista de usuarios/grupos.

4. El usuario administrador seleccionará mediante un checkbox las averías que considere oportunas vincular al usuario/grupo seleccionado.
5. Si quiere guardar la asignación realizada, el usuario debe pulsar el botón “Save”.

3.4.2.2.Flujos alternativos

Sera necesario añadir flujos alternativos para controlar los siguientes sucesos o excepciones:

- Errores en el guardado de las asignaciones.
- Errores en la carga de la lista de averías.
- Errores en la carga de la lista de usuarios/grupos.
- Errores al no vincular ninguna avería y pulsar el botón de “Save”.

3.4.3. Precondiciones

Para que este caso de uso pueda desarrollarse es necesario crear o disponer de una serie de usuarios y/o grupos. Además es indispensable disponer de la lista de posibles averías que se pueden disparar en el almacén.

3.4.4. Postcondiciones

No se han determinado.

3.4.5. Diagrama de actividad

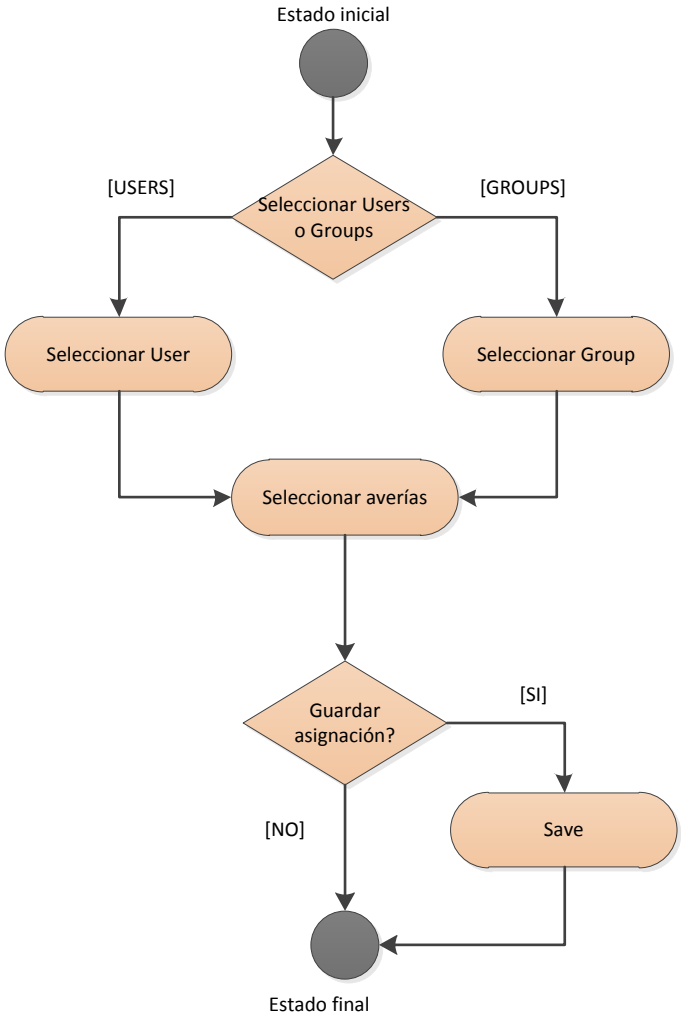


Ilustración 16: Diagrama de actividad para "Asignación de averías"

3.5. Caso de Uso "Gestión de reglas"

3.5.1. Descripción del escenario

Este caso de uso representa la interacción entre el usuario administrador y el sistema cuando el primero desea gestionar la creación y asignación de reglas a los usuarios. Se describe a una "regla" como un intervalo o conjunto de intervalos de tiempo que se asignarán al usuario y durante los cuales estará disponible para atender las incidencias del almacén. Además dichas reglas tendrán un valor de prioridad y de tiempo de espera, los cuales indicarán la prioridad sobre otros usuarios y el tiempo de envío de la notificación respecto al tiempo de activación de la avería.

3.5.2. Flujo de eventos

El caso de uso comienza cuando el usuario administrador selecciona en el menú principal la opción *Edit Rules*

3.5.2.1.Flujo básico – Gestión Reglas

1. Si el usuario administrador desea crear y asignar una regla, lo primero que tendrá que hacer es seleccionar una de las opciones: *User* o *Group.* Hecho esto, se cargará una lista con los usuarios o grupos creados con anterioridad.

2. El usuario tendrá que marcar uno de los usuarios/grupos de la lista mediante un checkbox. Solo se permite la selección de un ítem de la lista.

3. Una vez marcado el ítem, se seleccionarán automáticamente aquellas averías que el usuario/grupo tiene asignadas previamente.

4. El usuario podrá añadir intervalos temporales si selecciona la opción "+" o eliminar los ya creados si selecciona la opción "-". Los parámetros a añadir son el día de comienzo y fin, y la hora de entrada y salida.

5. El usuario dispondrá de dos textboxes para introducir la prioridad de la regla (valor de 1 a 10) y el tiempo de espera (en minutos).

6. Los pasos 4 y 5 son intercambiables.

7. Si el usuario pulsa el botón "Save" guardará toda la información introducida en los pasos anteriores y se vincularán los datos a las averías y usuario/grupo seleccionado. Si se pulsa el botón "Cancel" se borrarán los datos introducidos y se volverá al menú principal.

3.5.2.2.Flujos alternativos

Será necesario añadir flujos alternativos para controlar los siguientes sucesos:

- Errores en la lectura de la base de datos (carga de la lista de usuarios/grupos o averías).
- Errores en la escritura en la base de datos de la información introducida.
- Campos de datos vacíos o con un formato incorrecto.

3.5.3. Precondiciones

- Deben de crearse con anterioridad usuarios o grupos.
- Deben asignarse con anterioridad averías a los usuarios o grupos presentes.

3.5.4. Postcondiciones

No se han determinado.

3.5.5. Diagrama de actividad

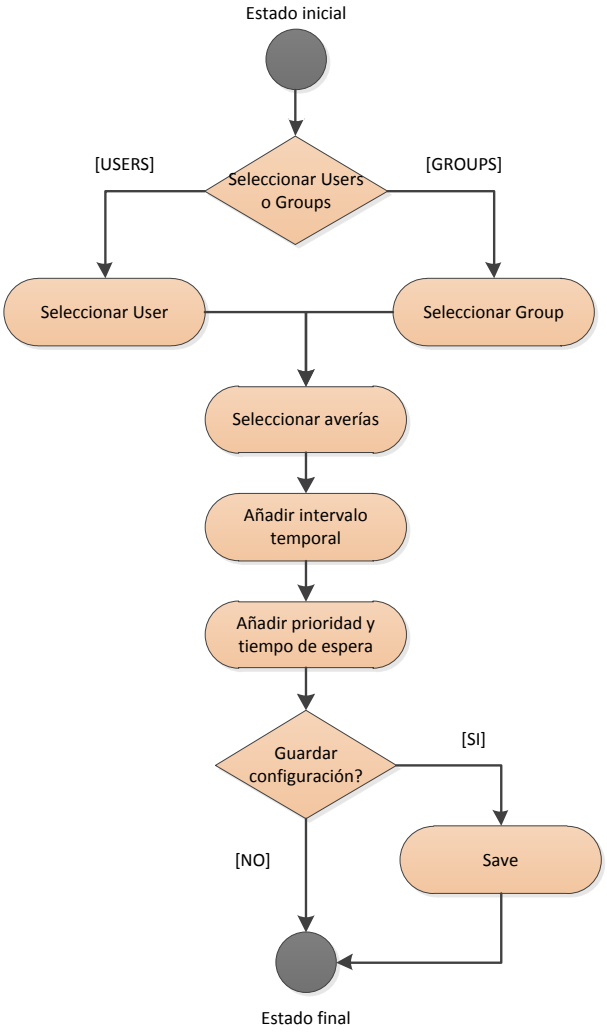


Ilustración 17: Diagrama de actividad para "Gestión de reglas de notificación"

3.6. Caso de Uso “Visualizar averías activas”

3.6.1. Descripción del escenario

Este caso de uso representa la interacción del usuario y el sistema cuando el primero desea comprobar el número de averías activas que tiene asignadas y los detalles de las mismas.

3.6.2. Flujo de eventos

El caso de uso comienza cuando el usuario se dispone a visualizar las averías disparadas o activas, que tiene asignadas.

3.6.2.1. Flujo básico – Visualizar averías activas

1. Si el usuario desea visualizar las averías activas pulsará el icono correspondiente a la aplicación Mecalux Failures Manager en su terminal móvil (Smartphone).

2. Si es la primera vez que el usuario inicia la aplicación, tendrá que configurar los parámetros de la conexión (Dirección ip a la que se ha de conectar) e introducir sus parámetros de inicio de sesión (nombre de sesión, password e identificador de la empresa para la que trabaja).

3. En caso de segundos y sucesivos arranques de la aplicación, el paso 2 se omitirá.

4. Una vez arrancada la aplicación, se conectará a la dirección IP introducida en el paso 2 y cargará las averías que el usuario logueado tiene asignadas. Se cargará en formato de lista, siendo el valor mostrado el de la máquina que dispara la incidencia.

5. El usuario podrá visualizar el detalle de la avería (Fecha de la avería, nombre de computador, descripción, sugerencia,…), pulsando sobre el nombre de la avería.

3.6.2.2. Flujos alternativos

Será necesario añadir flujos alternativos para controlar los siguientes sucesos:

- Errores en la conexión con el servidor de datos.

- Errores de validación de usuario.

3.6.3. Precondiciones

Para que este caso de uso comience debe haberse ejecutado primero el caso de uso "Gestión de usuarios" y como resultado del mismo tendrá que existir en la base de datos el usuario que pretenda visualizar las averías.

3.6.4. Postcondiciones

No se han determinado.

3.6.5. Diagrama de actividad

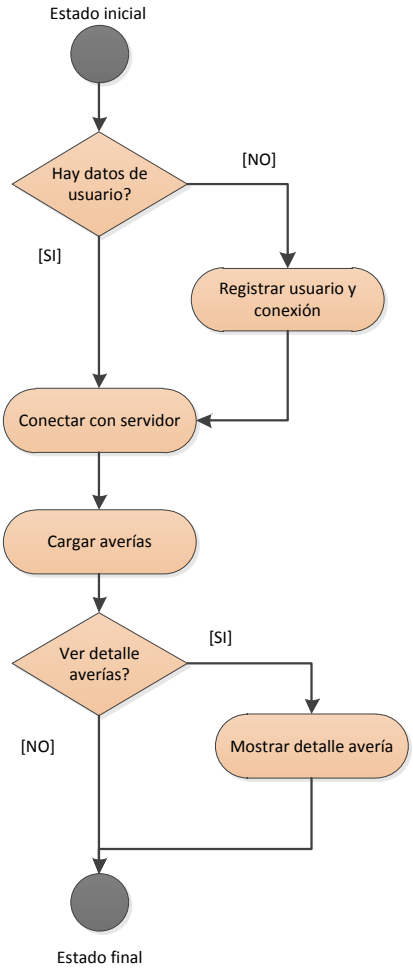


Ilustración 18: Diagrama de actividad para "Visualizar averías activas"

3.7. Caso de Uso "Confirmar averías"

3.7.1. Descripción del escenario

Este caso de uso representa la interacción entre el usuario y el sistema de visualización de averías cuando el primero desea confirmar la notificación de una avería. Esto permitirá informar al sistema de que el usuario ha recibido y comprendido la incidencia y que se dispondrá a solucionarla.

3.7.2. Flujo de eventos

El caso de uso comienza cuando el usuario accede al detalle de una avería en el visualizador de averías y selecciona en el menú la opción “Ack User”.

3.7.2.1. Flujo básico – Confirmar averías

1. El usuario accede al menú de detalle de avería.
2. El usuario pulsa el botón de “Ack User” y confirma la acción.
3. El visualizador de averías envía la confirmación al sistema de datos.
4. El usuario vuelve al menú principal.

3.7.2.2. Flujos alternativos

Será necesario añadir flujos alternativos para controlar los siguientes sucesos:

- Errores al cargar el detalle de la avería.
- Errores de conexión al confirmar la avería.
- Errores en la escritura de la confirmación en la base de datos.

3.7.3. Precondiciones

Para que este caso de uso pueda comenzar es necesario que se haya ejecutado el caso de uso “Visualizar averías activas” y como resultado se muestren las averías que el usuario quiere confirmar.

3.7.4. Postcondiciones

La avería confirmada se elimina de la lista de averías activas.

3.7.5. Diagrama de actividad

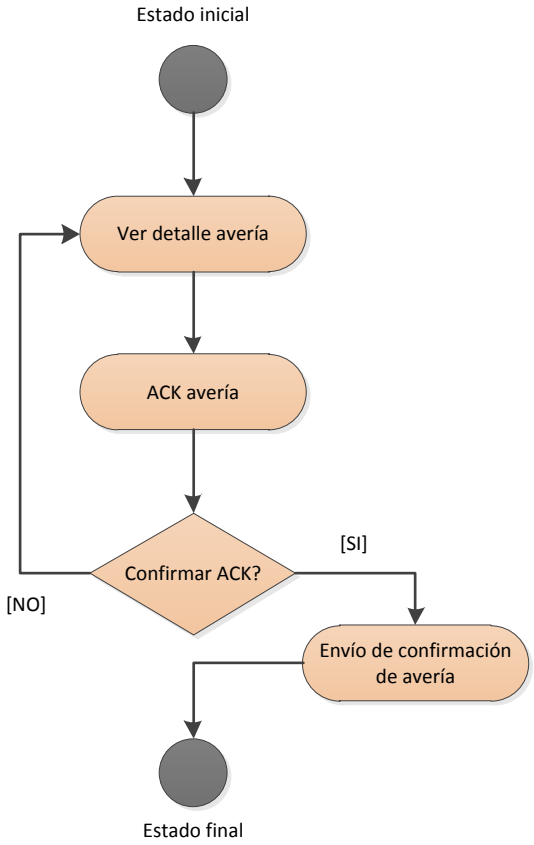


Ilustración 19: Diagrama de actividad para "Confirmar avería"

3.8. Caso de Uso "Filtrar averías"

3.8.1. Descripción del escenario

Este caso de uso representa la interacción del usuario y el sistema visualizador de averías cuando el primero desea realizar un filtrado de las averías activas que se muestran. Se permitirá introducir el nombre identificador de aquella máquina de la cual se deseen ver las averías.

3.8.2. Flujo de eventos

El caso de uso comienza cuando el usuario está visualizando las averías activas y desea realizar un filtrado por nombre de máquina del total de averías que se muestran.

3.8.2.1. Flujo básico – Filtrar averías

1. El usuario selecciona dentro del menú de opciones la opción “Faults filter”.

2. Se mostrará junto a la lista de averías un textbox en el que el usuario introducirá el identificador de aquella máquina, de la cual quiera ver las averías.

3. El filtrado de la información será automático, no siendo necesario pulsar ningún botón adicional.

3.8.2.2. Flujos alternativos

Será necesario añadir flujos alternativos para controlar los siguientes casos:

- El sistema no muestra ninguna avería activa.

- El identificador de la máquina a filtrar no existe.

3.8.3. Precondiciones

Para que este caso de uso pueda comenzar es necesario que se haya ejecutado el caso de uso “Visualizar averías activas” y como resultado se muestren las averías que el usuario quiere filtrar.

3.8.4. Postcondiciones

La lista de averías se reduce a las pertenecientes a la máquina indicada.

3.8.5. Diagrama de actividad

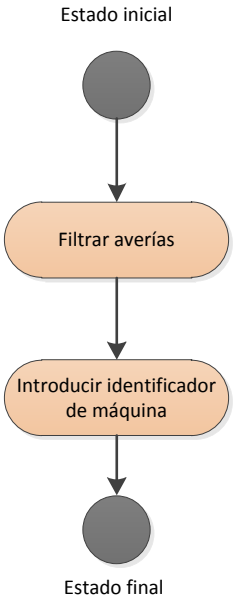


Ilustración 20: Diagrama de actividad para "Filtrar averías"

4. DIAGRAMAS DE CLASES

Después de realizar los diagramas de casos de uso y de actividades para cada uno de ellos, se está en disposición de crear los diagramas de clases. En ellos se representan las clases que se identifican en el sistema y las relaciones que hay entre ellas.

Además de las relaciones entre clases también se identifican los métodos y atributos de cada una de ellas.

4.1. Descripción del gestor de usuarios y averías

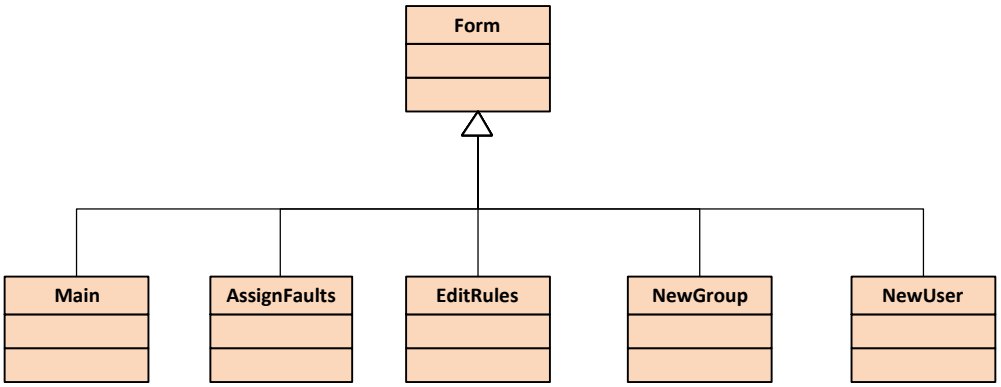


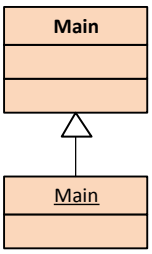
Ilustración 21: Diagrama de clases del gestor de usuarios y averías

4.1.1. Clase Form

Clase base que representa una ventana o cuadro de dialogo. Constituye la interfaz de usuario de una aplicación. Se aplicará un proceso de herencia sobre ella para tener todas las funcionalidades de una ventana en los cuadros de dialogo a desarrollar.

4.1.2. Clase Main

Diagrama de objeto

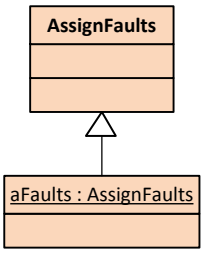


Descripción de la clase

Esta clase se encarga de implementar la lógica necesaria para el inicio de sesión de los usuarios en el sistema, así como proporcionar la información necesaria de los usuarios y averías presentes en dicho sistema.

4.1.3. Clase AssignFaults

Diagrama de objeto

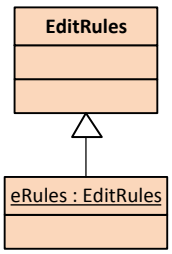


Descripción de la clase

Esta clase se encarga de implementar la lógica necesaria para mostrar información de los usuarios/grupos del sistema y las averías que pueden atender. Además de procedimientos que permitan vincular dichos usuarios y averías.

4.1.4. Clase EditRules

Diagrama de objeto

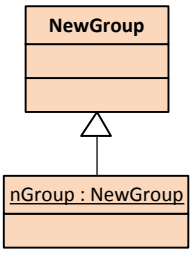


Descripción de la clase

Esta clase se encarga de implementar la lógica necesaria que permita introducir la información necesaria de los turnos de trabajo y prioridad de notificación que pueda tener un usuario o grupo del sistema.

4.1.5. Clase NewGroup

Diagrama de objeto

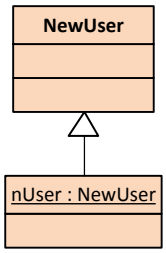


Descripción de la clase

Esta clase se encarga de implementar la lógica que permita a un usuario administrador introducir nuevos grupos al sistema con el objetivo de realizar agrupaciones de usuarios entorno a estos grupos.

4.1.6. Clase NewUser

Diagrama de objeto



Descripción de la clase

Esta clase se encarga de implementar la lógica que permita a un usuario administrador introducir nuevos usuarios (empleados) al sistema. Estos usuarios serán los responsables de atender y solucionar las averías que se puedan producir en el almacén.

4.2. Diagrama de clases del servicio web

El servicio web o aplicación servidora aglutinará toda la lógica de la aplicación, dejando al cliente gestor de usuarios y al visor de averías como simples interfaces de usuario. A continuación se muestra el diagrama

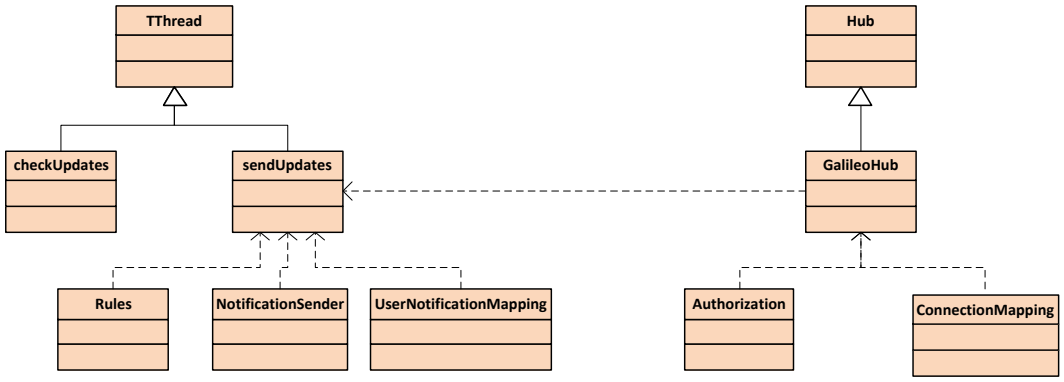


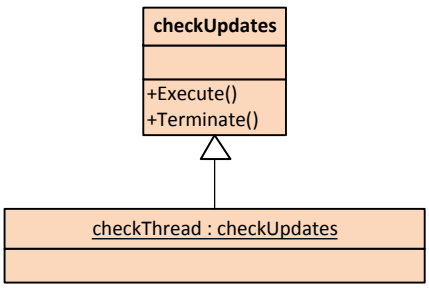
Ilustración 22: Diagrama de clases servicio web

4.2.1. Clase TThread

Clase perteneciente a la librería Mecalux.ITSW.Core.Abstraction desarrollada por Mecalux para el manejo de hilos de ejecución.

4.2.2. Clase checkUpdates

Diagrama del objeto

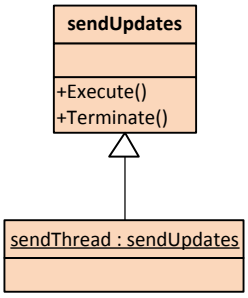


Descripción de la clase

Esta clase se encarga de implementar el proceso de comprobación de la base de datos. Funcionará como un hilo de ejecución paralelo al programa principal, permitiendo comprobaciones periódicas de la base de datos para detectar nuevas averías.

4.2.3. Clase sendUpdates

Diagrama del objeto

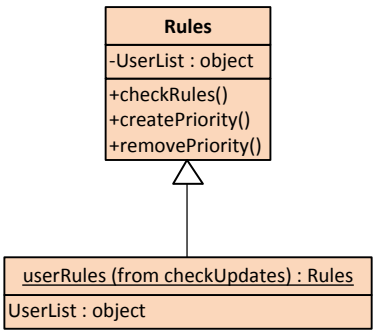


Descripción de la clase

Esta clase se encarga de implementar el proceso de envío de notificaciones de nuevas averías. Para ello recogerá información periódicamente de un almacén temporal donde se guardarán los avisos de nuevas averías.

4.2.4. Clase Rules

Diagrama del objeto

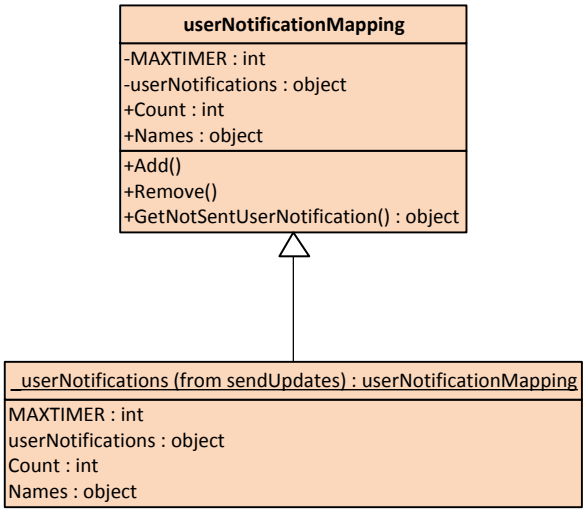


Descripción de la clase

Esta clase se encarga de implementar la lógica necesaria para determinar que usuarios cumplen los requisitos de prioridad y turno de trabajo para enviarles una determinada notificación de avería.

4.2.5. Clase UserNotificationMapping

Diagrama de objeto



Descripción de la clase

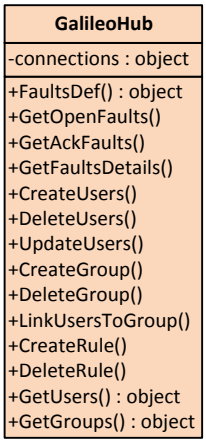
Esta clase de encarga de implementar el proceso de vinculación del usuario o identificador de usuario con la información recabada de la avería. Esto permitirá decidir posteriormente la dirección de envío de la notificación de incidencia.

4.2.6. Clase Hub

Esta clase se encarga de proveer los métodos necesarios para comunicar el Hub del servicio web con todas aquellas conexiones SignalR que se produzcan. La clase GalileoHub heredará todas sus propiedades.

4.2.7. Clase GalileoHub

Diagrama de objeto

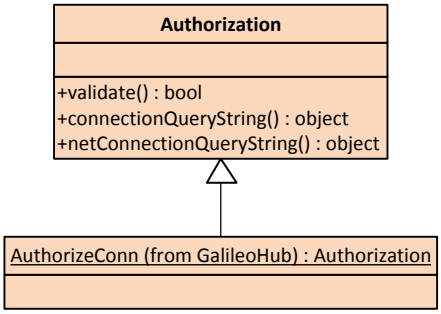


Descripción de la clase

Esta clase se encarga de implementar todos aquellos métodos que permitan al cliente pedir y recibir una determinada información. De acuerdo a las características de SignalR todos los métodos que se quieran invocar de forma remota han de encontrarse en esta clase.

4.2.8. Clase Authorization

Diagrama de objeto

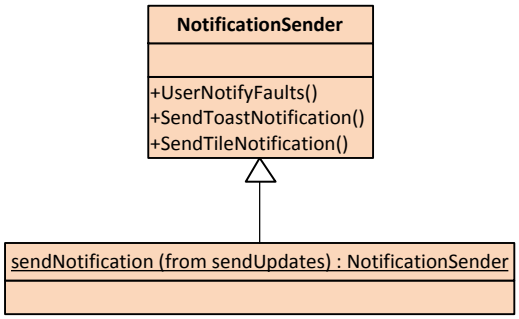


Descripción de la clase

Esta clase se encarga de implementar los métodos necesarios para la autenticación y validación de los usuarios que se conecten al servicio web.

4.2.9. Clase NotificationSender

Diagrama de objeto

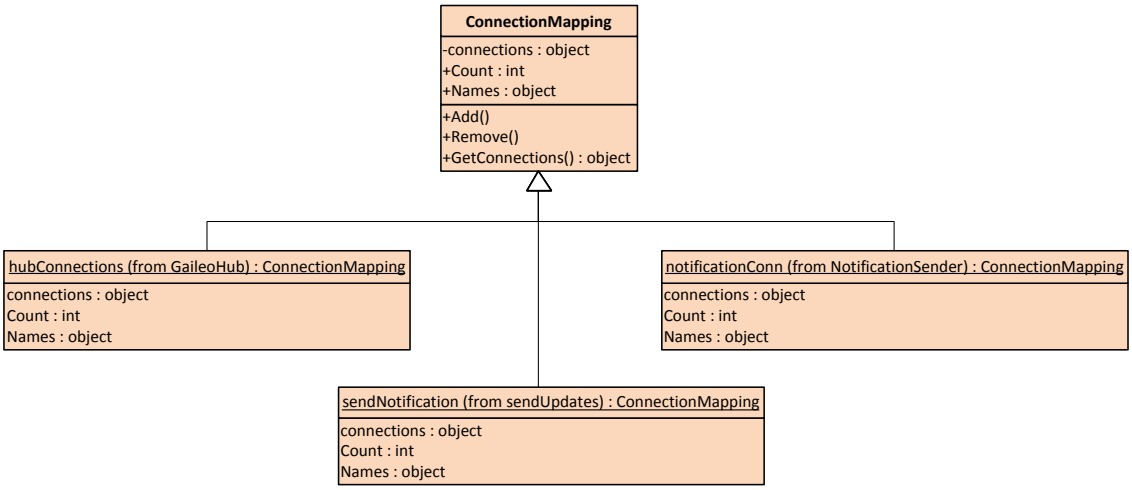


Descripción de la clase

Esta clase se encarga de implementar la lógica necesaria para el envío de avisos de avería al usuario. Tendrá que encargarse del envío de información a través de la conexión SignalR como el envío de notificaciones PUSH a la aplicación móvil.

4.2.10. Clase ConnectionMapping

Diagrama de objeto



Descripción de la clase

Esta clase se encarga de implementar la lógica necesaria para almacenar y vincular al usuario que se conecte al sistema con la propia conexión SignalR. Este proceso ayudará a reconocer la identidad de cada conexión con el objetivo de hacer envíos de información personalizados.

3.3. Diagrama de clases del visor de averías

Como se ha comentado en apartados anteriores, la aplicación de visualización de averías se desarrollará con tecnologías web. Todas las funcionalidades que ofrece convergen en torno a una interfaz de usuario. Por ello el diagrama de clases que presenta la aplicación se desarrolla en torno a ella.

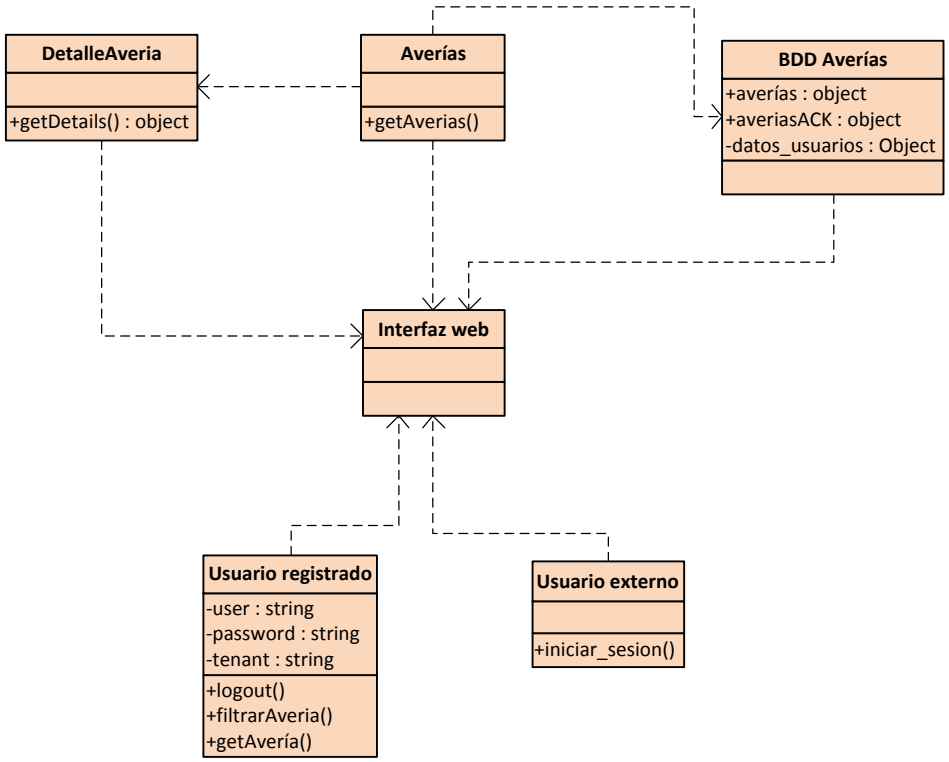


Ilustración 23: Diagrama de clases del visor de averías

4.2.1. Clase Interfaz web

Esta clase se encargará de implementar todos aquellos controles que permitan al usuario solicitar datos al servidor como visualizarlos. Será el eje central de la aplicación, pues la visualización de contenidos es su principal objetivo.

4.2.2. Clase Usuario Registrado

Esta clase se encargará de implementar el comportamiento del usuario que esté registrado en la aplicación. El hecho de estar registrado, implica no tener que realizar el paso de "logueo" cada vez que se inicia la aplicación. Las opciones que podrá realizar dicho usuario serán las de ver las averías, filtrar las averías que desee y cerrar sesión si desea.

4.2.3. Clase Usuario externo

Esta clase se encargará de implementar el comportamiento y la lógica que permita a un usuario no identificado registrarse en el sistema y dentro de la aplicación. Para que un usuario acceda a la aplicación tendrá que existir en la base de datos asociada. La creación de dicho usuario no se realizará desde el visor de averías, sino que se realizará previamente en el gestor de usuarios.

4.2.4. Clase Averías

En esta clase implementará la lógica que permita al usuario registrado pedir al servicio web la información necesaria para visualizar las averías activas que tiene asignadas. Además de controlar la creación de la vista de detalles de avería. Esta vista permitirá al usuario visualizar en el interfaz la información complementaria de una determinada avería.

4.2.5. Clase DetalleAvería

Esta clase implementará la lógica que permita a la aplicación pedir la información al servicio web de una determinada avería indicada por el usuario. Utilizará además esos datos para crear la página de visualización que mostrará el interfaz. Aparte de lo indicado, se facilitará al usuario una opción para confirmar la avería detallada. Significa que al confirmar una avería, el usuario está comprometido en solucionarla.

5. DESCRIPCIÓN DEL MODELO DE DATOS

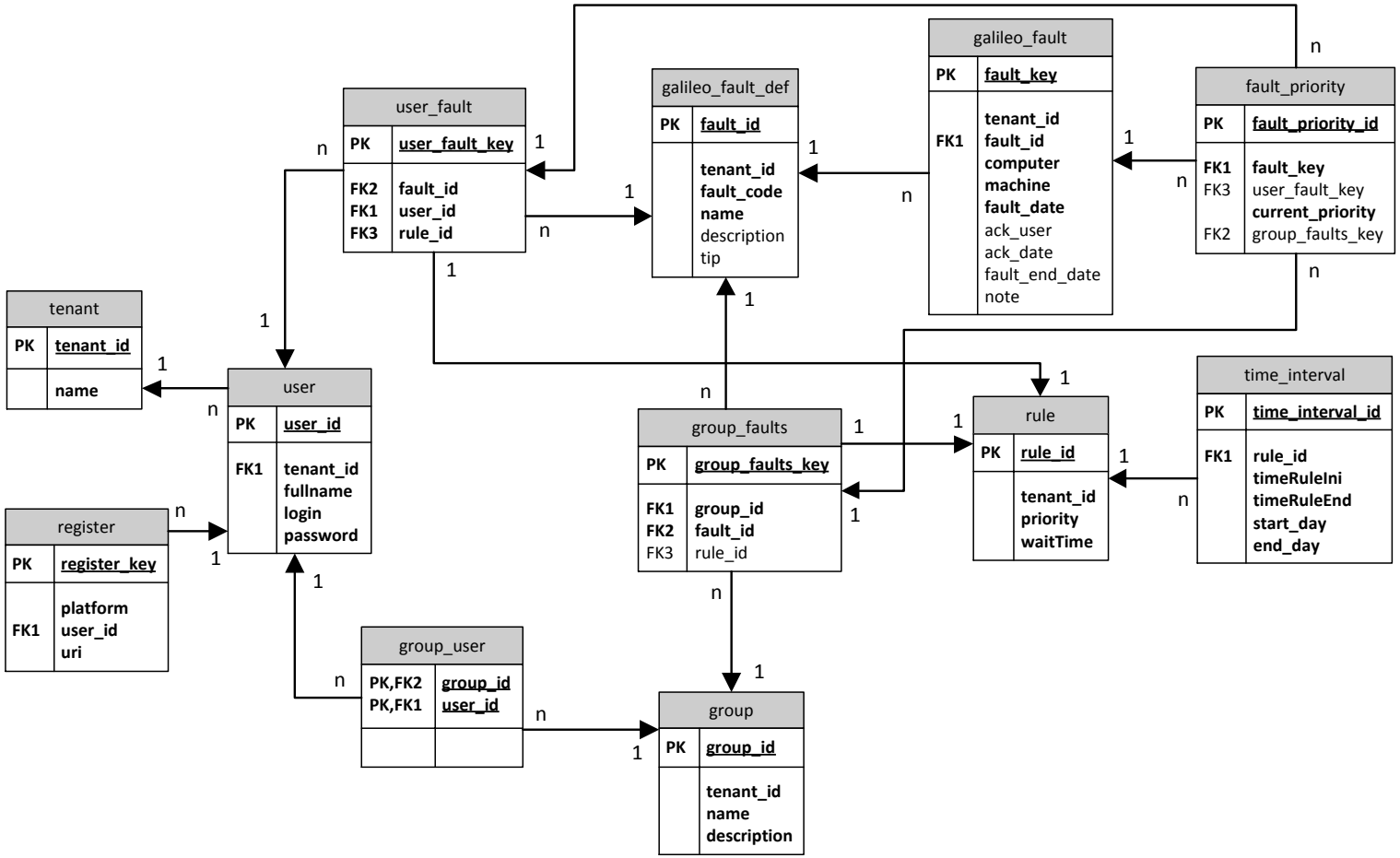


Ilustración 24: Descripción del modelo de datos

5.1. Descripción de las entidades

En este apartado se da una descripción de todas las tablas que forma la base de datos, así como una breve explicación de todos los campos que forman cada una de las tablas. Además en el diagrama anterior, se pueden distinguir las relaciones que unen a cada entidad.

5.1.1. user

Se almacenan los datos referentes a los usuarios que van a utilizar el sistema. Sus campos son:

- user_id: Identificador único para cada usuario. Este identificador es la clave principal de la tabla.
- tenant_id: Identificador del tenant (cliente) en el cual trabaja el usuario.
- fullname: Nombre completo del usuario.
- login: identificador o nombre de inicio de sesión del usuario.
- password: contraseña para el inicio de sesión.

5.1.2. tenant

Almacena los diferentes clientes (empresas) que utilizan el sistema. Sus campos son:

- tenant_id: Identificador único para cada cliente. Este identificador es la clave principal de la tabla.
- name: Nombre descriptivo del cliente. Nombre comercial.

5.1.3. group

Almacena los datos referentes a los grupos de usuarios creados. Sus campos son:

- group_id: identificador único para cada grupo. Este identificador es la clave principal de la tabla.
- tenant_id: Identificador del tenant (cliente) al que está asociado el grupo.
- name: Nombre del grupo.
- description: Breve descripción de las labores que realiza el grupo.

5.1.4. group_user

Tabla destinada a romper la relación N a N existente entre las tablas USER y GROUP. Sus campos son:

- group_id: Identificador de grupo.
- user_id: Identificador de usuario.

5.1.5. galileo_fault_def

En esta tabla se almacenará una descripción completa de las averías que puede disparar el sistema, es decir, una definición de las averías posibles dentro del almacén, no las que se han producido. Sus campos son:

- fault_id: Identificador único de la avería. Este identificador será la clave principal de la tabla.
- tenant_id: Identificador del cliente donde se dispara la avería.
- fault_code: Codigo descriptivo del tipo de avería disparada.
- name: Nombre de la avería.
- description: Descripción detallada de la avería. Indica el tipo de fallo que se produce.
- tip: En este campo se indican sugerencias de solución para la avería. No es un campo obligatorio.

5.1.6. galileo_fault

En esta tabla, el sistema Galileo volcará todas las averías que dispare el sistema con su correspondiente información. Sus campos son:

- fault_key: Identificador único de avería disparada. Este identificador será la clave principal de la tabla.
- tenant_id: Identificador del cliente (empresa, almacen) donde se dispara la avería.
- fault_id: Identificador de avería. No es un valor único dentro de la tabla ya que puede dispararse varias veces una avería del mismo tipo.
- computer: Identificador de la computadora que dispara la avería. Puede haber varias en el mismo almacén.
- machine: Identificador de la máquina en la que se produce la avería.
- fault_date: Este campo indica la fecha y hora en la que se detecta la avería.
- ack_user: Este campo indica si algún operario ha confirmado la detección de la avería.

- ack_date: Valor de la fecha y hora en la que el usuario confirma la detección de la avería.
- fault_end_date: Fecha y hora en la que la avería se soluciona.
- note: Breve descripción o información de la avería y de cómo se ha solucionado el problema.

5.1.7. **user_fault**:

Tabla destinada a romper la relación N a N existente entre las tablas USER y GALILEO_FAULT_DEF y para determinar que averías se asignan a los usuarios. Sus campos son:

- user_fault_key: Identificador único de asignación de avería. Este identificador será la clave principal de la tabla.
- fault_id: Identificador de la avería asignada.
- user_id: Identificador del usuario al que se le asigna la avería.
- rule_id: Identificador de la regla que se ha asignado al usuario. La regla establece los parámetros temporales y de prioridad en los que tienen vigencia la asignación.

5.1.8. **group_faults**

Tabla destinada a romper la relación N a N existente entre las tablas GROUP y GALILEO_FAULT_DEF y para determinar que averías se asignan a los grupos. Sus campos son:

- group_faults_key: Identificador único de asignación de avería al grupo. Este identificador será la clave principal de la tabla.
- group_id: Identificador del grupo al que se le asigna la avería.
- fault_id: Identificador de la avería asignada al grupo.
- rule_id: Identificador de la regla que se ha asignado al usuario.

5.1.9. rule

En esta tabla se almacenará parte de la información relacionada con las reglas de notificación de averías. Sus campos son:

- rule_id: Identificador único de la regla. Este identificador será la clave principal de la tabla.
- tenant_id: Identificador el cliente (almacén) donde se aplicará la regla.
- priority: Este campo indica que prioridad tiene el usuario/grupo al que se asigna esta regla para solventar una avería disparada.
- waitTime: Indica el tiempo que tardará en notificarse dicha avería al usuario de siguiente prioridad.

5.1.10.time_interval

En esta tabla, relacionada con la tabla RULE, se indicarán los parámetros temporales de la regla. Sus campos son:

- time_interval_id: Es el identificador único del intervalo temporal. Este identificador será la clave principal de la tabla.
- rule_id: Indica el la regla a la que se aplicará este intervalo temporal.
- timeRuleIni: Especifica el horario de comienzo del intervalo.
- timeRuleEnd: Especifica el horario de fin del intervalo.
- start_day: Indica el dia del año en el que comienza el intervalo.
- end_day: Indica el dia del año en el que finaliza el intervalo.

5.1.11.fault_priority

A la hora de notificar una avería, el sistema tendrá que consultar el valor de prioridad que tienen los usuarios/grupos asignados. También se almacenará el valor de prioridad de la avería, de forma que cuando una notificación no reciba confirmación, la prioridad aumente y la notificación se envíe a más usuarios. Para ello será necesario consultar una serie de campos descritos a continuación:

- fault_priority_id: Se trata de un identificador único de prioridad de la avería. Será la clave principal de la tabla.
- fault_key: Este parámetro es el identificador de la avería disparada.
- current_priority: Indica el valor actual de prioridad de la avería. Será un valor creciente (1…10) que provocará el envío de notificaciones a más usuarios cuanto mayor sea el valor.

- user_fault_key: Identificador de asignación de avería. No es un valor obligatorio.
- group_fault_key: Identificador de asignación de avería a un grupo. No es un valor obligatorio.

5.1.12.register

Para notificar una avería a un usuario, este tendrá que disponer de la aplicación de visualización de incidencias. Para poder recibirlas la plataforma de notificaciones tendrá que tener la dirección de envío del terminal. Esta tabla almacenará este valor además de otros parámetros requeridos. Sus campos son:

- register_key: Identificador único de registro. Este campo será la clave principal de la tabla.
- platform: En este campo se almacenará la plataforma a la que corresponde el registro y la dirección de envío (Android, WP8, iOS).
- user_id: Identificador del usuario al que se envía la notificación.
- URI: Dirección única de envío de la notificación. Parámetro requerido por el cliente de notificaciones del terminal.

MANUAL DE USUARIO

6. MANUAL USUARIO MECALUX FAILURES MANAGER

6.1. Introducción

Mecalux Failures Manager es un nuevo producto diseñado para Mecalux Software Solutions, que pretende ser un complemento al sistema de control Galileo y al entorno de desarrollo Designer IV. El principal objetivo del proyecto es el desarrollo de un sistema que permita la visualización remota de las incidencias producidas en los almacenes controlados por Galileo. Además de la visualización, otro de los objetivos será el desarrollo de una aplicación liviana de gestión de usuarios y averías, que permita configurar una relación entre ambos y que trabaje conjuntamente con el mencionado visor de averías.

A continuación se realiza una exposición del manejo básico de las diferentes aplicaciones, necesaria para poder entender el funcionamiento de los objetos que se han desarrollado para este proyecto.

6.2. Conexión entre cliente y servidor

Como se ha comentado en los bloques anteriores, el sistema está compuesto por tres partes diferenciadas: una aplicación Windows encargada de la gestión de usuarios y averías, un visor de averías, desarrollado mediante una aplicación móvil, y un servidor o servicio Web que será el encargado de asumir toda la lógica del sistema y notificar a los clientes cuando se produce una avería en el almacén controlado por Galileo. Comentar que el servidor del sistema no dispone de una interfaz de usuario clásica. Para ésta entre en funcionamiento soló será necesario iniciar la aplicación.

Para que los diferentes clientes de la aplicación puedan conectarse con el servidor será necesario que configuren sus direcciones de conexión (IP) de forma similar, es decir, se indicará en qué dirección “publica” su información el servicio web y sabido esto, los clientes que deseen conectarse al servicio tendrán que indicar dicha dirección IP en su configuración correspondiente.

Ilustración 25: Configuración de conexión en aplicación móvil

En la ilustración superior se muestra una imagen del menú de configuración de la aplicación móvil. Será en esta pantalla donde el usuario introduzca la dirección IP del servidor. Solo así podrá acceder al servicio de averías.

6.3. Manejo de Mecalux Failures Manager

En este apartado se explica cómo manejar la aplicación, analizando las partes más significativas del interface de usuario. Debido a que la aplicación se compone de dos clientes diferentes se analizarán por separado.

Es importante indicar que las **ventanas o cuadros de dialogo de la aplicación** de gestión de usuarios y averías que se va a mostrar en el apartado 6.3.1 corresponde con una **versión de desarrollo no definitiva**, pues la versión final irá integrada con el SGA (Sistema de Gestión de Almacén) o con el SCADA desarrollado por Mecalux. La presente versión solo tiene como objetivo mostrar las diferentes funcionalidades del sistema.

6.3.1. Gestor de usuarios y averías

En este apartado se explicará cómo manejar la aplicación de gestión de usuarios. Se describirá con detalle el funcionamiento de todos los campos y ventanas que la componen.

6.3.1.1. Ventana principal

La ventana principal dispone de cuatro secciones diferenciadas, orientada cada una de ellas a una función determinada:

- **Login**: Este grupo de controles agrupa los campos y botones necesarios para el inicio de sesión del usuario en el servicio web.

- **Users**: Este grupo de componentes controla la selección, creación y eliminación de usuarios de nuestro sistema.

- **Data**: Grupo de controles donde se aglutina la información referente a usuarios. Para activar esta sección será imprescindible seleccionar antes un usuario en el menú *Users*.

- **Faults**: Este grupo se encarga del acceso a las opciones de asignación y configuración de averías.

En la figura siguiente se muestra el aspecto de la ventana principal de la aplicación así como sus principales elementos.

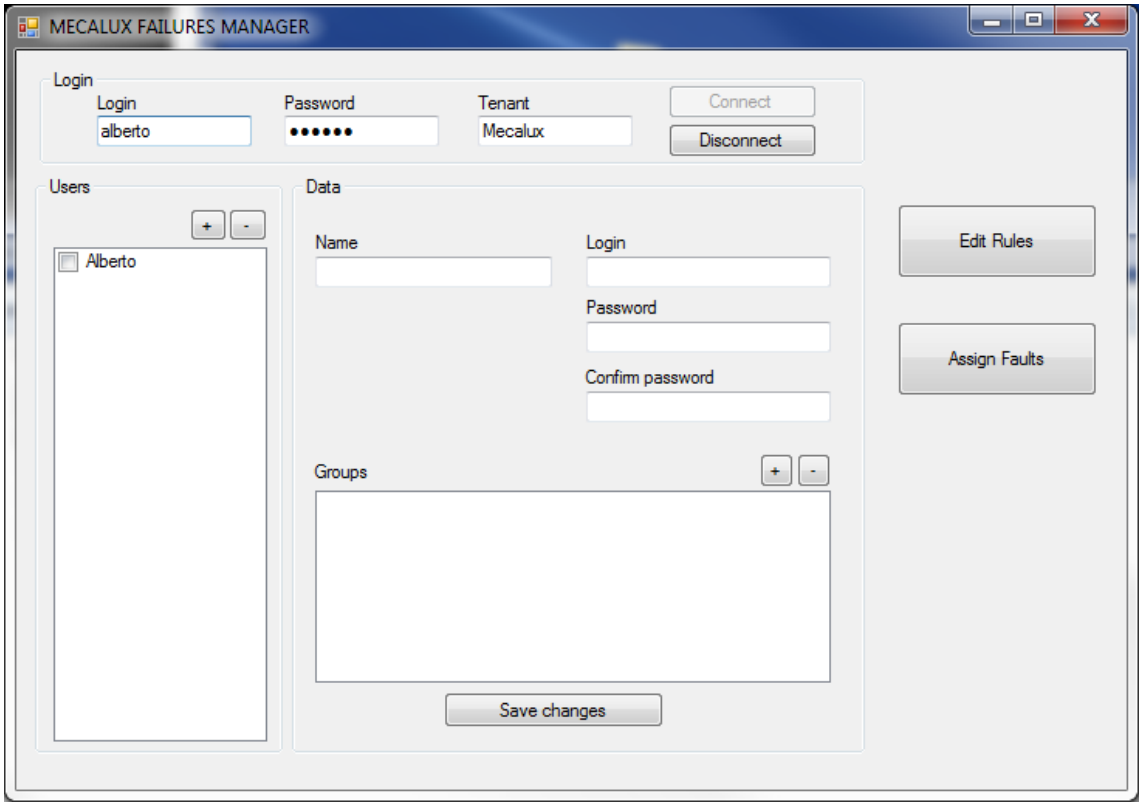
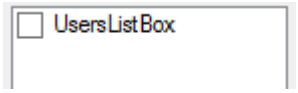


Ilustración 26: Ventana principal del gestor de usuarios y averías

- En el *GroupBox* **Users** se pueden encontrar los siguientes campos y controles:

Botón de creación de nuevo usuario. Esta opción del menú lanza la ventana de "Crear Nuevos Usuarios", donde se encontrarán los campos y opciones necesarias para añadir nuevos usuarios al sistema.

Botón de eliminación de usuarios. Esta opción del menú permite eliminar los usuarios del sistema. El proceso de borrado puede ser individual o múltiple, para ello se seleccionará en la lista de usuarios aquellos que se deseen eliminar y a continuación se pulsará el presente botón.



Lista de usuarios presentes en el sistema. Este campo permitirá a la aplicación cargar en una lista todos los usuarios que han sido creados y añadidos al sistema Mecalux Failures Manager. Se trata de un control del tipo CheckedListBox, por

lo que junto al nombre de usuario presentará un recuadro de selección. La selección de un usuario significará cargar sus datos en el menú Data. Si por el contrario la selección es múltiple no se cargará ningún dato en dicho menú.

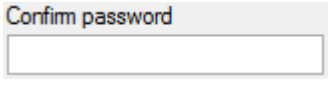
- En el *GroupBox* **Data** se pueden encontrar los siguientes campos y controles:

Este GroupBox está desactivado por defecto, es decir, con el arranque de la aplicación sus campos no serán editables. Para cargar o modificar los datos de un usuario el administrador tendrá que iniciar sesión previamente y a continuación "checkear" el usuario a modificar.

TextBox con el nombre del usuario. En este campo se mostrará el nombre completo del usuario seleccionado en la sección *Users.*

TextBox con el nombre de sesión del usuario. En este campo se mostrará el nombre de inicio de sesión del usuario seleccionado en la sección *Users*.

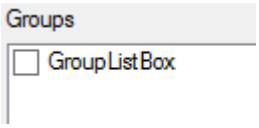
TextBox con la contraseña del usuario. En este campo se mostrará la contraseña para el inicio de sesión del usuario seleccionado en la sección Users. No se mostrará directamente la contraseña actual de dicho usuario, sino que los caracteres permanecerán ocultos.



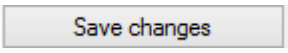
TextBox para confirmación de contraseña. En este campo se mostrará la contraseña para el inicio de sesión del usuario seleccionado en la sección *Users*. Este campo y el campo Password tendrán que tener el mismo valor si se modifica la contraseña del usuario.

Botón de creación de nuevos grupos. Esta opción del menú lanza la ventana de "Crear Nuevos Grupos", donde se encontrarán los campos y opciones necesarias para añadir nuevos grupos al sistema.

Botón de borrado de grupos. Si se pulsa este botón se eliminarán aquellos grupos seleccionados en el ListBox *Groups*.



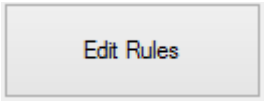
Lista de grupos de usuarios presentes en el sistema. Este campo permitirá a la aplicación mostrar todos aquellos grupos creados y añadidos al sistema Mecalux Failures Manager. Se trata de un control del tipo CheckedListBox, por lo que junto al nombre de grupo presentará un recuadro de selección. La selección de un grupo significará que se vincula al usuario seleccionado en el menú *Users*. Se permiten selecciones múltiples. La vinculación no se realizará hasta que el administrador guarde la configuración.



Botón de guardado de cambios. Este botón permite guardar todos cambios que se hayan realizado en el menú Data. Es decir, aquellos cambios de nombre, contraseña, etc, y las modificaciones de los grupos vinculados al usuario.

- En el *GroupBox* **Faults** se pueden encontrar los siguientes campos y controles:

Botón de asignación de averías. Este botón permite abrir la ventana de Asignación de averías, donde se permitirá vincular a usuarios y grupos con las averías deseadas.

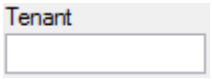


Botón de edición y creación de reglas. Este botón permite abrir la ventana de Edición de Reglas, donde se establecerán los periodos temporales (turnos de trabajo) en los que el usuario podrá atender averías.

- En el *GroupBox* **Login** se pueden encontrar los siguientes campos y controles:

Login. TextBox donde el administrador introduce su nombre de sesión.

Password. TextBox donde el administrador introduce su contraseña de inicio de sesión.



Tenant. TextBox donde el administrador introduce el nombre del cliente al que pertenece (Nombre de empresa).

Botón de conexión. Este botón permitirá el establecimiento de la conexión con el servidor web.

Botón de desconexión. Este botón permitirá la desconexión del cliente con el servidor web.

6.3.1.2.Añadir un nuevo usuario a Mecalux Failures Manager

Para añadir un nuevo usuario al sistema, será necesario pulsar sobre el botón  del menú Users. Esto abrirá la ventana de la siguiente ilustración:

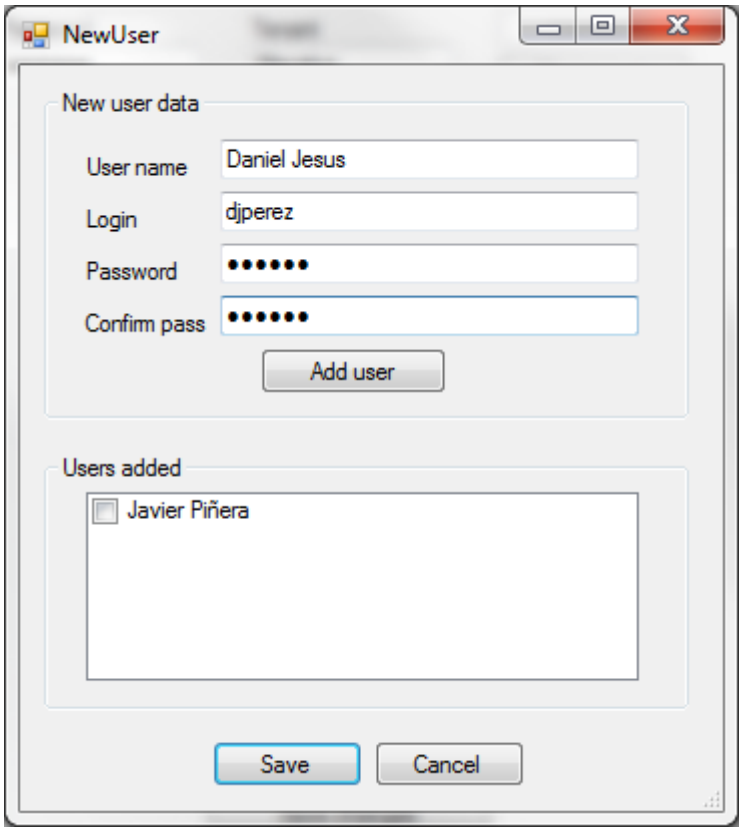


Ilustración 27: Ventana de creación de usuarios

Lo que debe hacer un administrador del sistema para añadir nuevos usuarios es introducir el nombre completo del trabajador en el campo *User name*. A continuación se encuentra el campo *Login*. Sirve para definir un nombre de usuario o de inicio de sesión. Será el valor que emplearán los usuarios para conectarse al servicio. En los campos *Password* y *Confirm password* se introducirá la contraseña empleada para iniciar sesión. Ambos valores han de ser iguales. Una vez hecho esto, será necesario pulsar el botón *Add user* para guardar provisionalmente los datos del nuevo usuario. Se podrá repetir este paso para añadir los usuarios necesarios a nuestra base de datos.

El siguiente paso será confirmar la creación de los usuarios. Para ello se dispondrá un CheckListBox con todos los trabajadores añadidos en el paso anterior. Será necesario seleccionar los deseados mediante un "check" en la casilla correspondiente y pulsar a continuación el botón *Save*. Si no hay ningún fallo en la escritura en la base de datos se mostrará un mensaje de éxito.

6.3.1.3. Añadir un nuevo grupo

El proceso de creación de un nuevo grupo en Mecalux Failures Manager se inicia desde la ventana principal de la aplicación, en el menú Data, mediante el botón  . Se mostrará entonces un diálogo que recoge toda la información del nuevo o nuevos grupos a crear:

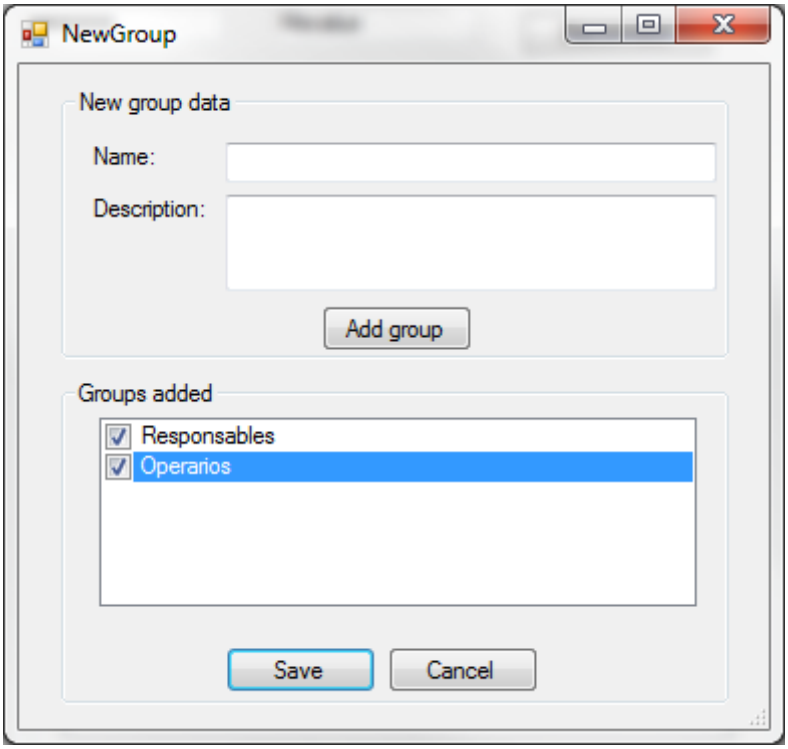


Ilustración 28: Ventana de creación de nuevos grupos

En este diálogo se recoge la siguiente información:

- **Name**: Nombre del grupo que se va a crear.
- **Description**: Descripción detallada de las actividades o tareas que va a realizar el grupo.
- **Groups added**: Lista provisional de los grupos a crear.

En proceso de creación de grupos es similar al comentado en el apartado anterior para los grupos. El administrador introducirá un nombre y una descripción del grupo a crear y pulsará el botón *Add group* para añadirlo a una lista provisional. Se podrá realizar este paso las veces deseadas. Para guardar los grupos creados en la base de datos, será

necesario seleccionar mediante un “check” aquellos que se deseen preservar y pulsar a continuación el botón *Save.* Si la creación se realiza correctamente se desplegará un mensaje de éxito.

6.3.1.4. Borrado de grupos creados

Si un grupo es innecesario o pasa a estar en desuso, se podrá eliminar del sistema siguiendo los siguientes pasos. En el menú *Data* se muestra una lista con todos los grupos añadidos a la aplicación. Se seleccionarán aquellos que se deseen borrar (se permite una selección múltiple) y se pulsará el botón  situado encima de la lista. Para ratificar el borrado se mostrará diálogo de confirmación.

6.3.1.5. Asignar a verías a un usuario o grupo.

El proceso de asignación de una avería a un usuario o grupo es iniciado desde la opción Assign Faults del menú principal de la aplicación. Al seleccionar esta opción, aparecerá un diálogo que permitirá seleccionar aquellos usuarios o grupos a los que se pretenden vincular averías:

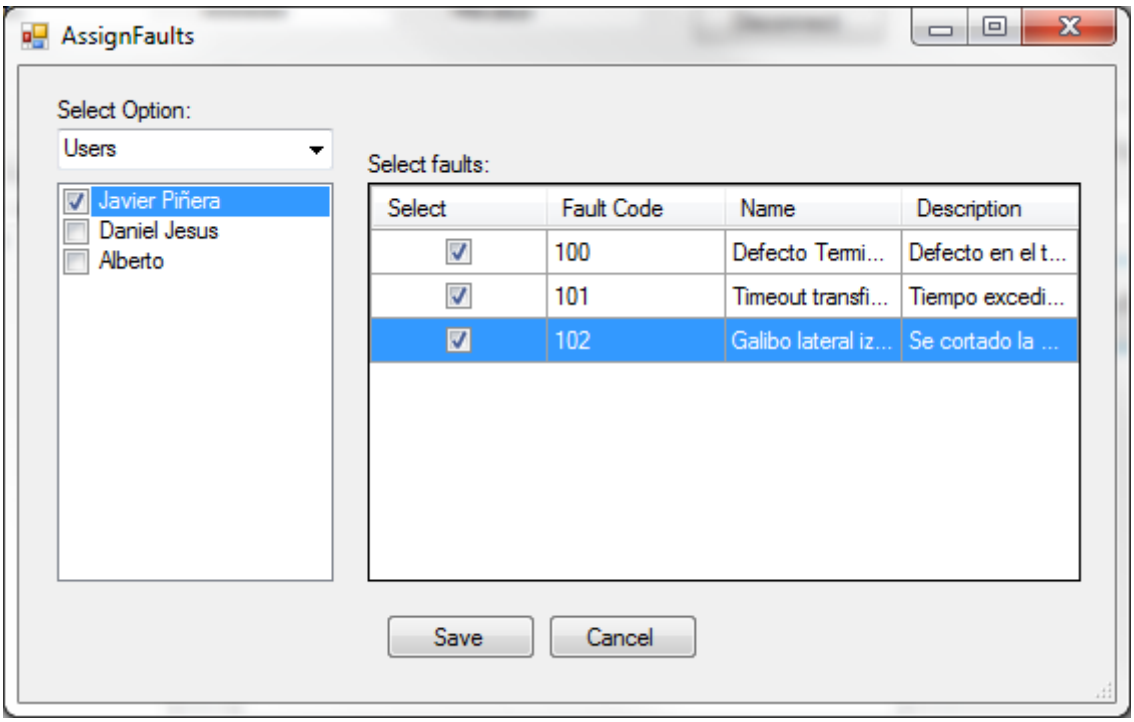


Ilustración 29: Diálogo de asignación de averías

Lo primero que se ha de hacer al desplegar este diálogo será escoger la opción de asignación, es decir, si las averías se vincularán a usuarios individualmente o a grupos directamente. Para ello se dispone de un *ComboBox* con ambas opciones.

Una vez seleccionado *Users* o *Groups*, se cargará una lista con todos aquellos presentes en el sistema. También se cargará en un *grid* una descripción de las averías que puede disparar el sistema. El objetivo será vincular a esos usuarios o grupos con averías acordes a su cualificación o responsabilidad. Para realizar esta vinculación solo será necesario "checkear" un usuario/grupo y la o las averías que se desee vincularle respectivamente. Para guardar la selección se pulsará el botón *Save.* Se mostrará un diálogo para confirmar el guardado.

6.3.1.6. Configuración de las reglas de notificación

El proceso de creación y configuración de las reglas de notificación se inicia desde la opción *Edit Rules* de la sección Faults del menú principal de la aplicación. Se mostrará entonces un diálogo que recogerá toda la información necesaria para la creación o modificación de las reglas para la notificación de averías:

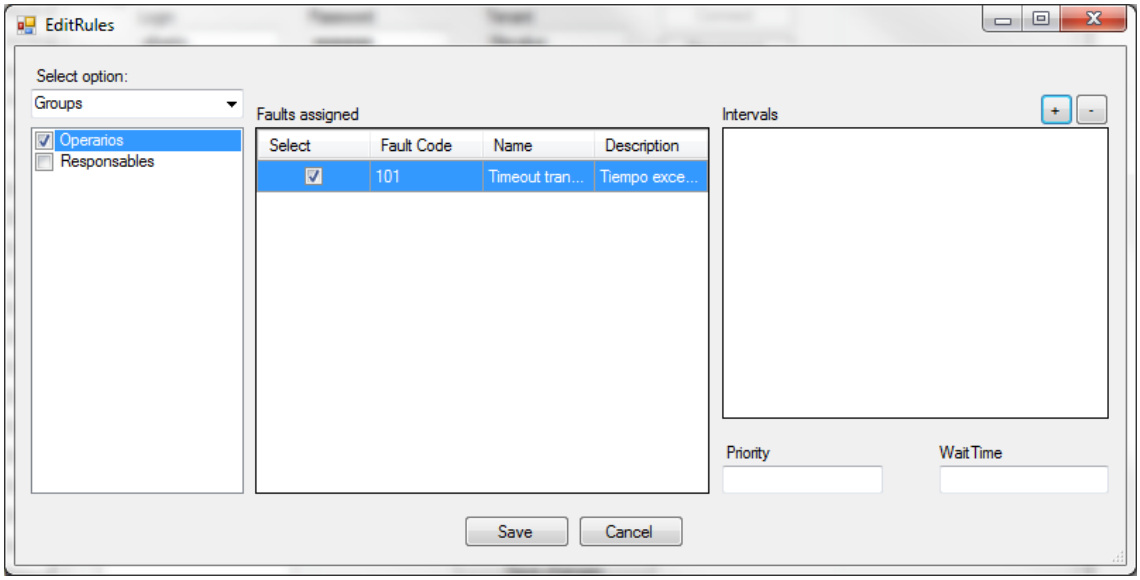


Ilustración 30: Diálogo de configuración de reglas de notificación

Este diálogo se compone de tres partes diferenciadas:

- Selección del usuario o grupo.
- Selección de la avería
- Introducción de los parámetros temporales y de prioridad.

El primer paso será elegir entre *Users* o *Groups* en el ComboBox de la parte superior izquierda. Dependiendo de la elección se cargarán aquellos usuarios o grupos que hayan sido añadidos al sistema previamente.

A continuación se seleccionará un usuario/grupo de la lista. Con esta selección se conseguirá cargar en el grid *Faults Assigned* aquellas averías que tenga asignadas dicha selección. Antes de establecer los parámetros temporales (turno de trabajo) se seleccionará aquella o aquellas averías que van a estar ligadas a este turno. Puede hacerse de forma individual, es decir, un usuario puede ocuparse de solventar una avería durante un determinado periodo y otra en otro periodo diferente.

Una vez seleccionada la o las averías correspondientes, se introducirán los parámetros temporales, es decir, las fechas de comienzo y fin del turno de trabajo. Para ello será necesario pulsar el botón  correspondiente a la ilustración siguiente:

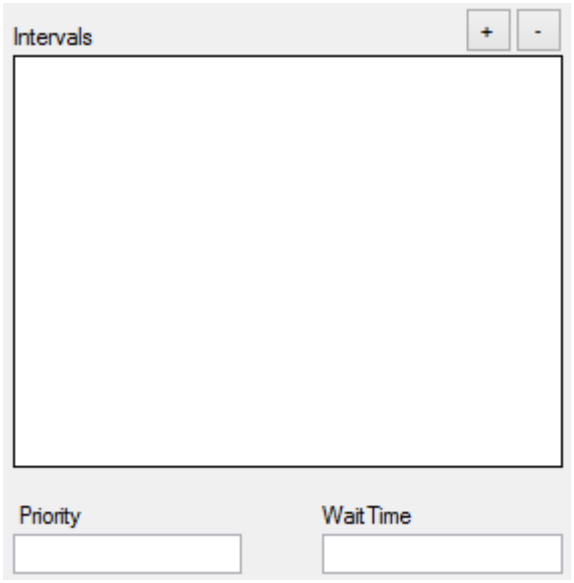


Ilustración 31: Recolección de datos temporales

La pulsación de este botón desplegará otro diálogo adicional donde se introducirán los siguientes datos:

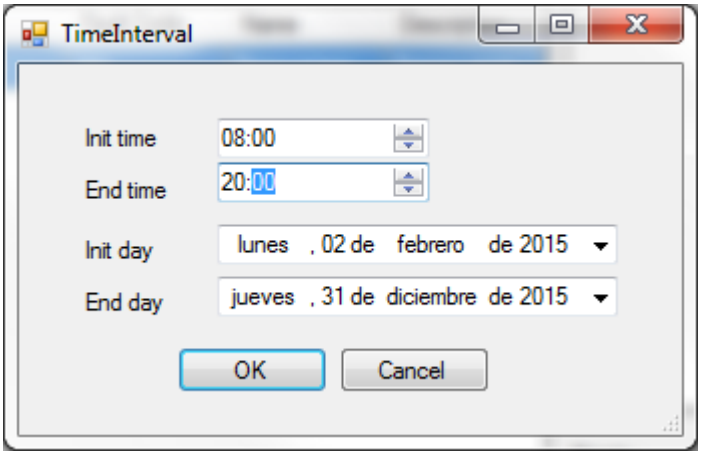


Ilustración 32: Introducción de los parámetros temporales

- Init time: Hora de comienzo del turno.
- End time: Hora de finalización del turno.
- Init day: Dia de comienzo del turno de trabajo.
- End day: Dia de finalización del turno de trabajo.

Para añadir un turno de trabajo a un usuario se introducirán los parámetros temporales que se consideren (hora de entrada y salida, y fecha de comienzo y fin del turno) y se pulsará el botón OK para guardar los cambios provisionalmente. Si por el contrario se pulsa el botón Cancel se volverá a la ventana anterior y los datos no se guardarán.

Si se ha pulsado el botón OK, los datos se almacenarán provisionalmente y la ventana se cerrará para volver al menú anterior. Si se desea eliminar un turno de trabajo por el motivo que sea será posible seleccionando aquel que desee borrar y pulsando a continuación el botón  de la sección *Intervals.*

El último paso para crear las reglas de notificación del usuario/grupo será introducir los valores de prioridad y de tiempo de espera.

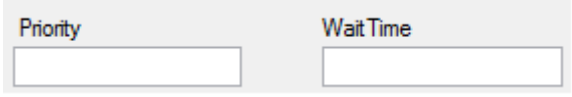


Ilustración 33: Atributos de prioridad y tiempo de espera

Con estos valores se especifica la prioridad que tendrá el usuario/grupo a la hora de atender una determinada avería, además del tiempo que se esperará para notificarle una avería atendiendo a tu tiempo de activación.

Para finalizar y guardar todos los cambios y valores introducidos se pulsará el botón *Save*. Si el guardado de los datos es correcto se mostrará un mensaje de información. Si por el contrario se pulsa el botón *Cancel* la ventana se cerrará y se perderán todos los datos introducidos.

6.3.2. Visualizador de averías (App móvil)

En este apartado se explicará cómo manejar la aplicación de visualización de avería. Como se comenta en apartados anteriores, se desarrolla mediante Phonegap, un framework de diseño de aplicaciones móviles a partir de lenguajes web.

En los siguientes apartados se mostrará el funcionamiento de la aplicación así como los diferentes menús que se puede encontrar el usuario.

6.3.2.1.Inicio de sesión en la aplicación

El proceso de inicio de sesión o registro del usuario en el visor de averías, se realizará una única vez siempre y cuando el usuario registrado no borre sus datos de la aplicación. La tarea de registro tendrá lugar en el menú que se muestra a continuación:

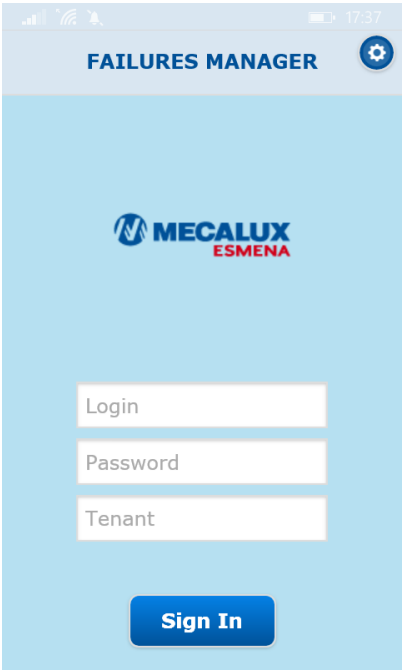


Ilustración 34: Menú de inicio de sesión

En el menú de la ilustración, se muestran tres partes diferenciadas. En la parte superior izquierda está disponible un botón  para acceder al menú de configuración. También se puede ver en la parte central tres TextBox donde se introducirán los datos del usuario que va a iniciar sesión. Pulsando el botón *Sign In*, se enviarán los datos al servidor para validar la conexión.

Una vez que el usuario es autenticado y validado en el sistema, se guardarán los datos del usuario en el terminal. El proceso de inicio de sesión se omitirá en los sucesivos arranques de la aplicación.

6.3.2.2. Menú de configuración

En este menú se reúnen las principales opciones de configuración de la aplicación. Dichas opciones son las siguientes:

- Configuración de la **dirección de conexión**.
- Activación o desactivación de **notificaciones**.
- **Borrado** de los **datos de usuario** del terminal.

Para introducir la dirección de conexión al servidor se proporcionan dos campos, donde el usuario puede introducir la dirección IP de conexión y el puerto (Port). Para guardar estos valores será necesario pulsar el botón *Save.*

Ilustración 35: Menú de configuración de la aplicación

Si lo que se desea es activar o desactivar la recepción de notificaciones, se proporciona un control de tipo "flip toggle" en el que se podrá elegir entre **ON** para recibir las notificaciones y **OFF** para no recibirlas. Las notificaciones estarán activadas por defecto.

Por último el usuario podrá eliminar todos los datos guardados en la aplicación (dirección, datos de usuario,…) pulsando el botón *Clear data.* Si se cambia de terminal se recomienda realizar esta acción.

6.3.2.3.Menú principal

Se considera menú principal a la vista general de averías. La única forma de acceder a este menú será validando al usuario que inicie la sesión. Este menú se utiliza para mostrar un resumen a modo de lista de las averías activadas que se han asignado al usuario. Consta de dos partes, por un lado un contador de averías, que indica el total de averías activadas y por otra parte la lista, un menú desplegable hacia abajo en el que se indica el identificador de máquina que dispara la avería y desplegando el menú el nombre y una breve descripción de dicha avería.

Si lo que se desea es ver el tipo de avería y la descripción, el usuario tendrá que pulsar sobre el nombre de la máquina, desplegando hacia abajo un pequeño menú en el que podrá visualizar dicha información.

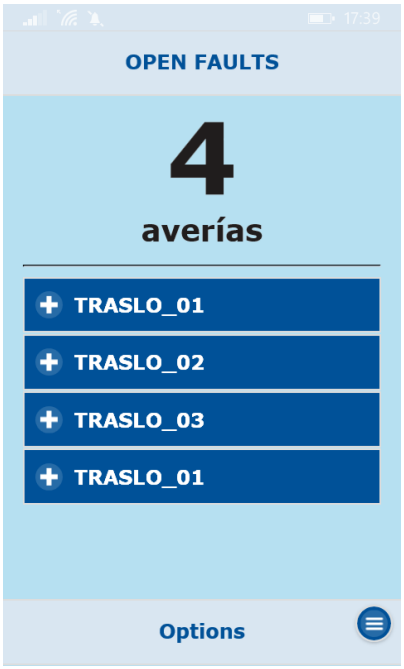


Ilustración 36: Menú principal. Vista de averías activas

Dentro del desplegable, se encuentra disponible un enlace que permite acceder a una vista de detalle de avería. En esta vista se pueden consultar todos los datos disponibles de la avería seleccionada (fecha de activación, máquina que la dispara, computador que recoge el fallo, etc…). Para más información ver apartado 6.3.2.4.

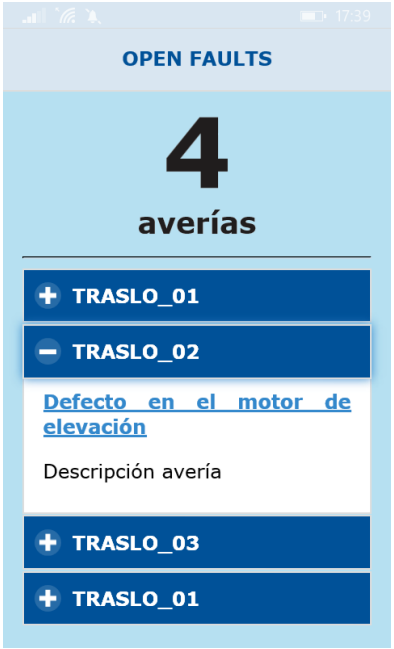


Ilustración 37: Menu principal. Vista de avería desplegada

Otro de los elementos que presenta el menú principal es un botón  que despliega un menú lateral en el que se muestra el resto de opciones de visualización. Ver apartado 6.3.2.5. para más detalles.

6.3.2.4. Ver detalle de avería

Para acceder a este menú será condición indispensable que el usuario pulse en el nombre de avería mostrada al desplegar la lista de incidencias. En ella se muestra la información completa de la avería, su código de error, nombre, fecha, sugerencias para solucionar dicha avería, etc.

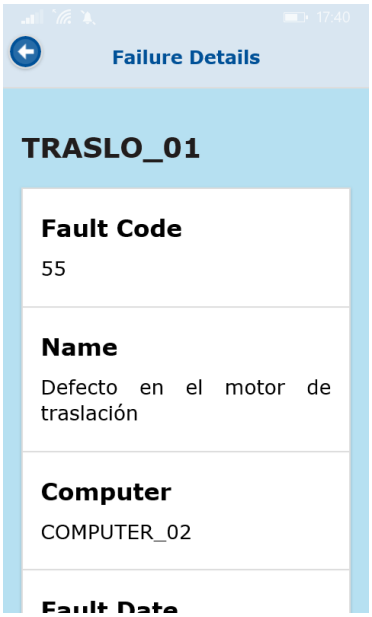


Ilustración 38: Vista de detalle de avería

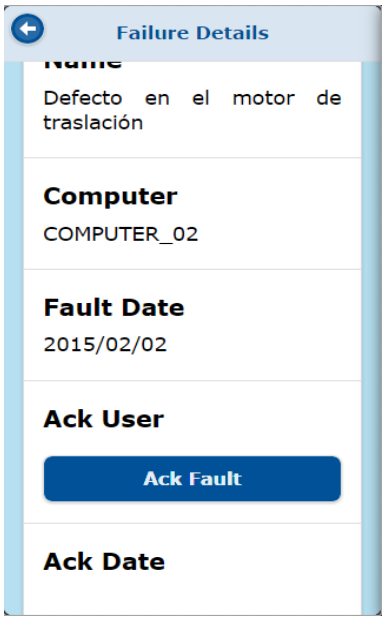


Ilustración 39: Vista de detalle de avería 2

En este menú el usuario también podrá confirmar la recepción de la avería. Para ello cuenta con un botón Ack User en el que se podrá indicar al sistema que el usuario se va a hacer cargo de dicha avería. Solo será necesario pulsar el botón y aceptar el mensaje de confirmación.

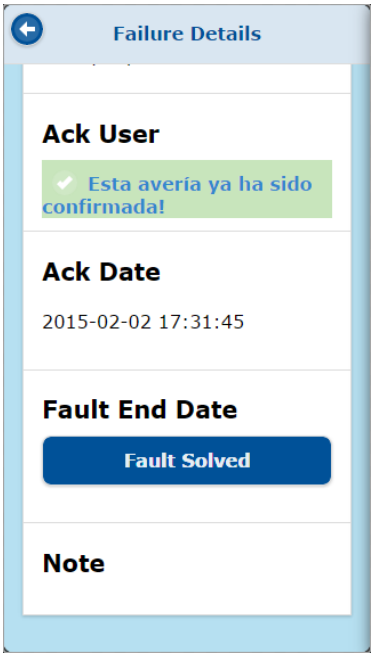


Ilustración 40: Vista de avería confirmada

6.3.2.5. Menú de opciones

El siguiente menú se despliega dentro del menú principal. En él se pueden ver las diferentes opciones que permite realizar la aplicación, que son las siguientes:

- **User faults**: Vista por defecto, muestra las averías activas y sin confirmar que tiene asignadas el usuario.
- **Ack faults**: En esta vista se mostrarán solo aquellas averías activas que el usuario ya ha confirmado mediante la pulsación del botón Ack User.
- **Filter faults**: Esta opción permite filtrar todas las averías que no sean la indicada. en el cuadro de texto que se despliega con la pulsación.
- **Logout**: Al pulsar esta opción el usuario podrá cerrar sesión. Esta opción borra los datos de dicho usuario, por lo que en el siguiente arranque de la aplicación tendrá que introducir esos datos.

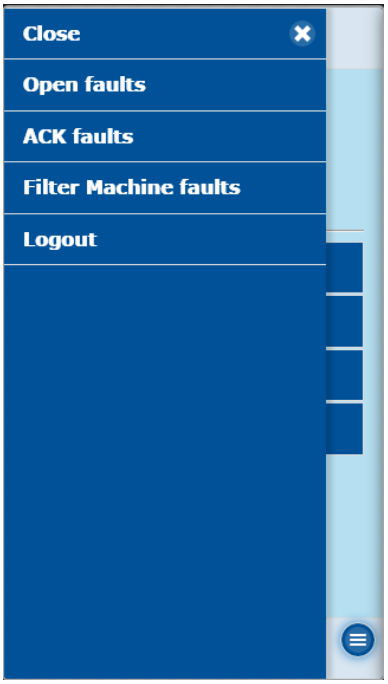


Ilustración 41: Menú de opciones

6.3.2.6.Salir de la aplicación

Este apartado solo será válido si la aplicación se está ejecutando en terminales con Windows Phone y Android.

Para salir de la aplicación solo será necesario pulsar el botón “back” del terminal. Si se realiza esta operación se mostrará el mensaje de confirmación siguiente:

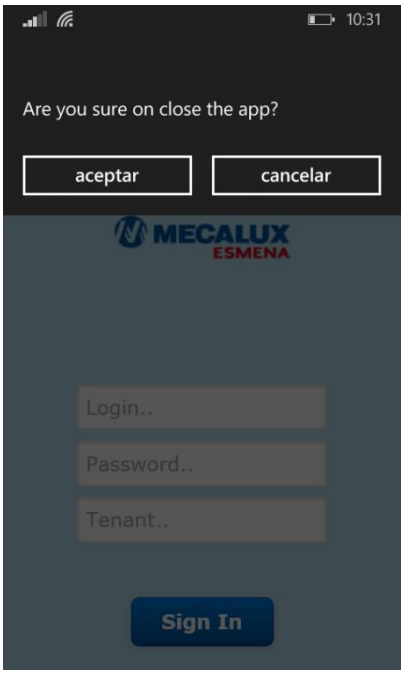


Ilustración 42: Cierre de la aplicación

Pulsando el botón “Aceptar” el usuario saldrá de la aplicación, conservando eso si sus datos guardados para próximos usos. Si pulsa “Cancelar” la aplicación permanecerá activa.

Si el usuario pulsa el botón “Home” o botón central, la aplicación pasará a segundo plano, pero seguirá ejecutándose. Será responsabilidad del usuario cerrarla correctamente.

MANUAL
DEL
PROGRAMADOR

7. MANUAL DEL PROGRAMADOR

7.1. Operaciones para la modificación del software

La aplicación se ha desarrollado utilizando el entorno de programación Visual Studio 2013. Este entorno de desarrollo crea un fichero con extensión *.sln* que contiene la información de la aplicación: ficheros que la componen y la relación entre los formularios y el código fuente. Se permite indicar que opciones de compilación son necesarias. La parte del proyecto referida a la aplicación móvil va aparte, pues tiene su propia estructura y compiladores.

El objetivo de este manual técnico es facilitar al programador la comprensión del código y dar facilidades a la hora de configurar los diferentes parámetros necesarios para la compilación y ejecución del código.

El código fuente de la aplicación está organizado en cuatro grandes bloques que se corresponden con otros tantos ficheros *.csproj*, excluyendo la aplicación móvil, cuya arquitectura no facilita un fichero de solución. Estos bloques se describen a continuación:

- **Client**: Este bloque contiene los ficheros que constituyen la aplicación de gestión de usuarios/grupos y averías. Para la creación de esta aplicación se ha añadido un nuevo proyecto de aplicación de Windows Forms a la solución. Para el acceso a este bloque deberá emplearse el fichero de proyecto *Client.csproj*

- **DataModel**: Este bloque contiene la biblioteca de clases donde se implementan las funcionalidades de la base de datos. Se ha desarrollado a través de una biblioteca de clases, añadida a la solución. Para la correcta visualización de este bloque será necesario disponer del paquete de ADO.NET Entity Framework.

- **SignalRSelfHosted**: Este bloque contiene los ficheros que constituyen el servicio web. Para el acceso a este bloque tendrá que emplearse el fichero de proyecto *SignalRSelfHosted.csproj*

- **FailuresApp**: Este bloque contiene los ficheros referentes a la aplicación móvil. Como se desarrolla bajo un entorno diferente a .NET no se ha vinculado a la

solución del resto del proyecto sino que se mantiene aparte. En apartados posteriores se explica con detalle la estructuración del código y los ficheros.

7.2. Configuración del software

Si se desea realizar alguna modificación en la aplicación o continuar el desarrollo en otro computador, será necesario añadir una serie de referencias y archivos al entorno de desarrollo para que la compilación posterior sea correcta.

7.2.1. Configuración de Visual Studio 2013 y MySQL

Para que el DataModel obtenga los datos correctamente de la base de datos MySQL será necesario añadir el siguiente "provider" al archivo **machine.config** encontrado en la carpeta de instalación del framework .NET.

```
<system.data>
  <DbProviderFactories>
      <add name="MySQL Data Provider" invariant="MySql.Data.MySqlClient"
description=".Net Framework Data Provider for MySQL"
type="MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data, Version=x.x.x.x,
Culture=neutral, PublicKeyToken=c5687fc88969c44d" />
  </DbProviderFactories>
</system.data>
```

Además de lo anteriormente comentado, será necesario instalar tanto el conector para .NET de MySQL como el plugin desarrollado para Visual Studio 2013. En el enlace que se proporciona en la bibliografía [ ] se encontrará toda la documentación necesaria. Se recomienda instalar la última versión estable siempre que sea posible.

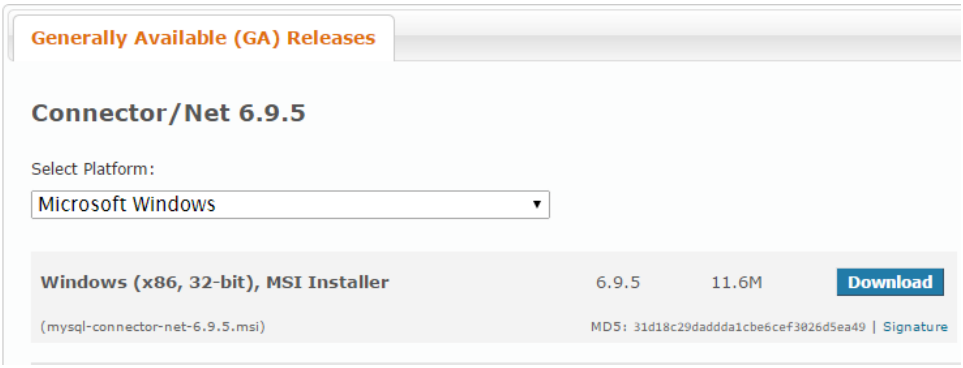


Ilustración 43: Descarga de conector MySQL para .NET

Para la añadir el complemento de MySQL para Visual Studio 2013 se facilita un enlace en la bibliografía **[10]** en la que se encuentra toda la información disponible. Se recomienda siempre que sea posible instalar la última versión estable publicada. El objetivo de este plugin es tener disponible en el entorno Entity Framework todas las opciones y herramientas de MySQL para poder reconocer e importar la base de datos sin problema.

Ilustración 44: Descarga plugin MySQL para Visual Studio

7.3. Ficheros de la aplicación

A continuación se enumeran y describen los componentes de cada uno de los grupos citados en el apartado anterior y que constituyen en conjunto Mecalux Failures Manager.

7.3.1. Ficheros de *Client*

La aplicación gestora de usuarios/grupos y averías, se desarrolla siguiendo la estructura de un programa de Windows Forms, cuyos componentes se describen a continuación:

- **AssignFaults**. Esta sección contiene los siguientes ficheros:
  - AssignFaults.Designer.cs: Contiene la definición e inicialización de los componentes de la interfaz de usuario de AssignFaults.

- AssignFaults.cs: Contiene las funciones y eventos que el usuario dispara al activar los componentes de la interfaz de AssignFaults.

- **EditRules**. Esta sección contiene los siguientes ficheros:
  - EditRules.Designer.cs: Contiene la definición e inicialización de los componentes de la interfaz de usuario de EditRules.
  - EditRules.cs: Contiene las funciones y eventos que el usuario dispara al activar los componentes de la interfaz de EditRules.

- **Login**. Esta sección contiene los siguientes ficheros:
  - Login.Designer.cs: Contiene la definición e inicialización de los componentes de la interfaz de usuario de Login.
  - Login.cs: Contiene las funciones y eventos que el usuario dispara al activar los componentes de la interfaz del registro de usuario Login.

- **Main.** Esta sección contiene los siguientes ficheros:
  - Main.Designer.cs: Contiene la definición e inicialización de los componentes de la ventana principal Main.
  - Main.cs: Contiene las funciones y eventos que el usuario dispara al activar los componentes de la ventana principal Main.

- **NewGroup**. Esta sección contiene los siguientes ficheros:
  - NewGroup.Designer.cs: Contiene la definición e inicialización de los componentes de la interfaz de usuario de NewGroup.
  - NewGroup.cs: Contiene las funciones y eventos que el usuario dispara al activar los componentes de la interfaz de NewGroup.

- **NewUser**. Esta sección contiene los siguientes ficheros:
  - NewUser.Designer.cs: Contiene la definición e inicialización de los componentes de la interfaz de usuario de NewUser.
  - NewUser.cs: Contiene las funciones y eventos que el usuario dispara al activar los componentes de la interfaz de NewUser.

- **TimeInterval**. Esta sección contiene los siguientes ficheros:
  - TimeInterval.Designer.cs: Contiene la definición e inicialización de los componentes de la interfaz de usuario de TimeInterval.
  - TimeInterval.cs: Contiene las funciones y eventos que el usuario dispara al activar los componentes de la interfaz de TimeInterval.

- **Packages.config**: Este fichero contiene una lista de los paquetes instalados en el programa a través del administrador de paquetes NuGet.

- **App.config**: Fichero auto generado.

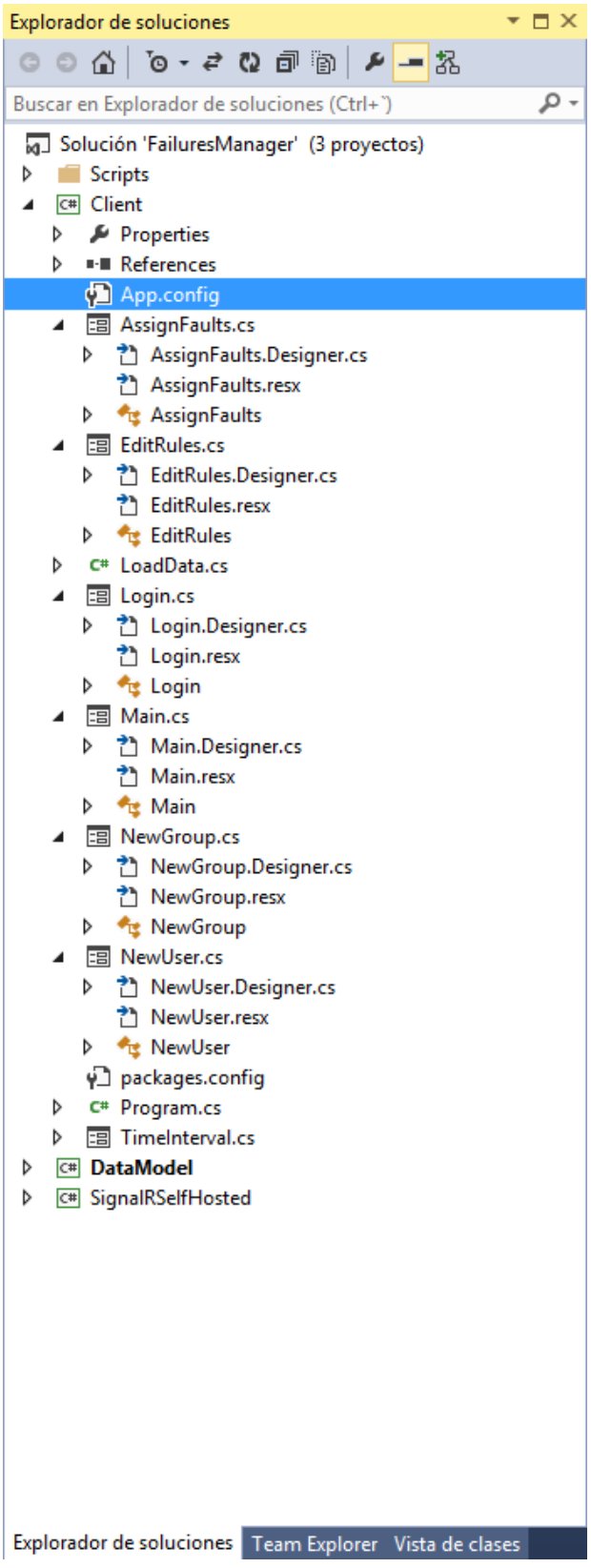


Ilustración 45: Lista de ficheros de la aplicación Cliente

7.3.2. Ficheros de *DataModel*

- **App.config:** Este fichero contiene las cadenas de conexión con la base de datos y la descripción de los proveedores de datos para Entity Framework.

- **DataModel.edmx:** Este fichero muestra una visualización de las entidades que forman la base de datos y sus relaciones.
    - DataModel.Designer.cs: Este fichero contiene las definiciones y propiedades de las entidades que componen la base de datos.

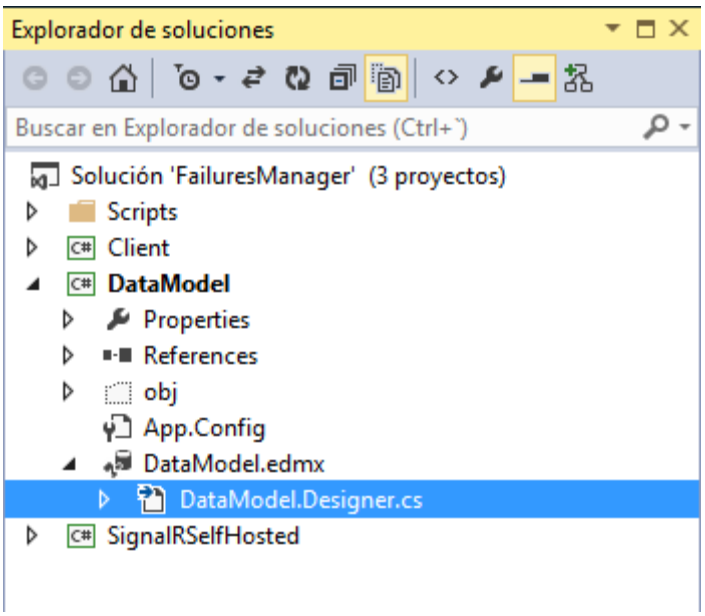


Ilustración 46: Lista de ficheros de DataModel

7.3.3. Ficheros de *SignalRSelfHosted*

El bloque SignalRSelfHosted se desarrolla bajo una aplicación de consola de Windows. Se añadirá al proyecto el paquete de SignalR, a través del administrador de paquetes NuGet. Los ficheros que componen el proyecto son los siguientes:

- **Authorization.cs:** Este fichero contiene la implementación de los métodos necesarios para validar o no a los usuarios en el sistema.
- **ConnectionMapping.cs**: Este fichero contiene la implementación de los métodos que nos permiten vincular a los usuarios del sistema con un identificador de conexión.

- **DBRules.cs**: Este fichero contiene la implementación de los métodos encargados de la asignación de averías a grupos y usuarios.

- **Hub.cs:** Este fichero es una de las bases de SignalR. Permite definir métodos que pueden ser llamados por la aplicación cliente de una forma sencilla.

- **Notification**: Este fichero contiene la definición de los datos usados en las notificaciones.

- **NotificationMapping**: En fichero contiene la implementación de los métodos necesarios para la vincular las notificaciones a los diferentes usuarios.

- **NotificationSender**: En fichero contiene los métodos necesarios para el envío de notificaciones a los clientes del servicio.

- **Program**.**cs**: Este fichero inicializa la aplicación de consola y el servicio web. Además de dos hilos de ejecución paralelos para la detección y envío de notificaciones.

- **Startup**: En este fichero se mapea las conexiónes del hub SignalR y se permite modificar la configuración del servicio.

- **UserNotificationMapping**

- **Packages.config**: Este fichero contiene una lista con los paquetes instalados en el proyecto a través del administrador de paquetes NuGet.

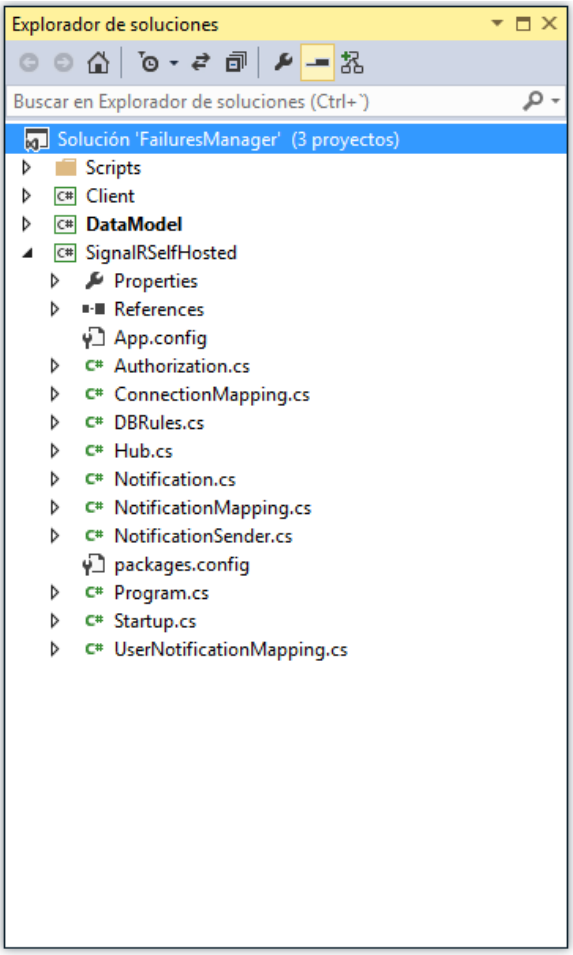


Ilustración 47: Lista de ficheros de SignalRSelfHosted

7.3.4. Ficheros de *FailuresApp*

El bloque FailuresApp engloba todos los ficheros de la aplicación móvil. La aplicación parte de una plantilla generada por Apache Cordova (Phonegap) por lo que en el presente apartado solo se hará mención a los ficheros desarrollados por el diseñador del proyecto.

Dentro de la organización del proyecto de Phonegap, distinguiremos tres puntos o apartados claves. Se trata de la carpeta *www*, la carpeta *plugins* y el fichero *config.xml.*

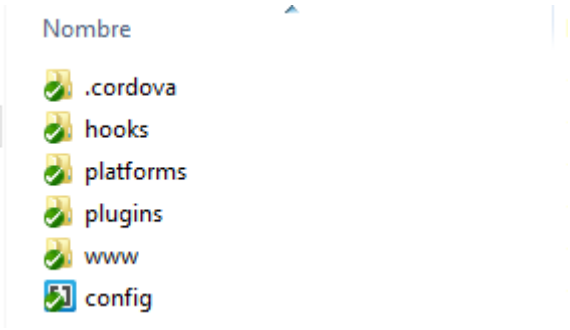


Ilustración 48: Estructura de ficheros de la aplicación móvil

En la carpeta *www* se guardarán todos los archivos que se refieren al código de la aplicación, como son los ficheros *.html*, *.js* y *.css* necesarios en la elaboración de la App.

Comentar que los ficheros .css referentes a jQuery Mobile son parte de una plantilla configurada de forma automática para que muestre de la forma deseada los controles y elementos del código html. Esta plantilla se obtiene gracias a la aplicación remota que ofrece jQuery en su página web **[12]**.

En la carpeta plugins se almacenarán todos aquellos plugins instalados por el desarrollador. Los plugins se encargan de dotar a la aplicación Phonegap de características adicionales que no están incluidas de serie. Por último el fichero config.xml tiene la labor de indicar la configuración de la App y los plugins que han de añadirse cuando se realiza una compilación en la nube a través de Phonegap Build, la herramienta de Adobe para la compilación de aplicaciones Phonegap. En el apartado siguiente se explica con más detalle en contenido de estas carpetas.

7.3.4.1.Directorio www

- **Index.html**: En este fichero se almacena todo el código html que mostrará la aplicación. Las aplicaciones en Phonegap están pensadas para reunir todo el código html en un solo fichero.

- **Css/index.css**: Este fichero almacena parte de la información referente a los estilos de la aplicación.

- **Css/jquery.mobile.structure-1.4.5.min**: Se trata de una plantilla donde se almacena la estructura de los estilos obtenidos con la aplicación de jQuery ThemeRoller.

- **Css/themes/customTheme5.min**: Se trata de la otra parte de la plantilla obtenida con ThemeRoller donde se especifican más parámetros de los estilos de la App.

- **Js/index.js**: Este fichero javascript contiene todos los métodos que harán funcionar la aplicación, desde la conexión con el servidor hasta la modificación de los elementos html.

- **Js/jquery.signalR-2.0.0:** Este fichero proporciona los métodos necesarios para dotar a la aplicación de características SignalR y poder establecer conexión con el servicio web.

- **Js/jquery-1.10.2**: Este fichero contiene todas las opciones y métodos que proporciona la librería de javascript jQuery.

El resto de ficheros de la aplicación son aportados por la plantilla generada por Phonegap al crear un nuevo proyecto. No se han modificado.

7.4. SignalRSelfHosted

El eje central del sistema será la aplicación servidora o servicio web, la cual será la encargada de tramitar las peticiones que lleguen desde los diferentes clientes y también realizar el envío de notificaciones de averías a dichos clientes.

Para este proyecto se plantea el desarrollo de este servicio bajo una aplicación de consola de Windows ya que no será necesario ningún interfaz visual, simplemente que la aplicación se inicie y se detenga en caso necesario.

En un futuro, como posible ampliación se ha previsto migrar el sistema hacia una aplicación web pura, es decir, que no necesite de un terminal para su ejecución sino que aproveche las ventajas ofrecen las plataformas “cloud” actuales como Azure o similares.

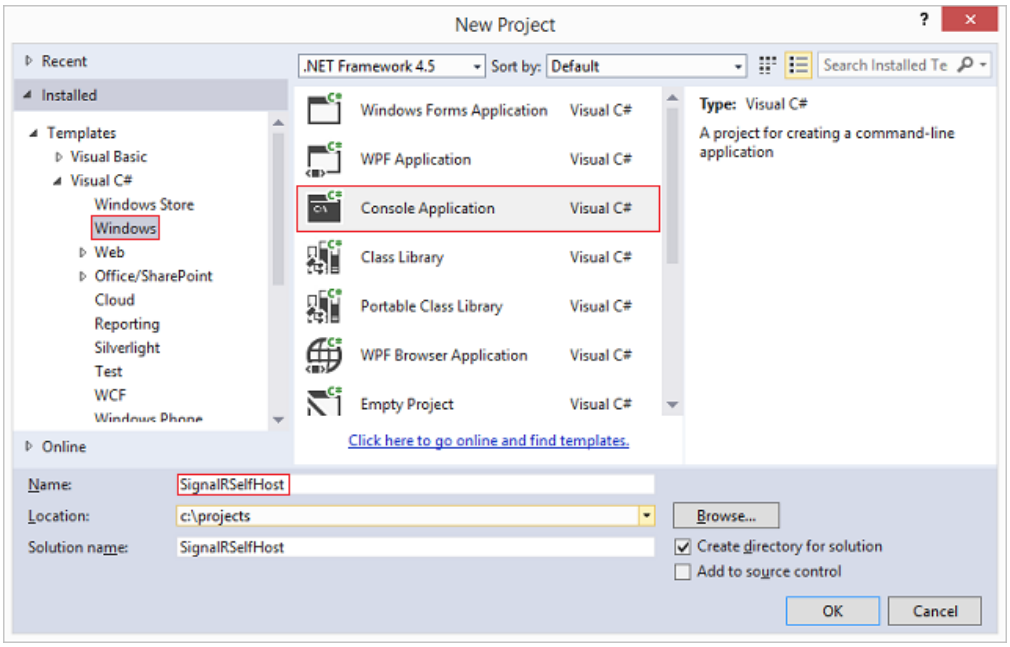


Ilustración 49: Creación aplicación consola Visual Studio 2013

Aunque se trata de una aplicación de consola sobre ella corre un servicio web SignalR. Para añadir esta característica es necesario añadir una serie de paquetes a través del Administrador de paquetes NuGet. Añadiendo la siguiente línea de código en NuGet se descargarán todos aquellos paquetes necesarios para dotar a la aplicación de las características que ofrece SignalR.

```
Install-Package Microsoft.AspNet.SignalR.SelfHost
```

Además de las librerías de SignalR será necesario añadir otras que proporcionen funcionalidades básicas para el funcionamiento de la aplicación.

```
Install-Package Microsoft.Owin.Cors
```

Este comando añade la librería Microsoft.Owin.Cors al proyecto. Se usará para dar soporte "cross-domain" es decir, que aplicaciones y clientes SignalR hospedados en diferentes dominios puedan comunicarse.

A continuación se agregarán a la aplicación de consola las siguientes clases. En primer lugar una "Clase de Inicio OWIN". En esta clase se registran las comunicaciones del servicio web y se configuran las opciones que ofrece. Se debe invocar esta clase cuando se desea arrancar el servicio.

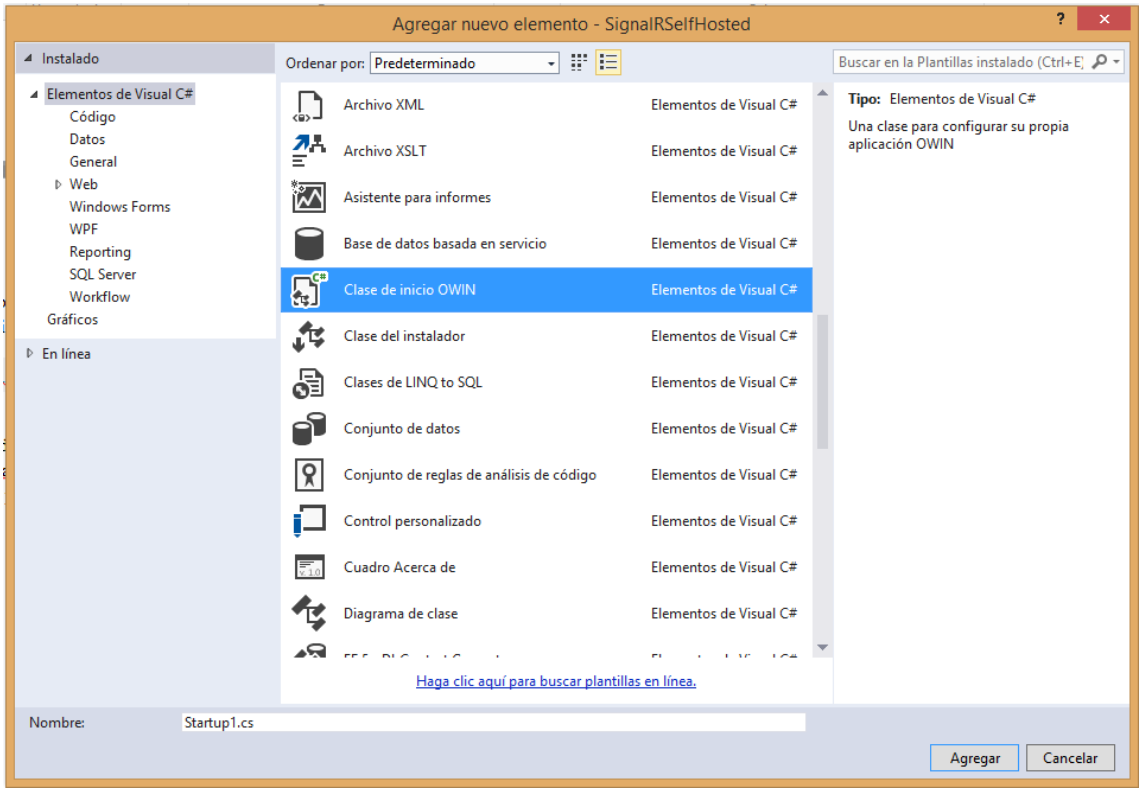


Ilustración 50: Agregar clase OWIN a proyecto SignalR

La siguiente clase necesaria para comenzar es "SignalR Hub Class". El **Hub** será en centro de todas las comunicaciones entre cliente y servidor. Obedece a un sistema de peticiones remotas, por lo que el cliente podrá realizar llamadas a todos aquellos métodos que se definan en su interior. Para dotar a nuestra clase de las características de un Hub bastará que herede sus propiedades de la clase "Hub". En el código mostrado a continuación se muestra un ejemplo sencillo con el que comenzar una aplicación más compleja.

Si la plantilla no está disponible en el Visual Studio instalado, el código necesario para comenzar será muy sencillo:

```
EJEMPLO

using System;
using System.Web;
using Microsoft.AspNet.SignalR;
namespace SignalRChat
{
    public class ChatHub : Hub
    {
        public void Send(string name, string message)
        {
```

```
            // Call the broadcastMessage method to update clients.
            Clients.All.broadcastMessage(name, message);
        }
    }
}
```

Ilustración 51: Plantilla ejemplo del Hub SignalR.

7.4.1. Funciones del Hub

La clase Hub permitirá comunicar mediante llamadas remotas al cliente y al servicio web SignalR de forma bidireccional y así intercambiar información. Las funciones habilitadas en el proyecto para realizar dicha tarea son las siguientes:

- `public override Task OnConnected()`

  Este método se dispara automáticamente cuando un usuario o cliente se conecta al servicio.

- `public override Task OnDisconnected(bool stopCalled)`

  Método ejecutado automáticamente cuando un usuario o cliente se desconecta del servicio.

- `public override Task OnReconnected()`

  Método ejecutado automáticamente cuando un usuario o cliente se reconecta en un tiempo menor al fijado por el servicio. Este tiempo de reconexión está definido en el método *Configuration* de la clase Startup. Puede ser modificado por el usuario.

- `public List<galileo_fault_def> FaultsDef()`

  Este método busca en la base de datos las definiciones de todas las averías que pueden producirse en el almacén. Estas definiciones se encuentran almacenadas en la tabla *galileo_fault_def* de la base de datos.

  Devuelve:

  Lista de objetos galileo_fault_def con toda la información requerida de las averías que puede disparar el sistema.

- `public List<int> FaultDefUser(string login, string tenant)`

  Función encargada de buscar en la base de datos el "fault_code" de todas las averías asignadas al usuario.

  Parámetros:

  login: cadena de caracteres que representa el nombre de usuario del trabajador en el sistema.

  tenant: cadena de caracteres que representa en nombre de la empresa donde trabaja el cliente.

  Devuelve:

  Lista de enteros que representa los "fault_code" de las averías asignadas al usuario.

- `public List<int> FaultDefGroup(string groupName, string tenant)`

  Esta función se encarga de buscar en la base de datos el "fault_code" de cada avería asignada al grupo especificado.

  Parámetros:

  groupName: cadena de caracteres que representa el nombre del grupo para el cual se buscan los códigos de avería.
  tenant: cadena de caracteres que representa el nombre de la empresa del grupo especificado.

  Devuelve:

  Lista de enteros que representa los "fault_code" de las averías asignadas al grupo.

- `public void GetAckUserFaults(string userLogin, string tenant)`

  Esta función se encarga de realizar la petición a la base de datos de todas las averías ya confirmadas (ACK) por parte del usuario especificado.

  Parámetros:

  userLogin: cadena de caracteres con el nombre de usuario del empleado que realiza la conexión.
  tenant: cadena de caractres que representa en nombre de la empresa donde trabaja el cliente especificado.

Devuelve:

Nada. La información de averías obtenida no la devuelve explícitamente esta función, sino que el Hub se encarga de enviarla al cliente cuando esté disponible.

- `public void GetOpenFaultsUser(string userLogin, string tenant)`

Esta función se encarga de buscar en la base de datos todas las averías activas y sin confirmar que tiene el usuario especificado.

Parámetros:

userLogin: cadena de caracteres con el nombre de usuario del empleado que realiza la conexión.
tenant: cadena de caracteres que representa en nombre de la empresa donde trabaja el cliente especificado.

Devuelve:

Nada. La información de averías obtenida no la devuelve explícitamente esta función, sino que el Hub se encarga de enviarla al cliente cuando esté disponible.

- `public List<string> GetFaultDetails(int fault_key)`

Esta función se encarga de buscar en la base de datos la información detallada de la avería especificada.

Parámetros:

fault_key: Entero que representa el identificador único de la avería disparada por el sistema.

Devuelve:

Lista de *string* con la información detallada de la avería indicada.

- `public void GetMachineFaults(string idMachine, string userlogin, string tenant)`

Esta función se encarga de buscar la información de todas las averías producidas por una máquina especificada.

Parámetros:

idMachine: Cadena de caracteres que representa el identificador de la máquina.
userLogin: cadena de caracteres con el nombre de usuario del empleado que realiza la conexión.
tenant: cadena de caracteres que representa en nombre de la empresa donde trabaja el cliente especificado.

Devuelve:

Nada. La información de averías obtenida no la devuelve explícitamente esta función, sino que el Hub se encarga de enviarla al cliente cuando esté disponible.

- `public void CreateUsers(IEnumerable<user> newPeople, string tenantName)`

Este método se encarga de crear un número determinado de usuarios para el cliente (tenant) indicado.

Parámetros:

newPeople: Lista (IEnumerable) de objetos de la clase *user*, que contiene los datos de los usuarios que se van a crear en la base de datos.

tenantName: Cadena de caracteres que representa el nombre del cliente (tenant) para el cual se crearán los usuarios.

Devuelve:

Nada.

- `public void DeleteUser(IEnumerable<string> logins, string tenantName)`

Este método se encarga de borrar de la base de datos un determinado número de usuarios para el cliente (tenant) especificado.

Parámetros:

logins: Lista de *string* con los nombres de usuario de los empleados a borrar de la base de datos.
tenantName: Cadena de caracteres que representa el nombre del cliente (tenant) para el cual se borrarán los usuarios.

Devuelve:

Nada.

- `public void UpdateUser(string name, user modUser, string tenantName)`

Este método se encarga de actualizar con nuevos datos, los ya presentes en la bas de datos, para un usuario y cliente (tenant) especificado.

Parámetros:

name: Cadena de caracteres que representa el nombre de usuario "antiguo" del empleado que se va a modificar.
modUser: Objeto de la clase *user* con la nueva información de usuario.
tenantName: Cadena de caracteres con el nombre del cliente (tenant) al que pertenece el usuario.

Devuelve:

Nada.

- `public void CreateGroup(IEnumerable<group> newGroups, string tenantName)`

Esta función se encarga de crear un determinado número de grupos en la base de datos para el cliente (tenant) indicado.

Parámetros:

newGroups: Lista (IEnumerable) de objetos de la clase group que contiene la información (nombre y descripción) de los grupos a crear.
tenantName: Cadena de caracteres que representa el nombre del cliente (tenant) para el que se van a crear los grupos.

Devuelve:

Nada.

- `public void DeleteGroup(IEnumerable<string> nameGroup, string tenantName)`

Esta función se encarga de borrar de la base de datos los grupos especificados para el cliente (tenant) indicado.

Parámetros:

newGroups: Lista (IEnumerable) de *string* con los nombres de los grupos que se van a borrar.
tenantName: Cadena de caracteres que representa el nombre del cliente (tenant) para el que se van a borrar los grupos.

Devuelve:

Nada.

- `public void LinkUsersToGroup(string login, IEnumerable<string> groupNames, string tenant)`

  Este método se encarga de vincular a un usuario determinado al grupo o lista de grupos que se desee.

  Parámetros:

  login: Cadena de caracteres que representa en nombre de usuario a vincular.

  groupNames: Lista (IEnumerable) de string con los nombres de los grupos a los que se vinculará el usuario.

  tenant: Cadena de caracteres que representa el nombre del cliente (tenant) para el que se realizará la vinculación.

  Devuelve:

  Nada.

- `public void CreateRule(string name,string tenant, rule newRule, List<time_interval> intervals,List<int> faultsAssigned,bool option )`

  Esta función se encarga de asignar a los usuarios las reglas temporales necesarias para el envío de notificaciones cuando se producen averías en el almacén.

  Parámetros:

  name: Cadena de caracteres que representa el nombre de usuario para el que se creará la regla.

  tenant: Cadena de caractres que representa el nombre del cliente (tenant) donde trabaja el usuario.

  newRule: Objeto de la clase *rule* con la información de la regla a crear.

  intervals: Lista de objetos de la clase time_interval con los parámetros temporales que se asignarán al usuario (turno de trabajo).

  faultsAssigned: Lista de enteros con el identificador de averías que tiene asignadas ese usuario.

  Devuelve:

  Nada.

- `public IDictionary<string,string> GetUsers2(string tenant)`

Esta función se encarga de buscar todos los nombres de usuario (login) almacenados en la base de datos para un cliente (tenant) especificado.

Parámetros:

tenant: Cadena de caracteres que representa el nombre del cliente (tenant) para el cual se buscarán los nombres de usuario.

Devuelve:

Un objeto diccionario en el que el parámetro *key* es el nombre de inicio de sesión y el *value* es el nombre del empleado.

- `public IEnumerable<string> GetGroups(string tenant)`

Esta función se encarga de buscar todos los nombres de los grupos asociados al cliente (tenant) indicado.

Parámetros:

tenant: Cadena de caracteres que representa el nombre del cliente (tenant) del cual se buscará el nombre de los grupos.

Devuelve:

Lista (IEnumerable) de *string* que contiene el nombre de todos los grupos vinculados al cliente (tenant) indicado.

- `public IEnumerable<user> UserData(string login, string tenant)`

Esta función se encarga de buscar en la base de datos toda la información referida al usuario del cliente (tenant) especificado.

Parámetros:

login: Cadena de caracteres que representa el nombre de inicio de sesión del usuario para el que se piden los datos.
tenant: Cadena de caracteres que representa el nombre del cliente (tenant).

Devuelve:

Un objeto de la clase *user* con toda la información almacenada del usuario indicado.

- `public List<string> GetUserGroups(string login, string tenant)`

Este método se encarga de buscar en la base de datos a que grupos está vinculado un determinado usuario.

Parámetros:

login: Cadena de caracteres que representa el nombre de inicio de sesión del usuario.
tenant: Cadena de caracteres que representa el nombre de cliente (tenant) en el que se buscarán esos grupos.

Devuelve:

Lista de *string* con los nombres de los grupos a los que el usuario especificado está vinculado.

- `public string AckUserFault(string userLogin, string tenant, int faultkey)`

Esta función se encarga de indicar a la base de datos que el usuario ya ha recibido la notificación y confirmado la avería.

Parámetros:

userLogin: Cadena de caracteres que representa el nombre de inicio de sesión del usuario que confirma la avería.
tenant: Cadena de caracteres que representa el cliente (tenant) donde trabaja el usuario.
faultkey: Entero que representa el identificador único de la avería a confirmar.

Devuelve:

Un *string* que representa la fecha y hora en la que se realiza la confirmación de la avería.

- `public void UserSelectFaultDef(string login, string tenant, List<int> faultsCode)`

Este método se encarga de asignar o vincular a un usuario una serie de averías indicadas.

Parámetros:

login: Cadena de caracteres que representa el nombre de inicio de sesión del usuario.
tenant: Cadena de caracteres que representa el nombre de cliente (tenant) donde trabaja el usuario.

faultsCode: Lista de enteros con el identificador de cada avería que se va a vincular al usuario.

Devuelve:

Nada.

- `public void GroupSelectFaultDef(string groupName, string tenant, List<int> faultsCode)`

  Esta función se encarga de asignar o vincular a un usuario una serie de averías indicadas.

  Parámetros:

  groupName: Cadena de caracteres que representa el nombre de grupo al que se vincularán la o las averías indicadas.
  tenant: Cadena de caracteres que representa el nombre de cliente (tenant) del grupo indicado.
  faultsCode: Lista de enteros con el identificador de cada avería que se va a vincular al grupo.

- `public void AddNotificationUri_User(string uri, string appPlatform, string login, string tenant)`

  Esta función se encarga de vincular una dirección URI a un usuario de forma que pueda recibir notificaciones cuando se registra e inicia la plataforma de notificaciones.

  Parámetros:

  uri: Cadena de caracteres que representa una URI, es decir, una dirección de red asociada a la plataforma de notificaciones del terminal de visualización de averías.
  appPlatform: Cadena de caracteres donde se indica el nombre de la plataforma de notificaciones de la presente URI. Los valores posibles son: Android, iOS o Win32NT.
  login: Cadena de caracteres que representa el nombre de inicio de sesión del usuario que se vinculará a la URI.
  tenant: Cadena de caracteres que representa el nombre del cliente (tenant) donde trabaja el usuario.

  Devuelve:

  Nada.

7.4.2. Funciones de Authorization

En esta clase de implementan los métodos necesarios para validar a los usuarios conectados a través de los diferentes clientes. Para validar a los usuarios se analiza la cadena de texto "querystring" incluida en la url de la petición HTTP. Debido a diferencias entre las peticiones realizadas por la aplicación móvil y el cliente de Windows se han definido dos métodos de validación diferentes, pero con el mismo objetivo y resultado. A continuación se describen estas variantes.

- `public static bool validate(string login, string password, string tenant)`

  Esta función se encarga de validar al usuario indicado, es decir, busca en la base de datos un usuario con el *login* y *password* para el cliente (tenant) indicado.

  Parámetros:

  login: Cadena de caracteres que representa el nombre de inicio de sesión del usuario.
  password: Cadena de caracteres que representa la contraseña de inicio de sesión del usuario.
  tenant: Cadena de caracteres que representa el nombre del cliente (tenant) donde trabaja el usuario.

  Devuelve:

  Bool. Valor booleano que representa si el usuario existe o no. Si el valor es *true* el usuario existe. *False* en caso contrario.

- `public static string[] connectionQueryString(string query)`

  Esta función se encarga de buscar el nombre de sesión y contraseña de un usuario dentro de la cadena de texto de una petición HTTP, cuando el inicio de sesión se realiza desde la aplicación móvil. A su vez este método llama a la función *validate* para comprobar si el usuario es válido.

  Parámetros:

  query: Cadena de caracteres que representa una parte de la URL de una petición HTTP.

  Devuelve:

  Array de *string*. En la posición 0 del array estará el valor *true* o *false*, indicando que el usuario existe o no respectivamente. En la posición 1 estará el nombre de sesión del usuario y en la posición 2 su contraseña.

- `public static string[] netConnectionQueryString(string login, string password, string tenant)`

Esta función se encarga de buscar el nombre de sesión y contraseña de un usuario dentro de la cadena de texto de una petición HTTP, cuando el inicio de sesión de realiza desde el cliente de Windows (Gestor de usuarios/grupos y averías).

Parámetros:

login: Cadena de caracteres que representa el nombre de sesión del usuario.
password: Cadena de caracteres que representa la contraseña de usuario para el inicio de sesión.
tenant: Cadena de caracteres que representa el nombre de cliente (tenant) donde trabaja el usuario.

Devuelve:

Array de *string*. En la posición 0 del array estará el valor *true* o *false*, indicando que el usuario existe o no respectivamente. En la posición 1 estará el nombre de sesión del usuario y en la posición 2 su contraseña.

- `public static void Log(string user, string tenant,string logMessage)`

Función encargada de registrar en un documento de texto las conexiones y desconexiones que se producen el servicio, guardando el nombre de sesión del usuario, el cliente (tenant) al que pertenece y la fecha y hora.

Parámetros:

user: Cadena de caracteres que representa el nombre de inicio de sesión del usuario.
tenant: Cadena de caracteres que representa el nombre de cliente (tenant) donde trabaja el usuario.
logMessage: Cadena de caracteres que indica si el usuario se ha conectado o desconectado.

Devuelve:

Nada.

- `public static string GetGroupID(string groupName, string tenant)`

Esta función se encarga de buscar en la base de datos el identificador único del grupo, para el nombre y cliente (tenant) especificado.

Parámetros:

groupName: Cadena de caracteres que representa el nombre del grupo buscado.
tenant: Cadena de caracteres que representa el nombre del cliente (tenant) del grupo buscado.

Devuelve:
Cadena de caracteres que representa el identificador único del grupo.

- `public static string GetUserID(string login, string tenant)`

Este método busca en la base de datos el identificador único de usuario para el nombre de sesión y cliente (tenant) que tiene el usuario buscado.

Parámetros:
login: Cadena de caracteres que representa el nombre de sesión del usuario buscado.
tenant: Cadena de caracteres que representa el nombre del cliente (tenant) para el que trabaja el usuario.

Devuelve:
Cadena de caracteres que representa el identificador único del usuario.

- `public static int GetTenantId(GalileoContext aContext, string aName)`

Esta función se encarga de buscar en la base de datos el identificador único del cliente (tenant) indicado.

Parámetros:
aContext: Objeto que permite la comunicación con la base de datos especificada.
aName: Cadena de caracteres con el nombre del cliente (tenant) buscado.

Devuelve:
Entero que representa el identificador único del cliente (tenant).

7.4.3. Funciones y atributos de ConnectionMapping

Una de las opciones principales de SignalR es la posibilidad de enviar a los usuarios información en tiempo real sin necesidad de una petición por parte de éstos, por ello, será necesario vincular su identificador de conexión con su identificador de usuario. Se consigue así tener identificada el lugar (o conexión) donde se encuentra el usuario al que mandar la información. A continuación se describen los métodos y atributos necesarios para realizar este proceso.

- **Variables**:

  - `private static readonly Dictionary<T, HashSet<string>> _connections`

    Variable de tipo diccionario que permite almacenar pares de valores. Como parámetros clave tendremos los identificadores de usuario. Como valores tendremos una lista con los identificadores de conexión de ese usuario. Comentar que un usuario puede tener varias conexiones abiertas, por lo que se almacenarán varios identificadores de conexión.

  - `public Dictionary<T, HashSet<string>> Connections`

    Variable pública que permite conocer desde cualquier parte del código el cuadro de conexiones de cada usuario. Será útil a la hora de enviar las notificaciones al usuario, pues puede que se tenga que enviar la misma notificación a varias conexiones.

  - `public int Count`

    Variable pública que permite conocer el número de usuarios que están conectados al servicio SignalR.

  - `public List<T> Names`

    Lista de *string* que representan el identificador de cada usuario que está conectado al servicio.

- **Métodos**:

  - `public void Add(T key, string connectionID)`

    Este método se encarga de añadir el identificador de conexión del usuario indicado a su lista de conexiones. Si ese identificador de conexión ya existe no se añade otra vez.

    Parámetros:

    key: Cadena de caracteres que representa el identificador único de usuario.
    connectionID: Cadena de caracteres que representa el identificador único de la conexión que se vinculará al usuario.

    Devuelve:

    Nada.

  - `public IEnumerable<string> GetConnections(T key)`

    Esta función se encarga de obtener una lista con todos los identificadores de conexión del usuario especificado. Si el usuario no tiene ninguna conexión abierta en el momento de la ejecución, se retornará una lista vacía.

    Parámetros:

    key: Cadena de caracteres que representa el identificador único de usuario.

    Devuelve:

    Lista (IEnumerable) de *string* que contiene todos los identificadores de c onexión del usuario indicado.

  - `public void Remove(T key, string connectionID)`

    Esta función se encarga de borrar el identificador de conexión de un usuario cuando este deja de estar conectado. Si el usuario no tiene ninguna conexión abierta en el momento de ejecutar el método se borrará a dicho usuario del diccionario.

Parámetros:

key: Cadena de caracteres que representa el identificador único de usuario.

connectionID: Cadena de caracteres que representa el identificador de la conexión a borrar del diccionario.

Devuelve:

Nada.

7.4.4. Funciones y variables de DBRules

En esta clase tendrán lugar todas las operaciones necesarias para determinar que usuario o usuarios tienen asignada una determinada avería en el momento de su activación. A continuación se describen las variables y métodos necesarios para realizar este proceso.

- **Variables**:

  - `public static List<UserNotification> UsersList`

    Lista de objetos de la clase UserNotification, que contiene las notificaciones para una determinada avería. La información contenida en los objetos muestra el identificador de usuario, de avería, la prioridad del usuario, etc.

- **Métodos**:

  - `public static void checkRules(int faultId, int faultKey)`

    Este método se encarga de comprobar si una avería activa tiene una regla de notificación asignada en las tablas user_fault y group_fault de la base de datos.

    Parámetros:

    fault_id: Entero que representa el id de la avería definida.

    faultkey: Entero que representa el id de la avería disparada.

    Devuelve:

    Nada.

- `private static void createPriority(int faultKey, string user_fault_Key, string group_fault_Key)`

  Este método se encarga de definir una prioridad para la avería producida, es decir, con el parámetro prioridad se consigue que la notificación de avería solo se envíe a usuarios con una prioridad igual no menor, consiguiendo así evitar la saturación del sistema de notificaciones.

  Parámetros:

  faultkey: Entero que representa el identificador único de la avería disparada.
  user_fault_key: Identificador del registro de la tabla user_fault que contiene el usuario al que se asigna la avería.
  group_fault_key: Identificador del registro de la tabla group_fault que contiene el grupo al que se asigna la avería.

  Devuelve:

  Nada.

- `private static void checkValidUsersGroups(int faultKey, string ruleId)`

  Esta función se encarga de comprobar que usuarios/grupos pueden atender la avería indicada, según la regla que tienen asignada. Se evalúa si el fault_date se encuentra dentro del intervalo temporal de la rule_id.

  Parámetros:

  faultKey: Entero que representa el identificador único de la avería disparada.
  ruleId: Cadena de caracteres que representa la regla de notificación que se comprobará para indicar que usuarios pueden atender la avería.

  Devuelve:

  Nada.

- `public static void RemovePriority(List<int> faultKeys)`

  Función encargada de eliminar de la tabla *fault_priority* una determinada lista de averías cuando éstas se hayan solucionado.

  Parámetros:

  faultKeys: Lista de enteros con los identificadores de avería de todas aquellas que ya estén solucionadas.

Devuelve:

Nada.

7.4.5. Funciones de NotificationSender

En esta clase tienen lugar todas aquellas operaciones ligadas al envío de notificaciones a la aplicación móvil. El servicio web comprobará los usuarios que pueden atender la avería y creará la notificación adecuada. Si el usuario no se encuentra conectado al servicio, se utilizará la plataforma de notificaciones del terminal asociado a la aplicación móvil. A continuación se describirán los métodos empleados para realizar estas tareas.

- `public static void UserNotifyFaults(string userID, UserNotification info)`

  Este método se encarga del envío de notificaciones al usuario. Para ello es necesario conocer el identificador de dicho usuario y la información que contendrá la notificación. Además se comprobará en qué plataformas móviles está registrado y se enviará una notificación personalizada a cada una.

  Parámetros:

  userID: Cadena de caracteres que representa el identificador único de usuario.

  info: Objeto de la clase *UserNotification* que contiene toda la información que se enviará en la notificación.

  Devuelve:

  Nada.

- `private static void SendToastNotification(string userID, string info)`

  Este método se encarga de enviar notificaciones de tipo “toast” a la aplicación móvil del usuario registrada en un terminal con Windows Phone 8.

Parámetros:

userID: Cadena de caracteres que representa el identificador único de usuario.

info: Cadena de caracteres con la información que irá en la notificación "toast".

Devuelve:

Nada.

- `private static void SendTileNotification(string userID, string info)`

Este método se encarga de enviar notificaciones de tipo "tile" a la aplicación móvil del usuario registrada en un terminal con Windows Phone 8.

Parámetros:

userID: Cadena de caracteres que representa el identificador único de usuario.

info: Cadena de caracteres con la información que irá en la notificación "tile".

Devuelve:

Nada.

- `private static void DeleteUri(string badUri)`

Este método tiene como función eliminar aquellas URIs (dirección a la que enviar las notificaciones) que ya no estén activas o no sean válidas.

Parámetros:

badUri: Cadena de caracteres que representa la URI vinculada al cliente de notificaciones del terminal móvil.

Devuelve:

Nada.

- `private static IEnumerable<string> GetUserPlatforms(string userID)`

Esta función se encarga de buscar en la base de datos todas las plataformas móviles donde ha registrado el usuario la aplicación móvil. Con ello se consigue que sea posible enviar una notificación personalizada a cada plataforma móvil.

Parámetros:

userID: Cadena de caracteres que representa el identificador único de usuario.

Devuelve:

Lista (IEnumerable) de *string* con la lista de plataformas móviles vinculadas al usuario.

- `private static IEnumerable<string> GetUris(string userID)`

Esta función se encarga de buscar en la base de datos todas las URIs que se han vinculado a un determinado usuario, con el objetivo de enviar la notificación a cada una de ellas.

Parámetros:

userID: Cadena de caracteres que representa el identificador único de usuario.

Devuelve:

Lista (IEnumerable) de *string* que representa las URIs vinculadas al usuario.

7.4.6. Funciones de Startup

La clase Startup se utiliza exclusivamente para la configuración y arranque del servicio SignalR, por lo que no se introducirán más métodos de los necesarios para este propósito a no ser que sea necesario.

- `public void Configuration(IAppBuilder app)`

En esta función se establecen los diferentes parámetros de configuración disponibles en SignalR. Será necesario configurar la aplicación para permitir conexiones desde clientes que estén en dominios diferentes, por lo que se introducirá la siguiente línea de código:

```
map.UseCors(CorsOptions.AllowAll);
```

Además se permitirá el envío de datos entre cliente y servidor del tipo JSON.

```
var hubConfiguration = new HubConfiguration
{
  EnableJSONP = true

};
```

El resto de parámetros son los indicados en la plantilla por defecto.

7.4.7. Funciones y atributos de Program

En la clase Program, que contiene el Main de la aplicación, se arranca tanto el servicio web como los 2 hilos de adicionales en los que se realiza la comprobación de averías en la base de datos y el envío de notificaciones.

Para la realización de los 2 hilos adicionales se ha utilizado la librería,

```
Mecalux.ITSW.Core.Abstraction
```

en la que se definen los métodos necesarios para su arranque y finalización.

- **Variables**:
-

  - `private readonly static UserNotificationMapping<string> _userNotifications`

    Objeto de la clase *UserNotificationMapping* con el que se accederá a los métodos de la clase *UserNotificationMapping*, para configurar la asignación de averías a usuario.

- **Métodos**:

  - `static void Main(string[] args)`

    Método o hilo principal de la aplicación de consola. En él se arranca el servicio SignalR y los 2 hilo de ejecución adicionales.

- **Clases adicionales:**

Clases encargadas de la ejecución de los hilos. Desarrolladas por Mecalux.

- `class checkupdates : TThread`
- `class sendupdates: TThread`

7.5. DataModel

Para el desarrollo de este proyecto se ha tomado la decisión de aislar el modelo de datos en una librería de clases (.dll) de modo que se pueda trabajar con ella de forma sencilla desde el servicio web y desde el cliente.

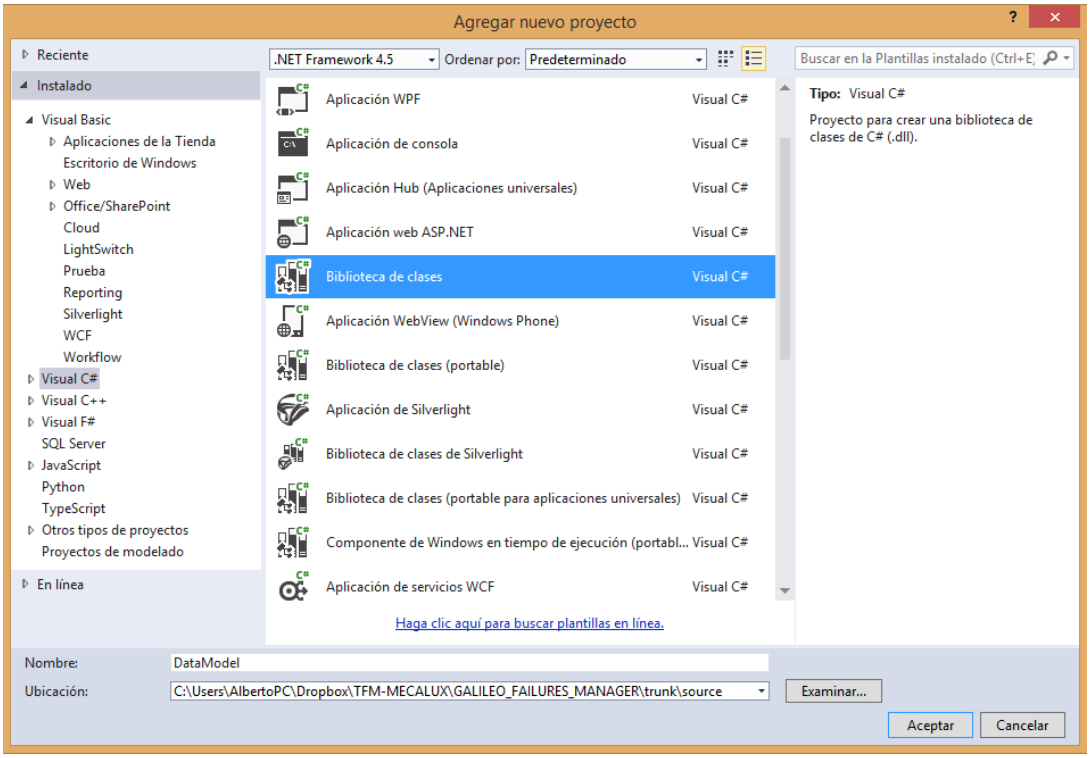


Ilustración 52: Creación de una biblioteca de clases

Para dotar a nuestra librería de las características y funcionalidades que ofrece Entity Framework es necesario agregar un nuevo elemento al proyecto (al proyecto DataModel), seleccionando la opción indicada en la siguiente ilustración.

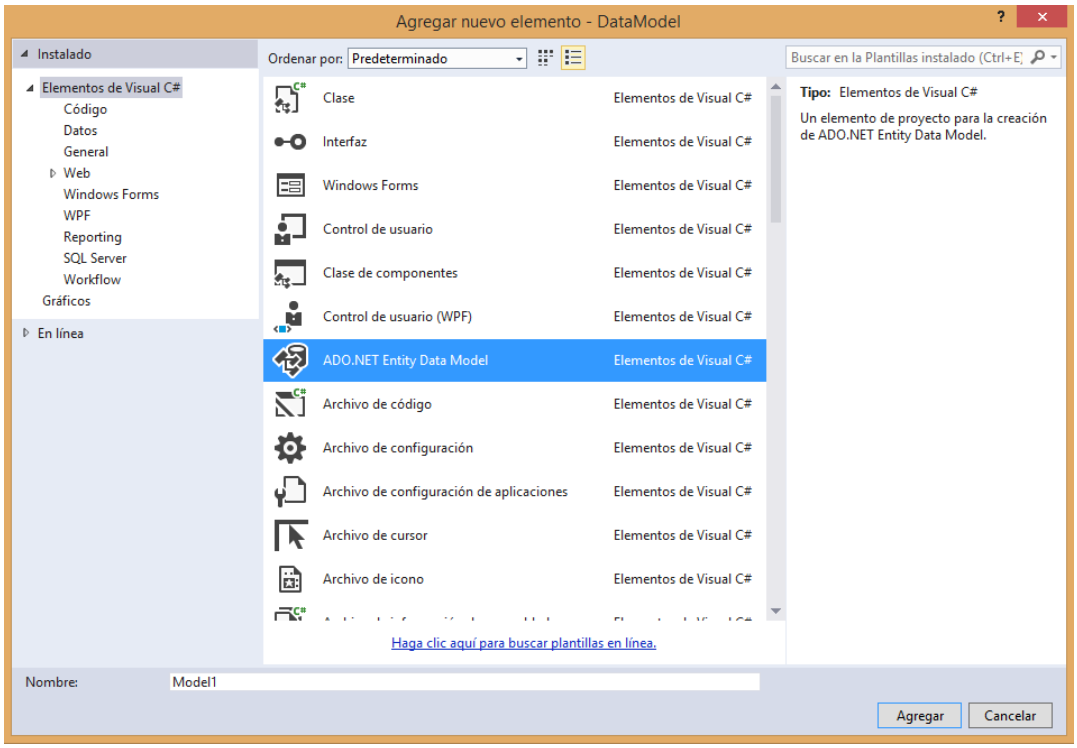


Ilustración 53: Agregar modelo de datos al proyecto.

Una vez se ha creado el modelo de datos habrá que importar las tablas con sus respectivos atributos de la base de datos MySQL que se ha usado para el desarrollo de la aplicación. Gracias a las bondades de Entity Framework este proceso no será obligatorio, ya que existe la posibilidad de crear el modelo de datos sin necesidad de código SQL. Se añadirán las tablas necesarias con sus respectivos campos y posteriormente se indicará el tipo de base de datos que se desea generar (MySQL, SQL Server, Oracle).

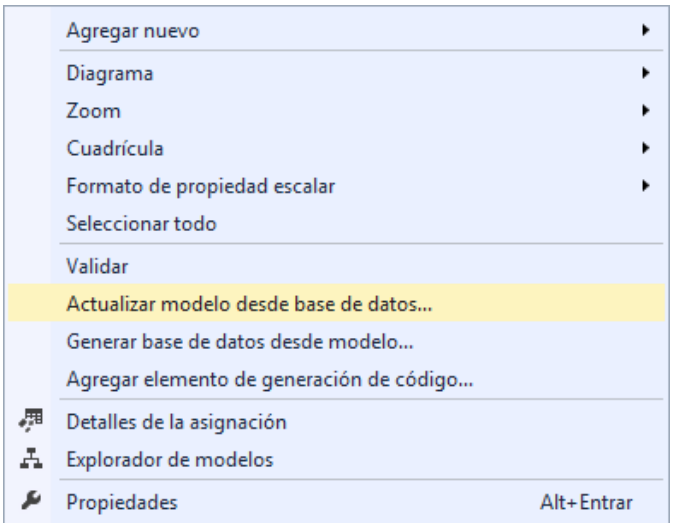


Ilustración 54: Añadir modelo a Entity Framework.

El proceso seguido en este caso es el inverso, se crea la base de datos con MySQL Workbench y se importan las tablas a nuestro modelo de datos con Entity Framework. Una vez importado el modelo se podrá exportar a al tipo de base de datos que se desee (SQL Server, Oracle,…).

Para que tanto el cliente como el servicio web tengan acceso al modelo de datos será necesario añadir una referencia a este proyecto en ambas partes, es decir, hablar que añadir una referencia a la librería creada y añadir la etiqueta *using* correspondiente cuando se quiera acceder al modelo de datos.

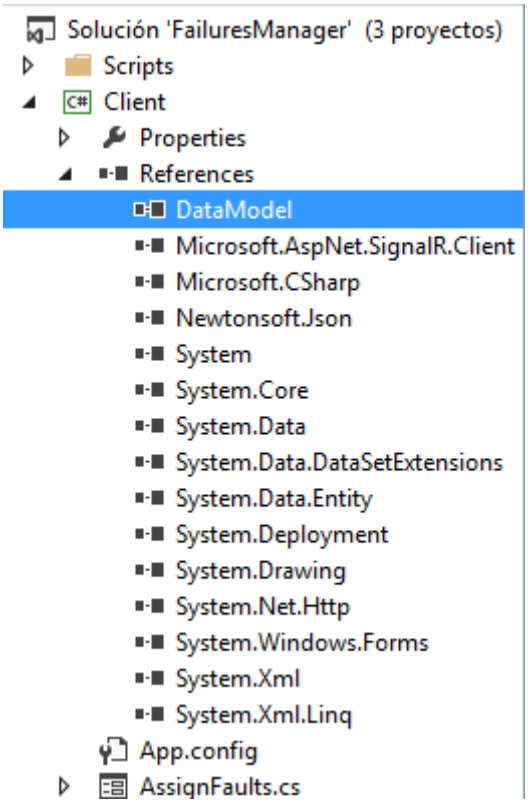


Ilustración 55: Añadir referencia al modelo de datos en cliente y servicio.

7.5.1. Paquetes adicionales

Para el correcto funcionamiento del modelo de datos de Entity Framework con MySQL será necesario añadir una serie de paquetes a través del administrador de paquetes NuGet. Introduciendo el comando indicado a continuación se cargarán al proyecto DataModel todos los paquetes necesarios para que sea posible la comunicación entre Entity Framework y nuestra base de datos MySQL.

```
Install-Package MySQL.Data.Entities
```

7.6. Client

La aplicación cliente encargada de la gestión de usuarios/grupos y averías se ha desarrollado a través de una aplicación de Windows Forms. Para dotar al cliente de las características de SignalR y permitir comunicaciones en tiempo real con el servicio web es necesario añadir una serie de librerías al proyecto.

A través del administrador de paquetes NuGet se añadirá al proyecto Client la librería desarrollada para clientes SignalR.

```
Install-Package Microsoft.AspNet.SignalR.Client
```

Con la instrucción indicada arriba se añadirán todos los componentes necesarios para comunicar cliente y servidor. No será necesario descargar el paquete completo de SignalR en el cliente.

Además de las librerías SignalR será necesario añadir una referencia al modelo de datos (Data Model) para poder interpretar y trabajar con los datos del sistema.

Para dotar al cliente de todas las opciones que se comentan en el apartado de Análisis y Diseño, se han creado una serie de ventanas o Forms, que nos permitirán realizar dichas tareas. Se indicará en los apartados siguientes el objetivo de cada función y parámetro del cliente.

7.6.1. Funciones de AssignFaults

En esta ventana tendrá lugar el proceso de asignación de averías a usuarios. Los métodos encargados de realizar dicha tarea son los siguientes.

- `private void LoadUsers()`

  Este método busca en la base de datos los nombres de todos los usuarios creados para el cliente (tenant) correspondiente.

- `private void LoadGroups()`

  Este método busca en la base de datos los nombres de todos los grupos creados para el cliente (tenant) correspondiente.

- `private void LoadFaultsDef()`

  Esta función se encarga de buscar en la base de datos y mostrar en pantalla la información de las posibles averías que puede disparar nuestro sistema.

- `private void Save_button_Click(object sender, EventArgs e)`

  Este evento se dispara cuando el usuario pulsa el botón "Save". Guardará la selección de averías que desea asignar al usuario o grupo correspondiente.

- `private void Cancel_button_Click(object sender, EventArgs e)`

  Este evento se dispara cuando el usuario pulsa el botón "Cancel". Se encarga de cerrar la ventana actual y borra los datos introducidos.

- `private void Users_Groups_ListBox_SelectedIndexChanged(object sender, EventArgs e)`

  Esta función se encarga de llamar a los métodos LoadUsers o LoadGroups si el usuario selecciona "Users" o "Groups" en el ComboBox indicado. Además de cargar los datos, si se selecciona un usuario o grupo de la lista, indicará que averías tiene asignadas dicha selección.

  Devuelve:
  Nada.

- `private void Users_Groups_ListBox_ItemCheck(object sender, ItemCheckEventArgs e)`

  Este método hace posible que solo se pueda seleccionar un ítem en el ListBox de usuarios/grupos.

- `private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)`

  Esta función se encarga de actualizar el ListBox de usuarios/grupos si se cambia de opción (Usuarios o Grupos).

- `private void CheckFaultsDef(List<int> faultsAssigned)`

  Este método se encarga de seleccionar en la vista de averías (Select faults) aquellas averías que ya tiene asignadas el usuario/grupo seleccionado en el ListBox.

Parámetros:

faultsAssigned: Lista de enteros con los identificadores de avería ya asignados al usuario/grupo seleccionado.

- `private void ClearGrid()`

  Esta función se encarga de deseleccionar todas las filas del Grid, con el objetivo de que no haya confusiones cuando se modifican los usuarios/grupos seleccionados.

7.6.2. Funciones y variables de EditRules

En esta ventana tiene lugar la creación y asignación de las reglas de notificación a los usuarios o grupos especificados.

- **Variables globales:**

  - `private static List<galileo_fault_def> _faults`

    Lista de objetos de la clase *galileo_fault_def* que servirá para almacenar todas aquellas averías que se asignen al usuario o grupo que se ha seleccionado en el interfaz de usuario.

  - `private List<time_interval> _intervals`

    Lista de objetos de la clase *time_interval* que servirá para almacenar los intervalos temporales (turnos de trabajo) que se pretendan asignar al usuario o grupo indicado.

- **Métodos**:

  - `private void SaveRuleButton_Click(object sender, EventArgs e)`

    Esta función se ejecuta cuando el usuario pulsa el botón Save. Se encarga de crear la regla de notificación con la prioridad, tiempo de espera y turnos de trabajo especificados. Además se encarga de vincularla al usuario o grupo seleccionado.

- `private void optionCombo_SelectedIndexChanged(object sender, EventArgs e)`

  Esta función se ejecuta cuando se modifica la opción seleccionada en el ComboBox de opciones. Se encarga de cargar en el ListBox la lista de usuarios o grupos si uno u otro es seleccionado.

- `private void CancelRuleButton_Click(object sender, EventArgs e)`

  Esta función se ejecuta cuando el usuario pulsa el botón Cancel. Se encarga de borrar el contenido de las variables *_faults* y e *_intervals* además de cerrar la ventana actual.

- `private void AddTimeInterval_Click(object sender, EventArgs e)`

  Esta función se ejecuta cuando se pulsa el botón "+". Se encarga de abrir la ventana *TimeInterval* y actualizar el grid de intervalos temporales cuando éstos se añaden.

- `private void DelTimeInterval_Click(object sender, EventArgs e)`

  Esta función se ejecuta cuando se pulsa el botón "-". Se encarga de eliminar del grid Intervals y de la lista *_intervals* todos aquellos turnos que seleccione el usuario en dicho grid.

- `private void LoadFaultsDef()`

  Esta función se encarga de buscar en la base de datos todas las definiciones de averías y guardarlas en la variable *_faults*.

- `private void LoadTIntervals()`

  Este método se encarga de rellenar o actualizar el grid de intervalos temporales.

- `private void User_Group_ListBox_SelectedIndexChanged(object sender, EventArgs e)`

  Esta función se encarga de buscar en la base de datos las averías asignadas al usuario o grupo seleccionado y a continuación envía los valores obtenidos al método CheckFaultsDef.

- `private void User_Group_ListBox_ItemCheck(object sender, ItemCheckEventArgs e)`

  Este método se encarga de limitar la selección de usuarios o grupos del ListBox a un solo ítem, es decir, impide seleccionar dos o más usuarios o grupos en la lista.

- `private void CheckFaultsDef(List<int> faultsAssigned)`

  Esta función se encarga de seleccionar o checkear las averías cuyos identificadores corresponden con los recibidos en el parámetro *faultsAssigned* en el grid Faults assigned.

  Parámetros:

  faultsAssigned: Lista de enteros que representan los identificadores de avería a seleccionar en el grid.

- `private void UnCheckFaults()`

  Esta función se encarga de deseleccionar todas las averías marcadas en el grid. Esta función es llamada cuando un usuario cambia el usuario/grupo seleccionado en la lista.

- `private void EditRules_FormClosing(object sender, FormClosingEventArgs e)`

  Este evento se dispara cuando se pulsa el botón de cierre de la ventana actual. Antes de cerrarla se vaciarán las listas *_faults* e *_intervals*.

7.6.3. Funciones y variables de Main:

En la ventana principal (Main) es donde se realizan la mayoría de las tareas que tienen que ver con la creación o modificación de usuarios y grupos. Además realiza también los procedimientos necesarios para conectar la aplicación cliente con el servidor SignalR. Se describirá a continuación el cometido de cada variable y función en la realización de estas tareas.

- **Variables globales:**

  - `private HubConnection SignalRConnect`

    A través de esta variable se crea la conexión entre cliente y servidor. Se le tendrá que especificar la dirección de conexión del servidor. Además como parámetros adicionales se pueden introducir "querystring" con los parámetros de inicio de sesión del usuario.

  - `public static  IHubProxy hubproxy`

    Este objeto de la clase IHubProxy tiene la función de permitir comunicar al cliente con el Hub del servidor SignalR. Gracias a este proxy se podrán invocar métodos en el servidor.

  - `const string SignalRServerURI`

    Esta cadena de caracteres representa la dirección de conexión con el servidor SignalR. Será necesario indicarle este parámetro al objeto SignalRConnect para establecer la comunicación.

  - `public static string SessionTenant`

    Esta variable almacenará el nombre del cliente (tenant) del usuario que inicia sesión. Se utilizará a la hora de realizar peticiones al servidor.

  - `private string currName`

    Esta variable tiene como función el almacenar el nombre actual de un usuario. Este valor será utilizado en las operaciones de modificación de datos de usuarios.

  - `private string currLogin`

    Esta variable tiene como función el almacenar el nombre actual de sesión de un usuario. Este valor será utilizado en las operaciones de modificación de datos de usuarios.

- `private string currPass`

  Esta variable tiene como función el almacenar la contraseña actual de un usuario. Este valor será utilizado en las operaciones de modificación de datos de usuarios.

- `public static Dictionary<string, string> user_login`

  Este objeto de tipo diccionario tiene la función de almacenar los pares de valores nombre completo usuario y nombre de sesión de todos los usuarios del cliente (tenant) indicado. Se utilizará en operaciones de modificación de datos de usuarios.

- **Métodos**:

  - `private void EditRulesButton_Click(object sender, EventArgs e)`

    Este evento se ejecutará cuando el usuario pulsa el botón "Edit Rules". Su objetivo es el de abrir la ventana de edición de reglas.

  - `private void AddUserButton_Click(object sender, EventArgs e)`

    Este evento se ejecutará cuando el usuario pulsa el botón "+" en la sección Users. Se encarga de abrir la ventana de Nuevo Usuario y actualizar la lista de usuarios del cliente en caso de que se hayan introducido nuevos empleados.

  - `private void DelUserButton_Click(object sender, EventArgs e)`

    Esta función se encarga de eliminar de la base de datos todos aquellos usuarios que se hayan seleccionado (checked) en la lista de la sección Users.

  - `private void CreateGroup_Click(object sender, EventArgs e)`

    Este evento se ejecuta cuando el usuario pulsa el botón "+" de la sección Data. Se encarga de abrir la ventada de Nuevo Grupo y actualiza la lista de grupos si se introduce alguno nuevo.

- `private void DeleteGroup_Click(object sender, EventArgs e)`

  Este evento se ejecutará cuando el usuario pulse el botón "-" de la sección Data. Se encarga de borrar todos aquellos grupos que se encuentren seleccionados en la lista asociada.

- `private void Save_button_Click(object sender, EventArgs e)`

  Este evento se ejecuta cuando el usuario pulsa el botón "Save". Se encarga de guardar en la base de datos todos aquellos cambios que se hayan producido en la sección Data, tanto la información de usuario como los grupos a los que se vincula dicho usuario.

- `private void AssignFaults_button_Click(object sender, EventArgs e)`

  Este evento se ejecuta cuando el usuario pulsa el botón "Assign Faults". Se encarga de abrir la ventana de asignación de averías.

- `private void Login_button_Click(object sender, EventArgs e)`

  Este evento se ejecuta cuando el usuario pulsa el botón "Connect". Se encarga de recolectar la información de inicio de sesión de usuario (login, password, tenant) y de llamar a la función "ConnectAsync" para iniciar la conexión con el servidor SignalR.

- `private void Disconnect_button_Click(object sender, EventArgs e)`

  Este evento se ejecuta cuando el usuario pulsa el botón "Disconnect" en la sección Login. Se encarga de detener la conexión con el servidor SignalR si esta se encuentra activa.

- `private async void ConnectAsync(Dictionary<string,string> queryStringData)`

  Esta función se encarga de crear e inicializar con los atributos indicados la conexión del cliente con el servidor SignalR. Además en este método se definen los manejadores para todos aquellos eventos disparados por el servidor.

  Parámetros:

  queryStringData: Objeto diccionario con la información de inicio de sesión del usuario.

- `private void Connection_Closed()`

  Este método se encarga de borrar todos los campos de la ventana principal cuando el usuario cierra la sesión. Con ello se evita que queden almacenados sin querer datos comprometidos.

- `private void UsersListBox_SelectedIndexChanged(object sender, EventArgs e)`

  Este método se encarga de cargar los datos de un usuario en la sección "Data" cuando se selecciona su nombre en el apartado "Users". Si se selecciona más de un usuario no se cargará ningún dato, borrando los que pueda haber.

- `private void LoadUsers()`

  Este método se encarga de buscar en la base de datos todos los nombres de usuario para el cliente (tenant) indicado y los carga en el listbox. Además guarda en el diccionario *user_login* estos valores.

- `private void LoadGroups()`

  Esta función se encarga de buscar en la base de datos todos los nombres de grupos para el cliente (tenant) especificado y los carga en el listbox de la sección Data.

- `private void ClearFields()`

  Esta función se encarga de borrar todos los campos de la sección Data y deselecciona aquellos grupos marcados.

- `private void Main_FormClosing(object sender, FormClosingEventArgs e)`

  Si se cierra la ventana principal de la aplicación se cerrará la conexión con el servicio SignalR en caso de que esta esté iniciada.

7.6.4. Funciones y variables de NewGroup

En esta clase se realizan las tareas de adición de grupos al cliente (tenant) adecuado. Se detallarán a continuación las variables y métodos necesarios para llevar a cabo la tarea.

- **Variables globales:**

  - `private List<group> _groups`

    Lista de objetos de la clase group que almacenarán provisionalmente los grupos que añada el usuario. No se añadirán en la base de datos hasta que el usuario confirme la selección.

- **Métodos:**

  - `private void AddGroup_button_Click(object sender, EventArgs e)`

    Este evento se ejecutará cuando el usuario pulse el botón "Add Group". Se encargará de añadir al usuario a la lista *_groups*. Además cargará su nombre en el listbox.

  - `private void Save_button_Click(object sender, EventArgs e)`

    Este evento se ejecutará cuando el usuario pulse el botón "Save". Se encargará invocar la función *CreateGroup* para guardar en la base de datos aquellos usuarios añadidos a la lista y confirmados.

  - `private void Cancel_button_Click(object sender, EventArgs e)`

    Este evento se ejecutará cuando el usuario pulsa el botón "Cancel". Se encarga de borrar el contenido de la lista *_groups* y cierra la ventana actual.

7.6.5. Funciones y variables de NewUser

En esta clase se realizan las tareas de adición de usuarios al cliente (tenant) especificado. Se procesará la información introducida por el usuario administrador y se realizarán las peticiones pertinentes al servicio SignalR para guardar dicha información en la base de datos. A continuación se describirán las variables y métodos necesarios para realizar esta tarea.

- **Variables globales:**

  - `private List<user> _users`

    Lista de objetos de la clase user que almacenarán provisionalmente los usuarios que añada el usuario administrador. No se añadirán en la base de datos hasta que el usuario confirme la selección.

- **Métodos:**

  - `private void AddUser_button_Click(object sender, EventArgs e)`

    Este evento se ejecutará cuando el usuario pulse el botón "Add User". Se encarga de añadir cada usuario a la lista *_users* y actualizar el listbox con dicho usuario.

  - `private void Save_button_Click(object sender, EventArgs e)`

    Este evento se ejecutará cuando el usuario pulse el botón "Save". Se encargará de invocar a la función *CreateUsers* para guardar en la base de datos la información de los usuarios que estén en la lista *_users* y chequeados en el listbox.

  - `private void Cancel_button_Click(object sender, EventArgs e)`

    Este evento se ejecutará cuando el usuario pulsa el botón "Cancel". Se encarga de borrar el contenido de la lista *_users* y cierra la ventana actual.

7.6.6. Funciones y métodos de TimeInterval

En esta clase se realizan las tareas de adición de intervalos temporales o turnos de trabajo para su vinculación con los usuarios del servicio. Se detallará a continuación las variables y métodos necesarios para realizar dicha tarea.

- **Variables**:

  - `public static List<time_interval> _TIntervals`

    Lista de objetos de la clase time_interval donde se añaden todos los intervalos deseados por el administrador. Será un almacén temporal hasta la confirmación y guardado en la base de datos.

- **Métodos**:

  - `private void Cancel_button_Click(object sender, EventArgs e)`

    Este evento se ejecuta cuando el usuario pulsa el botón “Cancel” en la ventana actual. Se encarga de cerrar dicha ventana.

  - `private void Ok_button_Click(object sender, EventArgs e)`

    Este método se encargar de procesar y validar la información temporal introducida y almacenarla en la lista *_TIntervals*.

7.6.7. Funciones de Program

Clase autogenerada por Visual Studio al crear la plantilla de la aplicación de Windows Forms. Es la encargada de arrancar la interfaz de usuario. Para el presente proyecto no se ha modificado.

7.7. Failures App

La aplicación móvil ha sido desarrollada en Javascript y HTML en casi su totalidad. Para su desarrollo de ha partido de la plantilla generada en la creación de un proyecto Phonegap. Para crear dicho proyecto será necesario instalar previamente la plataforma Phonegap en el computador. La documentación **[3]** en su apartado "The Command-Line Interface" explica con claridad el proceso de instalación y creación de un proyecto Phonegap.

Además del código HTML-Javascript que se desarrolle, será necesario añadir una serie de plugins al proyecto para dotar a la aplicación de todas las funcionalidades que ofrece una aplicación nativa. En el apartado 7.7.1. se describe este proceso.

Una vez añadido todo el código fuente y demás plugins al proyecto habrá que compilar la aplicación. En Phonegap existen dos alternativas de compilación. Por un lado la compilación local de la aplicación, la cual requiere de la descarga de los SDK de las plataformas para las que se desee obtener la aplicación. Se trata de una opción desaconsejable si se pretende desarrollar para muchas plataformas. La otra opción será la compilación en la nube, a través de la plataforma Phonegap Build, la cual se ha elegido para el desarrollo de este proyecto. En el apartado 7.7.2. se explica brevemente el funcionamiento de esta plataforma.

7.7.1. Configuración plugins

Para dotar a nuestra aplicación hibrida de las funcionalidades de cualquier aplicación nativa para terminales móviles, será necesario añadir una serie de plugins a nuestro proyecto. Para realizar este proceso no es obligatorio descargar ningún fichero, sino que es posible indicar en el fichero config.xml la lista de los plugins deseados. El compilador en la nube de Adobe (Phonegap Build) se encargará de introducir en nuestro fichero de aplicación los plugins indicados en la configuración. Todo ello sin necesidad de un software adicional. Se puede consultar en el enlace **[11]** de la bibliografía la documentación ofrecida por Phonegap para la instalación de plugins en nuestra aplicación.

7.7.2. Proceso de compilación

Como se comentó anteriormente, para el desarrollo de esta aplicación se ha elegido la herramienta web Phonegap Build como forma de compilación. Esto libera al desarrollador de la descarga de todos los SDK de las plataformas para las cuales se vaya a compilar.

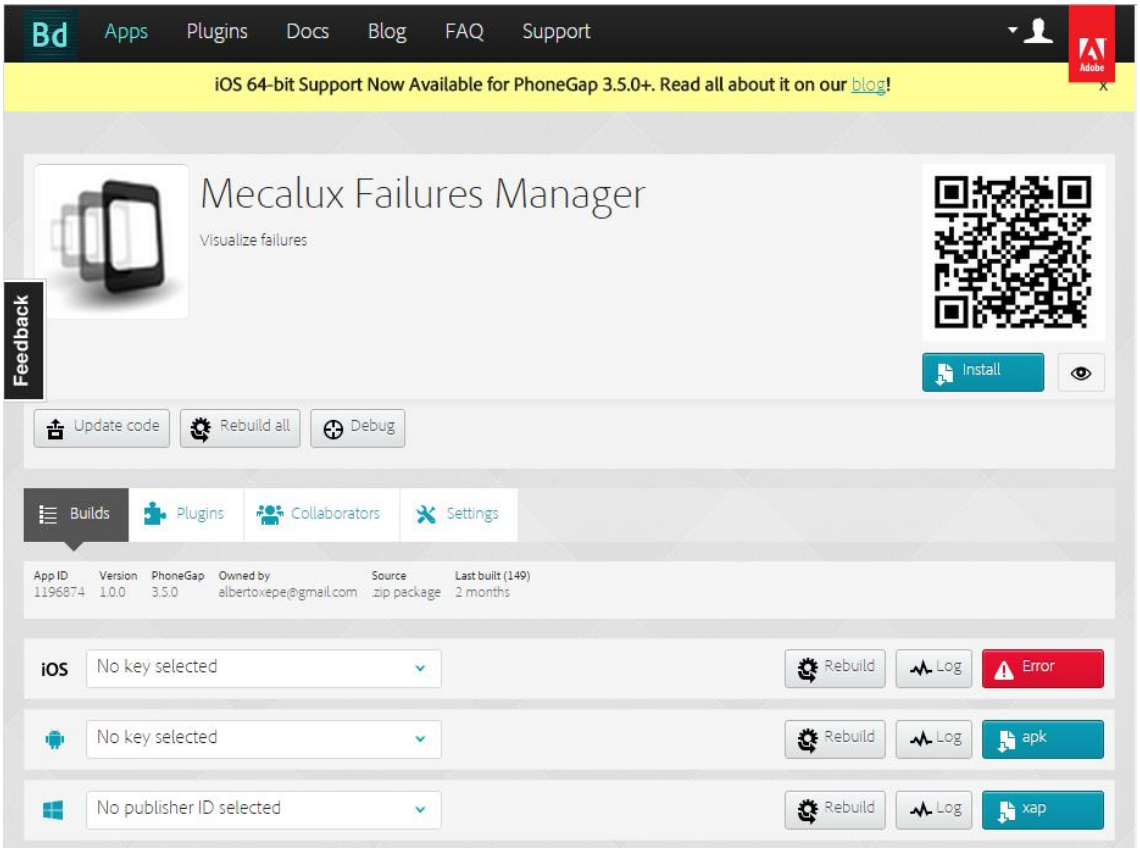


Ilustración 56: Plataforma Phonegap Build

El funcionamiento de la herramienta es muy sencillo. Bastará con empaquetar en un fichero zip la carpeta www generada en nuestro proyecto Phonegap y el archivo config.xml. En dicho archivo se encuentra una lista de los plugins que PhonegapBuild tiene que añadir a la solución.

Una vez añadido lo necesario al fichero zip, se subirá a la web mediante el botón Update Code, presente en la ilustración anterior. Una vez cargado, la compilación se realizará de forma automática.

Cuando el proceso haya finalizado se dispondrá de un fichero de aplicación para las plataformas indicadas (en cuentas gratuitas solo iOS, Android y Windows Phone) que se podrá descargar con un simple click.

7.7.3. Documentación del fichero Index.js

En este fichero se aglutinan las funciones que hacen posible tanto la conexión de la aplicación con el servicio web como las modificaciones realizadas en tiempo de ejecución sobre el DOM (Document Object Model).

Además, contiene los métodos necesarios para el correcto funcionamiento de los plugins instalados en la aplicación a través de Phonegap.

No solo se realiza la implementación de los métodos, sino que también se trabaja con variables globales que permiten comunicar desde cualquier método al cliente con el servicio.

A continuación se describirá el contenido y la utilidad de los métodos y variables utilizadas en el presente fichero.

- **Documentación de las variables**:

  - var connection

    Variable utilizada para almacenar el objeto que conecta al cliente de SignalR con el Hub del servidor.

  - var proxy

    Variable que contiene el proxy o intermediario entre cliente y servidor en el servicio SignalR, necesario para realizar peticiones de datos entre uno y otro.

  - var pushNotification

    Variable donde se almacena el objeto necesario para registrar la aplicación en el servicio de notificaciones de la plataforma móvil (para el presente proyecto Windows Phone 8).

- **Documentación de los métodos:**

  - function onBodyLoad()

    Función disparada cuando el body del fichero index.html se carga correctamente. A su vez, este método disparará en evento onDeviceReady para indicar que el dispositivo ya está listo para ejecutar la aplicación.

  - function onDeviceReady()

    Método disparado cuando el dispositivo ha cargado la aplicación. En él se registran los eventos asociados al "click" sobre el botón "back" del dispositivo, y los eventos *pause* y *resume*, que se activan cuando la aplicación pasa a "segundo plano" o a ejecución en pantalla. Además si se trata de la primera ejecución de la aplicación, iniciará el registro en la plataforma de notificaciones.

  - function registerMPNS()

    Este método inicia el proceso de registro de la aplicación en la plataforma de notificaciones.

  - function channelHandler(event)

    Método disparado cuando el registro de la aplicación en la plataforma de notificaciones se realiza correctamente.

  - function jsonErrorHandler(error)

    Este método se dispara cuando el registro en la plataforma de notificaciones no se puede realizar.

    Parámetros:

    error: Se trata de un objeto devuelto por la función registerMPNS que contiene el código de error asociado al fallo en el registro de la App y un mensaje en el que se detalla el motivo del fallo.

- function onNotificationWP8(e)

  Este método es disparado cuando la aplicación recibe una notificación PUSH. Tiene la labor de mostrar la información recibida en pantalla.

  Parámetros:

  e: Este objeto es enviado por la plataforma de notificaciones (en el caso de WP8 recibe el nombre de MPNS) y contiene la información que debe mostrarse en la notificación.

- function unregister()

  Este método tiene la labor de deshabilitar el registro de la aplicación en la plataforma de notificaciones, de modo que una vez ejecutado la aplicación dejará de recibir notificaciones del servidor.

- function succesHandler(e)

  Este método se dispara cuando el método unregister() desactiva correctamente las notificaciones de la aplicación.

  Parámetros:

  e: Este objeto contiene un código y mensaje que indican el éxito de la desactivación de notificaciones.

- function errorHandler(e)

  Este método se dispara cuando se ha producido un fallo en el proceso de desactivación de las notificaciones de la aplicación.

  Parámetros:

  e: Objeto que contiene el código y mensaje de error asociado al fallo en el proceso de desactivación de notificaciones.

- function addUri()

  El objetivo de este método es el de vincular y enviar al servidor, la URI obtenida por la plataforma de notificaciones y la identificación del usuario, de modo que cuando se quiera notificar una avería a un determinado usuario se tenga su dirección de envío.

- function onBackKeyDown()

  Este método se ejecuta cuando el usuario pulsa el botón “back” del terminal. Al pulsar esa tecla se preguntará al usuario si desea o no salir de la aplicación. Si la respuesta es afirmativa se detendrá la conexión con el servicio SignalR y se cerrará la aplicación.

- function onPause()

  La ejecución de este método se dispara cuando la aplicación pasa a segundo plano. Su objetivo es el de ejecutar la propiedad *enable* del plugin *backgroundMode*, lo que permite que la aplicación no se cierre y continúe ejecutándose con normalidad.

- function onForeground()

  Este método se dispara cuando la aplicación vuelve del segundo plano. Ejecuta la propiedad *disable* del plugin *backgroundMode* para desactivar la ejecución en segundo plano.

- function checkConnection()

  Este método permite conocer al usuario el estado de la conexión y el tipo de red al que se está conectado (**Wifi**, 3G, etc.).

- function changeUrl()

  Este método tiene como objetivo permitir la modificación de la URL a la que se tiene que conectar el cliente para establecer la conexión con el servicio SignalR.

- function clearData()

  Este método permite al usuario borrar todos los datos guardados por la aplicación, como son los datos de inicio de sesión, URI de notificación, etc.

- function beep()

  La ejecución de este método tendrá lugar cuando se reciban notificaciones y la aplicación esté en ejecución. Permite disparar un aviso sonoro por parte del terminal.

- function autoConnect()

  Este método comprobará si hay datos guardados del usuario (inicio de sesión) y conectará automáticamente con el servicio web si son válidos.

- function connect()

  Este método se encarga de iniciar la conexión con el servicio SignalR. Además soporta los manejadores de eventos necesarios para detectar y mostrar los datos enviados por el servicio.

- function logout()

  Este método se encarga de preguntar al usuario si desea cerrar su sesión. En caso afirmativo se disparará el método *loguoutOk* .

- function logoutOk(buttonindex)

  Este método es ejecutado cuando el usuario indica que quiere cerrar sesión. Se encargará de borrar del almacenamiento de la aplicación los datos del usuario en cuestión.

- function getFaults()

  Este método tiene como objetivo pedir al servicio SignalR información sobre todas las averías activas que tiene el usuario registrado en la aplicación.

- function getAckFaults()

  Este método tiene como objetivo pedir al servicio SignalR información sobre todas las averías activas y confirmadas que tiene el usuario registrado en la aplicación.

- function loadFaultsCollapsible(faultsCount, faults)

  Este método es el encargado de cargar la información de las averías activas en la pantalla principal de la aplicación.

  Parámetros:

  faultsCount: Indica el número de averías que se mostrarán en la pantalla principal.

faults: Este objeto contiene la información de las averías activas del usuario registrado en la aplicación (nombre de máquina y avería, y descripción).

- function createPage(id)

  Método empleado para crear una "pagina" de jQuery Mobile en la que cargar la vista de detalle de la avería. Se ejecutará cuando un usuario pulse sobre en nombre de la avería indicada.

  Parámetros:

  id: Identificador único de la avería seleccionada.

- function getFaultDetails(id)

  Este método pide al servicio web la información adicional de la avería seleccionada.

  Parámetros:

  id: Identificador único de la avería seleccionada.

- function FaultAck(id)

  Método encargado de preguntar al usuario si está seguro de confirmar la avería seleccionada. En caso afirmativo este método llamará a *onConfirmAck.*

  Parámetros:

  id: Identificador único de la avería seleccionada.

- function onConfirmAck(buttonindex)

  Método empleado para indicar al servicio web que el usuario ya ha recibido la notificación de avería y que se pondrá a solventarla lo antes posible.

  Parámetros:

  buttonindex: Entero usado a modo de identificador del botón pulsado en el mensaje de confirmación lanzado por el método *FaultAck*. Su valor será "1" si se pulsa *OK* y "0" si se pulsa *Cancel.*

- function filterMachine()

  Método empleado para filtrar todas aquellas averías cuyo nombre de máquina no corresponde con el introducido por el usuario en el campo habilitado para ello.

- function OnOffNotifications()

  Este método se encarga de preguntar al usuario si desea desactivar la recepción de notificaciones de la aplicación. Si la respuesta es afirmativa se ejecutará el método *confirmTurnOffPush*

- function confirmTurnOffPush(buttonindex)

  Método o evento encargado de llamar a la función *unregister* para iniciar la desactivación de las notificaciones de la aplicación.

  Parámetros:

  buttonindex: Entero usado a modo de identificador del botón pulsado en el mensaje de confirmación lanzado por el método *OnOffNotifications*. Su valor será "1" si se pulsa *OK* y "0" si se pulsa *Cancel.*

PLAN DE PRUEBAS

8. PRUEBAS DE REQUISITOS FUNCIONALES

El objetivo del presente apartado es el de indicar las pruebas que se han de realizar para comprobar que se cumplen los requisitos funcionales, derivados de los casos de uso presentados en el apartado de Análisis y Diseño. En los apartados siguientes se indicará que pruebas se han realizado para los siete casos de uso comentados.

8.1. Gestión de usuarios

Para comprobar se cumple el caso de uso "Gestión de usuarios" se deberán realizar las siguientes pruebas:

- Comprobar que se puede crear uno o varios usuarios en la aplicación.
- Comprobar que se pueden eliminar uno o varios usuarios de la aplicación.
- Comprobar que se muestra la información de usuario en pantalla al seleccionarlo.
- Comprobar que se pueden modificar los campos de datos del usuario seleccionado.
- Comprobar que se pueden guardar los datos modificados en la base de datos.

8.2. Gestión de grupos

Para comprobar que se cumple el caso de uso "Gestión de grupos" se deberán realizar las siguientes pruebas:

- Comprobar que se pueden crear uno o varios grupos de usuarios en la aplicación.
- Comprobar que se pueden eliminar uno o varios grupos de usuarios de la aplicación.
- Comprobar que se puede vincular un usuario a uno o varios grupos de usuarios.

8.3. Asignación de averías

Para comprobar que se cumple el caso de uso "Asignación de averías" se deberán realizar las siguientes pruebas:

- Comprobar que la aplicación muestra dos opciones de selección: Users o Groups.

- Comprobar que al seleccionar la opción Users se carga la lista de usuarios creados en el listbox asociado.

- Comprobar que al seleccionar la opción Groups se carga la lista de grupos creados en el listbox asociado.

- Comprobar que al iniciar el cuadro de dialogo de asignación de averías se muestra la lista con la definición de las averías que puede disparar el sistema de control.

- Comprobar que se puede seleccionar un usuario o grupo de la lista y seleccionar una o varias averías del grid.

- Comprobar que se puede asignar una o varias averías a un usuario/grupos.

- Comprobar que si al seleccionar un usuario/grupo con averías asignadas previamente, éstas se muestran de diferente color en el momento de la selección.

8.4. Gestión de reglas

Para comprobar que se cumple el caso de uso "Gestión de reglas" se deberán realizar las siguientes pruebas:

- Comprobar que la aplicación muestra dos opciones de selección: *Users* o *Groups*.

- Comprobar que al seleccionar la opción *Users* se carga la lista de usuarios creados en el listbox asociado.

- Comprobar que al seleccionar la opción *Groups* se carga la lista de grupos creados en el listbox asociado.

- Comprobar que al seleccionar un usuario/grupo se muestra en el grid de averías aquellas que éste tiene asignadas.

- Comprobar que se pueden añadir turnos de trabajo en el panel *Intervals.*
- Comprobar que se pueden eliminar turnos de trabajo en el panel *Intervals*.
- Comprobar que se pueden modificar los campos *Priority* o *WaitTime*.
- Comprobar que se añaden correctamente las reglas de notificación.

8.5. Visualizar averías activas

Para comprobar que se cumple el caso de uso "Visualizar averías activas" se deberán realizar las siguientes pruebas sobre la aplicación móvil:

- Comprobar que la aplicación móvil es capaz de identificar al usuario que inicia sesión.
- Comprobar que la aplicación móvil muestra la lista con las averías activas que tiene asignadas el usuario.
- Comprobar que en el segundo y posteriores inicios de la aplicación móvil no se piden los datos al usuario.
- Comprobar que al pulsar sobre la avería se despliega la lista, mostrando el nombre y descripción de la avería.

8.6. Confirmar averías

Para comprobar que se cumple el caso de uso "Confirmar averías" se deberán realizar las siguientes pruebas sobre la aplicación móvil:

- Comprobar que el usuario puede acceder al menú de detalle de avería.
- Comprobar que el usuario puede pulsar el botón de confirmación de avería.
- Comprobar que al confirmar la avería, se actualiza el campo ACK Date en la ventana de detalle.

8.7. Filtrar averías

Para comprobar que se cumple el caso de uso “Filtrar averías” se deberán realizar las siguientes pruebas sobre la aplicación móvil:

- Comprobar que al seleccionar la opción “Filtrar averías” se muestra el cuadro de búsqueda en la pantalla de visualización.

- Comprobar que al introducir el nombre de una maquina existente en el cuadro de búsqueda se filtrarán las averías producidas por el resto de máquinas.

- Comprobar que al introducir el nombre de una máquina inexistente en el cuadro de búsqueda no se filtrará ninguna avería.

9. PRUEBAS DE REQUISITOS NO FUNCIONALES

Se realizarán las pruebas pertinentes para asegurar que se cumplen los requisitos hardware y software indicados en el proyecto. Se deberán realizar las siguientes pruebas:

- Comprobar el correcto funcionamiento de la aplicación de gestión de usuarios/averías y el servicio web en los sistemas operativos Microsoft Windows XP/7/8.

- Comprobar el correcto funcionamiento de la aplicación móvil de visualización de averías en el sistema operativo Windows Phone 8.

- Comprobar si los requisitos mínimos del ordenador son válidos.

- Comprobar si los requisitos mínimos del Smartphone son válidos.

- Comprobar que las aplicaciones cliente y servidor pueden comunicarse entre sí.

- Comprobar que existe manual de usuario.

PLANIFICACIÓN
Y
PRESUPUESTO

10. PLANIFICACIÓN Y PRESUPUESTO

10.1. Descripción del documento

En el presente documento se incluye una estimación del coste que supone llevar a cabo el proyecto, así como la planificación temporal del mismo.

En cuanto al presupuesto hay que señalar que se detallan los recursos hardware, software y humanos necesarios, así como el presupuesto desglosado del proyecto y el presupuesto final del mismo.

En la planificación temporal, se desglosan, en primer lugar las tareas y fases que constituyen este proyecto y en último lugar se muestra el diagrama de Gantt con las diferentes tareas.

10.2. Documentos referenciados

A continuación se detallan los documentos relacionados con la planificación y el presupuesto de este proyecto.

10.2.1. Documentos del proyecto

No se hace referencias a ninguno otro documento del proyecto.

10.2.2. Documentos externos

Se hace referencia al trabajo fin de master **"Mejora en el sistema de gestión y visualización de averías de Galileo (Mecalux)"** desarrollado por Alberto Alvarez García para Mecalux Software Solutions.

10.3. Planificación

A la hora de elaborar la planificación del proyecto se ha considerado dividir esta en dos partes. Por un lado se estimará que bienes materiales, tanto de software como de hardware serán necesarios para el desarrollo de la aplicación. Y por otro se calculará el tiempo estimado de desarrollo de cada una de las tareas que componen el proyecto. En los apartados siguientes se podrá visualizar el desglose de los materiales necesarios.

10.3.1. Estimación de recursos necesarios

En las tablas siguientes se realiza una estimación de los diferentes productos hardware y software que serán necesarios para la elaboración del proyecto.

10.3.1.1. Recursos hardware

MECALUX SOFTWARE SOLUTIONS *Mecalux Failures Manager* - MEDICIONES RECURSOS HARDWARE			
ID UNIDAD	**DESCRIPCIÓN**	**UNIDAD DE MEDICIÓN**	**NUMERO DE UNIDADES**
HW01	Ordenador tipo PC	Uds.	1
HW02	Smartphone(WP8, Android o iOS)	Uds.	1

Tabla 6: Mediciones de recursos hardware.

10.3.1.2. Recursos software

MECALUX SOFTWARE SOLUTIONS *Mecalux Failures Manager* - MEDICIONES RECURSOS SOFTWARE			
ID UNIDAD	**DESCRIPCIÓN**	**UNIDAD DE MEDICIÓN**	**NUMERO DE UNIDADES**
SW01	Microsoft Windows 7	Uds.	1
SW02	Microsoft Windows 8.1	Uds.	1
SW03	Microsoft Visual Studio 2013	Uds.	1
SW04	MySQL Workbench	Uds.	1
SW05	Adobe Brackets	Uds.	1
SW06	Phonegap App Developer	Uds.	1
SW07	Microsoft Office 2013	Uds.	1
SW08	Google Chrome	Uds.	1

SW09	Internet Explorer	Uds.	1
SW10	.NET Framework 4.5	Uds.	1
SW11	Notepad++	Uds.	1
SW12	Microsoft Visio	Uds.	1

Tabla 7: Mediciones de recursos software.

10.3.2.Recursos humanos

Las labores de dirección técnica han sido realizadas por los directores del proyecto: D. Javier Piñera y D. José Antonio Cancelas. Se estima la dedicación a estas labores en dos horas semanales.

El programador encargado del desarrollo del proyecto debe disponer de conocimientos en aplicaciones para la plataforma Windows, concretamente en el lenguaje C#, experiencia en el trabajo con bases de datos y tener nociones de desarrollo en tecnologías web (HTML, Javascript, CSS).

Para el peritaje de los costes de los recursos humanos se han estimado unas seis horas diarias en las fases de análisis y diseño, y seis horas diarias en la fase de implementación, pruebas y documentación.

MECALUX SOFTWARE SOLUTIONS *Mecalux Failures Manager* - MEDICIONES RECURSOS HUMANOS			
ID UNIDAD	**DESCRIPCIÓN**	**UNIDAD DE MEDICION**	**NUMERO DE UNIDADES**
HU01	Análisis y diseño de software	Horas	200
HU02	Desarrollo de software	Horas	450
HU03	Dirección	Horas	50

Tabla 8: Mediciones de recursos humanos

10.4. Planificación temporal

A la hora de elaborar la planificación temporal y las diferentes etapas del proyecto, se ha de indicar que el presente desarrollador del proyecto no disponía de conocimientos de determinados lenguajes de programación, por lo que ha sido necesario intercalar las etapas de documentación de acuerdo a la tarea a desarrollar. En los apartados siguientes se muestra las diferentes etapas que se plantean en el desarrollo del proyecto así como un diagrama de Gantt en el que se detalla el periodo temporal necesario para la realización de las tareas mencionadas.

10.4.1. Etapas del proyecto

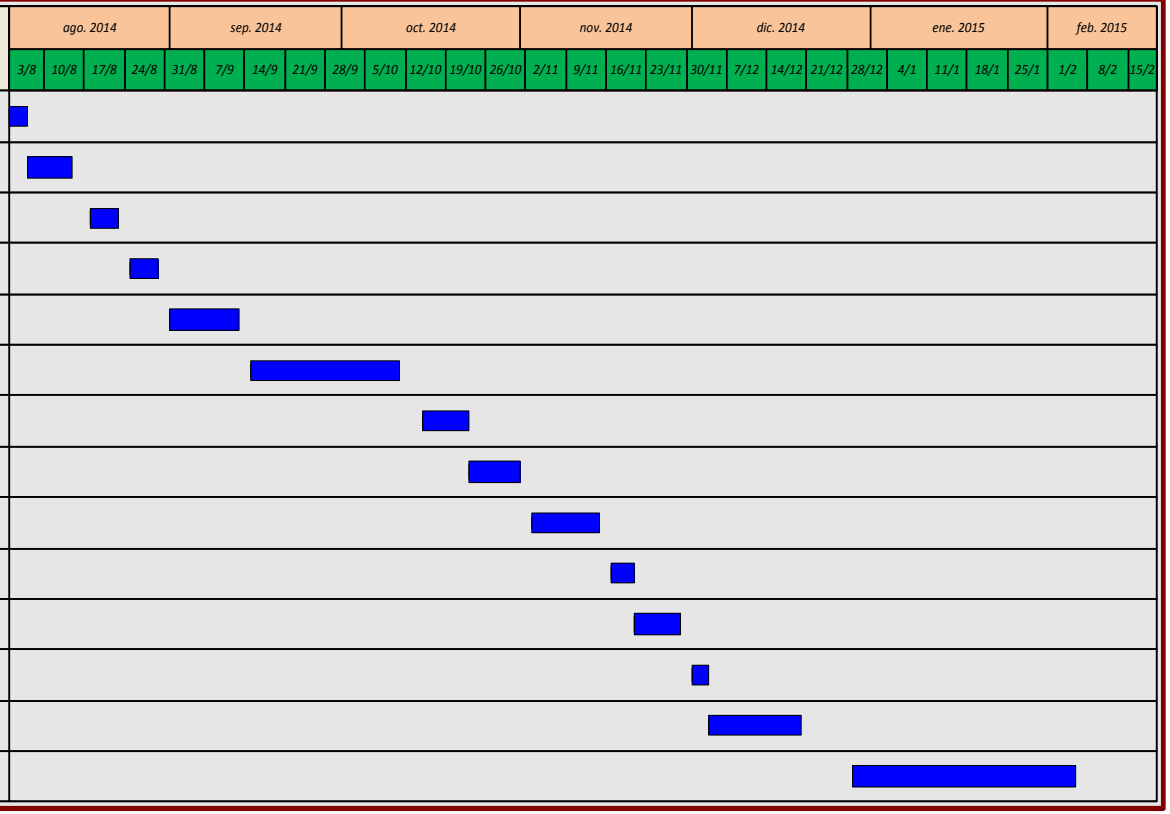
Las etapas que aparecen en el siguiente cuadro corresponden las diferentes tareas que se han tenido que llevar a cabo para la resolución del proyecto.

ID Etapa	Descripción
1	Planteamiento del problema y búsqueda de soluciones
2	Documentación sobre WCF
3	Creación de una plantilla Cliente – Servidor con WCF
4	Documentación sobre Entity Framework
5	Adición de un modelo de datos a la aplicación
6	Desarrollo del sistema Cliente – Servicio WCF
7	Documentación SignalR
8	Introducción de un servicio SignalR en la aplicación
9	Modificación del proyecto. Se elimina servicio WCF. Solo se emplea SignalR.
10	Documentación sobre HTML/Javascript/CSS
11	Desarrollo de un visor de averías con tecnologías web
12	Documentación Phonegap
13	Desarrollo de un visor de averías con Phonegap
14	Documentación adicional

Tabla 9: Etapas del proyecto

10.4.2. Diagrama de gantt

Id.	Nombre de tarea	Comienzo	Fin	Duración
1	**Planteamiento problema y busqueda de soluciones**	04/08/2014	06/08/2014	,6s
2	**Documentación WCF**	07/08/2014	14/08/2014	1,2s
3	**Creación de una plantilla Cliente – Servidor con WCF**	18/08/2014	22/08/2014	1s
4	**Documentación Entity Framework**	25/08/2014	29/08/2014	1s
5	**Adición de un modelo de datos a la aplicación con Entity Framework.**	01/09/2014	12/09/2014	2s
6	**Desarrollo de aplicación Cliente – Servidor WCF**	15/09/2014	10/10/2014	4s
7	**Documentación SignalR**	15/10/2014	22/10/2014	1,2s
8	**Introducir servicio SignalR al proyecto**	23/10/2014	31/10/2014	1,4s
9	**Modificación proyecto. Eliminación servicio WCF. Solamente con SignalR.**	03/11/2014	14/11/2014	2s
10	**Documentación sobre HTML + Javascript + CSS**	17/11/2014	20/11/2014	,8s
11	**Desarrollo de un visor de averías con tecnologías web (HTML/Javascript/CSS)**	21/11/2014	28/11/2014	1,2s
12	**Documentación Phonegap**	01/12/2014	03/12/2014	,6s
13	**Desarrollo de un visor de averías con Phonegap**	04/12/2014	19/12/2014	2,4s
14	**Documentación adicional**	29/12/2014	05/02/2015	5,8s



11. PRESUPUESTO

11.1. Mediciones

Consultar el apartado 8.3.1. “Estimación de recursos necesarios”.

11.2. Cuadro de precios

En las tablas siguientes se muestran los cuadros de precios relativos al software, hardware y personal necesario, detallando el precio unitario de cada concepto.

11.2.1.Recursos hardware

MECALUX SOFTWARE SOLUTIONS *Mecalux Failures Manager* - CUADRO PRECIOS RECURSOS HARDWARE			
ID UNIDAD	**DESCRIPCIÓN**	**UNIDAD DE MEDICIÓN**	**PRECIO UNITARIO**
HW01	Ordenador tipo PC	Uds.	750
HW02	Smartphone(WP8, Android o iOS)	Uds.	150

Tabla 10: Cuadro de precios de los recursos hardware

11.2.2.Recursos software

MECALUX SOFTWARE SOLUTIONS *Mecalux Failures Manager* - CUADRO PRECIOS RECURSOS SOFTWARE			
ID UNIDAD	**DESCRIPCIÓN**	**UNIDAD DE MEDICIÓN**	**PRECIO UNITARIO (EUROS)**
SW01	Microsoft Windows 7 PRO	Uds.	171.72
SW02	Microsoft Windows 8.1 PRO	Uds.	279.00
SW03	Microsoft Visual Studio 2013 PRO	Uds.	387.00
SW04	MySQL Workbench Community	Uds.	0
SW05	Adobe Brackets	Uds.	0
SW06	Phonegap App Developer	Uds.	0
SW07	Microsoft Office 2013	Uds.	119.00
SW08	Google Chrome	Uds.	0
SW09	Internet Explorer	Uds.	0
SW10	.NET Framework 4.5	Uds.	0
SW11	Notepad++	Uds.	0
SW12	Microsoft Visio	Uds.	399.00

Tabla 11: Cuadro de precios de los recursos software

11.2.3.Recursos humanos

MECALUX SOFTWARE SOLUTIONS *Mecalux Failures Manager* - CUADRO PRECIOS RECURSOS HUMANOS			
ID UNIDAD	**DESCRIPCIÓN**	**UNIDAD DE MEDICION**	**PRECIO UNITARIO (EUROS)**
HU01	Análisis y diseño de software	Horas	20
HU02	Desarrollo de software	Horas	12
HU03	Dirección	Horas	60.10

Tabla 12: Tabla de precios de los recursos humanos

11.3. Presupuestos parciales

En las tablas siguientes se detalla el importe parcial de los recursos hardware, software y humanos necesarios para la elaboración del proyecto. Para el desarrollo de estos presupuestos parciales se ha tenido en cuenta el número de unidades de cada concepto así como el precio unitario de cada uno.

11.3.1.Recursos hardware

MECALUX SOFTWARE SOLUTIONS *Mecalux Failures Manager* - PRESUPUESTO RECURSOS HARDWARE		
ID UNIDAD	**DESCRIPCIÓN**	**IMPORTE (EUROS)**
HW01	Ordenador tipo PC	750
HW02	Smartphone(WP8, Android o iOS)	150
PRESUPUESTO RECURSOS HARDWARE		900

Tabla 13: Presupuesto parcial de los recursos hardware

11.3.2.Recursos software

MECALUX SOFTWARE SOLUTIONS *Mecalux Failures Manager* - PRESUPUESTO RECURSOS SOFTWARE		
ID UNIDAD	**DESCRIPCIÓN**	**IMPORTE (EUROS)**
SW01	Microsoft Windows 7 PRO	171.72
SW02	Microsoft Windows 8.1 PRO	279.00
SW03	Microsoft Visual Studio 2013	387.00
SW04	MySQL Workbench	0
SW05	Adobe Brackets	0
SW06	Phonegap App Developer	0
SW07	Microsoft Office 2013	119.00
SW08	Google Chrome	0
SW09	Internet Explorer	0
SW10	.NET Framework 4.5	0
SW11	Notepad++	0
SW12	Microsoft Visio	399.00
PRESUPUESTO RECURSOS SOFTWARE		1355.72

Tabla 14: Presupuesto parcial de los recursos software

11.3.3.Recursos humanos

MECALUX SOFTWARE SOLUTIONS *Mecalux Failures Manager* - PRESUPUESTO RECURSOS HUMANOS		
ID UNIDAD	**DESCRIPCIÓN**	**IMPORTE (EUROS)**
HU01	Análisis y diseño de software	4000
HU02	Desarrollo de software	5400
HU03	Dirección	3000
PRESUPUESTO RECURSOS HUMANOS		12400

Tabla 15: Presupuesto parcial de los recursos humanos

11.4. Presupuesto final

MECALUX SOFTWARE SOLUTIONS *Mecalux Failures Manager* - PRESUPUESTO FINAL	
CAPITULO	**IMPORTE TOTAL (EUROS)**
Recursos Hardware	900
Recursos Software	1355.72
Recursos Humanos	12400
TOTAL	14655.72

Tabla 16: Presupuesto final

		EUROS
Presupuesto de Ejecución de Material	=	14655.72
Beneficio Industrial (6%)	=	879.34
Costes Generales (15%)	=	2198.36
Suma de Gastos y Beneficios		**17733.42**
I.V.A. (21%)	=	3724.02
Presupuesto de Ejecución por Contrata		**21457.44**

Asciende el presupuesto de ejecución por contrata a la expresada cantidad de **veintiún mil cuatrocientos cincuenta y siete euros (21457.4)**.

Gijón, a 5 de Febrero de 2015

Firmado: **Alberto Álvarez García (DNI 53518086-F)**

BIBLIOGRAFÍA

12. BIBLIOGRAFÍA

[1] Introducción a SignalR (Microsoft):http://www.asp.net/signalr/overview/getting-started/introduction-to-signalr

[2] Phonegap: http://phonegap.com/

[3] Documentación completa de Phonegap: http://docs.phonegap.com/en/4.0.0/index.html

[4] Descripción y características Phonegap: http://es.wikipedia.org/wiki/PhoneGap

[5] Entity Framework: http://msdn.microsoft.com/es-es/library/bb399567%28v=vs.110%29.aspx

[6]Windows Communication Foundation: http://msdn.microsoft.com/es-es/library/ms731082%28v=vs.110%29.aspx

[7] Manual de uso Software – Designer IV (Mecalux – Dpto. ITSW)

[8] Galileo vs PLC (Mecalux – Dpto. ITSW)

[9] Manual de uso Software – Sistema de Control Galileo IV (Mecalux – Dpto. ITSW)

[10] Instalación del conector MySQL para .NET: http://dev.mysql.com/downloads/connector/net/

[11] Añadir plugins a un proyecto Phonegap: http://docs.build.phonegap.com/en_US/configuring_plugins.md.html#Plugins

[12] Theme Roller jQuery Mobile: http://themeroller.jquerymobile.com/

[13] ASP.NET SignalR Incredibly simple real-time features for your web apps, Jose M.Aguilar, Campus MVP.NET

[14] Documentación HTML+Javascript+CSS: http://www.w3schools.com/html/default.asp

[15] Tutorial de desarrollo de aplicaciones Phonegap: http://www.desarrolloweb.com/manuales/aplicaciones-moviles-phonegap.html

[16] Documentación plugin Phonegap de notificaciones: https://github.com/phonegap-build/PushPlugin/tree/1979d97

[17] Envío de notificaciones Windows Phone 8: https://msdn.microsoft.com/en-us/library/windows/apps/hh202945(v=vs.105).aspx