UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA INFORMÁTICA

# APLICACIÓN WEB DE GESTIÓN DE LA PRODUCCIÓN PARA ROBOTS DE ALMACÉN

# MANAGEMENT PRODUCTION WEB APP FOR WAREHOUSE ROBOTS


Realizado por
**Álvaro Martín Cabrerizo**

Tutorizado por
**Luis Llopis Torres**
**Enrique Soler Castillo**

Departamento
**Departamento de Lenguajes y Ciencias de la Computación**

UNIVERSIDAD DE MÁLAGA
MÁLAGA, DICIEMBRE DE 2020



Fecha defensa: diciembre de 2020

# Resumen

Debido a la gran expansión de la tecnología en los ámbitos empresariales, tanto a nivel directivo como a nivel operativo, es cada vez más fácil la integración de nuevas herramientas en cualquier empresa que quiera abstraer su modelo de negocio y dejar las tareas rudimentarias a tecnologías mejor optimizadas como pueden ser los robots. En nuestro caso, se ha desarrollado una aplicación web que sirva como núcleo gestor de todas las comunicaciones entre los operarios y unos nuevos robots de almacén que están en desarrollo. Estos robots se están desarrollando siguiendo la directiva "low-cost", con idea de integrarlos fácilmente en cualquier empresa que disponga de un almacén con inventario móvil constante. Además, la aplicación web será capaz de tratar la información para mostrar los indicadores de rendimiento claves que servirán para la toma de decisiones de la empresa, así como de llevar un registro de las misiones y de los objetos que se están transportando. Se ha optado por utilizar las facilidades que tienen los nodos IoT Hub de Azure para centralizar todo el paso de mensajes en la nube y, así, poder permitir que el robot sea independiente de su entorno si los operarios se encuentran lejos de la plataforma de trabajo. Además, Azure permite una muy buena escalabilidad que se ajusta precisamente al presupuesto y las necesidades que tenga la empresa cliente. Para estudiar el comportamiento de los robots se han desarrollado simuladores en Python que implementan los mensajes que un robot va a emitir. Por último, la aplicación web se ha desarrollado usando las tecnologías de Angular (front-end) conectadas a .Net Core (back-end), que es el encargado de comunicarse con los nodos en Azure y gestionar la información en la base de datos.

**Palabras clave:** **Gestión de la producción, robot de almacén, aplicación web, IoTHub Azure, indicadores de rendimiento.**

# Abstract

Due to the huge expansion of technology in business areas, both at top-level and low-level, it is increasingly easier to integrate new tools in any company that wants to abstract its business model and leave rudimentary tasks to better technologies optimized such as robots. In our case, a web application has been developed to serve as core of all communications between operators and new warehouse robots that are under development. These robots are being developed following the "low-cost" directive, with the idea of easily integrating them into any company that has a warehouse with constant mobile inventory. In addition, the web application will be able to process the information to show the key performance indicators that will be used for the decision-making of the company, as well as to keep a record of the missions and the objects that are being transported. It has been decided to use the facilities that the Azure IoT Hub nodes have to centralize the entire message flow in the cloud and thus be able to allow the robot to be independent of its environment if the operators are far away from the work platform. Also, Azure allows a very good scalability that adjusts precisely to the budget and the needs of the client company. In order to study the behavior of robots, we developed Python simulators to implement the messages that a robot is going to emit. Finally, the web application has been developed using Angular technologies (front-end) connected to .Net Core (back-end), which is in charge of communicating with the nodes in Azure and managing the information in the database.

**Keywords:** **Management production, warehouse robot, web application, IoTHub Azure, key performance indicators**

# Índice

**Índice** ..... **1**

**Introducción** ..... **1**
**1.1 Motivación** ..... **1**
**1.2 Objetivos** ..... **2**
**1.3 Estructura de la memoria** ..... **3**
**1.4 Metodología de trabajo empleada** ..... **4**

**Tecnologías** ..... **9**
**2.1 Net Core y Entity Framework** ..... **9**
**2.2 Visual Studio Code y Visual Studio Community** ..... **10**
**2.3 MS SQL Server** ..... **11**
**2.4 Angular y Node.js** ..... **11**
**2.5 Python** ..... **12**
**2.6 Azure** ..... **13**
**2.7 OpenApi, Swagger y Postman** ..... **14**

**Fases del desarrollo** ..... **17**
**3.1 Diseño** ..... **17**
3.1.1 Arquitectura ..... 17
3.1.2 Definición de actores ..... 19
3.1.3 Casos de uso y de secuencia ..... 19
3.1.4 Definición de mensajes ..... 21
3.1.5 Modelado de datos ..... 22
**3.2 Implementación** ..... **25**
3.2.1 Recursos en Azure ..... 25
3.2.2 Simulador PMR ..... 27
3.2.3 Conectividad plataforma IoT-ERP ..... 29
3.2.4 API para conectividad con ERP ..... 31
3.2.5 Mockups Aplicación Web ..... 32
3.2.6 Desarrollo de Back-End ..... 32
3.2.7 Desarrollo Web ..... 33
3.2.8 Conexión Back-Front End ..... 35
3.2.9 Prueba de ciclo completo ..... 36
**3.3 Indicadores claves de rendimiento (KPIs)** ..... **37**

**Conclusiones y Líneas Futuras** ..... **41**

**Referencias** ..... **43**

**Diagramas de secuencia** ..... **45**
**Diagramas de secuencia entre ERP y WebApp** ..... **58**

**Mensajes MQTT** ..... **63**

**Mockups Aplicación Web** ..... **69**

# 1

# Introducción

## 1.1 Motivación

La popularización de sistemas automáticos autónomos, de la robótica flexible y colaborativa está impulsando la petición de nuevos productos por parte de las empresas, lo cual está abriendo nuevos mercados.

En los últimos años, la demanda de una plataforma móvil robotizada (en adelante PMR) se ha visto incrementada, siendo una petición que han recibido tanto integradores, ingenierías y centros de investigación. La oportunidad de negocio para las empresas que puedan suministrar esta tecnología es clara, pero para alcanzar un volumen de negocio adecuado, la solución propuesta debe mantener un buen ratio de costo-beneficio.

Se busca un prototipo funcional que responda a las necesidades de manipulación presentes en un entorno de fabricación ágil, donde las capacidades del PMR puedan ser explotadas al máximo, tanto desde el punto de vista de la conectividad, como desde el punto de vista del producto. El proyecto está concebido como un producto modular con enfoque al mercado.

El caso de uso del proyecto está tomado en referencia a una empresa española de construcción de piezas para aeronáutica que tiene un sistema de almacenaje idónea al que se le puede integrar la solución propuesta fácilmente. Además, se explorará la

expansión de su negocio a otros mercados (principalmente al de automoción) de la mano de las tecnologías digitales a utilizar en este proyecto. La empresa posee una planta de ensamblaje en Sevilla, con 23000 m2 y distribuida en tres plantas. Para la fabricación de sus diferentes productos, sus materiales y piezas se mueven por y entre las diferentes plantas.

Figura 1.1 Uno de los almacenes de piezas

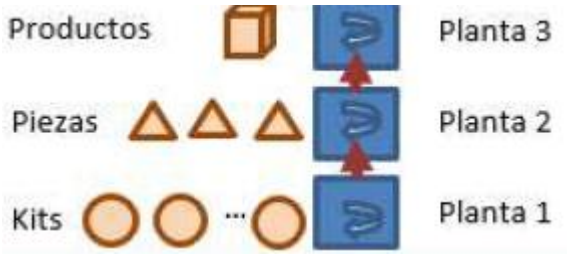



Figura 1.2. Distribución de componentes por plantas

Para transportar este tipo de piezas y productos, los empleados recorren las plantas con carritos y también los llevan en ascensor a través de las diferentes plantas. Este transporte requiere mucho tiempo y no tienen ningún valor añadido. Además, al ser una instalación donde se implementó la fabricación ágil, cuenta con espacios abiertos y bien diferenciados. Estas condiciones lo hacen perfecto para probar el PMR propuesto en este proyecto.

## 1.2 Objetivos

El objetivo principal de este proyecto es desarrollar toda la infraestructura informática necesaria para que el PMR pueda desarrollar las misiones para las que ha sido concebido, estas son, recoger con un sistema de visión por computador los materiales

en los almacenes de piezas (figura 1.1 y 1.2) y llevarlos a los trabajadores directamente para que estos no tengan que perder tiempo moviéndose entre las plantas de producción.

Para ello, realizaremos una aplicación web con la que los operarios puedan comunicarse con los robots, y ver en tiempo real los indicadores de rendimiento propios que servirán posteriormente para la consecuente toma de decisiones en la empresa.

Además, se requerirá de una librería propia con la que el PMR se pueda comunicar directamente con nuestra aplicación y toda la configuración en la nube que haga de intermediaria entre el PMR y la aplicación en cuestión. Más concretamente:

- Poder ser utilizado muy fácilmente por no expertos para adaptarse rápidamente a nuevas variantes de productos.
- Estar integrado en líneas de producción, con conectividad a los sistemas de gestión o sistemas de planificación de recursos empresariales (a continuación ERPs), y conectividad 4.0 como cualquier otro medio productivo.
- La solución PMR desarrollada será más económica de las actuales combinaciones para PMR de empresas alemanas o norteamericanas.
- Hacer uso de sistemas de conectividad *Internet of Things* (IoT en adelante) para medir los indicadores claves de rendimiento (KPIs en adelante) del sistema y optimizar las tareas programadas por el ERP.
- Implementar una solución estable para exportar información del PMR hacia la nube.
- Proporcionar una aplicación de usuario para la utilización y gestión de la plataforma a partir de la cual comandar las diferentes acciones a realizar.

## 1.3 Estructura de la memoria

El presente documento se ha estructurado siguiente la cronología del proyecto desde su inicio hasta el fin. En primera instancia, el lector se encontrará con el capítulo referente al diseño de todas las partes partícipes y necesarias en nuestra aplicación. Este

capítulo contempla desde la definición de actores hasta el modelado de datos final que se llevará a desarrollo. A continuación, pasaremos a detallar la parte de implementación, comentando en cada fragmento las tecnologías que hemos utilizado y las elecciones que hemos tomado al encontrarnos con dificultades no contempladas. Una vez que hayamos explorado la parte de diseño e implementación, vamos a detallar y definir los indicadores claves de rendimiento que serán utilizados en la aplicación para la toma de decisiones de los ejecutivos. Por último, presentaremos los resultados obtenidos en los test de validación de la herramienta para comprobar que su funcionamiento es correcto. En la parte final del documento se añaden las referencias utilizadas en el documento así como, uno apéndices con la información complementaria que no tiene cabida en el cuerpo de la memoria.

## 1.4 Metodología de trabajo empleada

El proyecto va a seguir la metodología más estandarizada en nuestros días que, a su vez, ha sido la impartida durante los años de formación en la carrera.

En primera instancia, se pondrá en consenso los casos de usos y los diagramas de secuencia necesarios para el correcto funcionamiento de la aplicación.

Una vez que la parte de recolección de requisitos esté cerrada, se procederá con la fase de diseño que incorporará todas las tareas necesarias para documentar perfectamente cada punto de la plataforma web

Luego, se pasará a la implementación de la herramienta siguiendo una metodología Ágil [1] que se pueda adoptar a los cambios imprevistos, puesto que el proyecto general se enmarca dentro de la categoría I+D y es lo que mejor se amolda en nuestro caso. En concreto, se seguirán las recomendaciones que contempla la metodología Scrum [2] intentando sobre todo mantener revisiones de los hitos que se van haciendo mediante reuniones periódicas con la empresa. Además, se irán haciendo diferentes iteraciones de desarrollo acorde a las funcionalidades que hagan falta presentar en cada revisión de trabajo. Para ilustrar mejor esta metodología, en las figuras 1.3 y 1.4 se puede ver de una manera más visual el proceso que se ha llevado a cabo.


Figura 1.3 Esquema metodología Agile

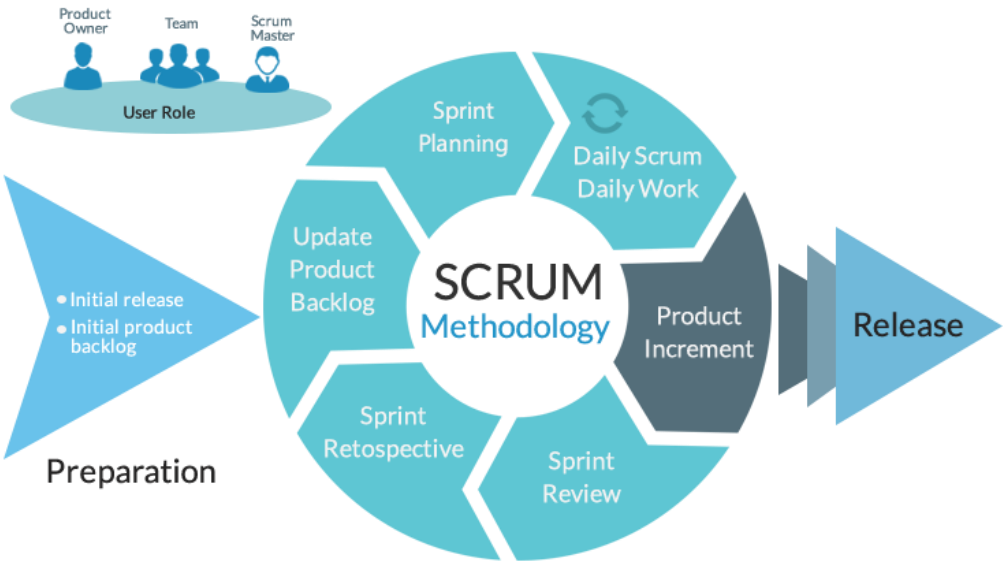



Figura 1.4 Esquema desarrollo Scrum

Para nuestro caso se han definidos las siguientes fases de desarrollo bien diferenciadas entre ellas y abstrayendo lo más posible cada fase como una unidad única e independiente de información. Estas son:

- Recolección de Requisitos: recoger todos los requisitos, funcionales y no funcionales, que requerirá la aplicación.
- Investigación y estudio: se procederá al estudio y aprendizaje de las herramientas que tengan que ser usadas y de las que no se disponga ningún tipo de formación.
- Fase de diseño: se desarrollará toda la documentación necesaria para una correcta implementación de la solución.
    - Definición de Actores: identificación de los actores que van a participar en la web.
    - Casos de uso: definición de los casos de usos acorde a los actores.
    - Diagramas de secuencia: diseño de los diagramas de secuencia para los diferentes casos de uso
    - Definición de mensajes: definir el formato estandarizado de los mensajes que se enviarán entre el robot y la plataforma
    - Modelado de datos: diseño de las entidades con los atributos necesarios para el desarrollo del proyecto.
- Implementación: desarrollo de la plataforma web con todas las funcionalidades especificadas. Se irán desarrollando subtareas siguiendo la metodología iterativa incremental propia del proceso Ágil.
    - Simulador de Robot: se hará una pequeña aplicación de consola que simule al robot para comprobar la conexión entre el robot y el nodo de Azure IoT Hub.
    - Back-end para recepción de mensajes: plataforma que se encargue de conectarse al nodo de Azure y recibir y tratar correctamente los mensajes del robot.
    - Desarrollo API: se implementará una API que se encargará de leer y escribir datos en la base de datos centralizada.

    - Comprobación de ciclo general de mensajería: se probará todo el ciclo que siguen los mensajes en ambas conexiones para comprobar la correcta conectividad de la herramienta.
    - Mockups Página Web: bocetos para la aplicación web que recojan todos los requisitos del proyecto.
    - Creación de recursos en Azure: será necesario dar de alta en Azure servicios como una base de datos para almacenar la información relevante de los mensajes.
    - Desarrollo Demo Web: se implementará una web con las funcionalidades mínimas necesarias para probar toda la cadena de conexión y visualización de datos.
    - Conexión Back-Front End: conectar ambas partes para ver si se reciben y procesan los datos correctamente.
- Implementación de los KPIs: definición y cálculo en la plataforma de los indicadores de rendimiento utilizando los datos proporcionados por los robots.
- Test y validación final: se comprobará el correcto funcionamiento de todo el sistema.

# 2

# Tecnologías

De manera muy breve, vamos a presentar las distintas herramientas y tecnologías que vamos a utilizar durante el desarrollo de nuestra aplicación. Comentaremos los argumentos más relevantes de por qué hemos elegido estas herramientas y no otras para la realización del mismo y cuáles son las ventajas de las que podemos hacer uso.

## 2.1 Net Core y Entity Framework

.Net Core [3] es una plataforma de desarrollo de código abierto para uso general. Con ella, podemos crear fácilmente aplicaciones de consola muy rápidamente tanto para Windows, macOs y Linux. Lo que nos interesa de esta tecnología es la gran cantidad de librerías y utilidades que contempla para el desarrollo de APIs en la nube y tecnologías IoT. Además, como esta plataforma pertenece a Microsoft, encontramos muchas más facilidades y documentación a la hora de conectarnos con los recursos de Azure.

Como complemento, haremos uso del paquete Entity Framework [4] para esta solución que nos permitirán desarrollar todo el software orientado a datos siguiendo la directiva *code-first*, es decir, programaremos todo el modelado de datos para después volcarlo automáticamente en los gestores de base de datos y generar el esquema directamente. Con este conjunto de tecnologías, podremos trabajar en un nivel más alto de abstracción cuando trabajemos con datos y podremos mantener toda la infraestructura dedicada a datos con menos código que en las aplicaciones tradicionales.

Al trabajar con *code-first*, el modelo conceptual se asigna al modelo de almacenamiento en código. Entity Framework puede deducir el modelo conceptual en función de los tipos de objeto y las configuraciones adicionales que defina. Los metadatos de asignación se generan durante el tiempo de ejecución basándose en una combinación de cómo se definen los tipos de dominio e información de configuración adicional que se proporciona en código. Por último, Entity Framework genera la base de datos según sea necesario en función de esos metadatos.


Figura 2.1 Logo .Net Core


Figura 2.2 Logo Entity Framework

## 2.2 Visual Studio Code y Visual Studio Community

Para todo el proyecto, hemos utilizado los dos entornos gratuitos por defecto que nos brinda Microsoft: Visual Studio Code [4] y Visual Studio Community [5], para poder programar cualesquiera de las partes. No hemos querido innovar y utilizar otras tecnologías, ya que, estos entornos tienen bien integrado todos los paquetes necesarios para trabajar con multitud de lenguajes de programación y además, ya vienen con todas las facilidades de Microsoft para conectarnos de una manera fácil e intuitiva a todas las herramientas de Azure que vamos a utilizar.

Figura 2.3 Logo Visual Studio Code


Figura 2.4 Logo de Visual Studio Community

## 2.3 MS SQL Server

Microsoft SQL Server [6] es un sistema de gestión de bases de datos relacional desarrollado por Microsoft. Esta herramienta será utilizada para el almacenamiento y la persistencia de los datos de la aplicación, de forma que puedan consultarse todos los datos referentes a las misiones que los robots tendrán que ejecutar. Además, se guardará toda la información que se vaya a utilizar por cualquier herramienta. Vamos a utilizar el programa Microsoft SQL Server Management Studio 18 para gestionar las bases de datos más fácilmente y de una manera más visual.


Figura 2.5 Logo SQL Server

## 2.4 Angular y Node.js

Para el desarrollo de toda la parte que estará en contacto con el usuario final (front-end), hemos optado por utilizar las tecnologías que nos brinda Angular [7] en su versión número 9 junto con los paquetes bastante flexibles que podemos encontrar de Node.js [8], que nos permiten una personalización total y rápida de páginas web receptivas a cambios y a tamaños. (lo que en informática se conoce como *webs responsives*).

Estas soluciones componen un conjunto de herramientas para JavaScript gratuitas y de código abierto creadas por Google para el desarrollo de páginas y aplicaciones web SPA (Single Page Application). Este modelo de carga permite la navegación por la página sin la necesidad de recargarla, haciendo que cuando el usuario entra en la web se muestre el contenido de todas las páginas a la vez. Esto hace que la primera carga nada más entrar sea más lenta pero, conseguimos que los cambios entre páginas sean

instantáneos. Además, solo se utiliza un fichero con el código de la web por lo que no se requiere de muchas llamadas a servidor.

Otra de las características de Angular es que utiliza TypeScript [9], una extensión del lenguaje JavaScript, la cual se convierte en última instancia a este último. Esta extensión, proporciona tipado estricto, interfaces, inyección de dependencias y muchas más ventajas que nos serán muy útiles.

La metodología de trabajo con Angular se enfoca en el uso en dos conceptos: el patrón MVC (modelo-vista-controlador) y la distribución en componentes. Esta forma de desarrollo permite organizar proyectos de grandes dimensiones en pequeños componentes y trabajar con la composición de éstos.

Por todo ello se ha decidido incluir estas tecnologías en el desarrollo del proyecto donde la visualización de información dinámica es una pieza clave.

Figura 2.6 Logo de Angular

Figura 2.7 Logo Node.js

## 2.5 Python

Para simular el paso de mensajes que el robot lanzará a la nube, necesitamos un lenguaje de programación bastante usado, con unas buenas librerías actualizadas y que nos permitan generar librerías y código ejecutable bastante rápido.

Python [10] es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, que soporta orientación a objetos, programación imperativa y, menor medida, programación funcional. Por ello, nos hemos decidido por este lenguaje ya que será

mucho más fácil integrar varias tecnologías que utilicen lenguajes diferentes con Python, que es de uso común y habitual por todo el mundo y todas las desarrolladoras de software. Hemos utilizado  la versión más actualizada al inicio de este proyecto (Python 3.7).

Figura 2.8 Logo de Python

## 2.6 Azure

Llegamos a la tecnología estrella de toda nuestra plataforma. El núcleo que va a gestionar toda la plataforma va a ser Azure. Hemos pensado todo el proyecto a través de esta tecnología porque nos ofrece muchas herramientas de desarrollo en la nube gratis y bajo de manda para proyectos basados en la nube e IoT. Esto nos va a permitir abstraernos del espacio físico de trabajo donde los robots estén funcionando y por comandar acciones y ver resultado desde cualquier parte con solo una conexión a internet.

Azure [11] es un conjunto de funcionalidades en la nube que Microsoft ha estado desarrollando en los últimos años debido a la gran expansión de negocios a plataformas virtuales y descentralizar su modelo de negocio para tener mayor presencia internacional. Gracias a las inversiones por parte de la empresa de Gates, estos años se han forjado una reputación en lo que respecta a computación en la nube y a infraestructura para computación lo que la hace muy buena opción para la creación de proyectos de este calibre. Además, los sistemas de pago son escalables en cualquier momento y según la necesidad de cómputo que requiramos, por lo que no nos es necesario pagar de más ni contratar servicios que no vayamos a utilizar.


Figura 2.9 Logo de Azure

## 2.7 OpenApi, Swagger y Postman

Unos de los objetivos anteriormente comentados es la necesidad de que el sistema se conecta a los ERPs de las empresas que quieran integran esta solución en su plan de producción. Por ello, hemos decidido optar por la metodología OpenApi [12] para realizar todas las comunicaciones que no podamos controlar al salir de nuestro control. La especificación OpenAPI define una interfaz estándar independiente del lenguaje para las interfaces que cumplen con la especificación REST o interfaces RESTful [13], permitiendo que tanto los humanos como las computadoras descubran y comprendan las capacidades del servicio sin acceso al código fuente, documentación o mediante la inspección del tráfico de la red. Cuando se define correctamente, un consumidor puede comprender e interactuar con el servicio remoto con una cantidad mínima de lógica de implementación.

Figura 2.10 Logo de Swagger


Figura 2.11 Logo de Postman

OpenAPI también provee herramientas de generación de para mostrar la API, herramientas de generación de código para generar servidores y clientes en varios lenguajes de programación, herramientas de prueba y muchos otros casos de uso. La implementación concreta que utilizar será la librería de código abierto NSwag [14], la cual proporciona herramientas para generar especificaciones OpenAPI a partir de

controladores de .NET Core existentes y código de cliente a partir de estas especificaciones. Utilizaremos el entorno de trabajo Swagger [15] que implementa esta solución. Para comprobar que las llamadas a la API devuelven los datos correctos, vamos a hacer uso de un programa bastante útil y muy conocido llamado Postman [16]. Este programa nos va a permitir simular las peticiones a la API y nos será de gran ayuda en la elaboración del proyecto.

# 3

# Fases del desarrollo

## 3.1 Diseño

Tras hacer un repaso general por las tecnologías y herramientas que vamos a utilizar en nuestro proyecto, nos adentramos en las distintas fases del desarrollo y cómo hemos ido gestionando el progreso de la aplicación. En este primer bloque, vamos a ver todas las partes relacionadas con el diseño de la aplicación: desde los actores que van a hacer uso de las solución, la arquitectura de desarrollo pasando por el modelado de datos que vamos utilizar. Una vez que veamos esta parte ya estaremos en disposición de entrar en el siguiente bloque que abarcará toda la implementación per se de nuestro trabajo.

### 3.1.1 Arquitectura

La arquitectura IoT propuesta para el proyecto es una arquitectura de computación sin servidor (serverless) basada en un proveedor de cloud comercial. Esta arquitectura tiene diferentes partes diferenciadas con objetivos diferentes:

- Broker MQTT: servidor de traducción y direccionamiento de mensajes para el protocolo MQTT
- WebApp: aplicación de gestión global y monitorización del sistema.
- Database: base de datos para persistencia de KPIs y gestión
- API de integración : API para el intercambio de información con el ERP de la empresa.

Adicionalmente, se propone la implementación de una librería para abstraer a la plataforma PMR de las comunicaciones de forma que simplifique e independice a la plataforma de las comunicaciones.

Cabe destacar que este diseño, basado en servicios independientes favorece la flexibilidad de la plataforma para adaptarse a diferentes escenarios de conectividad y uso.

En el siguiente diagrama se describe conceptualmente la arquitectura IoT propuesta para el proyecto.

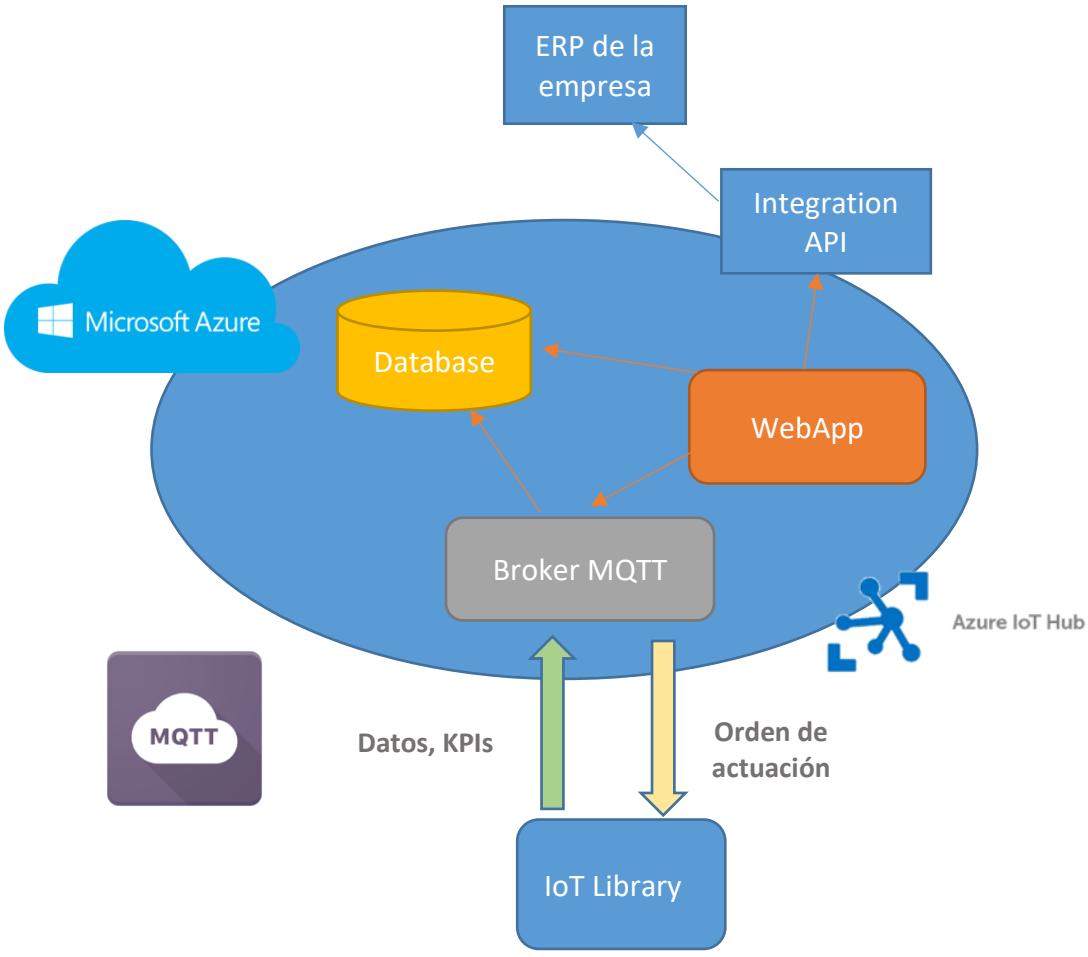



Figura 3.1 Arquitectura planteada

### 3.1.2 Definición de actores

Puesto que es un proyecto con unas funcionalidades muy específicas y que, en principio, tiene que ser básico para que cada empresa tenga la capacidad de integrar más funcionalidades si lo consideran oportuno, vamos a diferenciar únicamente dos actores, ya que son los que necesitamos para manipular la aplicación web. Estos son:

- Operator: operador común de los centros de producción que utilizarán la aplicación para pedir objetos a los robots y que estos se los traigan de manera independiente.
- Administrator: un actor para la administración interna de los propios operadores con todos los privilegios necesarios para la creación, modificación y eliminación de los distintos perfiles de los operadores de la empresa cliente.

Puesto que no se han concebido más funcionalidades específicas para la aplicación de las que ya han sido plasmadas con anterioridad, no vamos a hacer uso de ningún actor más de los que no sean estrictamente necesarios para no generar puntos abiertos en el desarrollo que no hayan sido contemplados.

### 3.1.3 Casos de uso y de secuencia

Una vez que tenemos los actores bien especificados y definidos, vamos a detallar los diferentes casos de uso [17] de la plataforma y su relación con el resto de elementos del sistema. Presentamos en un primer momento el diagrama general de casos de uso desde un punto de vista de usuario (figura 3.2).

Tras identificar los casos generales y tener una visión global del sistema, vamos a pararnos a detallar los diferentes casos de uso de la plataforma y su relación con el resto de elementos del sistema en la parte de **Manage Missions**. El objetivo principal de la definición de estos casos de uso es poder diseñar las comunicaciones entre los diferentes elementos implicados y guiar el diseño de la aplicación de usuario para la gestión de la producción.

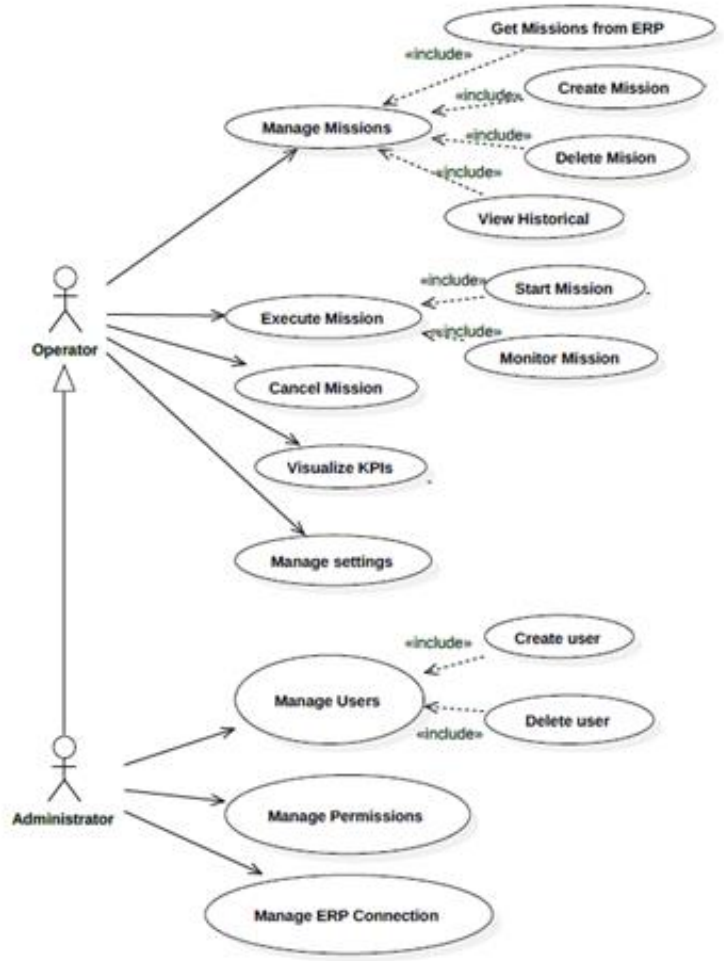



Figura 3.2 Diagrama general de casos de uso

Este caso de uso incluye todas las acciones relacionadas con la gestión de las misiones. De esta forma, agrupa los siguientes sub-casos de uso:

- Get Missions from ERP: obtención de órdenes de fabricación presentes en el ERP de la empresa susceptibles a ser transportadas por el sistema. En concreto, se obtendrán al menos para cada una de estas órdenes su estado, identificador dentro del ERP, el origen y el destino.
- Create Mission: permitirá crear misiones internas al sistema sin interactuar con el ERP. Estas misiones tienen como objeto realizar pruebas del sistema sin alterar el funcionamiento del ERP.

- Delete Mission: permitirá borrar misiones realizadas. Inicialmente solo será posible borrar aquellas misiones que no pertenezca al ERP.
- View Historical: visualización de todas las misiones realizadas por el sistema. Para cada una de las misiones se podrá ver el detalle de la misión, incluyendo las incidencias que se hayan producido durante la misma.

El siguiente paso en nuestra fase de diseño sería hacer los diagramas de secuencia [18] correspondientes a los casos de uso previamente comentados. Para no enturbiar el cuerpo de la memoria con tablas y esquemas, la siguiente información se podrá encontrar al final de dicha memoria, en el **Apéndice A**.

### 3.1.4 Definición de mensajes

Tal y como se ha descrito anteriormente en la arquitectura, las comunicaciones entre la plataforma IoT y el conjunto PMR (robot en adelante) se hará a través del protocolo de comunicaciones MQTT. MQTT es un protocolo de comunicaciones publish-subscribe que trabaja sobre TCP/IP que está especialmente diseñado para aplicaciones IoT.

A partir de los casos de usos descritos, se concluye que serán necesario intercambiar los siguientes tipos de mensajes entre la plataforma IoT y el robot:

- StartMission: mensaje que indica que debe realizarse una misión.
- UpdateStatus: indica el estado en el que se encuentra el robot. Este mensaje se utilizará de forma genérica para indicar el estado del robot durante una misión y su contenido debe permitir el cálculo de los KPIs.
- CancelMission: este mensaje indica al robot que debe cancelar una misión en curso.
- Alarm: mensaje destinado a notificar incidencias del robot durante una misión.
- GetStatus: solicita bajo demanda desde la plataforma IoT al robot a que actualice su estado actual.
- Request Action: mensaje desde el robot a la plataforma para requerir una acción por parte del operador.
- MisionStats: mensaje desde el robot a la plataforma con información necesaria para el cálculo de los KPIs.

Además de esto mensajes, se controlará desde la plataforma el mensaje *will*. Este tipo de mensaje es una característica de MQTT que permite controlar las desconexiones de los clientes. Para ver el sentido de la comunicación y el formato definido para cada mensaje, se añade en el **Apéndice B** dicha información por si el lector lo considera interesante.

### 3.1.5 Modelado de datos

Para cerrar con nuestro bloque de diseño, vamos a presentar el modelo de datos que hemos pensado que será necesario y suficiente para que la aplicación funcione.

En concreto para el ERP del caso de uso de la empresa, se expondrá la información necesaria para el proyecto a través de una serie de tablas de su base de datos. El siguiente diagrama muestra información sobre dichas tablas.

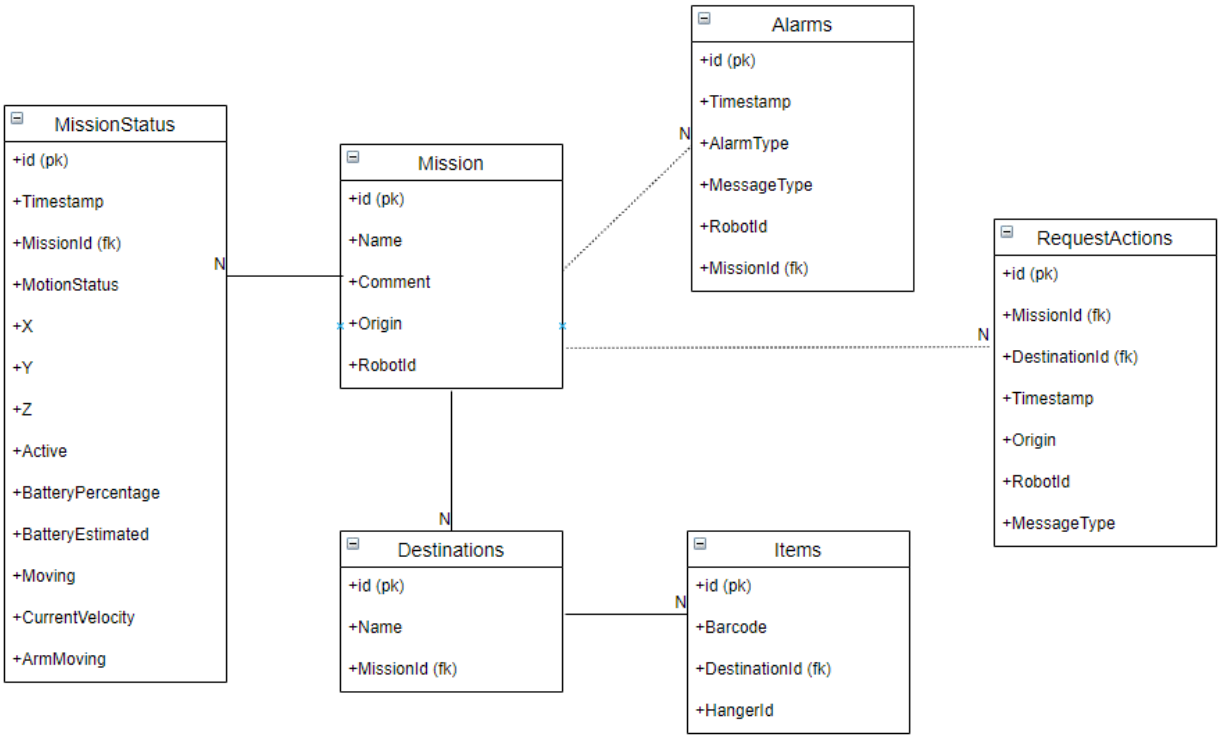
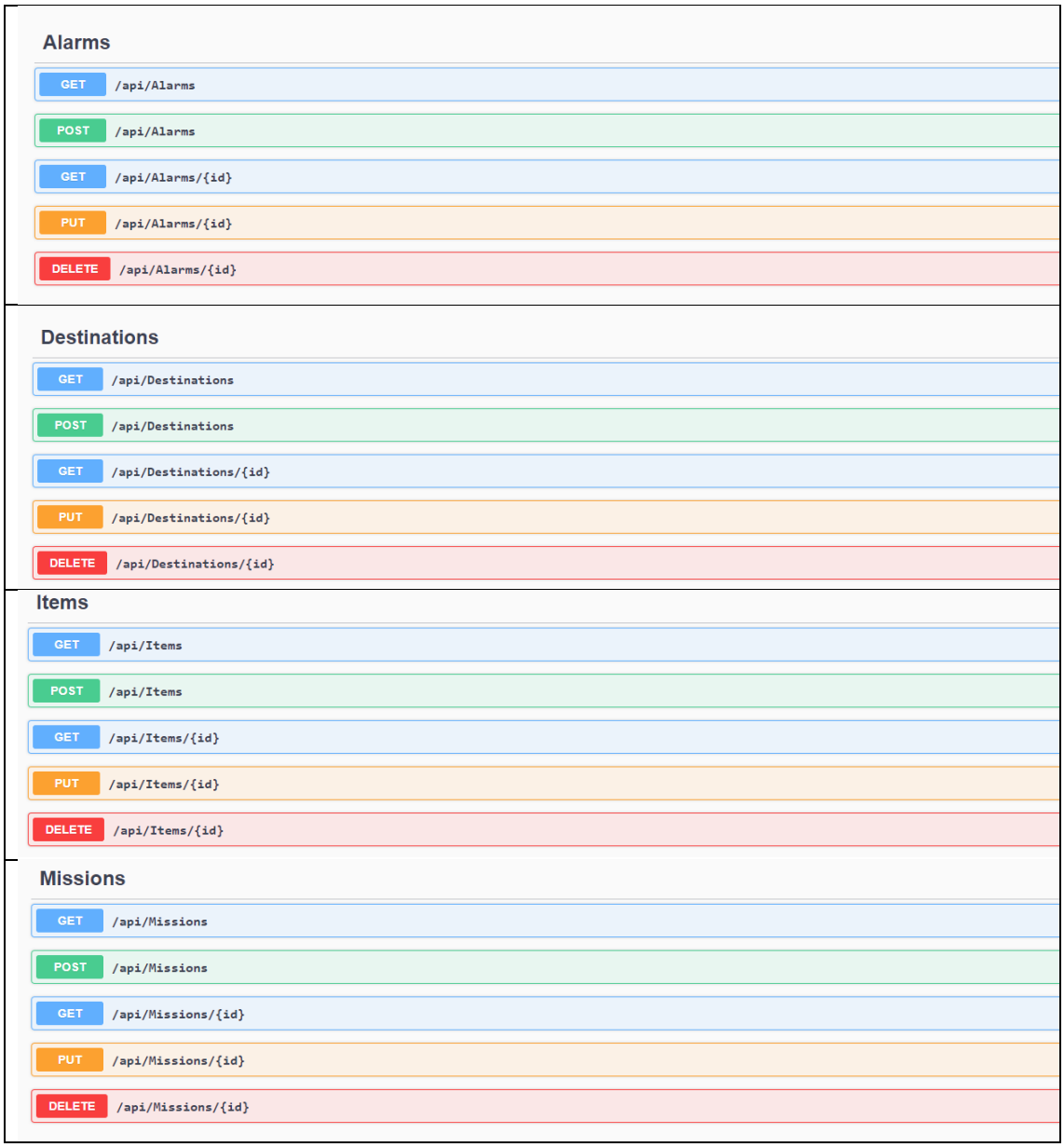



Figura 3.3 Tablas de datos para ERP y plataforma

Estas tablas serán compartidas entre la plataforma IoT y el sistema ERP, por lo que ambos sistemas podrán consultar y modificar el contenido para poder llevar a cabo las acciones necesarias por cada uno de los sistemas, siendo responsabilidad de la plataforma IoT actualizar el estado de las misiones y registrar las incidencias de las mismas, mientras que el ERP debe registrar las misiones en la tabla correspondiente para que puedan ser atendidas. La API REST de integración, se ha definido de tal forma que proporcione los mecanismos necesarios para poder trabajar sobre dichas tablas.

A continuación, se muestra la definición de las operaciones definidas sobre cada una de las entidades obtenidas a través de la herramienta Swagger.



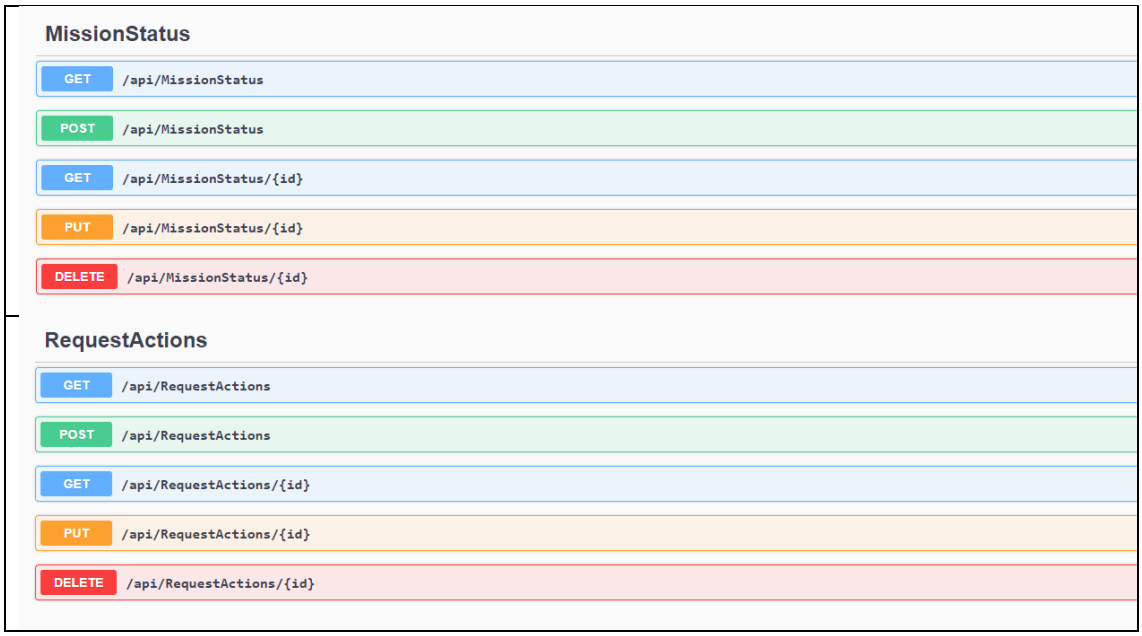



Tabla 3.1 Operaciones sobre entidades de la API Rest

Para acabar, se muestra en la siguiente imagen el modelo de datos para las operaciones definidas.

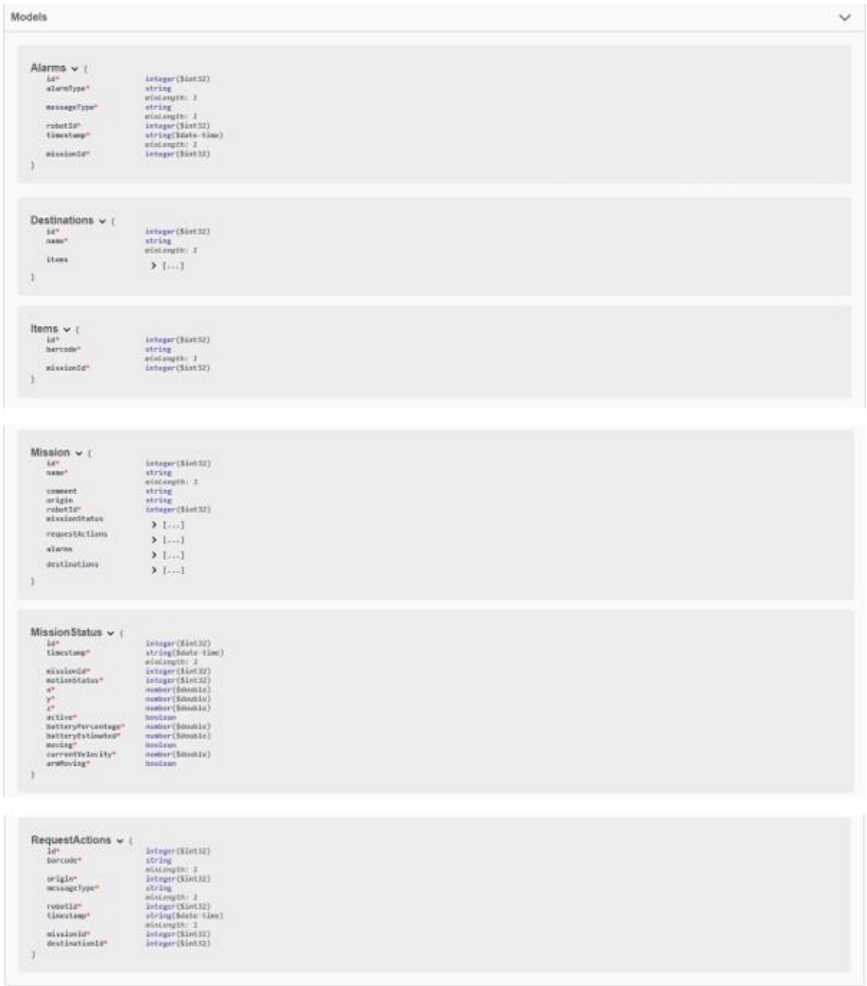



Figura 3.4 Modelo de datos

Los siguientes campos están restringidos a sus correspondientes valores:

- Alarms → **AlarmType** : este valor queda reservado para que la empresa cliente determine los tipos de incidencias diferentes que se pueden dar en su zona de trabajo. Depende de la gravedad de estos, se podrá programar en la aplicación web distintas funcionalidades para notificar al usuario con más hincapié en las alarmas que tengan que ser atendidas con más urgencia.
- MissionStatus → **status**: estado de una misión en concreto. Una misión puede tener cualquiera de los siguientes estados: PENDING, IN PROGRESS, BLOCKED, PAUSE, LOST y FINISHED. Inicialmente, cuando se guarden por primera vez los datos de una nueva misión en la BBDD del ERP, el estado de dicha misión será PENDING predeterminadamente.

## 3.2 Implementación

Dejamos atrás la parte de diseño para adentrarnos en la parte de implementación. La parte práctica de este proyecto se encargará de hacer realidad toda la planificación que hemos desarrollado en los anteriores capítulos. Para no empezar la casa por el tejado, vamos a centrarnos primero en dar de alta los recursos de Azure que vamos a utilizar, puesto que es lo primero que nos va a hacer falta para empezar a simular el paso de mensajes de los robots. Una vez que esto esté hecho, empezaremos con la implementación de los simuladores que enviarán los mensajes definidos (**Apéndice B**) como si fueran robots. Por último, ya estaremos en disposición de crear la consiguiente aplicación web que utilizará la información que reciba de los dos puntos anteriores.

### 3.2.1 Recursos en Azure

Para proyectos que no requieran muchos recursos (como es nuestro caso) con una cuenta gratuita de Azure podemos hacer uso de todas las herramientas que vamos a necesitar. Esto está muy bien puesto que una vez que el proyecto esté terminado y procede, a la hora de llevarlo a producción una empresa tan sólo tendrían que pagar por la cantidad exacta de recursos que les requiera su plataforma de producción. Pero para nuestro caso con los siguientes recursos no es más que suficiente.

- Suscripción de Azure: este recurso es obligatorio puesto que se enlazan todos los demás a él. Es el encargado de gestionar que todos los servicios que tenemos arrancados funcionen y en caso de que nos pasemos de lo contratado, cobrar un sobrecoste o darnos la opción de mejorar nuestro plan.
- Azure IoT Hub: lo más importante para nosotros. Tenemos que dar de alta el nodo IoT Hub de Azure [19] que trabajará como emisor y receptor intermedio.
- Cuenta de almacenamiento: necesitaremos dar de alta un servicio de almacenamiento para guardar las mensajes provenientes del robot para después ser utilizados por nuestra aplicación. Las cuentas de almacenamiento sí nos cobran por cantidad de datos pero para pocos volúmenes es gratis. Los datos irán en Azure Tables [20], ya que Azure dispone de varios sistemas diferentes de almacenamiento (como Cosmo Db, Blob Storage… [21]).
- Azure Function: por último y de forma también gratuita, crearemos la función que será la encargada de trocear los mensajes en bruto de ambas plataformas y almacenarlas correctamente en las Azure Tables con sus columnas y valores correspondientes.

Tras dar de alta los servicios el portal de Azure nos quedará de la siguiente manera:


Figura 3.5 Portal de Azure con los servicios en alta

### 3.2.2 Simulador PMR

Para simular el paso de mensajes entre nuestra aplicación gestora y los robots, hemos diseñado una librería de la que harán uso los desarrolladores encargados del robot y que sirva como puente intermedio entre nuestra plataforma y el entorno de trabajo que utilicen los robots. De esta forma, nos aseguraremos de que la comunicación sea bidireccional. Para ello, utilizamos el nodo IoT Hub de Azure creado en el paso anterior y que funcionará como una oficina de correos. A él le llegarán todos los mensajes provenientes de los robots y se encargará de enviarlos de vuelta a sus destinatarios (en nuestro caso, la aplicación .Net Core que será el back-end). Recapitulando, tenemos por un lado una librería en Python 3.7 con los métodos definidos para el envío de cada uno de los mensajes definidos que se conecta a un nodo de Azure. Por otro lado, tenemos una pequeña aplicación de consola para simular nuestra aplicación y poder simular el paso de mensajes.

A continuación veamos los métodos más importantes de la librería así como los resultados de la ejecución de dichos programas para dejar verificada que esta parte funciona.

- def RequestAction(method_request): envía un mensajes tipo RequestAction a la nube para ser recibido por nuestra aplicación de consola.
- def iothub_client_init(CONNECTION_STRING): da de alta el cliente de conexión iot hub necesario para enviarle los mensajes.
- def device_method_listener(device_client,source): una función que se queda escuchando el puerto de azure e imprime un mensaje si se le envía algo desde la aplicación de consola.
- def send_alarm(id,alarm_type,client): envía un mensajes tipo Alarm a la nube para ser recibido por nuestra aplicación de consola.
- def update_status(l_x,l_y,l_z,active_mission,motion_status,mission_id,percentage, estimated,is_moving,arm_moving,current_vel,client): envía un mensajes tipo UpdateStatus a la nube para ser recibido por nuestra aplicación de consola.
- def request_action(missionId,origin,destination,robotId,client):

- def mission_stats(missionId,pose_cov,rep,max_vel,robotId,client): envía las estadísticas que el robot ha recabado durante la misión y las envía únicamente al finalizar.

Como podemos observar, no es necesario hacer funciones para enviar todo tipo de mensajes, puesto que los robots, sólo pueden enviar a la nube unos tipos de mensajes, no todos. (Para más detalles consultar el sentido de los mensajes en Apéndice 3)

Si ejecutamos la aplicación vemos como se verifica el paso de mensajes en los dos sentidos utilizando la nube y dejamos preparado el código necesario para comunicarnos con el robot más delante desde la aplicación web. Si dejamos el programa funcionando vemos como Azure nos notifica la llegada de mensajes nuevos del robot (figura 3.7)

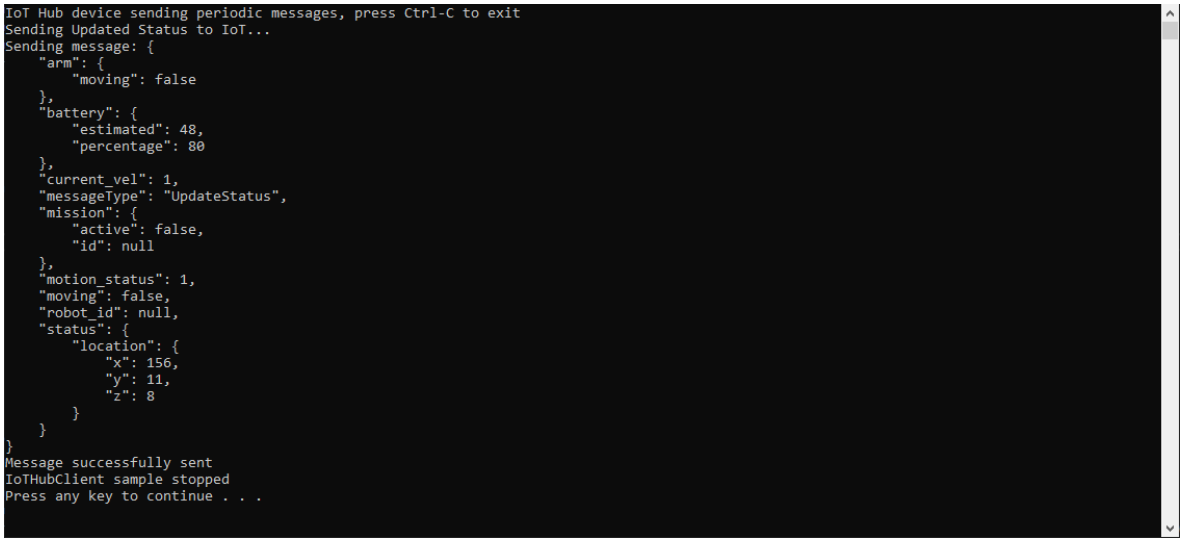


Figura 3.6 Envío de mensaje UpdateStatus del robot


Figura 3.7 Envío de mensaje MissionStats

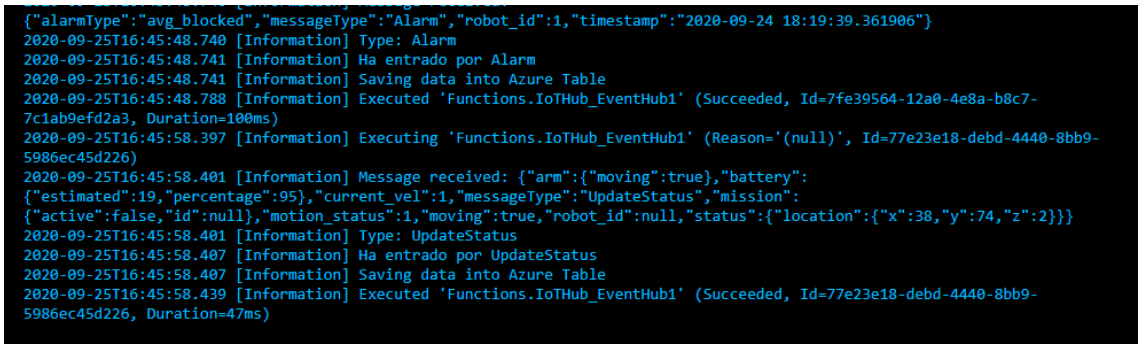



Figura 3.8 Mensajes de recepción de Azure

### 3.2.3 Conectividad plataforma IoT-ERP

Antes hemos visto que el sistema tiene que ser capaz de trabajar con datos que les vendrá del exterior (provenientes de un ERP) y que para facilitar la conexión utilizaremos una API que se encargará de traer los datos de fuera y almacenarlos en nuestra base de datos local para que la aplicación pueda hacer uso de ella. Nos encontramos con el primer problema en la ejecución de este trabajo. Los sistemas ERPs están normalmente integrados en redes privadas de las empresas que los tienen desplegados. Por ello, es muy difícil que una empresa nos dé un puerto propio al que hacer las peticiones de datos que se conecte fuera de la red privada de esta. Ante este problema que nos surge hemos desarrollado la siguiente solución que puede ser integrada en cualquier campo de trabajo empresarial.

El acceso al ERP desde la plataforma IoT se realizará a partir de una API REST que servirá de proxy con el ERP, ya que la información del ERP no está accesible a través de Internet. Para poder hacer visible la información necesaria a través de Internet, se desplegará esta API REST en un dispositivo que tenga conexión con el ERP de la empresa y ,a su vez, pueda publicar dicha API REST a través de Internet como un servicio.

Esta aproximación, utilizando una API REST que aísla las peculiaridades de cada ERP de la plataforma, permite definir un contrato genérico entre los ERP y la plataforma IoT, haciendo posible dar servicios a diferentes tipos de ERP sin la necesidad de modificar la plataforma, siendo necesario implementar en la API el servicio necesario para adaptar la información de cada ERP a la definida en el contrato de la plataforma.

Desde el punto de vista de conectividad, se plantean dos escenarios diferentes posibles:

1. Despliegue de un router IoT y un PC embebido que realiza el proxy entre la red del ERP e Internet.


Figura 3.9 Router con Raspberry con API Rest

2. Utilización de un proxy inverso. Ante la imposibilidad de instalar un dispositivo en la red del ERP, es posible utilizar un proxy inverso a través de un servidor de la red del ERP para publicar la API REST de forma segura a través de Internet.

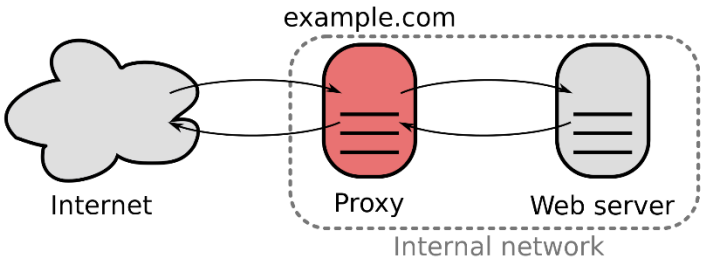


Figura 3.10 Proxy Inverso

### 3.2.4 API para conectividad con ERP

Para desarrollar la API que nos servirá para tomar los datos de los ERPs de las empresas, hacemos uso de las tecnologías de Entity Framework para generar los controladores necesarios para hacer las peticiones y además, generar directamente la base de datos. Estos dos pasos que parecen bastantes tediosos son realmente muy rápidos ya que, sirviéndonos de dicha tecnología sólo con programar los modelos de las entidades, lo demás se hace de manera automática. Una vez que tengamos la API arrancada y funcionando publicamos la aplicación para ser utilizado más adelante cuando proceda y la empresa nos dé las cadenas de conexión necesarias para hacer las peticiones. De momento, como estamos trabajando con funcionalidades esta herramienta la hemos dejado implementada pero no conectada al no disponer de una empresa disponible para probar la conexión en una red privada que no sea la propia. El modelo de datos nos queda muy escueto al primar la minimización de recursos necesarios como se puede ver en la figura 3.11 y la base de datos se generará acorde al mismo.

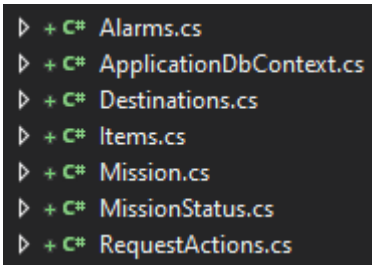


Figura 3.11 Modelo de datos API

### 3.2.5 Mockups Aplicación Web

Una vez que hemos desarrollado por separado las distintas partes que vamos a utilizar, es hora de empezar a plantearnos qué elementos tiene que tener la página web para cubrir todas las necesidades y requisitos que se han detallado en la parte de diseño. Como nuestros ideales es mantener una aplicación que de cara al usuario sea muy intuitiva y fácil de utilizar, vamos a optar por un diseño minimalista y que sea adaptable a los distintos tamaños de pantallas. Además, la idea de utilizar las tecnologías de angular y node.js es que si una empresa decide finalmente integrar esta solución sea muy fácil y rápido de modificar y agrandar según sus necesidades al trabajar en todo momento con componentes de Angular. Los prototipos o mockups de la aplicación web se presentan en el Apéndice C de este documento.

### 3.2.6 Desarrollo de Back-End

Para nuestra aplicación gestora de la información, vamos a seguir la misma metodología que utilizada en la API para la conectividad con el ERP. Vamos a implementar el modelo de datos necesario para que nuestra plataforma local pueda guardar en una nueva base de datos (la anterior iría generada en el ERP de la empresa y no tenemos control sobre ella) con nuevas tablas que tendrán que manejar los distintos tipos de mensajes que los robots emitan, así como los mensajes que viajan desde la aplicación web descrita en el paso anterior hasta los robots. Una vez que tengamos la nueva API a la que se conectará Angular (nos falta todavía un paso más para que esta conexión sea definitiva), ya podremos dar de alta las nuevas tablas. Dejamos en la siguiente figura el nuevo modelado de datos y las tablas generadas en la base de datos local. Recordemos que este paso es casi automático si tenemos ya el modelo de datos a mano.

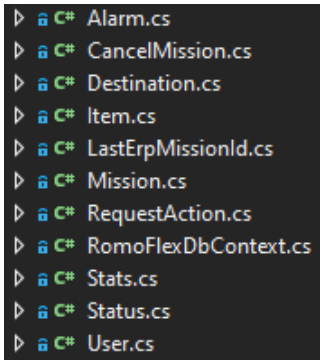



Figura 3.12 Modelo de datos API local

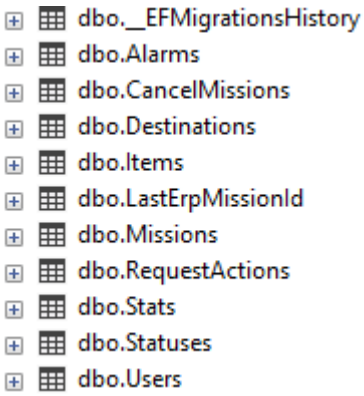



Figura 3.13 Tablas generadas en SQL Server

### 3.2.7 Desarrollo Web

Una vez que tengamos el back-end operativo, vamos a empezar a implementar las funcionalidades web con Angular que han quedado reflejadas en los prototipos del **Apéndice C**. Esta parte no requiere de mucho misterio ya que, consiste en maquetar con componentes de angular cada parte de los mockups diseñados y que de momento no podremos conectar las funcionalidades a falta del paso que veremos después de este. Por ello, presentamos varias pantallas ya maquetadas en la versión web para que el usuario pueda hacer una comparativa de estas con los prototipos iniciales de los que nos hemos basado.

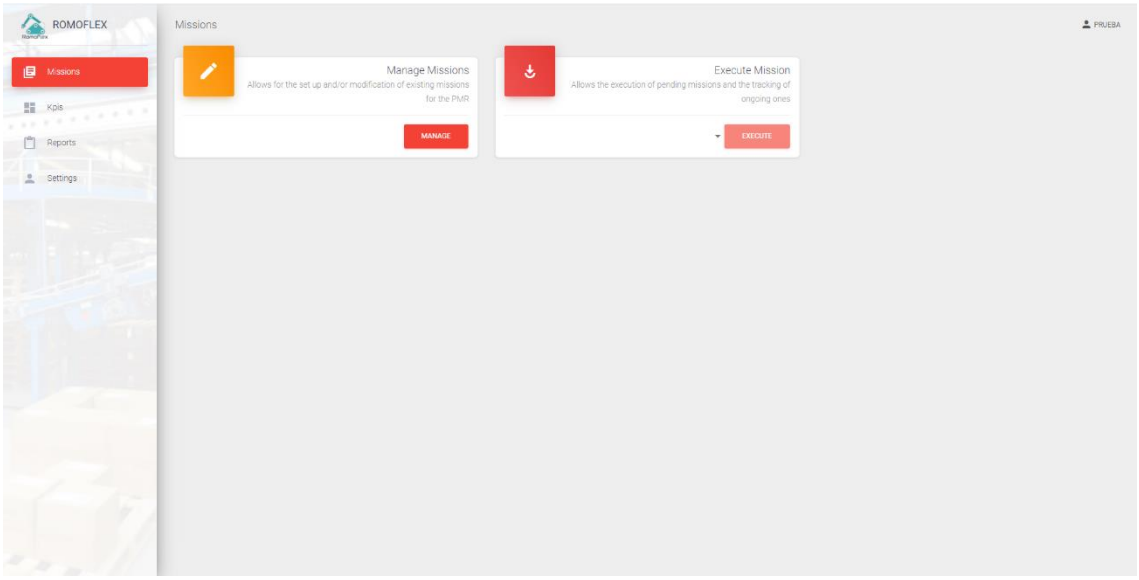



Figura 3.14 Página de gestión de las misiones

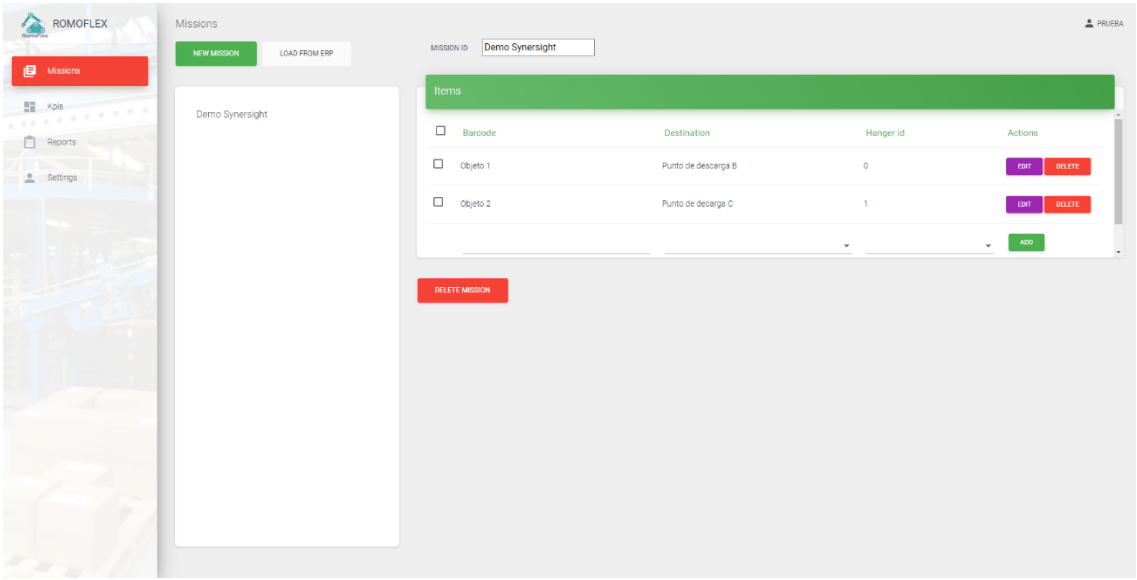



Figura 3.15 Página de creación-edición de misiones e importación

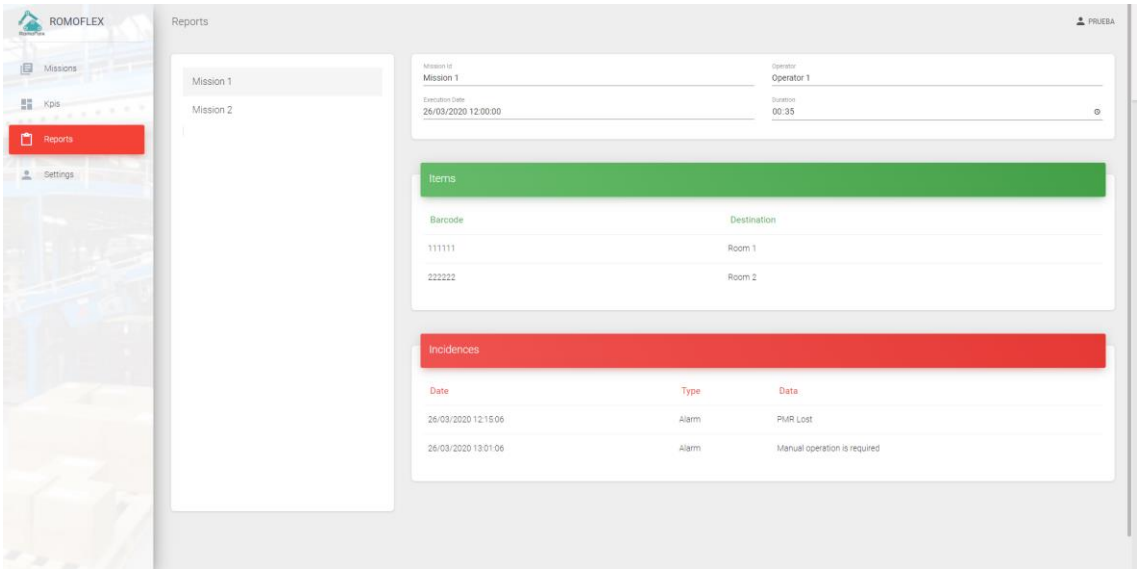



Figura 3.16 Página de reportes

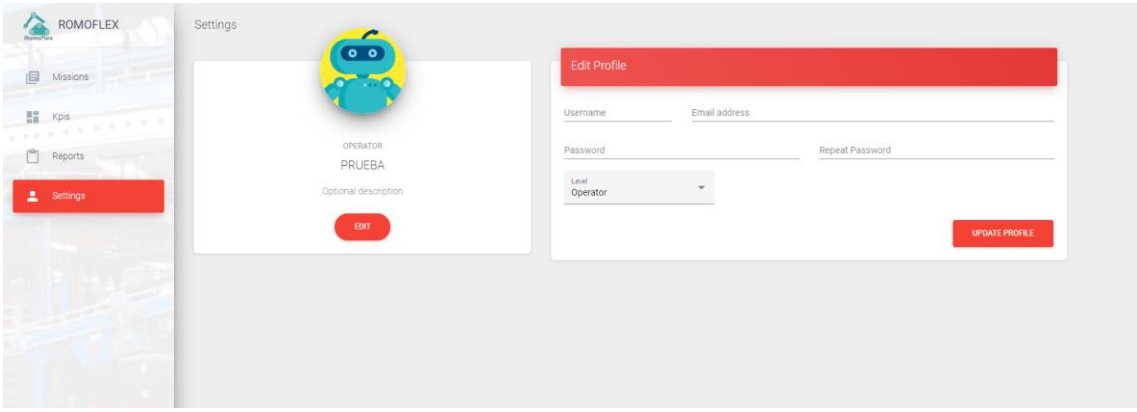



Figura 3.17 Página de ajustes de usuario

### 3.2.8 Conexión Back-Front End

Llegamos a la parte clave de todo este proyecto. La conexión entre la aplicación web y la aplicación gestora de la información es vital para que todo funcione correctamente. Antes hemos especificado que nos faltaba un paso para que las funcionalidades webs funcionaran y es que, necesitamos una forma de enlazar la API local que gestiona la aplicación del nodo de Azure con la web en cuestión, para que el usuario final pueda ver en tiempo real la información de las misiones así como, gestionar cualquiera de las que todavía no han sido lanzadas a la plataforma. Para hacer este paso nos hemos valido de la herramienta de OpenApi Swagger que utiliza la especificación de la API que hemos desarrollado (especificación que se genera automáticamente al lanzar la aplicación desde .Net Core teniendo instalado su respectivo paquete). Swagger ejecuta un programa que a cada entrada de la API definida le asocia un servicio en el lenguaje de programación que queramos para ser utilizados en el front-end. Por ello, elegimos el lenguaje de programación Typescript que es el que utiliza Angular y tras ejecutar la herramienta obtenemos los siguientes archivos (figura 3.18) listos para poder ser usados desde la aplicación web.

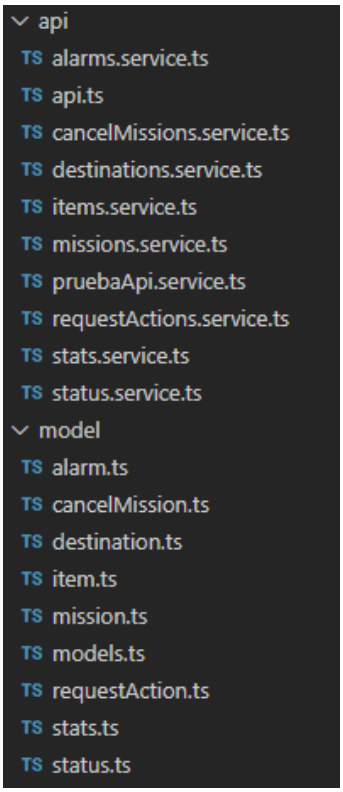



Figura 3.18 Servicios generados por Swagger

Con estos archivos, ya estamos en disposición de utilizar los métodos definidos en ellos para enlazar la parte visual de la aplicación y darle funcionalidades lógicas de verdad.

### 3.2.9 Prueba de ciclo completo

Por último, para acabar la parte de implementación y con ello, dejar comprobado que hasta este punto todo lo que llevamos hecho funciona y está conectado correctamente, hemos hecho una prueba de ciclo completo que consiste en lo siguiente:

1. El usuario crea una misión en la aplicación web.
2. La aplicación web manda la información al robot utilizando el nodo de Azure.
3. El robot recibe la misión y envía mensajes de actualización como UpdateStatus o Alarm.
4. Estos mensajes los recibe el IoT Hub de Azure y gracias a la función de Azure que tenemos implementada, se almacena la información en sus respectivas tablas de Azure.


Figura 3.19 Envío de mensaje StarMission desde la aplicación web

| ARM_MOVING | BATTERY_ESTIMATED | BATTERY_PERCENTAGE | CURRENT_VEL | MISSION_ACTIVE | MISSION_ID | MOTION_STATUS | MOVING | ROBOT_ID | STATUS_LOCATION_X | STATUS_LOCATION_Y | STATUS_LOCATION_Z |
|---|---|---|---|---|---|---|---|---|---|---|---|
| true | 16 | 73 | 1 | false | | 1 | true | | 18 | 27 | 6 |
| true | 34 | 99 | 1 | true | | 1 | false | | 118 | 67 | 9 |
| false | 31 | 83 | 1 | false | | 1 | false | | 168 | 10 | 5 |
| true | 16 | 73 | 1 | false | | 1 | true | | 18 | 27 | 6 |
| true | 16 | 73 | 1 | false | | 1 | true | | 18 | 27 | 6 |
| false | 1 | 16 | 1 | true | | 1 | false | | 64 | 60 | 3 |
| true | 47 | 34 | 1 | true | | 1 | false | | 8 | 86 | 2 |
| false | 25 | 43 | 1 | false | | 1 | false | | 145 | 13 | 8 |
| true | 35 | 100 | 1 | false | | 1 | false | | 107 | 42 | 9 |
| true | 39 | 2 | 1 | true | | 1 | false | | 140 | 74 | 1 |
| false | 10 | 61 | 1 | false | | 1 | false | | 86 | 20 | 9 |
| false | 47 | 63 | 1 | true | | 1 | false | | 99 | 53 | 6 |
| true | 5 | 65 | 1 | false | | 1 | false | | 74 | 65 | 6 |
| false | 38 | 58 | 1 | true | | 1 | true | | 33 | 95 | 1 |
| false | 16 | 60 | 1 | false | | 1 | true | | 78 | 42 | 0 |
| false | 10 | 26 | 1 | true | | 1 | true | | 171 | 25 | 0 |
| false | 22 | 6 | 1 | false | | 1 | true | | 144 | 48 | 4 |
| false | 6 | 93 | 1 | false | | 1 | false | | 196 | 81 | 9 |
| true | 45 | 52 | 1 | false | | 1 | true | | 145 | 98 | 8 |
| false | 37 | 89 | 1 | false | | 1 | true | | 73 | 80 | 3 |
| false | 30 | 18 | 1 | true | | 1 | true | | 28 | 75 | 1 |
| false | 6 | 61 | 1 | false | | 1 | true | | 97 | 22 | 4 |
| false | 24 | 31 | 1 | false | | 1 | true | | 134 | 13 | 9 |
| false | 50 | 66 | 1 | false | | 1 | true | | 166 | 53 | 7 |
| true | 17 | 46 | 1 | true | | 1 | false | | 108 | 94 | 0 |
| false | 20 | 2 | 1 | true | | 1 | false | | 21 | 99 | 9 |
| true | 49 | 81 | 1 | true | | 1 | false | | 0 | 21 | 9 |
| true | 16 | 73 | 1 | false | | 1 | true | | 18 | 27 | 6 |
| true | 25 | 0 | 1 | true | | 1 | false | | 141 | 83 | 5 |
| false | 48 | 80 | 1 | false | | 1 | false | | 156 | 11 | 8 |
| false | 47 | 31 | 1 | true | | 1 | false | | 99 | 73 | 8 |
| false | 14 | 94 | 1 | false | | 1 | false | | 110 | 42 | 0 |
| true | 19 | 95 | 1 | false | | 1 | true | | 38 | 74 | 2 |

Figura 3.20 Datos guardados en Azure Tables de los mensajes UpdateStatus que manda el robot

Una vez que tenemos el ciclo de mensajes cubierto, ya estamos en disposición de agregar las últimas funcionalidades a la aplicación, estas son, los indicadores claves de rendimiento que vamos a medir y mostrar en la web para que los usuarios hagan uso de esta información.

## 3.3 Indicadores claves de rendimiento (KPIs)

No debemos olvidar que una parte fundamental de este proyecto también es la relativa al tratado de datos. El usuario final, al fin y al cabo, necesitará las funcionalidades básicas para pedir materiales a los robots pero, para los ejecutivos de la empresa y encargados de la supervisión de los almacenes les será de gran ayuda tener unos datos elementales y personalizados de lo que está pasando con los robots en las plantas bajas de producción. También les será de utilidad para mejorar aquellos aspectos que les hagan cuello de botella o que puedan cambiar para incrementar su volumen de facturación anual. Los indicadores claves de rendimiento [22] son una serie de métricas que se utilizan para sintetizar la información sobre la eficacia y productividad de las acciones que se lleven a cabo en un negocio con el fin de poder tomar decisiones y determinar aquellas que han sido más efectivas a la hora de cumplir con los objetivos marcados en

un proceso o proyecto concreto. Ya que este proyecto se está desarrollando desde 0, hay que identificar cuáles serán esos indicadores de especial interés a tener en cuenta y definirlos correctamente para llevarlos posteriormente a la práctica y mostrarlos a los usuarios finales. Estos indicadores serán calculados en el back-end y se enviarán directamente a la página web para su impresión. Esto tiene que ser así ya que es este el encargado de leer los mensajes tanto de las tablas de Azure en la nube como de la base de datos local que guarda toda la información relativa a los mensajes.
A continuación, se describen los KPIs propuestos separados por capacidades:

- Navegación
    - Precisión de posicionamiento: este valor se pasará por el mensaje MisionStats, utilizando el valor PoseWithCovariance del Robot.

    - Repetibilidad de seguimiento de trayectoria: este valor vendrá incluido también en el mensaje MisionStats.

    - Número de pérdidas: se contabilizará el número de mensajes UpdateStatus que tengan el motion_status con el valor LOST.

    - Tiempo perdido en reubicación: tras recibir un LOST por parte del robot, se contabilizará el tiempo transcurrido hasta que se envíe un UpdateStatus con el motion_status actualizado.

    - Velocidad máxima: Vendrá dada como atributo en el mensaje MisionStats.

- Manipulación de cajas
    - Variedad de bolsas: número diferentes de bolsas con los que trabaje la empresa.
    - Peso máximo de bolsa: peso máximo que cada tipo de bolsa podrá aguantar.

- Bolsas perdidas: el operador indicará de manera manual en el panel del robot el error o pérdida de algún componente y la plataforma guardará la incidencia.
- N.º de bolsas por hora: este dato se calculará en la plataforma IoT contabilizando el número de bolsas que se han utilizado en las misiones. Los ítems ya vienen dados en el StartMission.

- General
  - Retorno de la inversión [23]: esto depende de la empresa y de cómo haga uso de la herramienta.
  - Seguridad: definición de accidente por cada empresa para poder mostrar los datos en la web.
  - Satisfacción de los operarios: los operarios meterán este valor directamente.
  - Tiempo de ciclo: este dato se calcula directamente en la plataforma con el tiempo transcurrido desde que empieza hasta que acaba un proceso completo.
  - Tiempo inactividad por ciclos de carga: tiempo que el robot no está trabajando porque se encuentra en la estación de carga de la empresa al no tener batería.
  - Tiempo inactividad por avería: a introducir manualmente por los operarios.

Los indicadores propuestos son simples ideas siguiendo la filosofía del trabajo y pensando en cuáles son aquellos datos que más utilidad tendrán en la toma de decisiones de la empresa. El objetivo de esta parte es mostrar unos cuantos como ejemplo para ver el funcionamiento del prototipo de la aplicación puesto que los KPIs

tendrán que ser definidos directamente por la empresa cliente y estos serán implementados en la parte correspondiente para ello en el back-end y front-end. En la figura 3.21 podemos ver cómo quedarían algunos de los indicadores anteriormente definidos en la aplicación final teniendo en cuenta los datos de los robots que han ejecutado alguna misión.

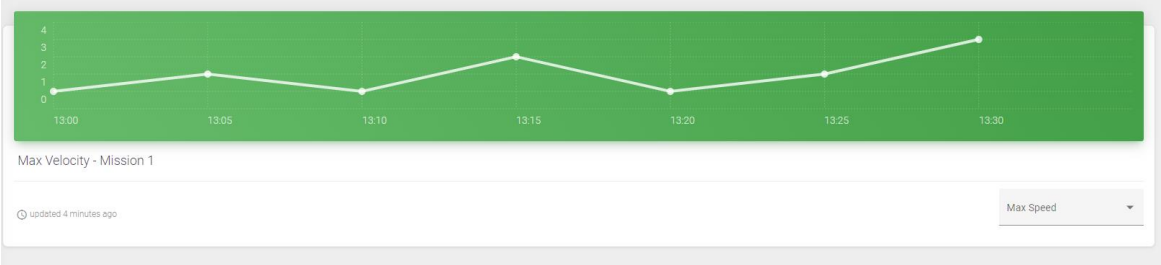



Figura 3.21 Indicador de Velocidad Máxima (m/s) de una misión

# 4

# Conclusiones y Líneas Futuras

En este proyecto se ha desarrollado una nueva solución que podría ser integrada con robots de bajo coste para empresas que quieran automatizar sus almacenes sin invertir mucho capital. Al utilizar la computación en la nube con Azure, toda la gestión de la información se abstrae directamente y por tanto, ganamos en los siguientes puntos:

1. Facilidad de integración y adaptación a los sistemas ERPs normalmente desfasados o a los que les es muy costoso integrar una nueva solución.
2. Libertad de uso de los operarios y administradores de la aplicación al no tener que estar en el mismo entorno físico que los robots.
3. Mínimo coste para expandir el modelo o añadir funcionalidades.
4. No requerir de una infraestructura informática propia y encargarnos de su mantenimiento, disminuyendo hasta casi nulo la capacidad de fallo hardware del sistema al estar en la nube.

Hemos visto cómo haciendo uso de tecnologías que están a mano de cualquier persona, y lo más importante, no requiriendo de muchos recursos, podemos gestionar todas las funcionalidades necesarias para lanzar a un robot en su búsqueda autónoma de materiales y piezas de construcción. Resulta muy interesante para las pequeñas y medianas empresas que quieran automatizar sus almacenes ya que, la mayor inversión

se requerirá en la obtención de los robots per se. Pero todo el sistema informático gestor se verá reducido bruscamente al disponer de las herramientas de Azure. Para esta aplicación de prueba hemos invertido un total de 2€ para guardar los datos en tablas de Azure. Contando que una empresa se haga con una decena de robots el gasto de la plataforma tras el primer mes será de unos pocas decenas de euros, lo que puede significar un incremento de su facturación al haber ahorrado tiempo y esfuerzo a los operarios que no tienen que pelearse con las cajas y materiales.

También cabe destacar la flexibilidad con la que ha sido pensada este proyecto. Toda la parte de requisitos ha sido diseñada por componentes únicos, por lo que para añadir funcionalidades y personalizar la aplicación es muy fácil de hacer para un tercero que no esté familiarizado con el código. Cualquier empresa interesada puede escalar el proyecto acorde a sus necesidades sin mucho esfuerzo.

Aunque aquí no se ha trabajado, sería muy interesante comentar como posible línea de trabajo futura el manejo de datos multimedia en tiempo real, para que los operarios puedan seguir en todo momento la situación del robot y mostrarla en un mapa interactivo que la empresa tendría que disponer. Pudiendo presentarse perfectamente esta información en la pestaña de ejecución de misiones.

Además, como característica complementaria relativa a las misiones, se podría instalar fácilmente un sistema de alertas y notificaciones en la pantalla principal que anuncie al operario de una manera más drástica que ha ocurrido algún error durante una misión. Para ser más incisivos con estos mensajes, ya que son bastante prioritarios, se podrían generar distintas notificaciones que vayan a los emails y los teléfonos de los operarios para darle más visibilidad a las alarmas puesto que, una vez que lancen misiones desde la aplicación web, nadie les obliga a estar delante de la pantalla viendo si llega un mensaje de alarma o que el robot requiera de una acción manual por parte de este.

# Referencias


[1] Rubin, K. S. (2012). Essential Scrum: A practical guide to the most popular Agile process. Addison-Wesley, pags. 29-58.

[2] Rubin, K. S. (2012). Essential Scrum: A practical guide to the most popular Agile process. Addison-Wesley, pags. 1-10.

[3] Microsoft Docs, "Información general sobre .Net Core" https://docs.microsoft.com/es-es/dotnet/core/about . Accedido por útlima vez en junio, 2020.

[4] Visual Studio Community, https://visualstudio.microsoft.com/es/vs/community/. Accedido por última vez en junio, 2020.

[5] Visual Studio Code https://code.visualstudio.com/. Accedido por última vez en junio, 2020.

[6] Microsoft, "Introducing SQL Server 2019" https://www.microsoft.com/en-us/sql-server/sql-server-2019. Accedido por última vez en junio, 2020.

[7] Google, "Angular" https://angular.io/. Accedido por última vez en junio, 2020.

[8] OpenJS Fundation, "Node js" https://nodejs.org/es/. Accedido por última vez en junio, 2020.

[9] Microsoft, "TypeScript" https://www.typescriptlang.org/. Accedido por última vez en junio, 2020.

[10] Python, "Python" https://www.python.org/. Accedido por última vez en junio, 2020.

[11] Microsoft, "Azure" https://azure.microsoft.com/es-es/. Accedido por última vez en junio, 2020.

[12] Linux Fundation, "OpenAPI Initiative" https://www.openapis.org/. Accedido por última vez en agosto, 2020.

[13] L. Richardson y S. Ruby. RESTful Web Services, O'Reilly Media Inc, 2008

[14] GitHub, "RicoSuter/NSwag" https://github.com/RicoSuter/NSwag. Accedido por última vez en agosto, 2020.

[15] SmartBear, "Swagger" https://swagger.io/. Accedido por última vez en agosto, 2020.

[16] Postman Inc, "Postman" https://www.postman.com/. Accedido por última vez en agosto, 2020.

[16] Postman Inc, "Postman" https://www.postman.com/. Accedido por última vez en agosto, 2020.

[17] VisualParadigm, "What Is Use Case Diagram?" https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ Accedido por última vez en agosto, 2020.

[18] Geeks for Geeks, "Unified Modeling Language (UML) | Sequence Diagrams" https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/ Accedido por última vez en agosto, 2020.

[19] Microsoft Azure, "Azure IoT Hub" https://azure.microsoft.com/es-es/services/iot-hub/. Accedido por última vez en agosto, 2020.

[20] Microsoft Azure, "Table Storage" https://azure.microsoft.com/es-es/services/storage/tables/ Accedido por última vez en agosto, 2020.

[21] Fabían Calvo, "Almacenamiento en los tiempos de Azure" https://blogs.encamina.com/sextosharepoint/2016/06/02/almacenamiento-en-los-tiempos-de-azure-ii/ Accedido por última vez en agosto, 2020.

[22] Wikipedia, "Indicador clave de rendimiento" https://es.wikipedia.org/wiki/Indicador_clave_de_rendimiento. Accedido por última vez en agosto, 2020.

[23] RdStation, "¿Qué es el ROI? https://www.rdstation.com/es/blog/roi/ Accedido por última vez en septiembre, 2020.

# Apéndice A

# Diagramas de secuencia

En este apéndice se encuentran los diagramas de secuencia y sus tablas de definición referentes al apartado 3.1 Diseño. Se proporciona una descripción textual de cada uno de los casos de uso y un diagrama de secuencia.

- Get Missions from ERP

| **Nombre** | **Get Missions from ERP** |
|---|---|
| **Actores** | Operador, WebApp, ERP |
| **Disparo** | Demanda del operador a través de la aplicación de usuario |
| **Garantías Mínimas** | El usuario recibirá un mensaje indicando el resultado de la operación realizada |
| **Garantías de éxito** | La plataforma obtiene del ERP un listado de las misiones existentes y su estado actual |
| **Escenario principal** | |
| 1. El operador solicita a través de la aplicación web obtener las misiones pendientes del ERP<br>2. El sistema consulta a través de la API de integración las misiones pendientes<br>3. La API devuelve una lista con las misiones pendientes presentes en el ERP<br>4. El sistema almacena en la BD de datos las misiones pendientes y muestra un listado de las mismas al usuario | |

Tabla A.1 Definición Get Missions from ERP

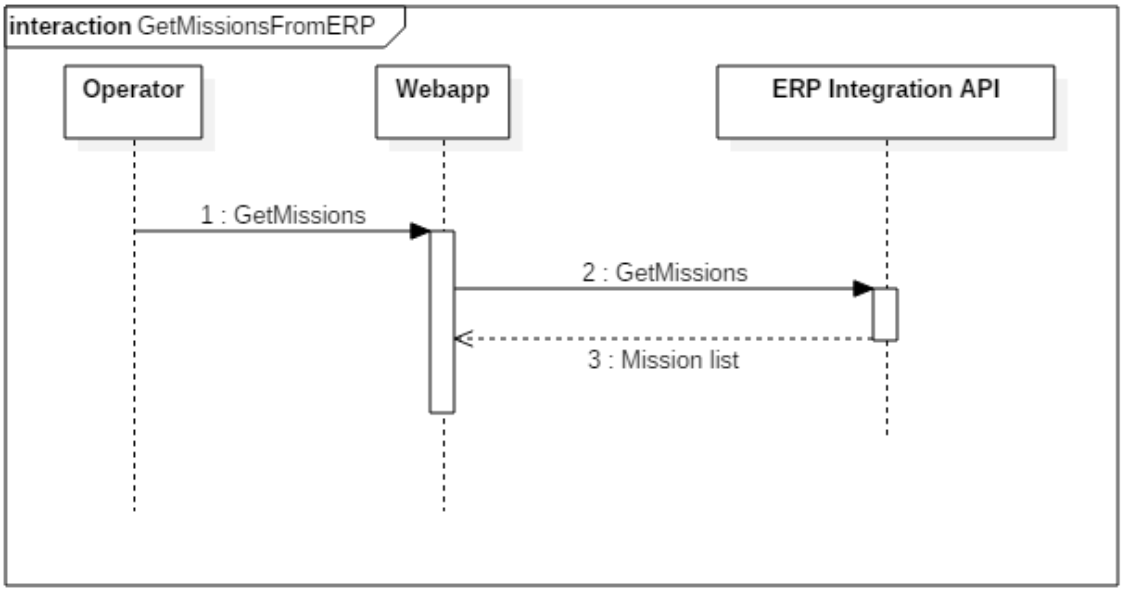



Figura A.1 Diagrama de secuencia GetMissionsFromERP

- Create Missions

| Nombre | **Create Missions** |
|---|---|
| **Actores** | Operador, WebApp |
| **Disparo** | Demanda del operador a través de la aplicación de usuario |
| **Garantías Mínimas** | El usuario recibirá un mensaje indicando el resultado de la operación realizada |
| **Garantías de éxito** | La plataforma genera una nueva misión que no altera el estado del ERP |
| **Escenario principal** | |
| 1. El operador solicita a través de la aplicación web crear una nueva misión<br>2. El sistema muestra un formulario al usuario para introduzca los datos de la nueva misión<br>3. La aplicación registra la nueva misión en la base de datos. | |

Tabla A.2 Definición Create Missions

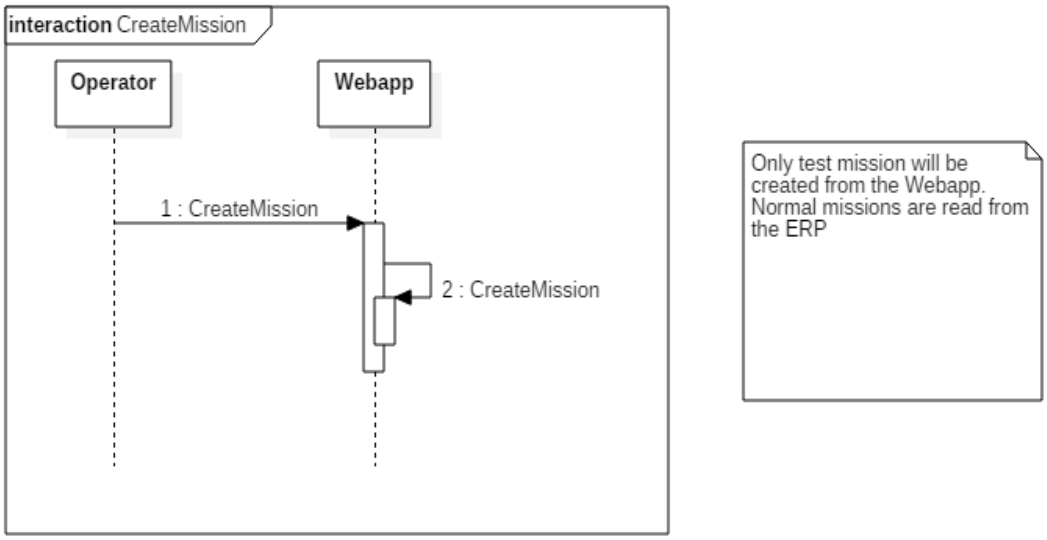



Figura A.2 Diagrama de secuencia CreateMission

- Delete Mission

| **Nombre** | **Delete Mission** |
|---|---|
| **Actores** | Operador, WebApp, ERP |
| **Disparo** | Demanda del operador a través de la aplicación de usuario |
| **Garantías Mínimas** | El usuario recibirá un mensaje indicando el resultado de la operación realizada |
| **Garantías de éxito** | La plataforma elimina misión |
| **Escenario principal** | |
| 1. El operador solicita a través de la aplicación eliminar una misión<br>2. El sistema solicita a través de la API de integración el borrador de la misión a eliminar<br>3. La API devuelve el resultado de realizar la acción en el ERP<br>4. El sistema elimina de la BD de la plataforma la misión y muestra un mensaje al usuario | |

Tabla A.3 Definición Delete Mission

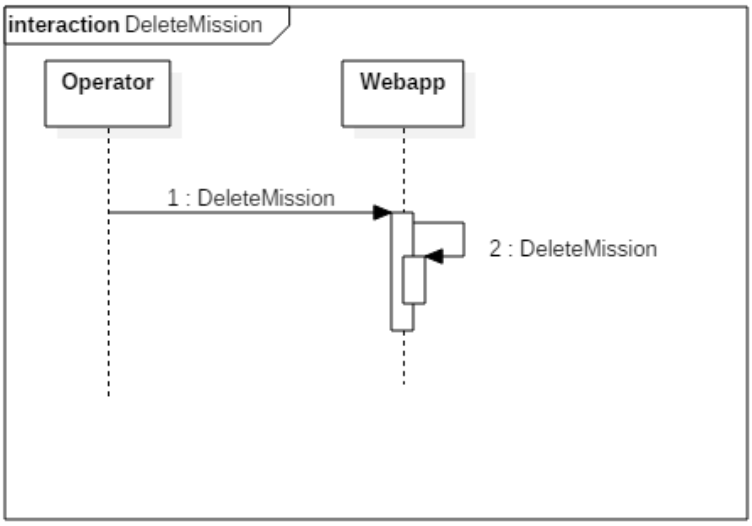



Figura A.3 Diagrama de secuencia DeleteMission

- View Historical

| **Nombre** | **View Historical** |
|---|---|
| **Actores** | Operador, WebApp |
| **Disparo** | Demanda del operador a través de la aplicación de usuario |
| **Garantías Mínimas** | El usuario recibirá el listado histórico de las misiones realizadas por el sistema |
| **Garantías de éxito** | La plataforma muestra las misiones ya terminadas |
| **Escenario principal** | |
| 1. El operador solicita a través de la aplicación ver el histórico de misiones<br>2. La aplicación muestra al usuario un listado de las misiones realizadas en función de los criterios establecidos por el usuario | |

Tabla A.4 Definición View Historical

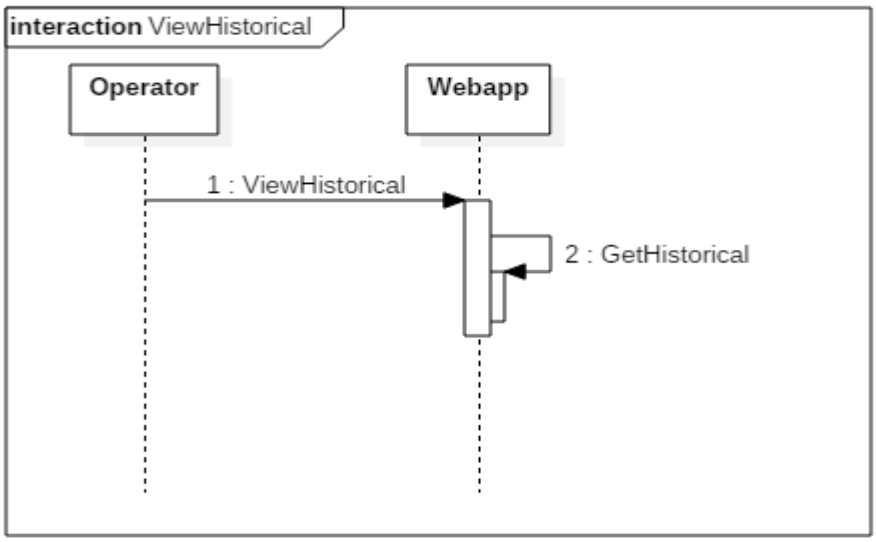



Figura A.4 Diagrama de secuencia ViewHistorical

- Start Mission

| **Nombre** | **Start Mission** |
|---|---|
| **Actores** | Operador, WebApp, ERP, Robot |
| **Disparo** | Demanda del operador a través de la aplicación de usuario |
| **Garantías Mínimas** | La misión comienza y se pasa al modo de monitorización de misión |
| **Garantías de éxito** | La aplicación pasa al modo de monitorización |
| **Escenario principal** | |
| 1. El operador solicita a través de la aplicación comenzar una misión<br>2. La aplicación web indica al Broket MQTT que envíe el mensaje de inicio de misión<br>3. El Broker MQTT indica que se ha enviado la solicitud de inicio de misión<br>4. Se actualiza el estado de la misión en el ERP y la base de datos<br>5. La aplicación web pasa al modo de monitorización de misión y mostrará la información de estado de misión de forma asíncrona, almacenando dicha información en la base de datos para poder consultarla posteriormente. | |

Tabla A.5 Definición Start Mission

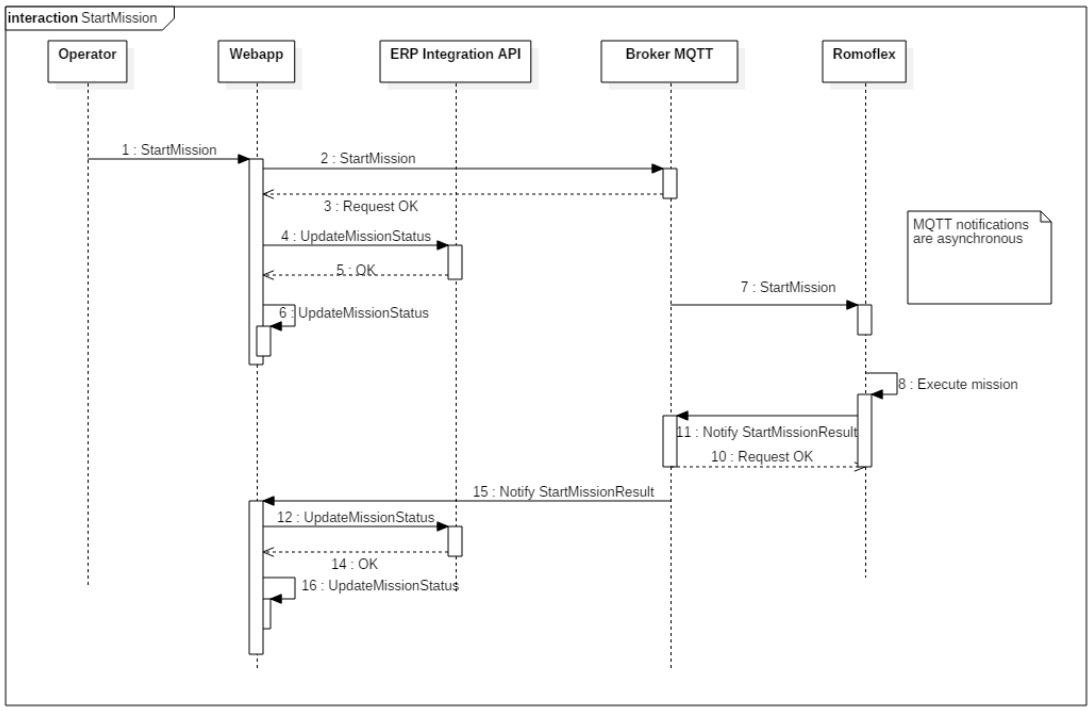



Figura A.5 Diagrama de secuencia StartMission

- Visualize KPIs

| Nombre | **Visualize KPIs** |
|---|---|
| **Actores** | Operador, WebApp |
| **Disparo** | Demanda del operador a través de la aplicación visualizar KPIs |
| **Garantías Mínimas** | El usuario recibirá listado de los KPI solicitados |
| **Garantías de éxito** | La plataforma realiza el cálculo de KPI y los muestra al usuario |
| **Escenario principal** | |
| 1. El operador solicita a través de la aplicación ver los KPI del sistema<br>2. La WebApp realiza el cálculo de los KPI solicitados<br>3. Se muestra al usuario un listado de KPI calculados en función de los criterios establecidos por el usuario | |

Tabla A.6 Definición Visualize KPIs

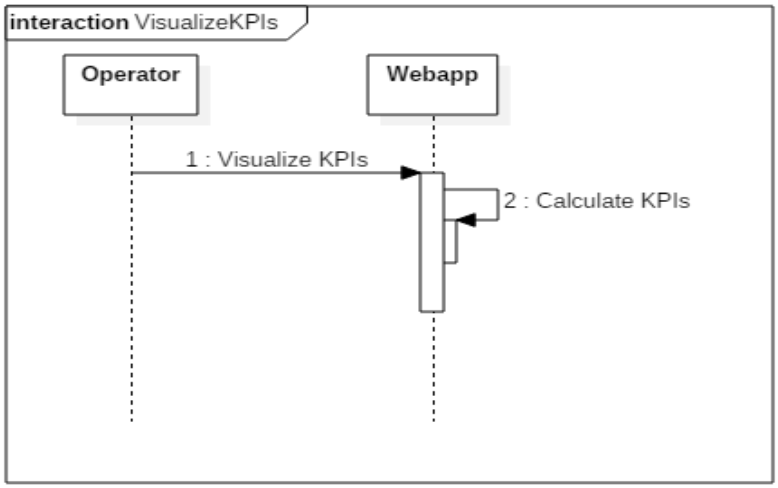



Figura A.6 Diagrama de secuencia VisualizeKPIs

- Monitor Mission

| **Nombre** | **Monitor Mission** |
|---|---|
| **Actores** | Operador, WebApp, Robot |
| **Disparo** | Demanda del operador a través de la aplicación para monitorizar una misión en proceso |
| **Garantías Mínimas** | El usuario recibirá la información de la misión |
| **Garantías de éxito** | La plataforma recoge los datos enviados por el robot y los muestra al usuario |
| **Escenario principal** | |
| 1. El operador solicita a través de la aplicación ver la monitorización de un robot en misión.<br>2. La WebApp recoge los datos necesarios<br>3. Se muestra al usuario un listado de la información que envía el robot de manera fácil e intuitiva | |

Tabla A.7 Definición Monitor Mission

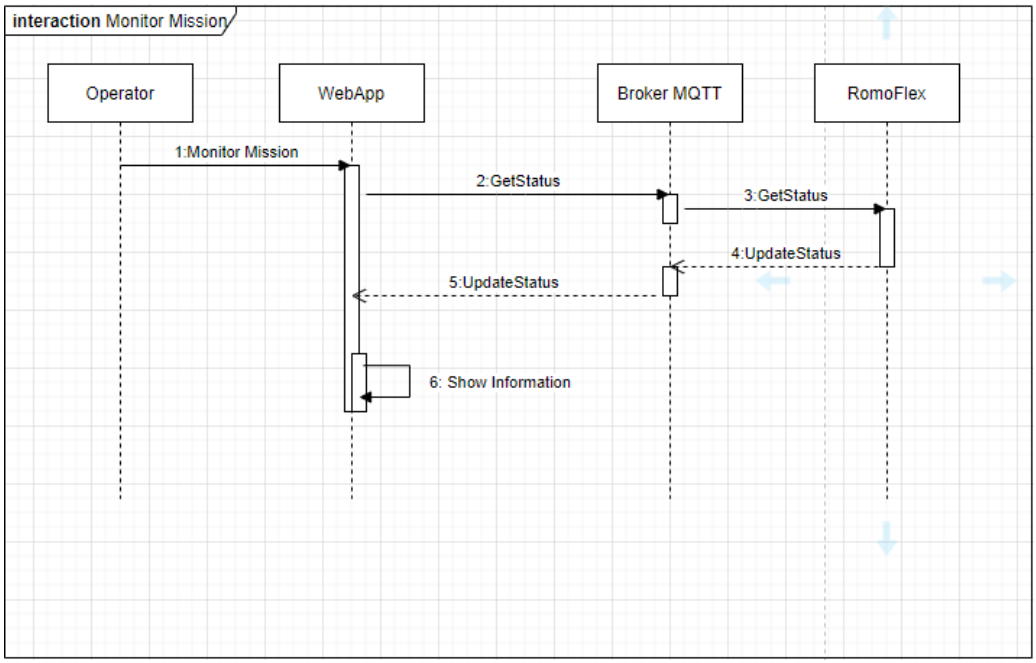



Figura A.7 Diagrama de Secuencia Monitor Mission

- Manage Settings

| **Nombre** | **Manage Settings** |
|---|---|
| **Actores** | Operador, WebApp |
| **Disparo** | Demanda del operador a través de la aplicación para gestionar las opciones de la misión |
| **Garantías Mínimas** | El usuario modificará los datos de la misión |
| **Garantías de éxito** | La plataforma modificará internamente las opciones de una misión en marcha |
| **Escenario principal** | |
| 1. El operador solicita a través de la aplicación gestionar las opciones de las misiones.<br>2. La WebApp muestra las misiones activas con sus propiedades.<br>3. El usuario modifica aquellos campos de datos que requieran ser cambiados. | |

Tabla A.8 Definición Manage Settings

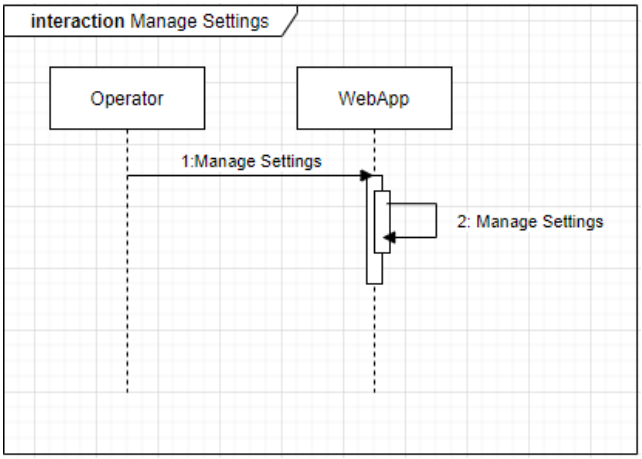


Figura A.8 Diagrama de secuencia Manage Settings

- Cancel Mission

| **Nombre** | **Cancel Mission** |
|---|---|
| **Actores** | Operador, WebApp, Robot |
| **Disparo** | Demanda del operador a través de la aplicación para cancelar una misión en progreso |
| **Garantías Mínimas** | El usuario cancelará una misión |
| **Garantías de éxito** | La plataforma emitirá el mensaje de cancelación a la plataforma IoT y esta al robot para parar su ejecución |
| **Escenario principal** | |
| 1. El operador solicita a través de la aplicación cancelar una misión.<br>2. La WebApp envía al robot el mensaje de cancelación.<br>3. El robot recibe la orden y para su tarea a espera de nuevas órdenes.<br>4. El usuario recibe la confirmación de la cancelación por una actualización de estado. | |

Tabla A.9 Definición Cancel Mission

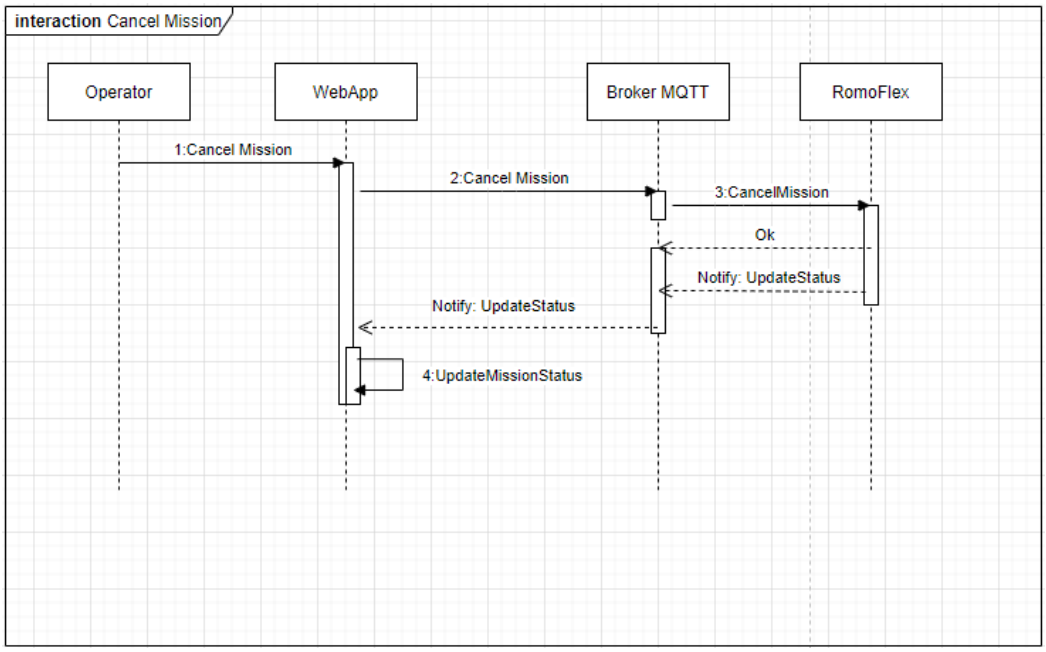



Figura A.9 Diagrama de secuencia Cancel Mission

- Create User

| **Nombre** | **Create User** |
|---|---|
| **Actores** | Administrador, WebApp |
| **Disparo** | Demanda del administrador a través de la aplicación para crear un nuevo usuario |
| **Garantías Mínimas** | El administrador creará un nuevo usuario |
| **Garantías de éxito** | La plataforma registrará y dará los permisos necesarios a un nuevo usuario |
| **Escenario principal** | |
| 1. El administrador solicita a través de la aplicación crear un nuevo usuario.<br>2. La WebApp muestra al administrador un formulario para introducir los datos del nuevo usuario.<br>3. La plataforma gestiona los permisos que se le darán al nuevo usuario.<br>4. El administrador recibe la confirmación de la nueva alta. | |

Tabla A.10 Definición Create User

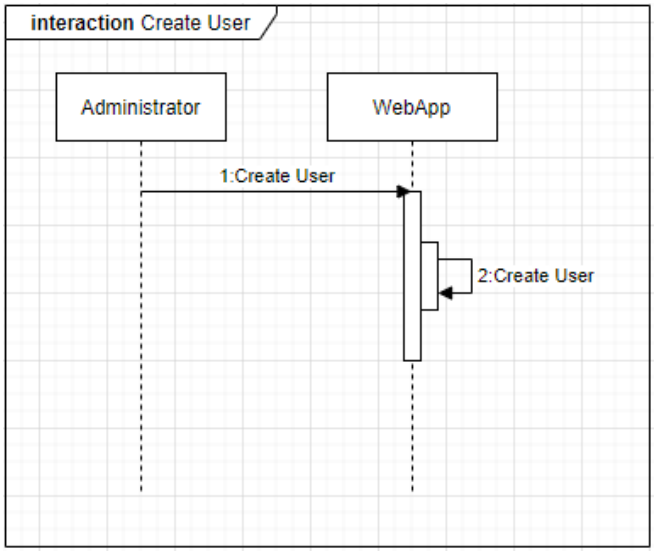


Figura A.10 Diagrama de secuencia Create User

- Delete User

| **Nombre** | **Delete User** |
|---|---|
| **Actores** | Administrador, WebApp |
| **Disparo** | Demanda del administrador a través de la aplicación para eliminar un usuario existente |
| **Garantías Mínimas** | El administrador eliminará un usuario de la plataforma |
| **Garantías de éxito** | La plataforma registrará la orden y eliminará de su base de datos todos los registros del usuario |
| **Escenario principal** | |
| 1. El administrador solicita a través de la aplicación eliminar un usuario.<br>2. La WebApp muestra al administrador todos los usuarios que hay registrados.<br>3. El administrador selecciona el usuario que quiere eliminar de la herramienta web.<br>4. La plataforma elimina todos los permisos y registros de dicho usuario. | |

Tabla A.11 Definición Delete User

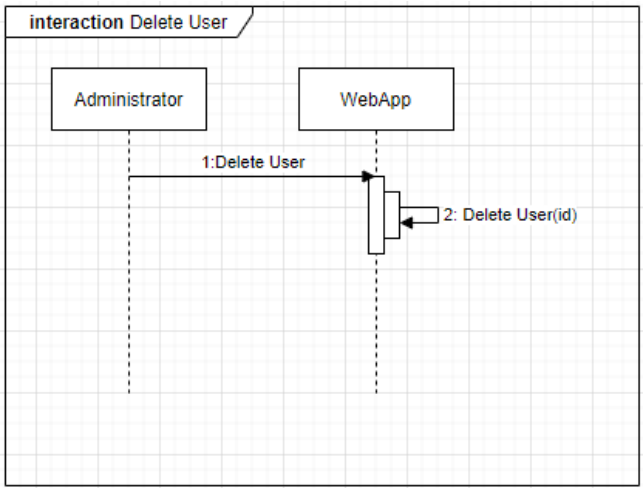


Figura A.11 Diagrama de secuencia Delete User

- Manage Permissions

| **Nombre** | **Manage Permissions** |
|---|---|
| **Actores** | Administrador, WebApp |
| **Disparo** | Demanda del administrador a través de la aplicación para cambiar los permisos que tienen los usuarios en la plataforma |
| **Garantías Mínimas** | El administrador modificará los permisos de los usuarios |
| **Garantías de éxito** | La plataforma registrará la orden del administrador y modificará los permisos de los usuarios |
| **Escenario principal** | |
| 1. El administrador solicita a través de la aplicación eliminar un usuario.<br>2. La WebApp muestra al administrador todos los usuarios que hay registrados.<br>3. El administrador selecciona el usuario al que quiere modificar sus permisos.<br>4. La plataforma modifica todos los permisos y registros la acción. | |

Tabla A.12 Definición Manage Permissions

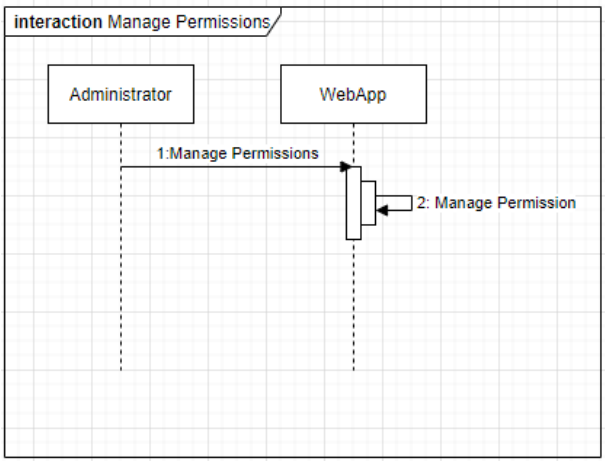


Figura A.12 Diagrama de secuencia Manage Permissions

- Manage ERP Connection

| **Nombre** | **Manage ERP Connection** |
|---|---|
| **Actores** | Administrador, WebApp |
| **Disparo** | Demanda del administrador a través de la aplicación para cambiar los datos referentes a la conexión con el ERP |
| **Garantías Mínimas** | El administrador modificará los datos de conexión |
| **Garantías de éxito** | La plataforma registrará la orden del administrador y modificará los datos de la conexión con el ERP |
| **Escenario principal** | |
| 1. El administrador solicita a través de la aplicación cambiar los datos de conexión.<br>2. La WebApp muestra al administrador los datos actuales de la conexión.<br>3. El administrador modifica los campos oportunos.<br>4. La plataforma guarda y registra la acción. | |

Tabla A.13 Definición Manage ERP Connection

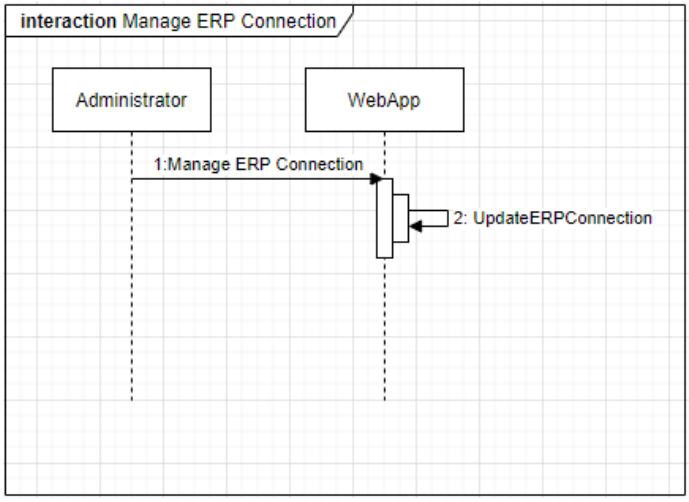


Figura A.13 Diagrama de secuencia Manage ERP Conncetion

## Diagramas de secuencia entre ERP y WebApp

Para mostrar mejor el ciclo de mensajes que se intercambiarán entre los distintos dispositivos, vamos a esbozar unos diagramas de secuencia que reflejen mejor dicha conectividad para dar al usuario una visión más completa de esta parte.

- Starting Mission from ERP

| **Nombre** | **Starting Mission from ERP** |
|---|---|
| **Actores** | Operator, ERP BBDD, ERP Integration API y WebApp |
| **Disparo** | Tras guardar un operador los datos pertinentes a una nueva misión en la BBDD del ERP, se lanza la orden de cargar datos y empezar misión desde la aplicación web |
| **Garantías Mínimas** | El usuario empezará una misión guardada en la BBDD del ERP |
| **Garantías de éxito** | La plataforma recibirá la orden y dará de alta la nueva misión enviando la acción al PMR |
| **Escenario principal** | |
| 1. El operadpr guarda los datos de la misión en la BBDD del ERP.<br>2. Tras esto, actualizará los datos en la aplicación web.<br>3. El operador puede selecciónar la misión y mandarla ejecutar.<br>4. Se envía la petición de la nueva misión a través del Broker MQQT hacia el PMR. | |

Tabla A.14 Definición Starting Mission from ERP

Como se puede apreciar en el siguiente diagrama, a partir del octavo mensaje, la secuencia corresponde con el diagrama presentado anteriormente en la figura A.5; por lo que se omite dicha parte para simplicidad del esquema.

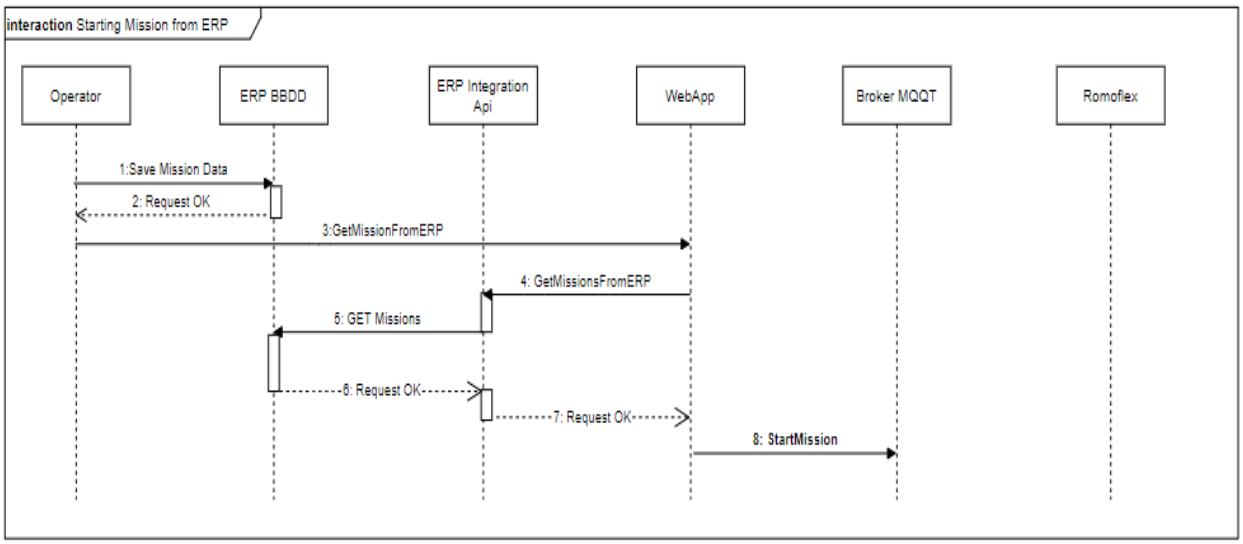



Figura A.14 Diagrama de secuencia Starting Mission from ERP

- Receive Incidences to ERP

| Nombre | **Receive Incidences to ERP** |
|---|---|
| **Actores** | ERP BBDD, ERP Integration API, WebApp, Broker MQTT, Romoflex |
| **Disparo** | Cuando el PMR registre una incidencia, esta tendrá que ser registrada en la BBDD del ERP |
| **Garantías Mínimas** | Se registrarán las incidencias en la BBDD del ERP |
| **Garantías de éxito** | La plataforma recibirá la orden y dará de alta en la BBDD del ERP la información de la incidencia proveniente del PMR |
| **Escenario principal** | |
| 1. El PMR lanza un mensaje de alarma al Broker MQTT.<br>2. La aplicación web registra la incidencia y, utilizando la Api de integración, manda los datos de esta hacia el ERP.<br>3. La base de datos del ERP recibe la petición y actualiza los datos de su tabla. | |

Tabla A.15 Definición Receive Incidences to ERP

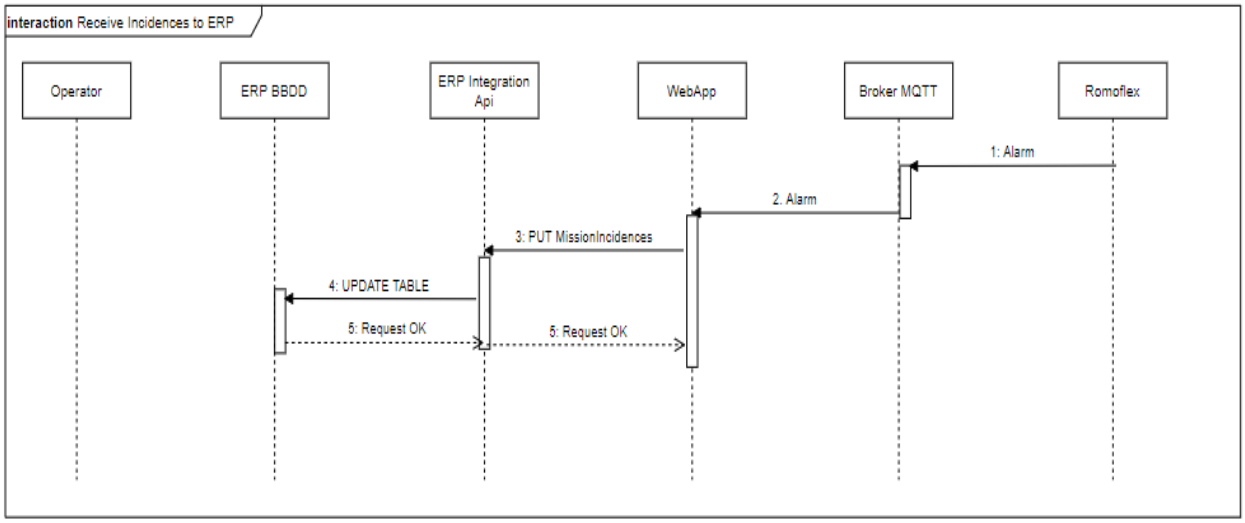



Figura A.15 Diagrama de secuencia Incidences to ERP

# Apéndice B

# Mensajes MQTT

Se muestra a continuación los mensajes en formato MQTT diseñados para ser utilizados en nuestro proyecto.

| Mensaje | StartMission |
|---|---|
| Sentido de la comunicación | Plataforma IoT → PMR |
| Formato | |

```
{ "MissionId": "mision1",
 "Origin": "1",
 "Destionation": [
          { "Name": "Dest1",
                  "Items": [
                           { "Name": "Item1",
                            "QrCode": "QR_CODE_1" } ] },
          { "Name": "Dest2",
                 "Items": [
                           { "Name": "Item2",
                             "QrCode": "QR_CODE_2" },
                          { "Name": "Item3",
                            "QrCode": "QR_CODE_3" } ] },
 "RobotId": "robot 1" }
```

Tabla B.1 Ejemplo mensaje StartMission

| Mensaje | UpdateStatus |
|---|---|
| Sentido | PMR → Plataforma IoT |
| Formato | |

```
{
 “MessageType”:”UpdateStatus”,
 "robot_id": "robot1",
 “motion_status”: “1”,
 "status": {
  "location": {
   "x": "xxx",
   "y": "yyyy",
   "z": "zz"
  },
  "mission": {
   "active": true,
   "id": "mision1"
  },
  "battery": {
   "percentage": 80,
   "estimated": "40h"
  },
  "moving": true,
  “current_vel” : “2”,
  "arm": {
   "moving": false
  }
 }
}
```

Tabla B.2 Ejemplo mensaje UpdateStatus

| Mensaje | CancelMission |
|---|---|
| Sentido | Plataforma IoT → PMR |
| Formato | |
| {<br>“MessageType”:”CancelMission”,<br>“MissionId”: “mission1”,<br>“RobotId”: “robot 1”<br>} | |

Tabla B.3 Ejemplo mensaje Cancel Mission

| Mensaje | Alarm |
|---|---|
| Sentido | PMR → Plataforma IoT |
| Formato | |
| {<br>“MessageType”:”Alarm”,<br>“RobotId”: “robot 1”,<br>“AlarmType”:”agv_blocked”,<br>“Timestamp”: 123123123123<br>} | |

Tabla B.4 Ejemplo mensaje Alarm

| Mensaje | GetStatus |
|---|---|
| Sentido | Plataforma IoT → PMR |
| Formato | |
| {<br>“MessageType”:”GetStatus”,<br>“RobotId”: “robot 1”<br>} | |

Tabla B.5 Ejemplo mensaje GetStatus

| Mensaje | RequestAction |
|---|---|
| Sentido | PMR → Plataforma IoT |
| Formato | |
| {<br>“MessageType”:”RequestAction”,<br>“MissionId”: “mision1”,<br>“Origin”: “1”,<br>“Destination”: “2”,<br>“RobotId”: “robot 1”<br>} | |

Tabla B.6 Ejemplo mensaje RequestAction

| Mensaje | MissionStats |
|---|---|
| Sentido | PMR → IoT |
| Formato | |


```
{
 “MessageType”:”MissionStats”,
 “MissionId”: “mision1”,
 “RobotId”: “robot 1”,
 “PoseeWithCovariance”: “n”,
  “Repeatability”: “20”,
   “Max_vel”: “1”
}
```


Tabla B.7 Ejemplo mensaje MissionStats

# Apéndice C

# Mockups Aplicación Web

Se presenta a continuación los prototipos que se han tenido en cuenta para el diseño de la aplicación web final con la que los operarios y administradores de las empresas tendrán que trabajar.

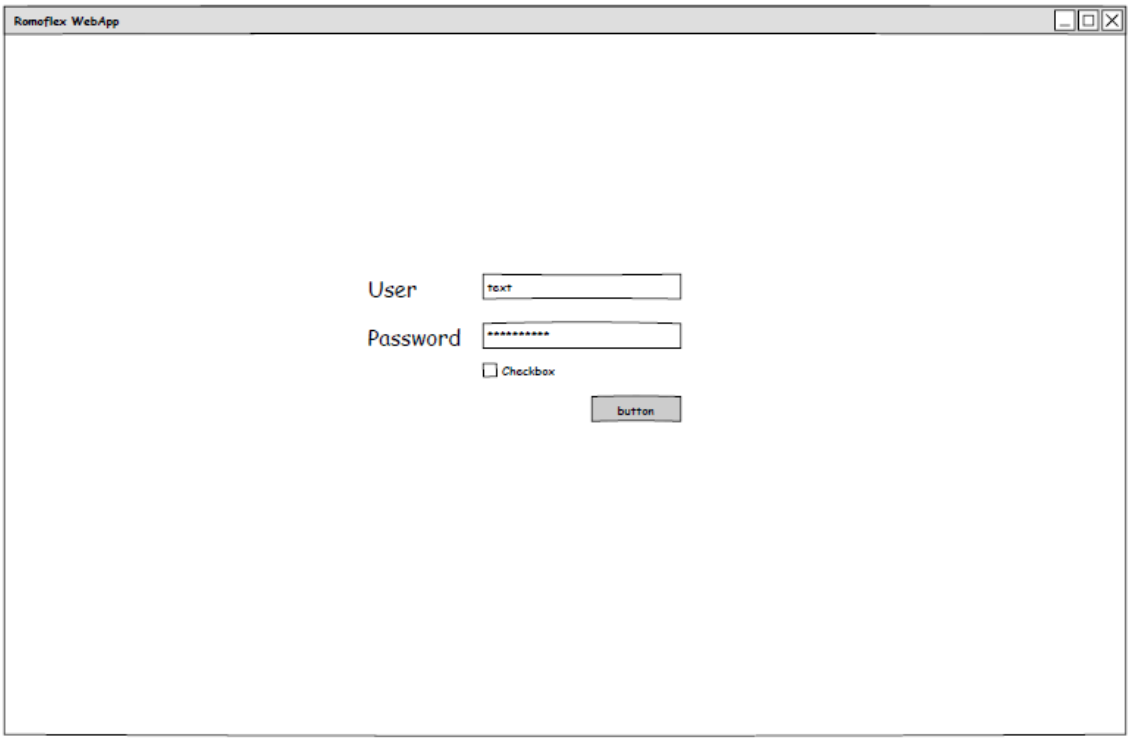



Figura C.1 Login

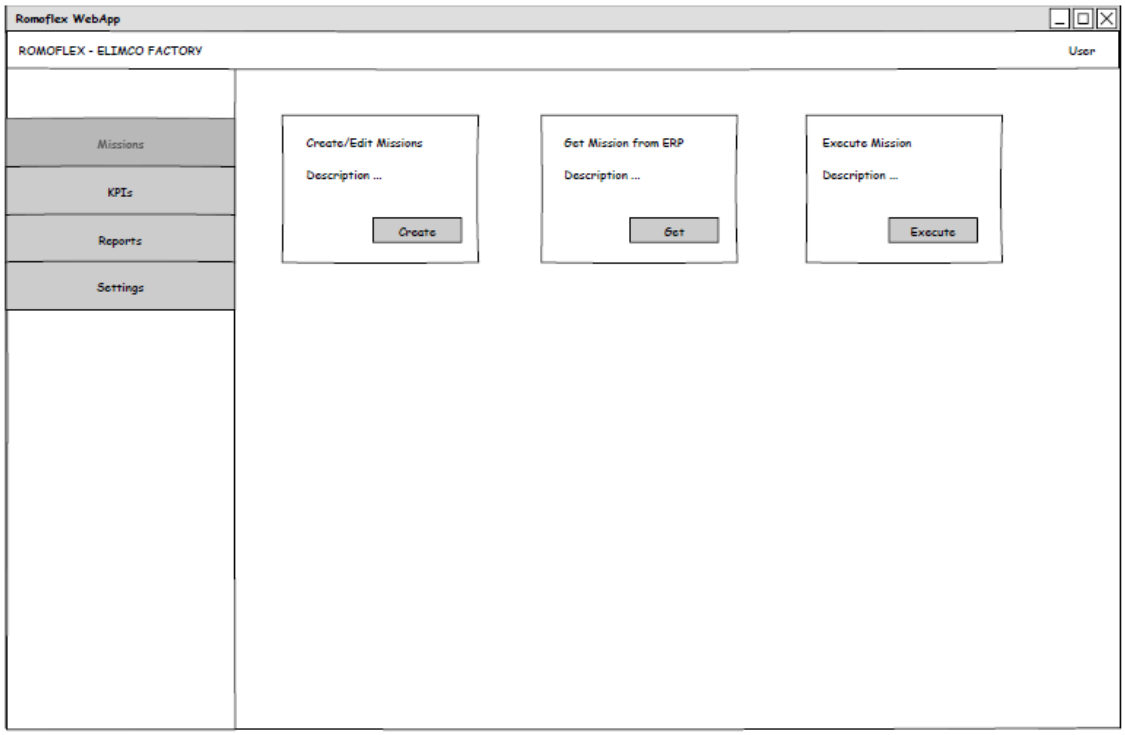



Figura C.2 Dashboard

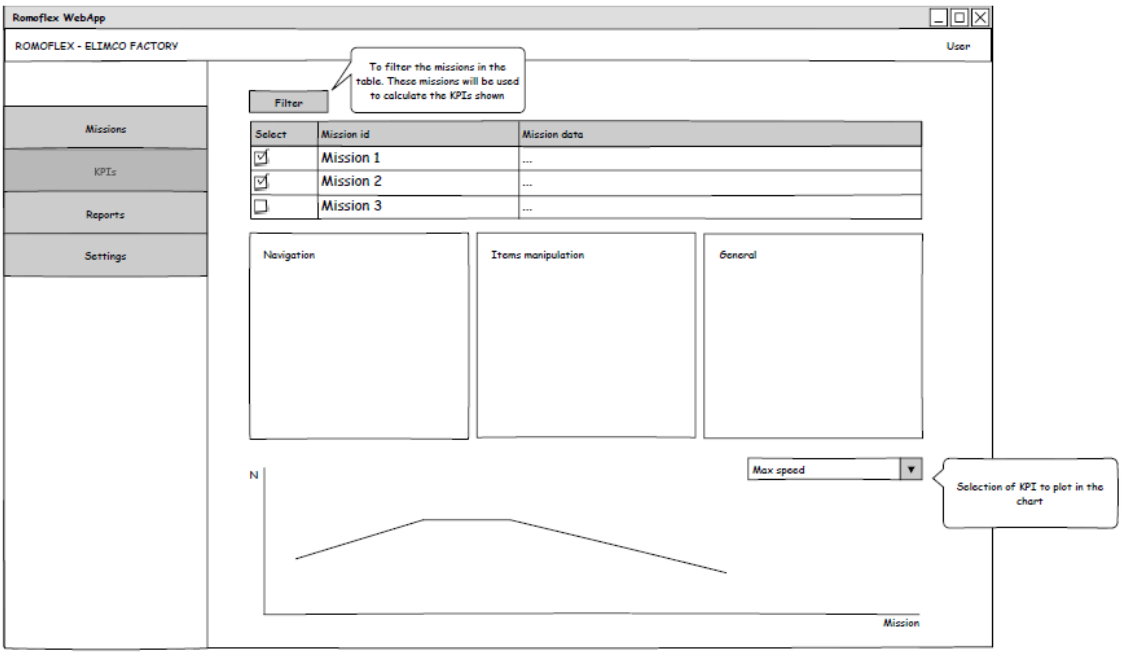



Figura C.3 KPIs

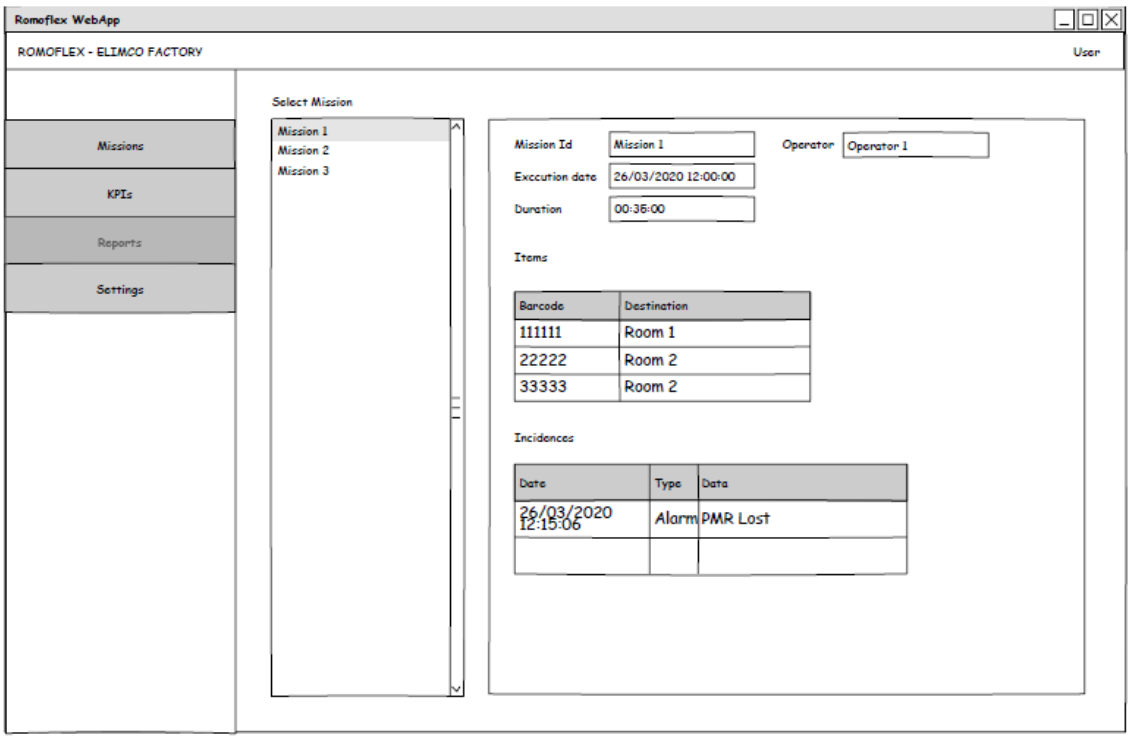



Figura C.4 Reports

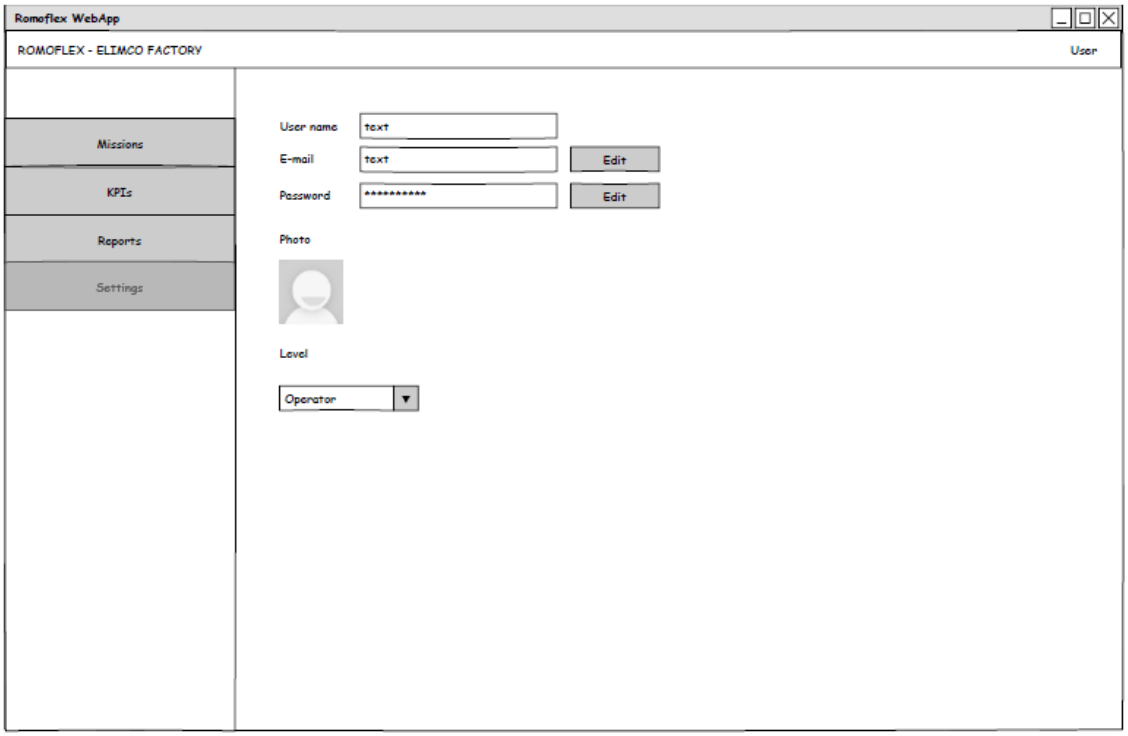



Figura C.5 Settings

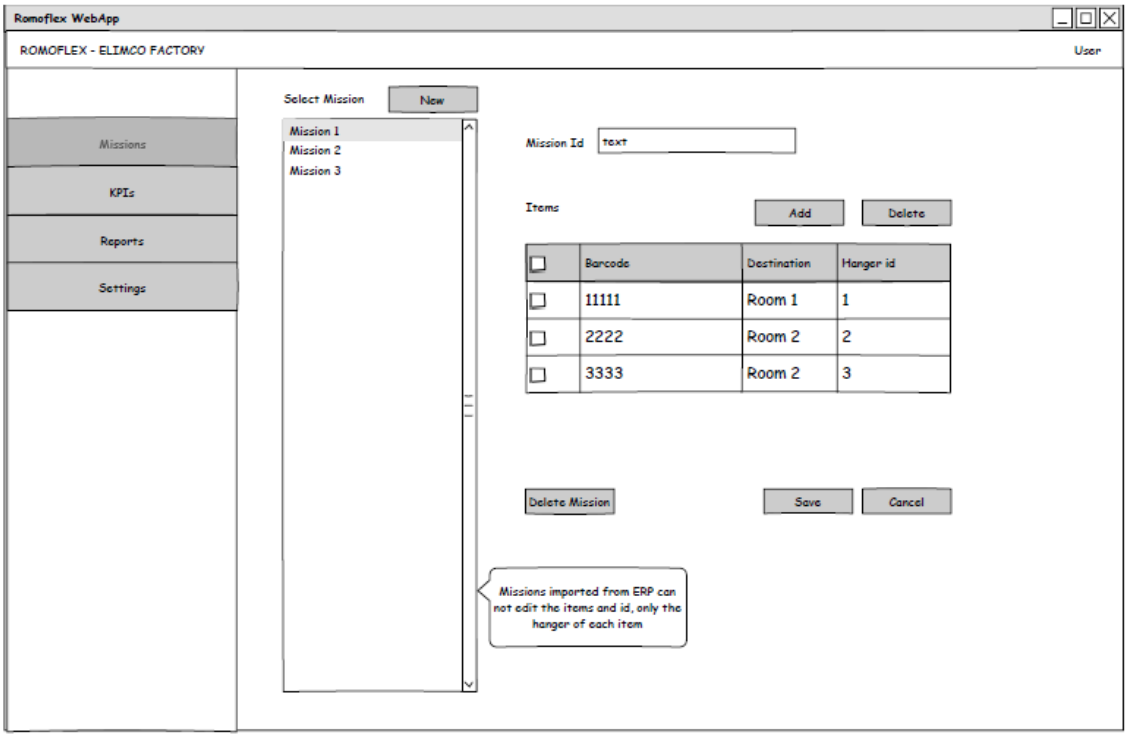



Figura C.6 Create/Edit Missions

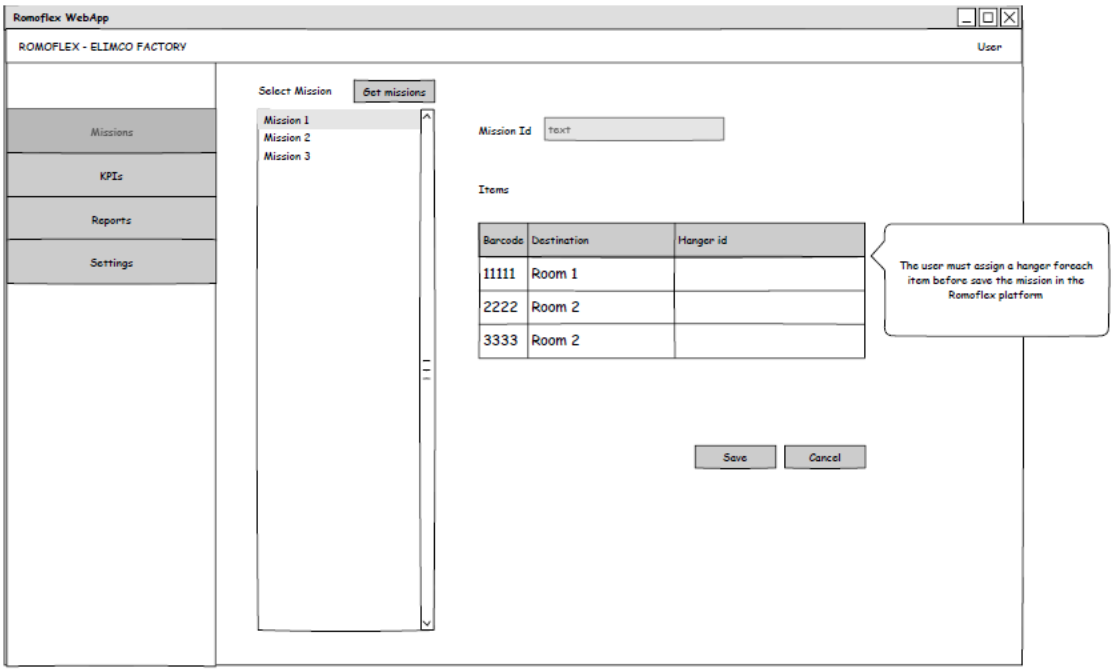



Figura C.7 Get Missions From ERP

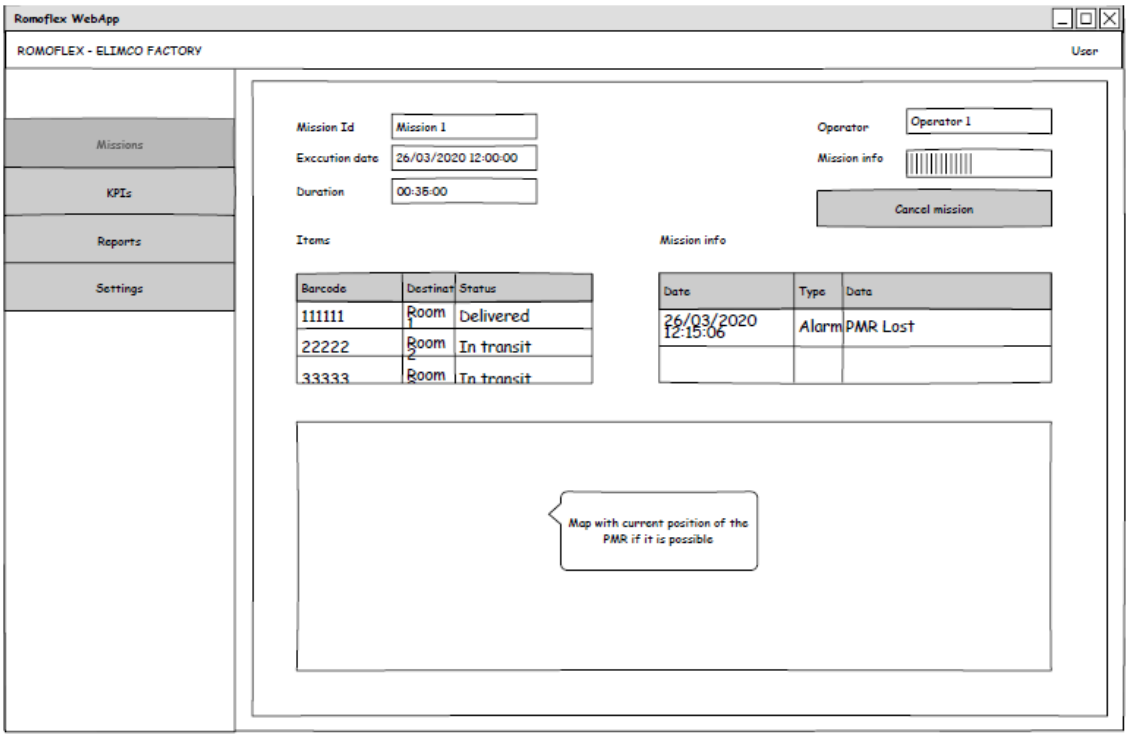



Figura C.8 Execute Mission

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga