

# TECHMED RESOLVE

## MEMORIA DE PROYECTO DE FIN DE CURSO

Proyecto de Fin de Ciclo Formativo de Grado Superior enfocado a la creación de una aplicación multiplataforma con Appian.

Alumno: Pedro Gómez Alonso

Tutor: Raquel Cerdá Losa

DAM 2022/23

## **Agradecimientos**

Quisiera expresar mi más sincero agradecimiento a mis profesores, en especial a Tomás Escudero, por disipar mis dudas cuando no estaba seguro de aceptar mi actual puesto como desarrollador de Appian y por instarme a tomar esa decisión. Esta elección ha transformado por completo mi vida.

Asimismo, deseo agradecer a Raquel Cerdá el haberme brindado la oportunidad de abordar y llevar a cabo mi proyecto relacionado con esta tecnología.

También quiero mostrar mi agradecimiento a mi empresa, Stemdo, por el apoyo brindado durante la realización de este proyecto, así como por la formación que me ha proporcionado.

Por último, es fundamental para mí expresar mi agradecimiento a mi pareja y familia, quienes siempre me han apoyado incondicionalmente, incluso cuando tomé la decisión de cambiar de rama profesional para dedicarme al desarrollo. Y en particular, gracias a mi padre por haberme inculcado el interés por la tecnología y la programación desde que era niño. Su influencia ha sido fundamental en mi desarrollo profesional y en la elección de esta carrera.

## **Índice general**

### **Contenido**

1. - Introducción.....	1
2. - Palabras y conceptos clave .....	3
3. – Módulos formativos aplicados en el proyecto .....	15
4. – Herramientas y lenguajes utilizados .....	18
5. – Fases del proyecto.....	19
5. – Conclusiones y mejoras del proyecto.....	51
6. – Bibliografía .....	52
7. – Anexos .....	52

## **1. - Introducción**

La idea de “**TechMed Resolve**”, nombre que se ha dado a la aplicación, surge por la necesidad de una automatización en todo el proceso de gestión de incidencias relacionadas con los aparatos de electromedicina de un hospital.

Cuando se avería una máquina, esto conlleva la realización de una serie de pasos muy definidos como podrían ser la detección de la avería, la creación de un parte, la asignación del parte a un técnico, la resolución, etc.

Para poder facilitar este tipo de procesos, un software **BPM** es especialmente útil, y se eligió **Appian** como plataforma BPM para llevar a cabo el proyecto, ya que este tipo de tecnologías están diseñadas precisamente para gestionar y automatizar procesos de negocio, y esto encajaba perfectamente con la casuística dada.

Por lo tanto, de forma resumida, el proyecto consiste en una aplicación que sirve para automatizar todo lo posible el proceso de creación, asignación y resolución de partes de reparación en un hospital ante las incidencias o averías ocasionadas en las diferentes máquinas de electromedicina.

TechMed Resolve está compuesta de 3 “**sub-aplicaciones**”:

1. **Aplicación para los empleados del hospital**: Será accesible por los empleados del hospital y les permitirá abrir incidencias de forma rápida y sencilla, pudiendo, además, fotografiar la máquina “en vivo” y adjuntarla en la apertura. Además, podrán ver el estado de los mismos en todo momento. En cualquier caso, tanto cuando abre la incidencia como cuando finaliza, será avisado via email de forma automática.
2. **Aplicación para los técnicos**: Será accesible por los técnicos de electromedicina, que tendrán un grid de tareas en el que podrán ver su asignación, y en las que les llegará toda la información de la incidencia generada por los empleados. Aquí es donde ellos mismos podrán cambiar el estado de las incidencias a “en curso”, “finalizado” (o a los diferentes estados que pueda presentar) y comenzar las reparaciones. Una vez finalizadas, podrán rellenar el parte creado de forma

automática. También podrán consultar el estado de los partes (abiertos o finalizados) en todo momento.

3. **Aplicación para los supervisores:** Desde esta aplicación, los supervisores podrán gestionar toda la información de los técnicos, las máquinas, etc. (tendrán un CRUD para cada una de ellas) así como ver una serie de informes que les permitirán sacar estudios y conclusiones. Esta aplicación además enviará un correo electrónico de forma automática cada viernes con un informe semanal indicando el estado de los equipos.

Además de lo anterior, se han desarrollado una serie de automatismos que permiten el envío de informes vía email cuando ocurren determinados eventos. También se ha incorporado un sistema de mensajería que permite la comunicación entre los diferentes equipos y que está disponible en las tres aplicaciones.

## **2. - Palabras y conceptos clave**

A continuación, se definirán una serie de conceptos que utilizados a lo largo del presente documento.

### **2.1. - BPM:**

Siglas de **Business Process Management** (Gestión de Procesos de Negocio). Es una metodología de trabajo utilizada por las empresas que se encarga de controlar el modelado, visibilidad y gestión de los procesos productivos de las mismas.

BMP implica adoptar una serie de pasos o acciones que modifican la forma de trabajar de la empresa con el objetivo de mejorar los procesos y facilitar la colaboración con un enfoque hacia el cliente.

### **2.2. - Low Code**

Low Code simplifica la creación de aplicaciones mediante interfaces visuales, permitiendo un desarrollo rápido y escalable al arrastrar y soltar componentes en pantalla. Utiliza configuraciones "point and click" en lugar de programación detallada, acelerando el proceso y haciéndolo más flexible. Los sistemas de Low Code ofrecen componentes reutilizables, eliminando la necesidad de crear elementos clave desde cero. A diferencia de No-Code, se requiere escribir cierto código, aunque en menor medida. Su popularidad ha crecido como una alternativa ágil al desarrollo convencional de software.

### **2.3. - Appian:**

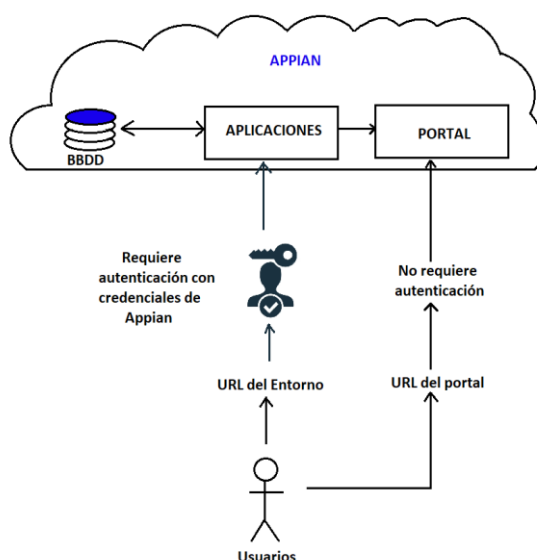
Plataforma **BPM** líder en el mercado enfocada a crear aplicaciones de software empresariales mediante **Low Code**. Permite a sus usuarios desarrollar, ejecutar y administrar aplicaciones sin la complejidad de construir y mantener la infraestructura típicamente asociada al desarrollo, despliegue y mantenimiento de una aplicación de software.

Appian es pionero en el desarrollo de soluciones empresariales con poco código para que se puedan entregar los proyectos en una fracción de lo que se tarda con otras tecnologías,

esto es hasta 20 veces más rápido (esta tecnología promueve que puedas crear una aplicación lista para producción en tan sólo 8 semanas).

Su configuración puede realizarse de dos maneras: la primera es en la nube, donde los administradores de Appian dan acceso a una instancia para tu empresa y se puede acceder desde cualquier parte del mundo. La segunda opción es tener una versión local en el servidor de tu compañía, aunque esa configuración debe ser realizada por un profesional de Appian.

### **¿Cómo acceden los usuarios a nuestras aplicaciones desarrolladas en Appian?**



Appian ofrece un entorno visual y colaborativo para el desarrollo eficiente de aplicaciones empresariales, alojado en la nube y es accesible a través de un navegador web o la aplicación móvil de Appian. Los usuarios inician sesión con sus credenciales de Appian en el entorno en el que se aloje, y dependiendo de sus permisos, pueden ejecutar la aplicación como usuarios finales o realizar modificaciones como desarrolladores. Tanto desarrolladores como usuarios finales acceden a la aplicación según sus permisos configurados en Appian.

Appian también ofrece la opción de crear "Portales", que son aplicaciones web públicas diseñadas para conectar a usuarios con datos y flujos de trabajo en Appian sin requerir un inicio de sesión. Los usuarios, como clientes externos, pueden acceder a estos portales

simplemente visitando una URL pública, lo que les permite ver o compartir información sin necesidad de tener una cuenta de usuario en Appian. Aunque no se han utilizado portales en la aplicación del proyecto actual, es relevante destacar esta posibilidad como una característica interesante.

#### **2.4. - Código de Appian: Expresiones y SAIL**

Como se ha comentado anteriormente, Appian se basa en el diseño de aplicaciones mediante Low Code, no obstante, se debe utilizar código si se desean realizar diseños complejos en nuestras interfaces, consultas a datos, o configuraciones personalizadas en la mayor parte de objetos de nuestra aplicación.

El código de Appian se utiliza para desarrollar lo que se conoce como **expresiones**. Por hacer una similitud con otros lenguajes de programación, una **expresión** vendría a ser algo así como un método que realizará una consulta, un cálculo o una operación y nos devolverá un resultado.

Este código también se utiliza junto con **SAIL** para el desarrollo de sus interfaces.

**SAIL** es un framework de Appian patentado para el desarrollo de interfaces mediante definiciones declarativas para generar experiencias de usuarios interactivas y multiplataforma.

#### **2.5. - Rule inputs:**

Haciendo un símil con otros lenguajes de programación, las rule inputs son parámetros de entrada/salida que se pueden pasar a expression rules e interfaces (objetos de appian que almacenan expresiones y que se definen en el siguiente apartado).

Por poner un ejemplo, imaginemos que un usuario rellena un formulario de registro. Una vez rellenado dicho formulario, los datos introducidos se podrían guardar en “Rule Inputs” para que fueran accesibles desde el exterior (en este caso usamos las rule inputs como “salida”). Una vez pulsado el botón de “Enviar” del formulario, se pondría en marcha una serie de procesos y cuando se llega a un determinado punto, se podría lanzar otra interfaz a otro equipo de trabajo para mostrar los datos que el primer usuario rellenó.



Para indicarle a esa interfaz los datos recopilados anteriormente, hay que pasárselos mediante el uso de rule inputs (en este caso, las usaríamos como “entrada”).

Para llamar a una rule input en el código de Appian, hay que hacerlo mediante el prefijo “**ri!**” Seguido del nombre de la rule input:

```
a!isNotNullOrEmpty(ri!username),
```

## 2.6. - Objetos más utilizados de Appian

Entre los objetos más utilizados e importantes se encuentran:

### 2.6.1. – Constantes

Las constantes en Appian, a diferencia de lo que ocurre en otros lenguajes de programación, son objetos. Sin embargo, su utilidad es la misma que en otros lenguajes ya que almacenan valores que no deben cambiar durante la ejecución de las aplicaciones.

Una constante puede referenciar valores de tipos de datos primitivos o datos más complejos como record types, la ubicación de un archivo, etc.

Para llamar a una constante en el código de Appian, se hace mediante el prefijo “**cons!**” seguido del nombre de la constante:

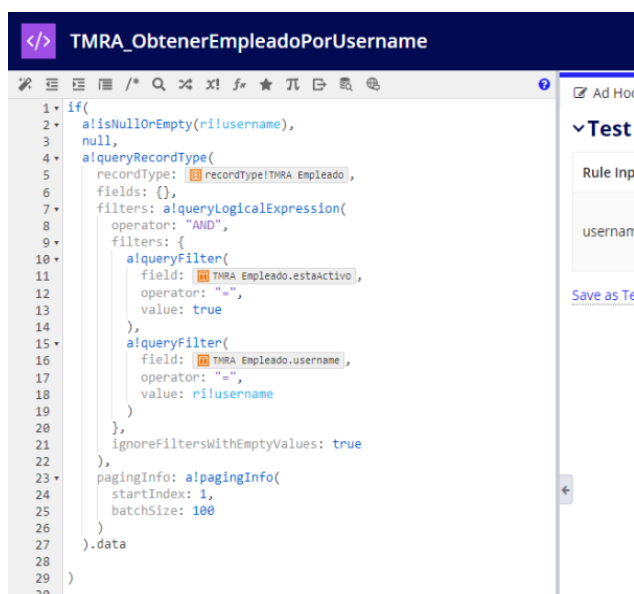
```
cons!TMRA_ESTADOS_USUARIOS_APPIAN[1].
```

### 2.6.2. - Expression Rules

Objeto de Appian que almacena una **expresión** que sirve para definir y evaluar expresiones lógicas o matemáticas dentro de una aplicación. Son fundamentales para manipular datos, realizar cálculos, tomar decisiones y controlar el flujo de ejecución en una aplicación Appian. Haciendo un símil con otros lenguajes de programación, sería algo similar a un objeto que almacena un método que podrá ser llamado desde otras partes de la aplicación.

A continuación, se muestra una captura de pantalla de una expresión en una Expression Rule que nos devuelve los datos de un empleado dado un nombre de usuario pasado por

Rule Input. Las Rule Input son parámetros de entrada-salida que se utilizan en Expression Rules e Interfaces para recibir datos de fuera del objeto o mandarlos hacia fuera.



```

1 if(
2   aisEmpty(rulusername),
3   null,
4   alqueryRecordType(
5     recordType: recordTypeTMRA Empleado,
6     fields: {},
7     filters: alqueryLogicalExpression(
8       operator: "AND",
9       filters: {
10        alqueryFilter(
11          field: TMRA Empleado.estaActivo,
12          operator: "=",
13          value: true
14        ),
15        alqueryFilter(
16          field: TMRA Empleado.username,
17          operator: "=",
18          value: rulusername
19        )
20      },
21      ignoreFiltersWithEmptyValues: true
22    ),
23    pagingInfo: alpagingInfo(
24      startIndex: 1,
25      batchSize: 100
26    )
27  ).data
28 )
29
30

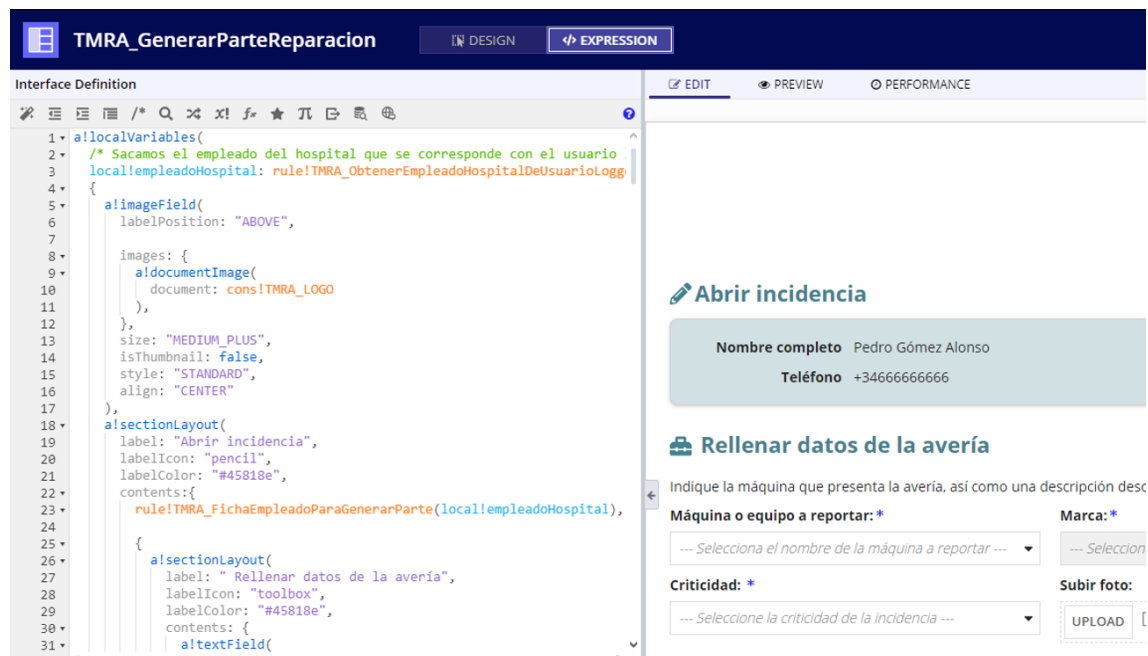
```

### 2.6.3. - Interfaces

Las interfaces son la forma en la que los usuarios finales interactúan con nuestra aplicación, y son un componente esencial de todas las aplicaciones de Appian. Estas pueden desarrollarse mediante dos modos: el “**Design Mode**” (o modo de diseño), que permite crear nuestra interfaz mediante configuraciones drag & drop y point and click, y el “**Expression mode**” (o modo de expresión) que nos permite crearla mediante código.

Lo más normal, es que primero se distribuyan los componentes mediante el design mode, y se pase al expression mode para realizar las configuraciones a nuestro gusto.

A continuación, se muestra una interfaz en el expression mode. Como se puede apreciar, hay diversas funciones de Appian para crear los componentes.



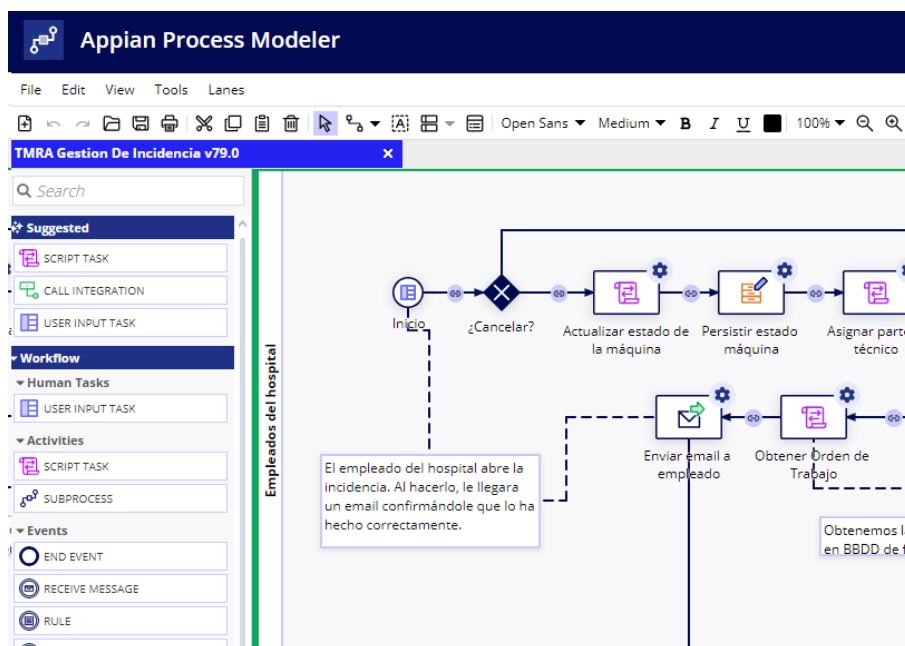
#### 2.6.4. - Process Model / Modelos de proceso

Cada vez que en Appian se realiza una **acción** como enviar un formulario, añadir o eliminar datos etc, se ejecuta un process model.

Los **process model** son objetos que nos permiten programar acciones en Appian mediante programación nodal. Esto, además, nos permite mostrar a los usuarios de negocio de forma clara y sencilla un proceso de negocio estructurado. Estos modelos de proceso en Appian se crean utilizando el lenguaje de diseño visual proporcionado por la plataforma.

En Appian, los process models utilizan nodos visuales en lugar de programación tradicional, simplificando la representación de tareas y decisiones en el flujo de trabajo. Este enfoque intuitivo facilita la colaboración entre equipos técnicos y usuarios de negocio. Además, ofrece la capacidad de definir reglas de negocio, integrar sistemas externos y establecer flujos de trabajo automatizados, todo dentro de un entorno gráfico que permite una rápida adaptación a cambios en los procesos de negocio.

A continuación, se muestra parte de uno de los process model del proyecto:



### Nodos utilizados en el proyecto

A continuación, se definen los nodos utilizados en los process model del proyecto:



**Nodo de inicio:** Sirve para indicar el inicio de un proceso. Si dentro del círculo viene el símbolo de una interfaz, quiere decir que el proceso se inicia con un formulario. Si viene con el símbolo de un reloj, quiere decir que está configurado como un BATCH que se ejecutará en determinado momento de forma automática. El nodo de fin, tiene el mismo aspecto, pero es simplemente un círculo vacío.

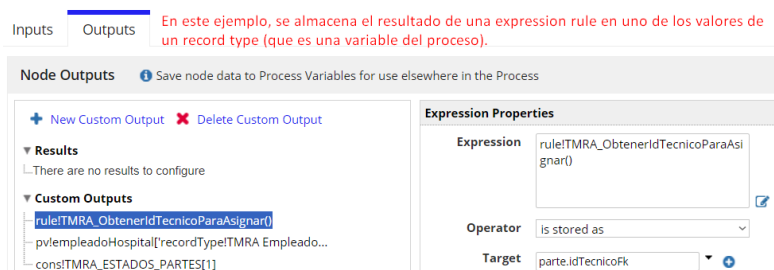


**Gateway XOR:** Sirve para que el flujo del proceso continúe en función de una condición. En la programación convencional, sería similar a un “if” o un switch. Si se cumple una determinada condición, el flujo sigue por un camino, y si no, por otro.

Conditions				
	Condition	Result	Path Label	Order
✖	If <input type="text" value="=pv/cancelar"/> is True	go to <input type="text" value="Fin"/>	<input type="text"/>	
	Else if no rules are TRUE	go to <input type="text" value="Actualizar estado de la m..."/>		



**Script Task:** Nodo que nos sirve para ejecutar expresiones y almacenar su resultado donde en variables del proceso para su utilización durante el mismo.



**Write Record:** Nodo que sirve para persistir en base de datos la información actual del record type que se le pasa por parámetro.



**Send Email:** Smart Service que nos permite el envío de un correo electrónico a quien deseemos. En este nodo se puede añadir una plantilla HTML para generar un correo más personalizado.



**User Input Task:** Nodo que carga una interfaz como tarea para el usuario al que deseemos asignársela. Le podremos pasar rule inputs como entrada y obtener el resultado de la interacción del usuario como otras rule inputs de salida.



**Add User:** Crea un usuario en el entorno de Appian con los datos que se le pasen por parámetro.



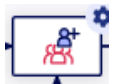
**Reactivate User:** En Appian, los usuarios no se pueden eliminar, simplemente se desactivan. Por lo que este nodo, nos permite reactivar un usuario.



**Update User:** Actualiza la información de un usuario en el entorno de Appian con los datos que se le pasen por parámetro.



**Deactivate User:** Desactiva al usuario indicado en el entorno de Appian.



**Add Group Members:** Añade al usuario indicado al grupo especificado.



**Execute Stored Procedure:** Ejecuta el procedimiento almacenado que deseemos de nuestra base de datos.



**Sync Records:** Sincroniza el record type que le indiquemos con su fuente de datos. Como el procedimiento almacenado se ejecuta en la BBDD, es necesario utilizar este nodo para sincronizar después.

#### **2.6.5. - Record Type**

Es un objeto que nos sirve para definir la estructura y los campos asociados con un conjunto específico de datos y se utiliza para representar entidades o información en una aplicación, almacenando datos relacionados con la entidad que representa.

Principalmente es empleado para replicar entidades de bases de datos relacionales y facilitar la realización de consultas complejas y la manipulación eficiente de datos.

Además de proporcionar una interfaz para interactuar con los datos, un "record type" en Appian ofrece herramientas como la generación automática de formularios, la definición de reglas de negocio y la integración con flujos de trabajo para automatizar procesos relacionados con la entidad representada.

Por nombrar algunas, entre las herramientas más utilizadas en un record type se encuentran:

- **Record actions**: Permiten definir y ejecutar acciones específicas sobre los datos almacenados en el mismo, brindando una interfaz fácil de usar para interactuar con la información dentro de las aplicaciones desarrolladas en la plataforma. Se les asocia un process model.
- **Filters**: Permite definir filtros a los datos del record type. Cuando se desee mostrarlos en un grid, se podrá hacer uso de estos filtros preconfigurados por nosotros.

#### 2.6.6. - **Site**:

Los sites son objetos de Appian con una interfaz que los usuarios finales ven y que sirve como punto de entrada para los usuarios finales a las aplicaciones desarrolladas en Appian. Digamos que es el lugar donde los usuarios finales interactúan con las aplicaciones.

Los sites tienen una URL asociada, que será la que será facilitada a los usuarios finales para que puedan acceder a nuestra aplicación. Estos usuarios, deben tener creado un usuario de Appian para poder acceder.

En estos objetos podremos configurar las páginas que compondrán nuestra aplicación que podrán ser interfaces, el resultado de un process model, de un record action o de una lista configurada de un record type.

A continuación, se puede ver el aspecto que tiene la ventana de configuración de un site.

## 2.7. – SCV de Appian

Es muy importante destacar que Appian tiene su propio **sistema de control de versiones** que es muy fácil de utilizar y muy intuitivo.

Cada vez que se realiza un guardado en un objeto de Appian (ya sea una interfaz, una expression rule, un process model o cualquier otro) se genera una nueva versión. Si nuestro componente falla, podemos volver a una versión anterior siempre que lo deseemos pulsando en la opción “versions”

PREDEFINED OBJECTS			
NEW	ADD EXISTING	DUPLICATE	ADD TO PACKAGE
SECURITY	DEPENDENTS	PRECEDENTS	MOVE
DELETE	MORE	RECYCLE	
	Properties	Versions	New Version
	Rename	Download	View Documentation
Name	Description		
<input type="checkbox"/> TMRA Gestion De Incidencia	Modelo de proceso para la gestión de las incidencias.		máquina
<input type="checkbox"/> TMRA_GridTareasTecnicoLoggeado	Grid con las tareas del técnico loggeado.		
<input type="checkbox"/> TMRA Tecnicos	Site para los técnicos de electromedicina.		
<input type="checkbox"/> TMRA_InterfazIncidenciasParaEmpleado	Interfaz que muestra las incidencias abiertas por el empleado loggeado y que per		
<input type="checkbox"/> TMRA_FichaEmpleadoParaGenerarParte	Tarjeta que muestra los datos del empleado del hospital que va a abrir el parte/r		
<input type="checkbox"/> TMRA_GenerarParteReparacion	Interfaz para abrir una incidencia sobre una máquina por los empleados del hosp		
<input type="checkbox"/> TMRA_FichaEmpleadoParaGenerarParteTablet	Tarjeta que muestra los datos del empleado del hospital que va a abrir el parte/r		
<input type="checkbox"/> TMRA_FichaEmpleadoParaGenerarParteSmartphone	Tarjeta que muestra los datos del empleado del hospital que va a abrir el parte/r		
<input type="checkbox"/> TMRA_DatosAperturaIncidenciaDispMovil	Interfaz que muestra los datos del empleado del hospital que abrió una incidenci		
<input type="checkbox"/> TMRA_RellenarParte	Interfaz para rellenar el parte de reparación para los técnicos.		
<input checked="" type="checkbox"/> TMRA_TIPOS_PANTALLA	Constante que almacena los tipos de ancho de pantalla que acepta la funcion isp		
<input type="checkbox"/> TMRA_ObtenerTipoPantalla	Devuelve el tipo de pantalla que se está utilizando.		



Si no deseamos volver a ninguna versión, pero queremos comparar la versión actual con alguna anterior, también podremos hacerlo.

## Versions

COMPARE

--- Select a Version ---

WITH

Latest

COMPARE

Created By

From

To

mm/dd/yyyy

mm/dd/yyyy

Clear Filters

Version ↓	Name	Description	Created		
Latest	TMRA_TIPOS_PANTALLA	Constante que almacena los tipos de ancho de pant...	11/30/2023 5:36 PM by Pedro Gómez Alonso		×
1	TMRA_TIPOS_PANTALLA		11/30/2023 12:32 PM by Pedro Gómez Alonso	↔	×

CLOSE

## 2.8. - Historia de usuario:

En la metodología Agile, una historia de usuario es una técnica para expresar requisitos de software desde el punto de vista del usuario final. Estas historias describen una funcionalidad específica que el sistema debe proporcionar y están escritas en un lenguaje natural comprensible tanto por los usuarios como por los desarrolladores.

Una **historia de usuario** generalmente sigue una plantilla simple que incluye:

- **Título**: Un breve nombre o frase que describe la funcionalidad que se desea implementar.
- **Descripción**: Una narrativa más detallada que explica qué es lo que el usuario desea lograr y por qué es importante para él o ella.
- **Criterios de aceptación**: Condiciones específicas que deben cumplirse para que la historia de usuario se considere completa y aceptada. Estos criterios son una forma de medir y verificar el éxito de la implementación.

### **3. – Módulos formativos aplicados en el proyecto**

Para la realización del proyecto me he apoyado en los conocimientos adquiridos en los siguientes módulos del ciclo formativo:

- **Programación:**

Evidentemente, se necesita tener una base en programación para poder desarrollar la aplicación con Appian, dado que, aunque en Appian se hace uso de herramientas low-code, se utiliza mucho más código del que podría parecer.

Además, a la hora de desarrollar modelos de procesos, aunque es mediante drag & drop, se deben tener nociones sobre esta materia, ya que, de lo contrario, difícilmente puedes diseñar un proceso, una interfaz compleja o una expression rule de forma correcta. Principalmente, los conocimientos adquiridos en este módulo se han aplicado en el desarrollo de estos objetos.

- **Bases de datos**

Se ha necesitado almacenar la información de los distintos usuarios, los partes, las máquinas, etc. en una base de datos relacional. Esta información es utilizada y forma el grueso de toda la aplicación, por lo que ha sido fundamental tener conocimiento de esta materia.

Los conocimientos adquiridos en este módulo se han aplicado en el diseño y creación de la base de datos, así como para crear un procedimiento almacenado que es llamado desde la aplicación.

- **Entornos de desarrollo**

Se deben tener conocimientos de testing, pruebas unitarias, etc. Además, es esencial contar con un dominio de las habilidades adquiridas en este módulo para la utilización efectiva de Git y GitHub. También se han de tener conocimientos para la creación de diagramas (diagrama de casos de uso, DER y demás) estudiados en esta materia.

Los conocimientos adquiridos en el módulo de entornos de desarrollo se han aplicado en:

- Creación de tests en cada una de las expression rules, interfaces y demás.
- Creación de repositorio de git y github y subida de contenido.
- Creación de diagramas para la memoria.

#### - **Lenguajes de marcas**

La aplicación realiza el envío de correos electrónicos automáticos y no automáticos mediante un Smart Service. Estos emails tienen como cuerpo del mensaje una plantilla diseñada en html con estilos CSS. Ambos son lenguajes de marcas vistos en este módulo.

Los conocimientos adquiridos en este módulo se han aplicado en la creación de las plantillas HTML y en darles estilo con CSS.

#### - **Programación de servicios y procesos**

En este módulo aprendimos, entre otras cosas, verbos HTTP y manejo de APIs.

Se ha realizado una integración con una API para hacer una simulación de mensajes ficticios extraídos de la misma. No tiene ninguna utilidad, simplemente se ha realizado como una muestra ya que no encontré ninguna API gratuita relevante para mi aplicación y no tuve tiempo de crear una yo mismo.

Los conocimientos adquiridos en este módulo se han aplicado en el acceso a la API (hay que conocer qué es, cómo se utiliza, qué son Query Params, etc.).

#### - **Desarrollo de interfaces**

En este módulo aprendimos conceptos y técnicas relacionadas con UIX que podrán ser aplicados a la hora de diseñar las interfaces de la aplicación, como es la elección correcta de una paleta de colores, tamaño de los botones, y componentes a utilizar para que la aplicación se vea de la forma más correcta

posible en el dispositivo para el que está pensado y que la experiencia de usuario sea la adecuada.

Los conocimientos adquiridos en este módulo se han aplicado en la creación de las interfaces de la aplicación.

- **Sistemas de gestión empresarial**

Al ser Appian una plataforma BPM, se ha considerado que el conocimiento de estos queda englobado en este módulo.

#### **4. – Herramientas y lenguajes utilizados**

**1º - Appian:** Dado que se ha necesitado un **BPM**, se ha pensado en Appian como tecnología principal a usar, por los motivos indicados en la introducción.

**2º - MySQL:** Se ha elegido MySQL como sistema de gestión de bases de datos relacionales de nuestra aplicación debido a que es gratuito y a que Appian, por defecto, viene con una base de datos ya conectada y configurada y que es administrada mediante phpMyAdmin.

**3º - PHPMyAdmin:** Como herramienta para gestionar la base de datos se ha utilizado phpMyAdmin, por el motivo indicado en el punto anterior.

**4º - Trello:** Se ha utilizado Trello para la organización de tareas, gestionar tiempos y hacer un seguimiento de qué funcionalidades quedaban por desarrollar, cuáles estaban terminadas y cuáles había que testear, etc. Además, ha servido como cuaderno de bitácora.

**5º - HTML y CSS:** Se han utilizado para la creación de plantillas de correos electrónicos que son enviados de forma automática a algunos usuarios de la aplicación.

**6º. – GitHub:** Utilizado para almacenar el código de las Expression Rules e Interfaces, así como otros anexos que se han considerado de interés. La URL del repositorio se encuentra en el apartado “Anexos” del presente documento.

**7º. – Draw.io:** Aplicación web que brinda una serie de facilidades para el dibujo de todo tipo de diagramas. En el caso del proyecto realizado, se ha utilizado para dibujar el diagrama de casos de uso de la aplicación.

**8º. – Balsamiq Wireframes:** Utilizada para dibujar wireframes y mockups de forma rápida, sencilla e intuitiva y en el caso de este proyecto, se ha utilizado para dibujar los wireframes de las principales interfaces de la aplicación.

**9º. – dbdiagram.io:** Para dibujar el diagrama Entidad Relación de la base de datos de nuestra aplicación, se ha hecho uso de esta herramienta que, mediante código sencillo y configuraciones drag & drop, permite el diseño de bases de datos de forma rápida y sencilla.

## 5. – Fases del proyecto

### 5.1. – Modelo de datos

La base de datos ha pasado por diversas modificaciones a lo largo del desarrollo de la aplicación, ya que se necesitaron tablas y campos con los que no se contaban en un principio. Tanto el **DER** como el código MySql se encuentran en el repositorio de github cuya URL se encuentra en el apartado de **anexos** del presente documento. No se ha añadido una captura de pantalla dado que, al ser bastante grande, pierde calidad al adjuntarlo en esta página.

En un principio, la base de datos constaba de 7 entidades, sin embargo, la versión final consta de 11. A continuación se explica cada una de ellas, y se acompaña de una tabla en la que vienen indicados los atributos, y el modelo de datos. Cabe destacar que el nombre de todas las tablas viene precedido de las siglas TMRA. Este es el prefijo de la aplicación que se ha desarrollado, y según las buenas prácticas, es muy recomendable nombrar las tablas así para facilitar su búsqueda, ya que el mismo entorno de Appian (y la misma BBDD) pueden almacenar datos de múltiples aplicaciones.

#### TMRA Mensaje:

La tabla de esta entidad almacena los mensajes enviados y recibidos de forma interna por los empleados que usan la aplicación. Sus atributos son los siguientes:

TMRA_Mensaje	Clave	Tipo Clave	Null	Tipo dato	Multivaluado	Tipo de atributo
id	SI	PK	NO	ENTERO	NO	SIMPLE
n_empleado_remitente_fk	SI	FK	NO	ENTERO	NO	SIMPLE
n_empleado_destinatario_fk	SI	FK	NO	ENTERO	NO	SIMPLE
asunto	NO	-	SI	CADENA	NO	SIMPLE
cuerpo_mensaje	NO	-	SI	CADENA	NO	SIMPLE
fecha_envio	NO	-	NO	FECHA	NO	SIMPLE
activo_remitente	NO	-	NO	ENTERO	NO	SIMPLE
activo_destinatario	NO	-	NO	BOOLEANO	NO	SIMPLE

### **TMRA Empleado:**

La tabla de esta entidad almacena la información genérica de todos los empleados que usen esta aplicación. Sus atributos son los siguientes:

TMRA_Empleado	Clave	Tipo Clave	Null	Tipo dato	Multivaluado	Tipo de atributo
n_empleado	SI	PK	NO	ENTERO	NO	SIMPLE
tipo_empleado_fk	SI	FK	NO	ENTERO	NO	SIMPLE
Nombre	NO	-	NO	CADENA	NO	SIMPLE
primer_apellido	NO	-	NO	CADENA	NO	SIMPLE
segundo_apellido	NO	-	NO	FECHA	NO	SIMPLE
telefono	NO	-	NO	ENTERO	NO	SIMPLE
telefono_emergencia	NO	-	NO	ENTERO	NO	SIMPLE
email	NO	-	NO	CADENA	NO	SIMPLE
username	NO	-	NO	CADENA	NO	SIMPLE
horario_trabajo	NO	-	NO	CADENA	NO	SIMPLE
fotografia	NO	-	SI	ENTERO	NO	SIMPLE
esta_activo	NO	-	NO	BOOLEANO	NO	SIMPLE

### **TMRA Empleado Hospital:**

La tabla de esta entidad almacena la información específica de todos los empleados que usen esta aplicación que formen parte del personal del hospital. Sus atributos son los siguientes:

TMRA_Empleado_hospital	Clave	Tipo Clave	Null	Tipo dato	Multivaluado	Tipo de atributo
id	SI	PK	NO	ENTERO	NO	SIMPLE
n_empleado_fk	SI	FK	NO	ENTERO	NO	SIMPLE
puesto	NO	-	NO	CADENA	NO	SIMPLE
area	NO	-	NO	CADENA	NO	SIMPLE
esta_activo	NO	-	NO	BOOLEANO	NO	SIMPLE

### **TMRA Tecnico**

La tabla de esta entidad almacena la información específica de todos los empleados que sean técnicos de electromedicina que usen esta aplicación. Sus atributos son los siguientes:

TMRA_Tecnico	Clave	Tipo Clave	Null	Tipo dato	Multivaluado	Tipo atributo
id	SI	PK	NO	ENTERO	NO	SIMPLE
n_empleado_fk	SI	FK	NO	ENTERO	NO	SIMPLE
categoria	NO	-	NO	CADENA	NO	SIMPLE
especialidad	NO	-	NO	CADENA	NO	SIMPLE
precio_hora	NO	-	NO	DECIMAL	NO	SIMPLE
nPartesAsignados	NO	-	NO	ENTERO	NO	SIMPLE
id_equipo_fk	SI	FK	NO	ENTERO	NO	SIMPLE
esta_activo	NO	-	NO	BOOLEANO	NO	SIMPLE

### **TMRA Tipo Empleado**

La tabla de esta entidad almacena el nombre del rol de cada tipo de empleado.

Esta tabla podría ser innecesaria, pero la creé por motivos de escalabilidad, por si se necesitaran en un futuro datos específicos relacionados con el tipo del empleado:

TMRA_Tipo_Empleado	Clave	Tipo Clave	Null	Tipo dato	Multivaluado	Tipo atributo
id	SI	PK	NO	ENTERO	NO	SIMPLE
nombre	NO	-	NO	CADENA	NO	SIMPLE
esta_activo	NO	-	NO	BOOLEANO	NO	SIMPLE



### **TMRA\_Equipo:**

La tabla de esta entidad almacena el nombre del equipo al que pertenecen los grupos de técnicos que supervisan los supervisores. Se ha creado por escalabilidad, por si se diera el caso de que se deseara que hubiera tal cantidad de técnicos que se necesitaran varios supervisores que supervisarán diferentes equipos de trabajo. Los atributos son los siguientes:

TMRA_Equipo	Clave	Tipo Clave	Null	Tipo dato	Multivaluado	Tipo atributo
id	SI	PK	NO	ENTERO	NO	SIMPLE
nombreEquipo	NO	-	NO	CADENA	NO	SIMPLE
esta_activo	NO	-	NO	BOOLEANO	NO	SIMPLE

### **TMRA\_Supervisor**

Esta tabla almacena la información específica de todos los empleados que sean supervisores que usen esta aplicación. Sus atributos son los siguientes:

TMRA_Supervisor	Clave	Tipo Clave	Null	Tipo dato	Multivaluado	Tipo atributo
id	SI	PK	NO	ENTERO	NO	SIMPLE
n_empleado_fk	SI	FK	NO	ENTERO	NO	SIMPLE
equipo_a_supervisar_fk	SI	FK	SI	ENTERO	NO	SIMPLE
esta_activo	NO	-	NO	BOOLEANO	NO	SIMPLE

### **TMRA\_Actualizacion\_Parte**

Esta tabla almacena la información de las actualizaciones que se han realizado en un parte de reparación durante la resolución de una incidencia (si procede), ya que esta puede durar varios días y puede pasar de un estado a otro, por lo que habrá que especificar qué se ha hecho en cada caso antes de cambiar su estado.

TMRA_Actualizacion_Parte	Clave	Tipo Clave	Null	Tipo dato	Multivaluado	Tipo atributo
id	SI	PK	NO	ENTERO	NO	SIMPLE
comentario	NO	-	NO	CADENA	NO	SIMPLE
fecha	NO	-	NO	FECHA	NO	SIMPLE
horas_trabajadas	NO	-	NO	DECIMAL	NO	SIMPLE
orden_trabajo_fk	SI	FK	NO	ENTERO	NO	SIMPLE
esta_activo	NO	-	NO	BOOLEANO	NO	SIMPLE

### **TMRA\_Tecnico\_Realiza\_Actualizacion**

Tabla intermedia entre la tabla de TMRA\_Tecnico y TMRA\_Actualizacion\_Parte. Sus atributos son los siguientes:

TMRA_Tecnico_Realiza_Actualizacion	Clave	Tipo Clave	Null	Tipo dato	Multivaluado	Tipo atributo
id	SI	PK	NO	ENTERO	NO	SIMPLE
id_tecnico_fk	NO	FK	NO	ENTERO	NO	SIMPLE
id_actualizacion_fk	NO	FK	NO	ENTERO	NO	SIMPLE

### **TMRA Parte:**

Tabla que almacena la información relativa a un parte de reparación. Sus atributos son los siguientes:

TMRA_Parte	Clave	Tipo Clave	Null	Tipo dato	Multivaluado	Tipo atributo
orden_de_trabajo	SI	PK	NO	ENTERO	NO	SIMPLE
criticidad	NO	-	NO	CADENA	NO	SIMPLE
estado	NO	-	NO	CADENA	NO	SIMPLE
fecha_inicio	NO	-	NO	FECHA	NO	SIMPLE
fecha_fin	NO	-	SI	FECHA	NO	SIMPLE
horas_duracion	NO	-	SI	DECIMAL	NO	SIMPLE
descripcion_averia	NO	-	SI	CADENA	NO	SIMPLE
foto_averia	NO	-	SI	ENTERO	NO	SIMPLE
descripcion_reparacion	NO	-	SI	CADENA	NO	SIMPLE
gastos_extra_reparacion	NO	-	SI	DECIMAL	NO	SIMPLE
precio	NO	-	NO	DECIMAL	NO	SIMPLE
firma_tecnico	NO	-	SI	ENTERO	NO	SIMPLE
id_empleado_hospital_fk	SI	FK	NO	ENTERO	NO	SIMPLE
id_tecnico_fk	SI	FK	NO	ENTERO	NO	SIMPLE
id_maquina_fk	SI	FK	NO	ENTERO	NO	SIMPLE
esta_activo	NO	-	NO	BOOLEANO	NO	SIMPLE

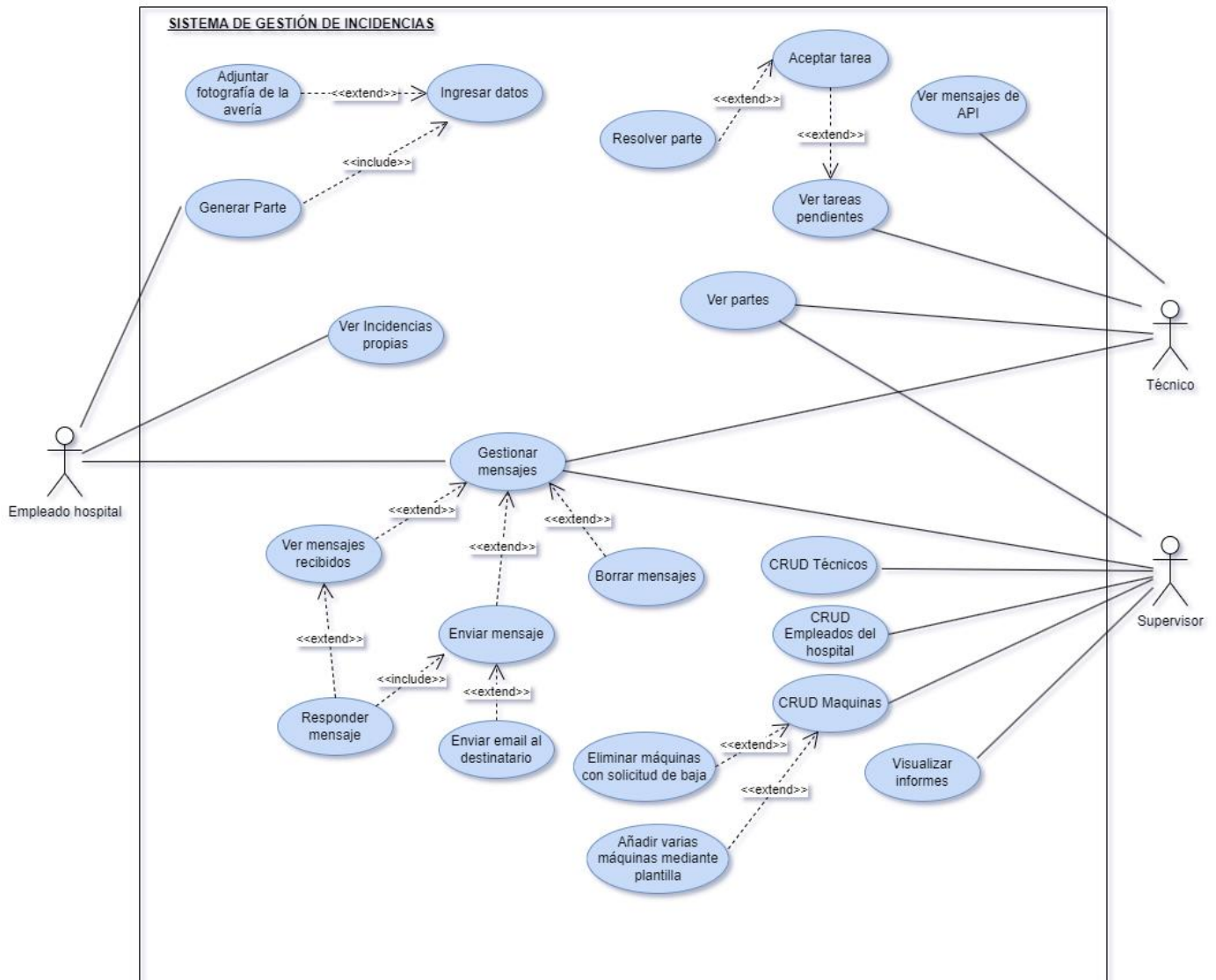
### **TMRA Maquina:**

Tabla que almacena la información relativa a las máquinas de electromedicina.

TMRA_Maquina	Clave	Tipo Clave	Null	Tipo dato	Multivaluado	Tipo atributo
id	SI	PK	NO	ENTERO	NO	SIMPLE
nombre	NO	-	NO	CADENA	NO	SIMPLE
estado	NO	-	NO	CADENA	NO	SIMPLE
marca	NO	-	NO	CADENA	NO	SIMPLE
modelo	NO	-	NO	CADENA	NO	SIMPLE
n_serie	NO	-	NO	CADENA	NO	SIMPLE
descripcion	NO	-	NO	CADENA	NO	SIMPLE
notas_adicionales	NO	-	NO	CADENA	NO	SIMPLE
foto	NO	-	NO	ENTERO	NO	SIMPLE
esta_activo	NO	-	NO	BOOLEANO	NO	SIMPLE

## 5.2. – Diagrama de casos de uso

A continuación, se muestra el diagrama de casos de uso de la aplicación:



### 5.3. - Diseño de las interfaces

A continuación, se muestra el diseño de las interfaces principales de la aplicación. No se incluye la interfaz de inicio de cada aplicación por economizar espacio en este documento, ya que las capturas de pantalla ocupan mucho y estas interfaces son simplemente informativas y no aportan nada relevante a mostrar.

#### 5.3.1. - Interfaces para los empleados del hospital

Dado que los empleados del hospital podrían querer fotografiar la máquina averiada mediante un smartphone o Tablet, se ha diseñado la aplicación para una mejor visualización en estos dispositivos, por lo que a continuación se mostrará el diseño de las interfaces tanto para un navegador web como para dispositivo móvil.

#### Ventanas principales de los menús

DISEÑO DE INTERFAZ PARA EL GRID DE INCIDENCIAS PARA NAVEGADOR WEB

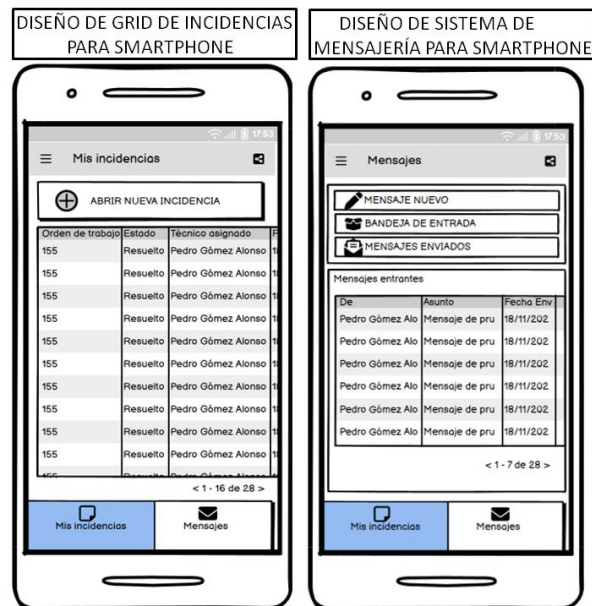
Orden de trabajo	Estado	Técnico asignado	Fecha de apertura	Fecha de cierre
155	Resuelto	Pedro Gómez Alonso	18/11/2023	18/11/2023
154	Resuelto	Pedro Gómez Alonso	17/11/2023	18/11/2023
153	Cerrado con problemas	Pedro Gómez Alonso	18/11/2023	18/11/2023
152	En curso	Pedro Gómez Alonso	17/11/2023	- Parte aún no cerrado -
151	Resuelto	Pedro Gómez Alonso	16/11/2023	16/11/2023
150	Resuelto	Pedro Gómez Alonso	10/11/2023	13/11/2023

1 - 16 of 30 >

DISEÑO DE INTERFAZ PARA EL SISTEMA DE MENSAJERIA PARA NAVEGADOR WEB

De	Asunto	Fecha Envío
Jose Luis Jimenez Olmeda - jjjimenez@tmra.io	Asunto 1	23/11/2023
Alfonso Garrido Rebollo - agarrido@tmra.io	Asunto 2	13/10/2023
Carlos Santano Del Casar - csantano@tmra.io	Asunto 3	04/09/2023
Cristina Garrido Parrillas - cgarrido@tmra.io	Asunto 4	23/08/2023

1 - 16 of 30 >



## Interfaz de nueva incidencia

Este es el diseño de la interfaz que se abre cuando el usuario pulsa en “Abrir nueva incidencia”:

DISEÑO DE LA INTERFAZ DE CREACIÓN DE INCIDENCIA PARA NAVEGADOR WEB

INICIO
MIS INCIDENCIAS
MENSAJES

**Abrir Incidencia**

**Nombre completo:** Pedro Gómez Alonso

**Teléfono:** +34000000000

**Puesto:** Enfermero

**Tfno. emergencia:** +34111111111

**Área:** Cirujía

**Correo electrónico:** pgomez@stemdo.io

**Rellenar datos de la avería**

Indique la máquina que presenta la avería, así como una descripción exacta que especifique lo que ocurre:

**Máquina o equipo a reportar: \***

... Seleccione el nombre de la máquina ...

**Marca: \***

... Seleccione el nombre de la máquina ...

**Modelo: \***

... Seleccione el nombre de la máquina ...

**Nº de serie: \***

... Seleccione el nombre de la máquina ...

**Criticidad: \***

... Seleccione el nombre de la máquina ...

**Subir foto:**

**UPLOAD** Adjunta una foto si lo desea

**Descripción de la avería: \***

Indica una descripción detallada de la avería para que el equipo de técnicos puedan resolverla.

0/250

CANCELAR
CONFIRMAR

DISEÑO DE INTERFAZ DE CREACIÓN DE INCIDENCIA  
PARA SMARTPHONE

**Abrir incidencia**

Nombre completo: Luis Dominguez Carrasco | Correo electrónico: ldomingue@tmra.io

Puesto: Enfermero | Puesto: Cirujía

Teléfono: +34688552211 | Tño. de emergencia: +3461125588

**Rellenar datos de la avería**

Indique la máquina que presenta la avería, así como una descripción exacta que especifique lo que ocurre.

Marca: -- Seleccione el nombre de la máquina a reportar --

Modelo: -- Seleccione el modelo --

Nº de serie: -- Seleccione el modelo --

Criticidad: -- Seleccione el modelo --

Adjuntar fotografía: CARGAR

Descripción de la avería: Indique una descripción detallada de la avería

CANCELAR | CONFIRMAR

### 5.3.2. - Interfaces para los técnicos de electromedicina

La aplicación de los técnicos está pensada para su uso en un navegador web que debería estar abierto en un ordenador de forma permanente en el taller. También podrían visualizarse en un smartphone, pero, ya que la aplicación está principalmente pensada para su uso en un PC y por economizar espacio en este documento, se presentan sólo los diseños para navegador.

Cabe destacar que el diseño del menú difiere de la aplicación de los empleados del hospital en el aspecto de que aparece en una barra lateral.

#### Grids de tareas y partes

Tanto la ventana de “Tareas” como la de “Partes” tienen un aspecto muy similar, ya que son grids (lo único que cambia es el contenido de los mismos), por lo que se muestra tan sólo el aspecto del grid de tareas para economizar espacio.

Tareas - Técnicos					
<a href="https://bdrf.applan.community/suite/sites/tecnicos-tmra/page/Tareas">https://bdrf.applan.community/suite/sites/tecnicos-tmra/page/Tareas</a>					
<div>Inicio</div> <div>Tareas</div> <div>Partes</div> <div>Mensajes internos</div> <div>Solicitudes externas</div>	Lista de tareas				
	ID	Nombre de la tarea	Estado	Técnico asignado	Fecha de creación
	7293	Gestión del parte	Aceptada	pgomez@stemda.io	02/12/2023
	7292	Gestión del parte	Aceptada	pgomez@stemda.io	15/09/2023
	7291	Gestión del parte	Aceptada	pgomez@stemda.io	20/08/2023

## Interfaz de mensajes internos

Esta interfaz es exactamente la misma que la de los mensajes de los empleados del hospital.

## Interfaz de mensajes externos

Esta interfaz es otro grid con un filtro que extrae comentarios ficticios de una API. No es realmente útil para la aplicación, tan sólo se creó para mostrar la capacidad de Appian para realizar integraciones con APIs. No se adjunta captura dado que, como se ha indicado, es un grid más.

## Interfaz de gestión del parte

Esta interfaz es la que se le muestra al técnico de electromedicina a la hora de pulsar sobre una tarea para resolver un parte de reparación.

### DISEÑO DE LA INTERFAZ DE GESTIÓN DEL PARTE

The mockup shows a web browser window titled 'Gestión del parte'. The address bar contains a URL. On the left is a sidebar with navigation links: Inicio, Tareas, Partes, Mensajes internos, and Solicitudes externas. The main content area is divided into sections:

- Datos del usuario que abrió la incidencia:** A table with fields: N° de empleado: 1, Teléfono: +34000000000, Nombre completo: Pedro Gómez Alonso, Tfno. emergencia: +34111111111, Puesto: Enfermero, Correo electrónico: pgomez@stemdo.io, Área: Cirujía.
- Datos de la máquina reportada:** A table with fields: Nombre: Desfibrilador, Marca: Zoll Medical, Modelo: AED Plus, N° de serie: AAAAAAAAAA3. Below it is a text field: Descripción avería: Esta es una descripción de la avería de prueba.
- Rellenar parte de reparación:**
  - Técnico asignado: Raúl Ramírez Fondón
  - Form fields: Estado\* (dropdown with 'Resuelto'), Fecha de apertura (04/12/2023), Fecha de cierre (04/12/2023), Horas de duración (3).
  - Descripción de la reparación: \* (text area with placeholder: Indique en detalle el resultado de la reparación, así como cualquier otro detalle que considere de interés).
  - \* Horas trabajadas: (text field with placeholder: Acepta decimales)
  - Costes asociados: (text field with placeholder: Coste total de los gastos derivados de la reparación (en euros))
  - Precio total: 20 €
  - \* Firma del responsable: (button: DRAW SIGNATURE)

At the bottom right is a button labeled 'CERRAR PARTE'.



### 5.3.3. - Interfaces para los supervisores

El diseño del menú de esta aplicación es igual al de los técnicos, con los botones situados en una barra lateral.

Por otra parte, esta aplicación tiene 3 CRUDS: uno para los empleados del hospital, otro para los técnicos y otro para las máquinas. Por lo tanto, se mostrará el diseño de sólo uno de ellos ya que el resto son similares.

#### Interfaz de un CRUD

DISEÑO DE LA INTERFAZ DEL CRUD DE LAS MÁQUINAS

Foto	Nombre	Marca	Modelo	Nº de serie	Estado	Acciones
<input checked="" type="checkbox"/>	Electrocardiógrafo	Nihon Kohden	EKG-2000	29YSP9PDKF	Operativo	
<input checked="" type="checkbox"/>	Desfibrilador	Medtronic	LIFEPAK 15	AAAAAAAAAA2	Fuera de servicio	
<input checked="" type="checkbox"/>	Monitor de signos vitales	Masimo	Rad-97 Pulse CO-Oximeter	AAAAAAAAAA3	En reparación	
<input checked="" type="checkbox"/>	Bomba de infusión	Baxter	SIGMA Spectrum Infusión Pump	AAAAAAAAAA4	Baja solicitada	

#### Interfaz de mensajes

Esta interfaz es exactamente la misma que la de los mensajes de los empleados del hospital, por lo que no se añade en este punto.

## Interfaces de los informes

Hay dos interfaces de informes. Ambas son muy similares, por lo que se muestra sólo una de ellas.

**DISEÑO DE LA INTERFAZ DEL INFORME DE MÁQUINAS**

**Tech Med Resolve**  
Estadísticas de las máquinas del hospital

**Estado de las máquinas**

**5** Operativos    **1** En reparación    **3** Fuera de servicio    **0** Dados de baja

**Máquinas por estado**  
Para ver la lista de máquinas operativas/en reparación/fuera de servicio/dados de baja, haz clic en la correspondiente sección del gráfico.

**N° de incidencias en las máquinas**

Incidentes reportados en las máquinas

ID	Tipo	Marca	Modelo	N° de incidencias rep.
1	Electrocardiógrafo	Nihon Kohden	EKG-2000	7
2	Desfibrilador	Medtronic	LIFEPAK 15	5
3	Monitor de signos vit	Masimo	Rad-97 Pulse CO-Oxi	5
4	Bomba de infusión	Baxter	SIGMA Spectrum Infu	2

Ver n° de incidencias por:  
Marca

### 5.3.4. - Interfaces comunes a las tres aplicaciones

Estas son interfaces que podrán visualizar varios roles de usuarios:

**DISEÑO DE LA INTERFAZ DE LA VISTA RESUMEN DE LOS PARTES**

**Orden de trabajo N° 155**

**Datos de apertura**

Fecha de inicio: 19/11/2023    Estado: Enfermero    Criticidad: Urgente  
Máquina afectada: Desfibrilador    Marca: Medtronic    Modelo: LIFEPAK 15  
N° de serie: AAAAAAAA4    Abierto por: Jose Luis Borreguero    Técnico responsable: Pedro Gómez Alonso

**Descripción de la avería:** Esta es una descripción de la avería de prueba

**Actualizaciones realizadas**

- Sin actualizaciones -

**Datos de cierre**

Fecha de cierre: 19/11/2023    Gastos asociados: 206    Precio total: 406  
Duración: 1 hora    Duración total: 1 hora    Cerrado por: Pedro Gómez Alonso

**Descripción de la reparación:** Esta es una descripción de reparación de prueba

**Firma del técnico:**

## Vista resumen del personal

Se añade tan solo la vista resumen de los técnicos, ya que la de los empleados del hospital es igual:

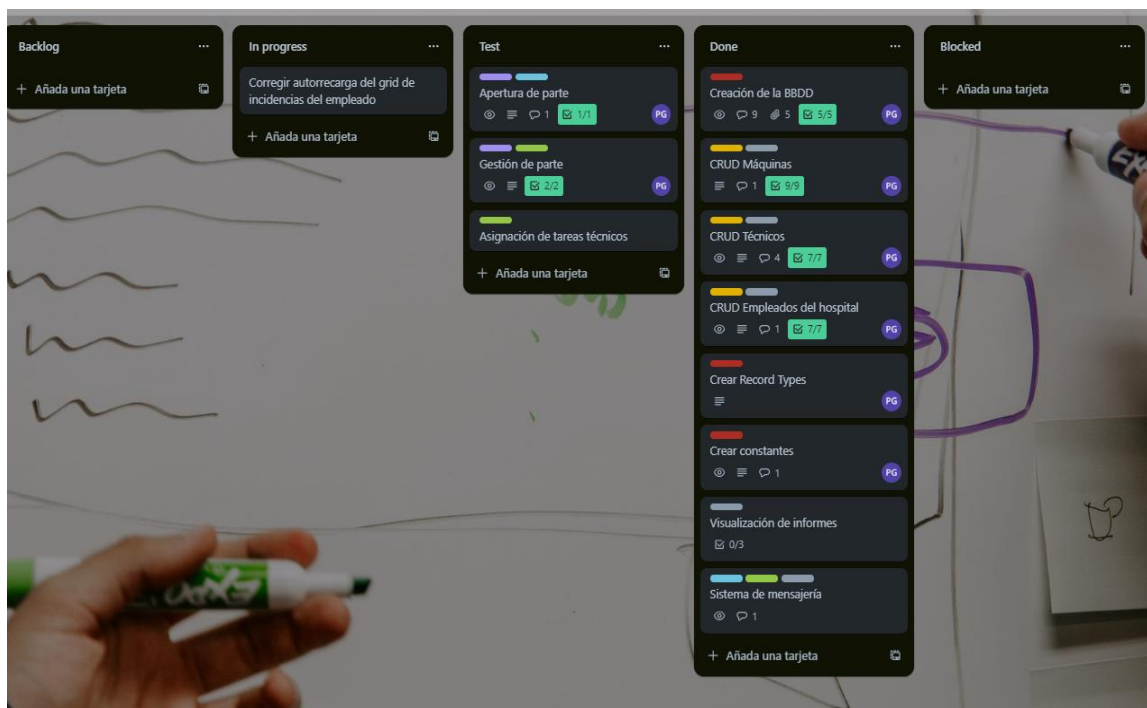
DISEÑO DE LA INTERFAZ DE LA VISTA RESUMEN DE LOS TÉCNICOS

The mockup shows a web browser window with the URL <https://ddt.ospin.comunidad/bulletin/tema-supervision/grupo-personal/pagina/tecnicos>. The page title is 'Técnicos - Supervisiones'. On the left is a sidebar menu with options: Inicio, Personal, Máquinas, Partes, Informes, and Mensajes. The main content area is titled 'Pedro Gómez Alonso' and contains a section 'Información del técnico' with the following details:

Nº de empleado: 27	Nombre: Pedro	Apellidos: Gómez Alonso	Username: pgomez
Teléfono: +00000000000	Teléfono de emergencia: +111111111	Email: pgomez@stemdo.io	Equipo: Técnicos de electromedicina
Categoría: Junior	Especialidad: Tecnología biométrica	Precio/hora: 30 €	Horario trabajo: Lunes a viernes - De 8:00 a 17:00

## 5.4. – Planificación del desarrollo:

Para planificar el trabajo realizado se ha utilizado la herramienta colaborativa **Trello**. Si bien es cierto que el “grupo” estaba compuesto por un solo integrante, Trello ha servido para organizar el trabajo y mejorar la efectividad, además de servir como cuaderno de bitácora durante todo el desarrollo.

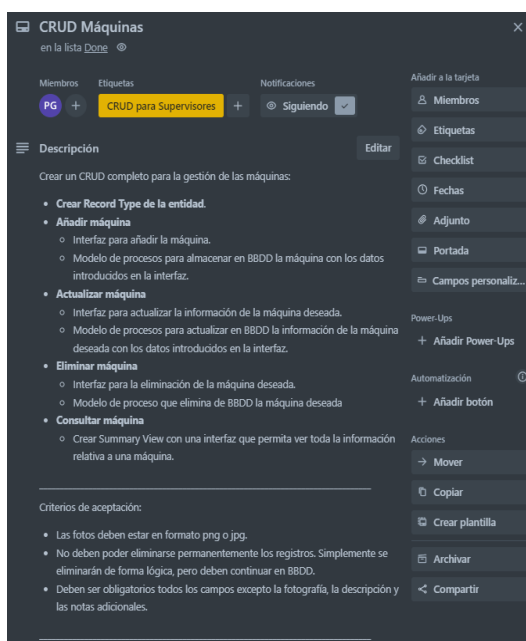


Gracias a esta herramienta, se ha podido ver en todo momento qué tareas quedaban por realizar, cuáles se habían completado pero que necesitaban ser testeadas, cuáles estaban completamente terminadas, etc. Además, si surgía cualquier tipo de idea para una nueva funcionalidad, se ha podido crear una nueva tarjeta de forma rápida y sencilla.

Como se puede apreciar en la imagen anterior, la estructura creada en Trello intentó simular la que se suele utilizar en una metodología Agile, teniendo las siguientes columnas:

- **Backlog**: En esta columna se iban añadiendo todas las tareas que había que realizar para la aplicación.
- **In progress**: Cada vez que se comenzaba a realizar una tarea, se movía la tarjeta correspondiente desde la columna “**Backlog**” a esta columna.
- **Test**: Cuando una tarea que se encontraba en “**In progress**” se finalizaba, se pasaba a la columna “**Test**”, ya que había que testear primero.
- **Done**: Una vez que una tarea se había finalizado, testado y comprobado que cumplía todos los criterios de aceptación, se pasaba a la columna “**Done**” para indicar que se había finalizado.
- **Blocked**: Esta columna se creó pero no se necesitó. Habría servido en caso de que una tarea se viera bloqueada al no poderse realizar hasta que se terminara otra. Al haberse desarrollado el proyecto por una única persona, esta casuística no podría haberse dado.

En la medida de lo posible, se han creado las tarjetas como si de historias de usuario se trataran (*ver Historias de usuario en el glosario*), de tal forma que incluyeran un título, una descripción y unos criterios de aceptación.



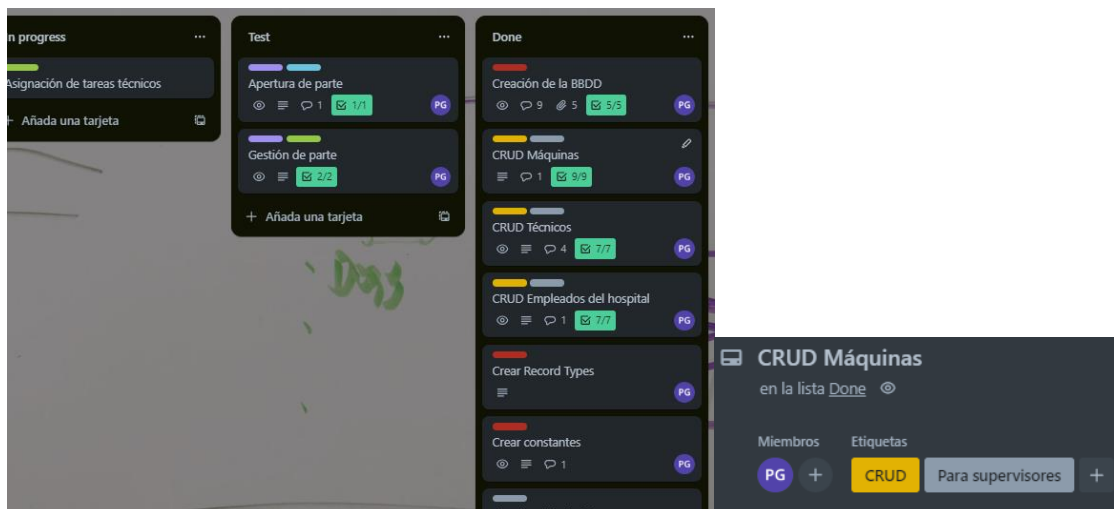
Además, para hacer un seguimiento más sencillo de cada tarjeta, se creó una checklist con las tareas específicas de la misma:

The image shows two checklists on a dark background. The first checklist is titled 'Checklist de interfaces para CRUD de Máquinas' and has a progress bar at 100%. It contains five items, all checked: 'Interfaz para añadir', 'Interfaz para actualizar', 'Interfaz para eliminar', and 'Interfaz para el Summary'. Below the items is a button 'Añada un elemento'. The second checklist is titled 'Checklist de PM para CRUD de Técnicos' and also has a progress bar at 100%. It contains three items, all checked: 'PM para añadir', 'PM para actualizar', and 'PM para eliminar'. Below the items is a button 'Añada un elemento'. Both checklists have buttons 'Ocultar los elementos marcados' and 'Eliminar' at the top right.

También se ha escrito un comentario en la tarjeta cada vez que se realizaba una acción relacionada con la misma. De tal forma que, de un simple vistazo, se podía ver cuándo se realizó dicha acción. Esto es especialmente útil si la aplicación falla en algún punto, ya que se puede saber rápidamente en qué momento falló, y facilita mucho la localización de la versión a la que habría que regresar en el sistema de control de versiones de Appian.

The image shows an activity log titled 'Actividad' with a 'Mostrar detalles' button. It contains four entries, each with a 'PG' icon, a name 'Pedro Gómez', a timestamp, and a comment. The first entry is 'Escriba un comentario...'. The second entry is 'Hecho'. The third entry is 'Falta añadir al modelo de proceso el smart services para la creación del usuario de appian.'. The fourth entry is 'Para hacer la prueba cambiar el primer apellido y la especialidad. De momento me cambia antes la especialidad y despues el apellido.'. Each entry has 'Editar' and 'Eliminar' buttons at the bottom.

Por último, se creó un etiquetado por colores para identificar el tipo de tarea y para qué estaba pensada a grandes rasgos.



Como se puede observar, si se está familiarizado con el código de colores propuesto, se puede discernir fácilmente el propósito asociado a cada tarea.

## 5.5. – Funcionalidad

A continuación, se va a explicar el desarrollo llevado a cabo en la aplicación.

Para evitar una gran extensión en la memoria, se ha explicado cómo utilizar la aplicación en un manual de usuario que se encuentra en el repositorio de github cuya URL se facilita en el apartado **Anexos**.

Además de esto, en ocasiones, durante la explicación de la funcionalidad que viene a continuación, se hará referencia al nombre de las expression rules desarrolladas y que son utilizadas en los distintos puntos de la aplicación. Si se desea revisar el código de las estas expression rules o de las interfaces, se han adjuntado también al repositorio antes mencionado dentro del directorio llamado “Expression Rules”. No se incluyen todas las capturas de pantalla aquí dado que son muchas.

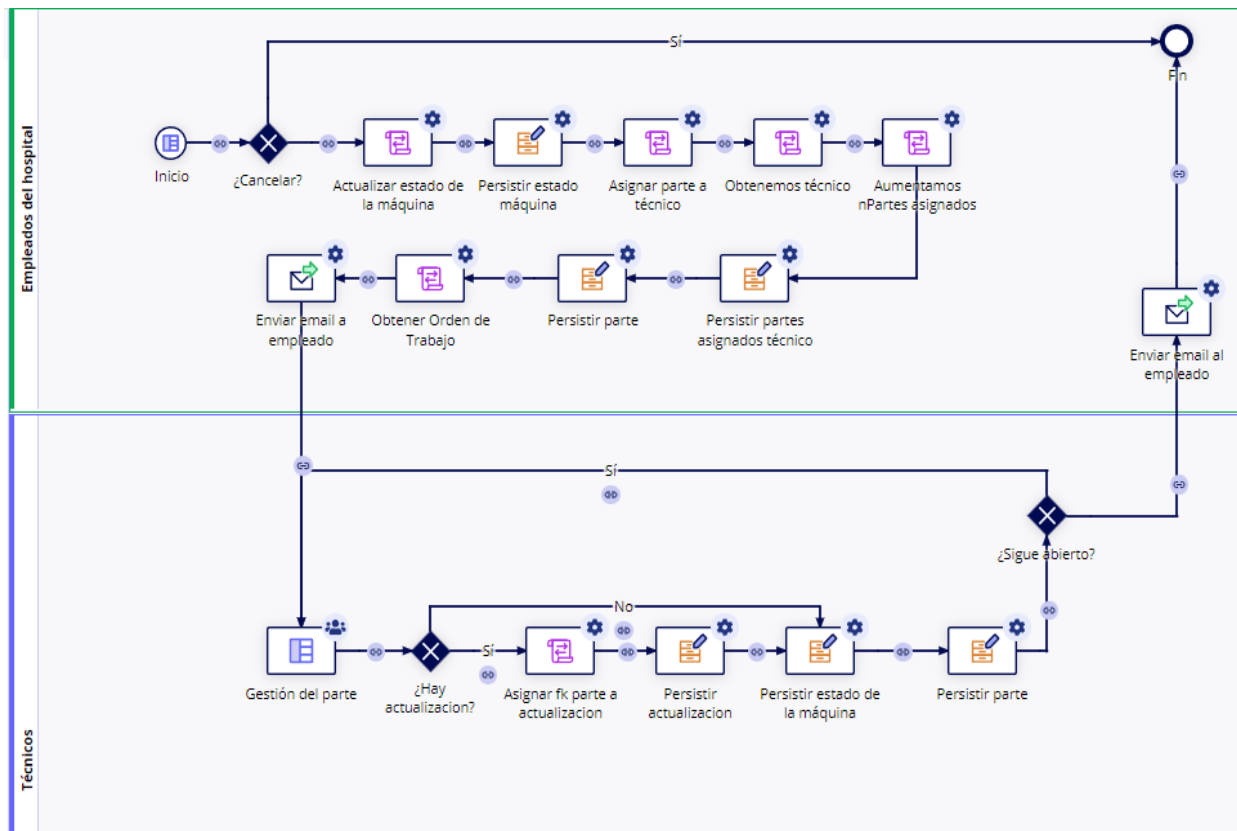
**Nota:** Al ser código propio de Appian, las Expression Rules e Interfaces, se visualizarán como texto plano. Para facilitar su visualización, se recomienda encarecidamente clonar el repositorio y abrir el código mediante la aplicación Notepad++ teniendo previamente instalado el siguiente plugin (requiere registro en Appian Community):

<https://community.appian.com/b/appmarket/posts/notepad-user-defined-language-for-appian>

### 5.5.1. - Gestión de las incidencias

Este es uno de los procesos principales de la aplicación.

Cuando un empleado del hospital pulsa en el botón de abrir una incidencia, se inicia el siguiente proceso (los nodos están explicados en el punto 2.6.3 del presente documento):



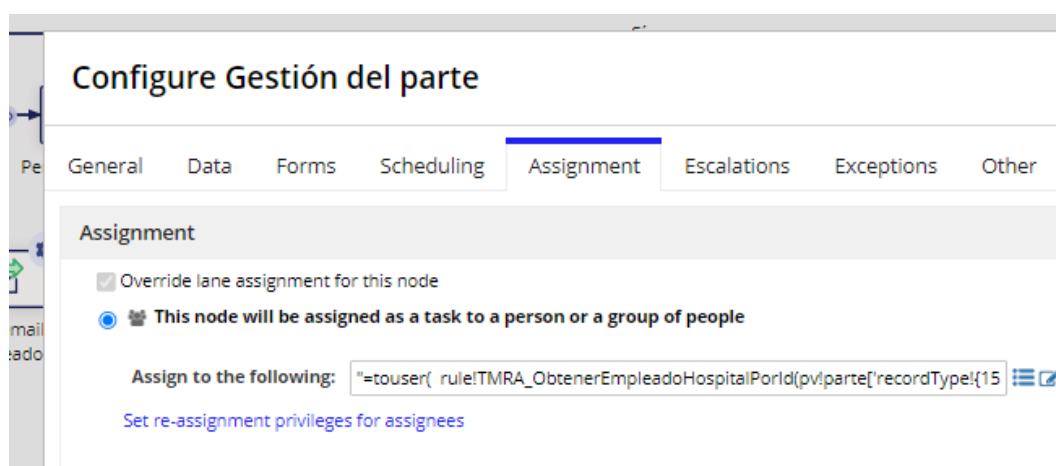
Como se puede ver, el proceso se inicia mediante una interfaz. Esta interfaz tiene el nombre de **TMRA\_GenerarParteReparacion** y muestra al usuario el formulario que solicita los datos de la máquina afectada.

Si el usuario pulsa en cancelar, el proceso finalizará, pero si pulsa en confirmar, tendrán lugar las siguientes acciones:

1. Mediante un script task, se actualiza el valor del campo “Estado” de la máquina a “Fuera de servicio”.
2. Mediante la expression rule “**TMRA\_ObtenerIdTecnicoParaAsignar**”, incluida en otro script task, se calcula el técnico al que se le asignará el parte en función del que tenga menos partes asignados.
3. Mediante otra script task, se aumenta en un punto el número de partes asignados que tiene el técnico.

4. Mediante otra script task, se obtiene la orden de trabajo (debe hacerse mediante una expresión, dado que en base de datos se autogenera la id y debemos obtenerla después).
5. Se envía un email al empleado que abrió la incidencia.
6. Todos los cambios realizados se persisten en BBDD.

Una vez llegado a este punto, el empleado del hospital ya no tendría que hacer nada más, ya que el siguiente nodo llamado “Gestión del parte” (que es un user input task), sólo será visible para los técnicos ya que, en su configuración, se indica que la asignación de dicha tarea es para el usuario con el id del técnico asignado que calculamos en nodos anteriores.



**Configure Gestión del parte**

General Data Forms Scheduling **Assignment** Escalations Exceptions Other

**Assignment**

☒ Override lane assignment for this node

☒ This node will be assigned as a task to a person or a group of people

Assign to the following:

[Set re-assignment privileges for assignees](#)

Una vez el técnico complete el formulario, tendrán lugar otra serie de acciones:

1. Si se ha realizado alguna actualización en el parte generado, es decir, si NO se ha cerrado el parte, se asignará la orden de trabajo como fk a dicha actualización y se persistirá en BBDD, en caso contrario, se saltará este paso.
2. Se persiste en BBDD la información sobre el parte de reparación introducida en el formulario rellenado por el técnico.
3. En el caso de que no se haya cerrado el parte, el flujo volverá a user input task asignada al técnico.
4. En el caso de que el parte se cierre, se enviará un email automático al empleado del hospital que abrió la incidencia indicándole que la incidencia se ha cerrado.

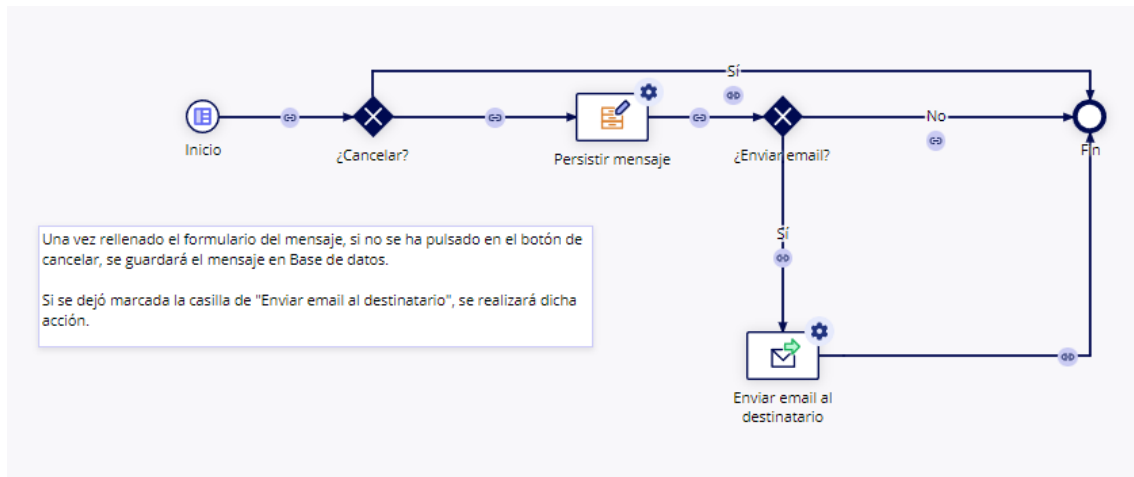
Como vemos, este proceso lo inicia un empleado del hospital desde su aplicación y lo continúa y finaliza un técnico desde la suya.



### 5.5.2. - Gestión de mensajes

#### - Mensaje nuevo

Cuando un usuario hace uso del sistema de mensajería y pulsa en el botón “Mensaje nuevo”, se ejecuta el siguiente proceso:

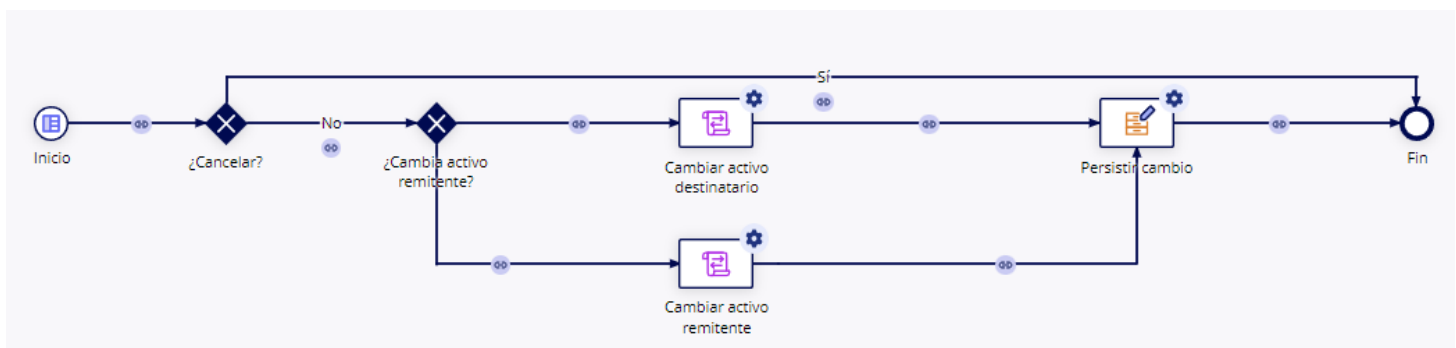


El proceso se inicia con una interfaz donde se le solicita al usuario el asunto y cuerpo del mensaje, así como el destinatario al que va dirigido.

Una vez rellenados dichos datos, si el usuario pulsa en cancelar, finalizará el proceso. Si pulsa en enviar, persistirá el mensaje en base de datos y, si se marcó la casilla de enviar un email al destinatario, se realizará dicho envío.

#### - Borrar mensaje

Por otra parte, cuando un usuario hace clic en el icono rojo de la basura dentro del grid de los mensajes, se ejecuta el siguiente proceso:



Este proceso empieza con un formulario de confirmación en el que se avisa al usuario de que se va a eliminar permanentemente el mensaje.

Si el usuario confirma la eliminación, se procede a un borrado lógico del mensaje de la siguiente manera:

- Si el mensaje se encontraba en la bandeja de entrada, se cambiará el valor del campo “activoDestinatario” del mensaje a false.
- Si se encontraba en la bandeja de salida, se cambiará el valor del campo “activoRemitente” del mensaje a false.

La interfaz está configurada para que sólo se muestren los mensajes con dichos estados como “true”.

Para ilustrarlo un poco mejor, se puede ver en la siguiente imagen cómo se indica que se muestre el grid de mensajes entrantes con el filtro añadido que especifica que únicamente se visualicen los que tengan el campo activoRemitente como true.

```

a!columnLayout(
  contents: {
    a!cardLayout(
      contents: {
        a!gridField_23r3(
          label: "Mensajes entrantes",
          labelPosition: "ABOVE",
          emptyGridMessage: "- No hay mensajes - ",
          data: a!recordData(
            recordType: TMRA Mensaje,
            filters: a!queryLogicalExpression(
              operator: "AND",
              filters: {
                a!queryFilter(
                  field: TMRA Mensaje.activoRemitente,
                  operator: "=",
                  value: true
                ),
                a!queryFilter(
                  field: TMRA Mensaje.nEmpleadoDestinatarioFK,
                  operator: "=",
                  value: if(
                    a!isNullOrEmpty(local!nEmpleadoLoggeado),
                    0,
                    local!nEmpleadoLoggeado
                  )
                )
              },
              ignoreFiltersWithEmptyValues: true
            )
          ),
          columns: {

```

## **- Responder mensaje**

Continuando con la gestión de los mensajes, si el usuario que ha iniciado sesión es el destinatario de un mensaje, cuando lo visualice, se le habilitará un botón de responder al mismo. Cuando pulse en él, se ejecutará el proceso visto anteriormente del envío del mensaje, pero con una diferencia:

El proceso tiene una variable llamada “esRespuesta” y otra llamada “mensajeAResponder”. Al pulsar en el botón de responder en la interfaz, se ejecutará el proceso pasándole por parámetro un true para la primera variable y el cuerpo del mensaje propiamente dicho para la segunda, como se puede ver en la siguiente imagen.

```
if(
  local!mensaje[ TMRA Mensaje.nEmpleadoRemitenteFk] = local!nEmpleadoLoggeado,
),
a!columnLayout(
  columns: {
    a!columnLayout(),
    a!columnLayout(
      width: "NARROW",
      contents: a!cardLayout(
        contents: {
          a!richTextDisplayField(
            value: {
              a!richTextItem(
                text: {
                  a!richTextIcon(icon: "reply"),
                  " RESPONDER"
                },
                style: { "STRONG" }
              )
            },
            align: "CENTER"
          )
        },
        link: a!startProcessLink(
          label: "Enviar mensaje",
          processModel: cons!TMRA_ENVIAR_MENSAJE,
          processParameters: {
            esRespuesta: true,
            mensajeAResponder: local!mensaje
          },
          bannerMessage: "Acción completada"
        ),
        height: "AUTO",
        style: "INFO",
        shape: "ROUNDED",
        marginBelow: "STANDARD",
        showShadow: local!bandejaDeEntrada
      )
    )
  }
)
```

De esta manera, tal y como está configurado, cuando se muestre la siguiente interfaz (la que solicita los datos para escribir el mensaje), no se solicitará al usuario que introduzca los datos del destinatario, ya que le llegarán por parámetro, y esa interfaz está configurada para que se oculten o muestren los componentes que la componen en función de estos valores.

```
a!sectionLayout(
  label: "Enviar mensaje",
  contents: {
    if(
      or(a!isNullOrEmpty(ri!esRespuesta), not(ri!esRespuesta)),
      a!columnLayout(
        columns: {
          a!columnLayout(
            contents: {
              a!dropdownField(
                label: "Rol que desempeña el destinatario:",
                labelPosition: "ABOVE",
                placeholder: "--- Seleccione el rol del empleado ---",
                choiceLabels: local!listaRolesUsuarios[ TMRA Tipo Empleado.nombre],
                choiceValues: local!listaRolesUsuarios,
                value: local!rolEmpleadoDestinatario,
                saveInto: {
                  local!rolEmpleadoDestinatario,
                  a!save(local!empleado,null)
                },
                searchDisplay: "ON",
                required: true,
                validations: {}
              )
            }
          )
        }
      )
    )
  }
)
```

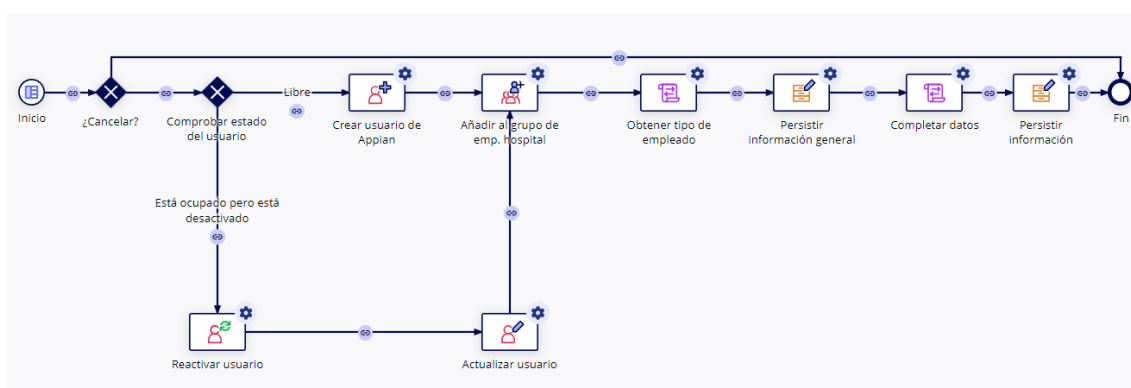
Si "esRespuesta" es falso o viene nulo o vacío mostrará la solicitud de datos del destinatario

## - CRUDs

La aplicación cuenta con 3 CRUDs completos: uno para los empleados del hospital, otro para los técnicos y otro para las máquinas. Se va a proceder a explicar tan sólo el de los empleados del hospital, ya que el resto son iguales. El CRUD de las máquinas es más simple, ya que cada proceso prescinde de todos los nodos relativos a los usuarios.


## - Crear empleado

Cuando un supervisor pulsa en el botón de añadir un nuevo empleado, se ejecutará el siguiente proceso:



El proceso comienza con el formulario de la interfaz “**TMRA\_CrearEmpleadoHospital**”, que solicita los datos del empleado al supervisor. Cuando se rellene el campo del username, comprobará si existe ya un usuario con ese username. Si está en uso, se le indicará al usuario y no permitirá escoger dicho username. A continuación, se muestra el fragmento de código de la interfaz que realiza esa comprobación:

```

    validations: {
      if(
        condition: rule!TMRA_ComprobarUsuario(r!recordEmpleado[ TMRA Empleado.username]) = cons!TMRA_ESTADOS_USUARIOS_APPIAN[3],
        valueIfTrue: "Nombre de usuario en uso.",
        valueIfFalse: ""
      )
    }
  
```

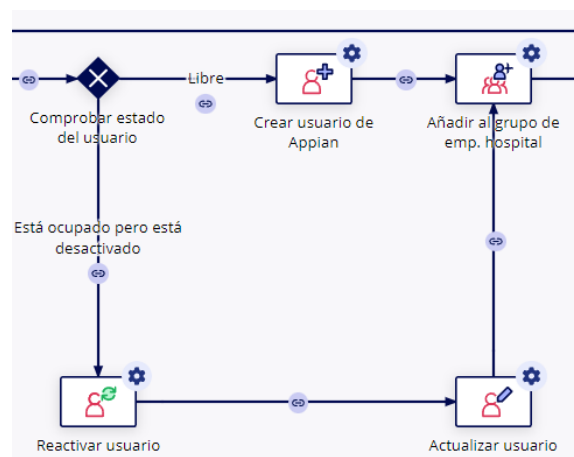
Como se puede apreciar, el código referencia a la Expression Rule “**TMRA\_ComprobarUsuario**” cuyo código muestro a continuación.

```

if(
  a!isNotNullOrEmpty(ri!username), /* Si el username no es nulo o viene vacío */
  if(
    not(isusername(ri!username)), /* Si el nombre de usuario no se ha usado. */
    cons!TMRA_ESTADOS_USUARIOS_APPIAN[1], /* Devuelve "libre" */
    if( /* Si existe, si el estado del usuario indica que está inactivo, devolverá "desactivado" */
      user(ri!username, "status") = 0,
      cons!TMRA_ESTADOS_USUARIOS_APPIAN[2],
      cons!TMRA_ESTADOS_USUARIOS_APPIAN[3] /* Si existe y no está inactivo, devolverá "ocupado" */
    )
  ),
  null, /* Si el username llega nulo o vacío nos devuelve un null */
)

```

Una vez se rellena el formulario (y si no se ha pulsado en cancelar), el proceso continuaría para la creación del nuevo usuario. No obstante, el username podría existir en Appian, pero estar desactivado (cabe recordar que los usuarios en Appian no se pueden eliminar), por lo que, si se intentara crear un usuario, daría error. Por eso, en el proceso se ideó este sistema:



Básicamente, lo que hace es que, si el username está libre y no existe, crea un usuario directamente y lo añade al grupo de empleados correspondiente. Sin embargo, si existe pero está desactivado, lo primero que hace es reactivar el usuario para, posteriormente, actualizar la información del mismo con la que escribió el supervisor en el formulario. Finalmente, se añadiría al grupo correspondiente y se crearía un registro en BBDD con la información del nuevo usuario.

De cara al supervisor no hay ninguna diferencia, ya que sólo verá que se ha creado un nuevo usuario.

### - **Actualizar empleado**

Cuando el supervisor desee editar la información de un empleado pulsando en el botón para tal efecto, se ejecutará este proceso:



Como se puede apreciar, es mucho más sencillo, ya que aquí se parte de la existencia previa de un empleado.

El proceso inicia con un formulario que solicita los nuevos datos del empleado. Si se pulsa en cancelar el proceso finaliza, de lo contrario, mediante la función `user()` almacenada en una **script task**, se obtiene el usuario con el username del empleado que se está actualizando y se almacena en una variable de proceso de tipo user:

```
touser(pv!recordEmpleado[  TMRA Empleado.username ] )
```

#### Configure Obtener usuario de Appian

General Data Forms Scheduling Assignment Exceptions Other

Inputs Outputs

Node Outputs Save node data to Process Variables for use elsewhere in the Process

+ New Custom Output ✖ Delete Custom Output

▼ Results  
There are no results to configure

▼ Custom Outputs  
touser(pv!recordEmpleado[recordType!TMRA Empleado.fields(2f088b8a-2ded-4b11-8000-000000000000)TMRA Empleado.username])

Obtenemos el usuario

Lo almacenamos en la variable "usuarioAppian"

Expression Properties

Expression: touser(pv!recordEmpleado[recordType!TMRA Empleado.fields(2f088b8a-2ded-4b11-8000-000000000000)TMRA Empleado.username])

Operator: is stored as

Target: usuarioAppian

Una vez se ha almacenado el usuario en la variable, esta se pasa como parámetro al nodo de actualizar el usuario para modificar su información.

Finalmente, con los nodos restantes (write records), persistimos los cambios en BBDD.

#### - Eliminar empleado

Cuando el supervisor pulsa en el botón para eliminar un empleado, se ejecuta el siguiente proceso:



Comienza con una ventana de confirmación, si no se pulsa en cancelar (lo que finalizaría el proceso), lo primero que haría el script task "Obtener empleado" es obtener y almacenar en dos variables de tipo record type, al empleado. Esto es porque, si se recuerda el modelo

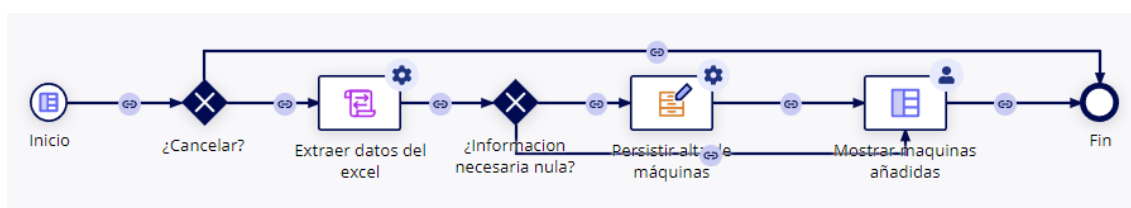
de datos, cada empleado tiene su información repartida en dos tablas: una de ellas es para su información genérica, y la otra para la información específica. Se necesita esta información para poder realizar el borrado en BBDD.

Una vez se han obtenido los record types del empleado, ahora se necesita obtener el usuario mediante el segundo script task, para posteriormente, desactivar el usuario con el siguiente nodo.

Finalmente, una vez que se ha desactivado, hacemos uso del último script task en el que, para realizar el borrado lógico de la información, modificamos el valor del campo “esta\_activo” del usuario en BBDD y persistimos la información.

### - Alta de varias máquinas mediante un fichero Excel

Los supervisores, opcionalmente tienen la posibilidad de dar de alta máquinas de forma masiva mediante la subida de una plantilla Excel. Cuando se pulsa en el botón para tal efecto se ejecuta este proceso:



Inicia con un formulario que permite descargar la plantilla para que se siga el mismo formato, y tras rellenarlo, permite su subida:

#### Carga de excel

Descarga la plantilla en el enlace que dejamos a continuación y rellénala. Una vez hecho esto, súbela cumplimentada. Esto dará de alta todas las máquinas de dicha plantilla en nuestro sistema.

La interfaz está configurada para que no se permita la subida de un archivo que no tenga extensión xlsx:

```

validations: {
  if(
    fv!files.extension <> "xlsx",
    "Tiene que ser un archivo excel",
    {}
  )
}
  
```

Una vez se sube, el fichero se almacenará en nuestro entorno de Appian, en un directorio creado previamente y referenciado mediante una constante. Además quedará almacenado en una rule input que será pasada al proceso y que es de tipo “Document”:

```

a!fileUploadField(
  labelPosition: "COLLAPSED",
  placeholder: "Carga tu archivo de excel",
  target: cons!TMRA PLANTILLA DE SUBIDA DE MAQUINAS CUMPLIMENTADAS,
  maxSelections: 1,
  value: ri!documentoExcel,
  saveInto: ri!documentoExcel,
  validations: {
    if(
      fv!files.extension <> "xlsx",
      "Tiene que ser un archivo excel",
      {}
    )
  }
)
}

```

Cuando el supervisor pulse en el botón de cargar datos, se ejecuta un script task que ejecuta la expression rule **“TMRA\_ExtraerDatosExcel()”** a la que le pasará por rule input la variable de tipo document que almacena el Excel. Esta expression rule extrae los datos del Excel pasado por rule input y crea un record type de máquinas por cada fila del mismo. Por lo tanto, devuelve una lista de record types que, mediante el script task, almacenamos en otra variable de proceso:

**Configure Extraer datos del excel**

General | **Data** | Forms | Scheduling | Assignment | Exceptions | Other

Inputs | **Outputs**

**Node Outputs** ⓘ Save node data to Process Variables for use elsewhere in the Process

+ New Custom Output ✖ Delete Custom Output

▼ **Results**  
There are no results to configure

▼ **Custom Outputs**  
1. rule!TMRA\_ExtraerDatosExcel( excel: pv!docu...

**Expression Properties**

Expression: rule!TMRA\_ExtraerDatosExcel( excel: pv!documentoExcel )

Operator: is stored as

Target: record

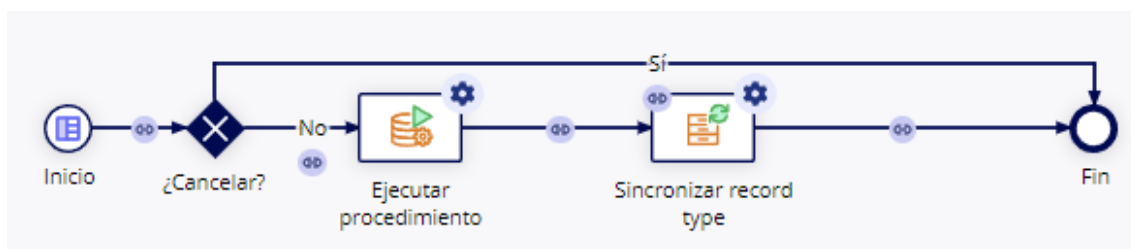
Después de esto, se realiza una comprobación. Si el record type no está vacío, persistirá la información en BBDD y mostrará al supervisor un grid con los registros añadidos. En caso contrario, mostrará este grid vacío, indicando que no se han añadido máquinas.



### - Ejecución de procedimiento almacenado para borrado de máquinas de baja

Los supervisores podrán eliminar de forma lógica todas las máquinas que tengan el estado con el valor del campo “Estado” como “Baja” pulsando un botón.

Cuando se pulsa dicho botón, se ejecuta el siguiente proceso:



Inicia con un formulario que pide confirmación, y si no se pulsa en cancelar, ejecutará un procedimiento almacenado creado en nuestra BBDD que modificará el campo esta\_activo a false en todas las máquinas cuyo estado sea “Baja”.

```

BEGIN
    UPDATE TMRA_Maquina SET esta_activo =
    false WHERE estado = 'Baja';
END
  
```

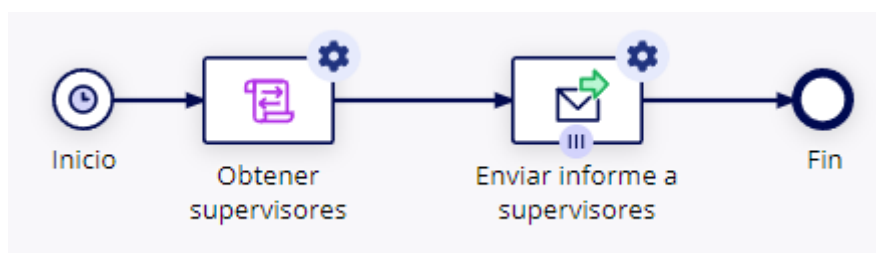
Finalmente, dado que el procedimiento almacenado tiene lugar en la BBDD, debe sincronizarse esta con nuestra aplicación, por lo que se utiliza el siguiente nodo para ello.

### - Envío de informe semanal (BATCH)

Una vez a la semana, los viernes a las 6 de la tarde concretamente, se envía un email a todos los supervisores con una tabla a modo de pequeño informe semanal del estado de las máquinas y partes. Este informe tiene el siguiente aspecto:

Informe Semanal - Estado de Incidencias y Equipos	
Buenas tardes Pedro Supervisor, a continuación se muestra el informe del estado de las incidencias y los equipos de la última semana:	
	Número
Incidencias Abiertas	31
Incidencias Resueltas	30
Incidencias en Curso	0
Incidencias en Suspensión	0
Incidencias Cerradas con Problemas	1
Máquinas Operativas	7
Máquinas en Reparación	0
Máquinas Fuera de Servicio	2
Máquinas Dadas de Baja	0

Este envío programado es el proceso configurado como un BATCH:



Como se puede apreciar, el nodo de inicio está configurado con un temporizador. Como ya se ha indicado, este se ejecuta todos los viernes a las 6 de la tarde. Una vez que lo hace, el primer script task obtiene una lista con todos los supervisores y la almacena en una variable del proceso llamada “supervisores”.

#### Configure Obtener supervisores

Esto es importante, ya que el siguiente nodo, que envía el email con el informe, está configurado como un **MNI**. MNI significa "Multiple Nodes Instance" (Instancia de Múltiples Nodos, en español). Configurar un nodo como MNI implica que dicho nodo puede ejecutarse varias veces en paralelo, creando instancias independientes de sí mismo. Esto es útil cuando se necesita manejar múltiples instancias de un mismo proceso o tarea al mismo tiempo.



En este caso, lo que se desea es que se ejecute este mismo nodo una vez por cada elemento en la lista de los supervisores (básicamente funcionaría de forma muy similar a un bucle for). Por lo tanto, en la configuración del mismo, lo indicamos de la siguiente manera:

## Configure Enviar informe a supervisores

General Setup Data Forms Scheduling Assignment Exceptions **Other**

### Multiple Instances

☒ **Automatically run multiple instances of this node**

☐ Run  instances of this node

☐ Run --No Number Variable found instances of this node

☒ Run one instance for each item in **supervisores**

☒ Spawn all ☐ Spawn new

☐ Run this many:

☒ **Run all instances at the same time**

Como ya se había obtenido una lista con todos los supervisores, y este nodo se ejecutará tantas veces como elementos hay en esa lista, tan solo hay que indicarle al nodo que el destinatario del email será supervisor de dicha lista que se corresponde con cada iteración:

## Configure Enviar informe a supervisores

General **Setup** Data Forms Scheduling Assignment Exceptions Other

### Email Configuration

\* From

\* To

[Add Cc](#) | [Add Bcc](#) | [Add Attachments](#)

Reply To

Subject

Priority

\*Required

### Message Body

☐ Enter text here ☒ Use a text or HTML template

\* Template

Runtime Template

### - Expression Rule para obtener el tipo de pantalla

Es conveniente recalcar que se creó una expresión rule para identificar el tipo de pantalla y así poder controlarlo en dicha interfaz y poder mostrar los componentes de una forma u otra en función del valor obtenido. El nombre de esta expression rule es “**TMRA\_ObtenerTipoPantalla**”:

```
a!localVariables(
  /*
  Almacenamos una lista de booleanos en la que mediante la funcion isPageWidth(),
  se comprobará si el tipo de ancho de pantalla se corresponde con los tipos de ancho de appian.
  Las comprobaciones se hacen en el mismo orden en el que están almacenados
  los valores posibles en la constante cons!TMRA_TIPOS_PANTALLA

  Como es imposible que se esté viendo en dos pantallas al mismo tiempo,
  sólo devolverá true el índice correspondiente al tipo de pantalla que
  se está utilizando.

  */
  local!listaPantallas: {
    a!isPageWidth("PHONE"),
    a!isPageWidth("TABLET_PORTRAIT"),
    a!isPageWidth("TABLET_LANDSCAPE"),
    a!isPageWidth("DESKTOP_NARROW"),
    a!isPageWidth("DESKTOP"),
    a!isPageWidth("DESKTOP_WIDE")
  },

  /*
  Iteramos la lista local!listaPantallas, y una vez que el elemento iterado sea true,
  devuelve el elemento de la constante de tipos de pantalla correspondiente
  al índice del elemento iterado. Por lo tanto, se devuelve
  */
  local!listaBooleanos: a!forEach(
    items: local!listaPantallas,
    expression: if(fv!item, cons!TMRA_TIPOS_PANTALLA[fv!index], {})
  ),

  /* Con index, obtenemos el elemento que deseemos de la lista que le pasemos por parámetro.*/
  index(
    local!listaBooleanos,
    1,
    {}
  )
)
```

Esta expresión rule es utilizada a lo largo de toda la aplicación de los empleados del hospital, que son los que realmente necesitarían usarla mediante la app.

### - Integración con API

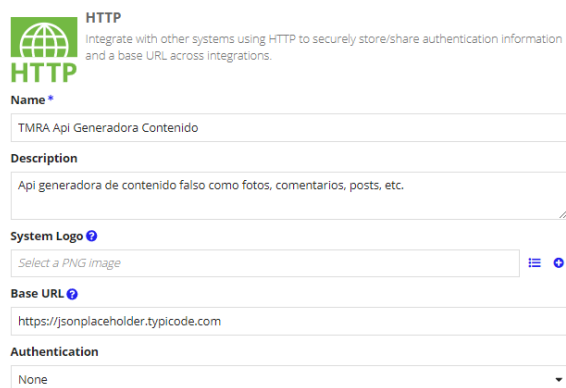
Dado que no se encontró ninguna API gratuita relevante para la aplicación y que no había tiempo, se optó por elegir una API que generaba comentarios ficticios. Aunque realmente no es útil para la aplicación, se añadió para la muestra de las posibilidades de Appian.

La URL de la API es la siguiente

<https://jsonplaceholder.typicode.com>

Para realizar la integración, primero hay que crear un objeto llamado Connected System y pasarle la URL de la API.

### Connected System Properties



**HTTP**  
Integrate with other systems using HTTP to securely store/share authentication information and a base URL across integrations.

**Name \***  
TMRA Api Generadora Contenido

**Description**  
Api generadora de contenido falso como fotos, comentarios, posts, etc.

**System Logo ?**  
Select a PNG image

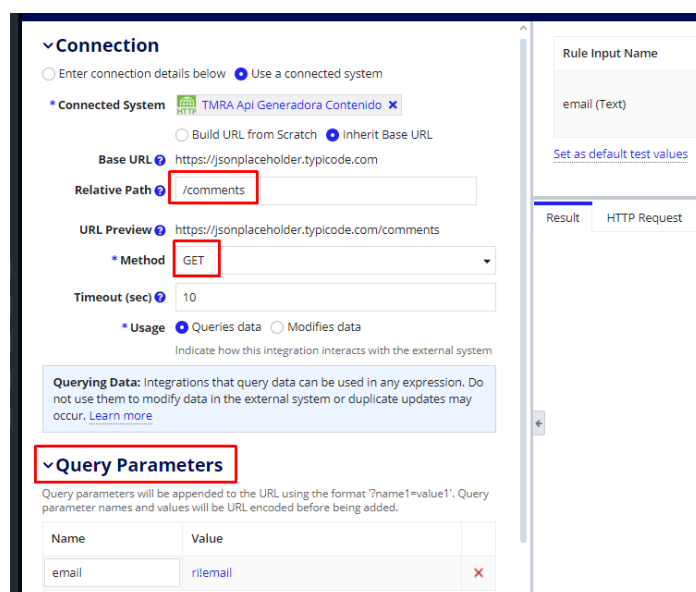
**Base URL ?**  
https://jsonplaceholder.typicode.com

**Authentication**  
None

Posteriormente hay que crear un objeto llamado Integration donde se configurará la integración, propiamente dicha. Habría que crear un objeto de este tipo por cada verbo HTTP que se desee utilizar.


En el caso del proyecto, tan solo se ha necesitado una, ya que sólo se han obtenido los comentarios de la API.

Una vez creado, se configura de forma muy sencilla indicando qué ruta relativa se necesita, qué verbo HTTP se va a utilizar, qué query params le vamos a pasar, e incluso cómo se desea obtener el resultado.



**Connection**

☐ Enter connection details below ☒ Use a connected system

**\* Connected System**  TMRA Api Generadora Contenido ✕

☐ Build URL from Scratch ☒ Inherit Base URL

**Base URL ?** https://jsonplaceholder.typicode.com

**Relative Path ?** /comments

**URL Preview ?** https://jsonplaceholder.typicode.com/comments

**\* Method** GET

**Timeout (sec) ?** 10

**\* Usage** ☒ Queries data ☐ Modifies data  
Indicate how this integration interacts with the external system

**Querying Data:** Integrations that query data can be used in any expression. Do not use them to modify data in the external system or duplicate updates may occur. [Learn more](#)

**Query Parameters**

Query parameters will be appended to the URL using the format '?name1=value1'. Query parameter names and values will be URL encoded before being added.

Name	Value
email	r@email

Una vez creada, podría ser llamada desde un process model, pero en el caso de este proyecto, ha sido llamada desde la interfaz “TMRA\_ComentariosFicticios”, que, como se puede apreciar en la siguiente imagen se hace mediante el prefijo “rule!”:

```
a!localVariables(
  local!comentarios: rule!TMRA_obtenerComentarios(null),
  local!email : "",
)

{
  a!sectionLayout(
    label: "Mensajes externos",
    contents: {
      a!columnLayout(
        columns: {
          a!columnLayout(
            contents: {
              a!dropdownField(
                label: "Email:",
                labelPosition: "ABOVE",
                placeholder: "--- Selecciona el email ---",
                choiceLabels: local!comentarios.result.body.email,
                choiceValues: local!comentarios.result.body.email,
                value: local!email,
                saveInto: local!email,
                searchDisplay: "AUTO",
                validations: {}
              )
            }
          )
        }
      )
    }
  )
}
```

## 5. – Conclusiones y mejoras del proyecto

Como conclusión final, cabe destacar que en menos de dos meses se ha desarrollado una aplicación completamente funcional y todo a cargo de una sola persona. Esto demuestra el potencial y la velocidad de desarrollo que se adquiere con Appian. Un equipo de desarrollo completo podría desarrollar aplicaciones bastante más complejas en el mismo tiempo (o menos).

Por otra parte, hay que recalcar que Appian es una tecnología que, aunque es aún joven, es líder en el sector, pero poco conocida por la gente que se inicia en el mundo de la programación y es más que probable que no haga sino crecer más y más.

Se ha querido explicar mejor el funcionamiento de Appian y no se ha podido hacerlo en su totalidad ni en profundidad debido a la limitación de la longitud de este documento.

Con respecto a las posibles mejoras del proyecto, habría sido deseable añadir procesos robóticos (RPA) o haber hecho uso de portales para mostrar otro de los puntos fuertes de Appian, por lo que el siguiente paso sería ese.

También se tenía prevista la creación de una API para su conexión con la aplicación, no obstante, por falta de tiempo ha sido imposible, por lo que también se tiene pensada su realización próximamente.

## 6. – **Bibliografía**

Toda la información utilizada en el presente proyecto ha sido extraída de la documentación oficial de Appian y del curso de certificación “Appian Certified Associate Developer” de Appian Community.

- **Documentación de Appian**: <https://docs.appian.com/suite/help/23.4/>
- **Appian Community**: <https://community.appian.com>

## 7. – **Anexos**

Todo el código de las interfaces y expression rules, así como los scripts de BBDD, código html de las plantillas de correo y demás información que se ha considerado de interés, se encuentra en el repositorio de github cuyo enlace se facilita a continuación:

<https://github.com/pedrogomezdev/TechMedResolve>