UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo e implementación una aplicación de escritorio para la contabilización y gestión de tareas o alertas y la mejora del rendimiento en empresas

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Martínez Lerín, Miguel

Tutor/a: Ruiz Font, Leonor

CURSO ACADÉMICO: 2022/2023

Miguel Martínez Lerín
ETSINF UPV

Resumen

Hoy en día todavía podemos encontrar muchas empresas, sobre todo en España, que siguen utilizando cantidades enormes de papel para contabilizar las tareas, peticiones o alertas que reciben en su actividad diaria. Esto nos presenta un elevado número de potenciales clientes que podrían actualizarse con las nuevas tecnologías de hoy en día.

En cualquier lugar disponemos de acceso a un ordenador por su bajo coste y por las necesidades actuales, pero muchas empresas siguen dejando constancia de sus datos por escrito, con los problemas que esto conlleva, como la dificultad luego para encontrar la información, el guardado y mantenimiento de esas hojas ya que con los años pasa a ser complicado su lectura, además de la tala de árboles necesaria para todo ese papel. Esto se puede solucionar con una aplicación para gestionar toda esta información, que nos permita conservar la información y realizar búsquedas y filtros.

Además, esto nos permitiría integrar esta aplicación con envíos de correos automáticos según necesidades, exportar/importar datos desde Excel o comunicaciones con otras aplicaciones como por ejemplo realizar envíos a la agencia tributaria, entre otras posibles funciones.

Todo esto nos abre un camino con enormes posibilidades que muchas empresas tienen en común, por lo que se puede hacer una aplicación que sea funcional para muchas de ellas, con diferentes módulos según las necesidades de cada una.

Palabras clave: Aplicación de escritorio, Base de datos, Mejora rendimiento, Gestión Tareas, Gestión Alertas, ERP.

Abstract

Today we can still find many companies, especially in Spain, that continue to use huge amounts of paper to record the tasks, requests or alerts they receive in their daily activity. This presents us with a large number of potential clients that could be updated with today's new technologies.

Anywhere we have access to a computer due to its low cost and current needs, but many companies continue to record their data in writing, with the problems that this entails, such as the difficulty later to find the information, save it and maintenance of these leaves as over the years it becomes difficult to read them, in addition to the felling of trees necessary for all that paper. This can be solved with an application to manage all this information, which allows us to keep the information and perform searches and filters.

In addition, this would allow us to integrate this application with automatic mailings according to needs, export / import data from Excel or communications with other applications such as sending to the tax agency, among other possible functions.

All this opens up a path with enormous possibilities that many companies have in common, so an application can be made that is functional for many of them, with different modules according to the needs of each one.

Keywords : Desktop application; Database; Improves performance; Task Management; Alerts management; ERP

Tabla de contenidos

Índice de ilustraciones

1. Introducción

1.1 Motivación personal

Me decante por la carrera de informática porque aparte de que me gustaban mucho los ordenadores, siempre me ha apasionado hacer cosas largas o repetitivas de forma automatizada, aunque al final me lleve más tiempo preparar el proceso que realizar la propia tarea.

Hace unos años, cuando mi pareja trabajaba para las oficinas de Metrovalencia fui a visitarla un día trabajando y vi la cantidad de papel que gastaban para documentar las tareas a realizar, informes sobre el trabajo realizado, datos sobre las llamadas, reclamaciones y problemas que tenían. También perdían varios minutos entre tareas escribiendo todos los datos en papal y luego mucho más tiempo al final de la jornada agrupando todos los datos de forma manual.

Esto me llevo a hacer mi primera aplicación ERP, software de gestión para empresas, era una aplicación de escritorio bastante básica donde en uno de los ordenadores había una base de datos y el resto de los equipos se conectaban a él. Fue un éxito y resultó en un descenso en el gasto de papel y un aumento en la productividad.

1.2 Motivación profesional

La mejor forma de aprender en el ámbito tecnológico es realizar algo que te apasione y puedas ver resultados. He realizado muchos proyectos que tengan más o menos utilidad solo por aprender nuevas tecnologías y no quedarme atrás.

Después de varios trabajos de desarrollador de software en empresas que proporcionan una aplicación a otras empresas, me di cuenta de que el punto en común sobre todo en España es que son aplicaciones muy grandes y con muchas funcionalidades pero que están orientadas a un entorno concreto, ya sea móvil, Windows, Apple, etc. Esto nos plantea limitaciones en temas de mantenibilidad, escalabilidad, portabilidad y modularidad entre otras.

Esto me llevó a cambiar de empresa buscando nuevas tecnologías ya que por desgracia no se planteaban la migración de aplicaciones a un entorno tecnológico más

avanzado. En ese punto me di cuenta lo atrasado que estaba respecto al mundo tecnológico actual.

Realizar un proyecto que me pueda ayudar a emplear nuevas tecnologías en el trabajo, desarrollando sobre un tema que me apasiona y que tiene posibilidades de crear mi propia empresa o poder utilizar este propio proyecto en mi empresa actual como servicio ERP para futuros clientes.

Por eso me decante en querer realizar una aplicación basada en pequeños módulos que sean más fáciles de mantener, que se pueda utilizar en cualquier tipo de dispositivo, que permita escalarla para aceptar el número de usuarios que se requiera y se pueda ceñir a unos estándares actuales tecnológicos.

1.3 Objetivos

- Utilizar las tecnologías más recientes para suplir las necesidades de las empresas actuales.
- Agilizar y facilitar el trabajo de gestión realizar por los usuarios.
- Eliminar el máximo posible uso de papel en la gestión y documentación de las empresas.
- Generar una base para una aplicación de gestión de tareas y alertas.
- Cumplir todas las necesidades de arquitectura que suelen requerir las aplicaciones multiplataforma en la nube.
- Hay que asegurar que la aplicación es fácil de mantener, no contiene duplicidades y es modular y distribuida.
- Poder ser utilizado para crear una empresa emergente por cuenta propia o que una empresa de software se interese en comercializarlo.

1.4 Impacto Esperado

El uso de una aplicación de gestión facilita el día a día del trabajador en la mayoría de las empresas que requieran cualquier tipo de control en el trabajo.

Evitar tener que usar tinta y papel para documentar y llevar el conteo de las tareas realizadas, ayudando a la disminución de la tala de árboles para favorecer los ODS (Objetivos de Desarrollo Sostenible) sobre la vida de ecosistemas terrestres.

Hacer las tareas repetitivas más sencillas automatizándolas, control del trabajo para evitar fallos por parte del usuario y generar informes automáticamente para favorecer el crecimiento industrial acogido en los ODS.

También utilizar este tipo de aplicaciones, favorece que una persona con algún tipo de diversidad funcional pueda realizar labores administrativas que se serían mucho más complicadas o imposible utilizando otros sistemas, lo que ayuda al ODS de reducir las desigualdades.

1.5 Metodología

Preparación

Para garantizar el cumplimiento de los objetivos, se definirán inicialmente unas tareas a realizar con un orden de prioridades y una estimación del tiempo requerido para implementarlas utilizando una aplicación tipo “Jira” o “Trello”.

Para poder estimar correctamente la duración, se realizará un estudio previo sobre las tecnologías y arquitectura a utilizar de forma que dichas estimaciones sean lo más certeras posible.

Elaboración

La implementación de las tareas se realizará en base al orden de prioridades establecido en la creación de estas, dejando para el final las menos importantes para que en caso de superar el tiempo de desarrollo, se pueda mostrar un producto final con varias funcionalidades.

En caso contrario, si se realizasen las tareas antes del tiempo esperado y sobrase tiempo de desarrollo, se utilizaría el tiempo restante en realizar mejoras visuales en la aplicación para que sea más atractiva para el usuario y la creación de más tests automáticos que aseguren la robustez de nuestro sistema.

Evaluación

Se realizarán pruebas a distintos usuarios durante y después del desarrollo para entender diferentes puntos de vista e intentar suplir necesidades o carencias que encuentren en la aplicación.

Uno de los objetivos principales es la interconexión entre los diferentes servicios implementados en diferentes tecnologías, entornos y alojamientos, el no cumplir dicho objetivo sería un proyecto totalmente fallido. La centralización de servicios en un solo entorno para poder gestionar el estado de cada uno de los servicios sin necesidad de conectar a cada uno de ellos también sería un objetivo muy importante para poder considerar este un buen proyecto.

1.6 Estructura

Próximos capítulos:

- Estado del arte:

Nos centraremos en la situación de la tecnología que se utiliza en las empresas actualmente en el ámbito informático, viendo las carencias y necesidades de este. Teniendo en cuenta también que posibles soluciones tenemos para suplirlas.

- Análisis del problema:

Necesidades del proyecto en base a los requerimientos del entorno. Las posibilidades para resolver los problemas de funcionalidad y escalabilidad eligiendo la más indicada y el presupuesto requerido para implementarla.

- Diseño de la solución

Requisitos del proyecto, arquitectura a implementar y las tecnologías a utilizar para cumplirlos.

- Desarrollo:

El punto principal donde comentaremos todos los problemas encontrados, las posibilidades para resolverlos y que solución se ha tomado en caso de haber podido resolver el problema.

- Implantación

Todos los requisitos necesarios para el despliegue del proyecto, tanto en las diferentes nubes/servidores utilizados, como los locales para el correcto funcionamiento del sistema.

- Pruebas

Un análisis a las pruebas de usuarios realizadas, pruebas automáticas en el propio sistema para asegurar el correcto resultado de las funcionalidades básicas.

- Conclusiones

Comentarios sobre la relación del proyecto con la carrera, que se han aprendido en su realización y que errores hemos encontrado durante el desarrollo del mismo.

- Trabajos futuros

El detalle de las tareas pendientes por realizar, los siguientes pasos y cambios que se realizarían en el proyecto que no han sido incluidos en este primer desarrollo.

- Referencias

Relación a las bases utilizadas para el desarrollo del proyecto, tanto para su implementación, los conceptos aprendidos para la preparación o la propia redacción de este documento.

2. Estado del arte

2.1 Estado actual

Cualquier empresa de hoy en día, sin importar su tamaño y sector de actividad, se organiza en secciones. Estos realizan todo tipo de tareas que varían en complejidad y que requieren una serie de recursos, seguir un orden cronológico y estar delimitadas por una fecha límite. Las herramientas de gestión de tareas, a pesar de su antigüedad, hoy en día siguen siendo más que necesarias.

El papel es un material versátil que se ha estado utilizando durante siglos [1], pero también cabe decir que, esto supone un gran daño ambiental y un fuerte impacto de los residuos en el medio ambiente. La pandemia de la COVID ha hecho plantearse a muchas empresas la digitalización de muchos de los trámites que se ejecutan y, ya no sólo se reduce el uso del papel, sino que, permite a los empleados poder trabajar desde cualquier parte del mundo sin necesidad de acudir a la oficina.

Así pues, es conveniente destacar que los programas de gestión de tareas se han ido abriendo camino en muchas de las empresas actuales. Ya sea un ERP para gestionar los activos de la empresa como un CRM para establecer un control de los clientes. Un 82% de las Pymes españolas tiene software ERP instalado [2]. Se ha pasado del uso tedioso de Excel, notas alrededor de una mesa a un gestor de tareas, mucho más fácil de utilizar, entender y además más visual.

Actualmente en el mercado, disponemos de múltiples gestores de tareas como:

- Asana

- Microsoft To Do

- Redmine

Por tanto, la gestión de tareas es importante ya que supervisa las tareas de un proyecto determinado, a través de sus etapas, de principio a fin [3]. La elección de un gestor de tareas para cualquier empresa es una decisión vital para el desarrollo de una empresa ya que el éxito o el fracaso de esta se decide con el trabajo en equipo, y la gestión de tareas es, sin duda, la mejor metodología para mejorarlo.

3. Análisis del problema

3.1 Análisis general

Para realizar un proyecto funcional que sea atractivo para el público se deben cumplir unos requerimientos.

Especificación de requisitos:

- El sistema debe presentar una alta usabilidad, el tiempo de aprendizaje del uso básico de la aplicación debe ser corto.
- El sistema debe ser fiable, la tolerancia a fallos ya sea por parte o por completitud de sus componentes no debe causar estados erróneos en el sistema.
- El sistema debe ser escalable, debe poder ser utilizado por grandes empresas con varias sedes en diferentes países o regiones o por pequeñas y medianas empresas.
- El sistema debe ser eficiente, debe aprovechar los recursos en base a la carga de trabajo o usuarios.
- El sistema debe permitir ser integrable con otras aplicaciones.
- El sistema debe poder ser utilizado como un Software-as-a-Service (SaaS) o como una aplicación particular para empresa.
- El sistema debe poder ser utilizado tanto en entorno “Cloud” como “On premises”.
- El sistema debe enviar informes diarios a los usuarios con las tareas importantes del día.
- Todos los componentes del sistema deben centralizar todos los datos de uso de la aplicación en un mismo registro.

Casos de uso:

- El usuario de la aplicación debe poder crear nuevas tareas con opciones disponibles.
- El usuario debe poder asignar tareas a otros usuarios, a él mismo o desasignarlas.
- El usuario debe poder marcar como resuelta o desmarcarla como tal.
- El usuario debe poder ver todas las tareas existentes en el sistema.
- El usuario debe poder filtrar las tareas por las diferentes propiedades.
- Un usuario administrador debe ser capaz de crear usuarios y categorías
- El usuario debe ver en la pantalla de inicio una vista personalizada de las tareas importantes.
- El usuario debe recibir un informe diario en su correo con las tareas importantes.

3.2 Análisis de seguridad

Cualquier aplicación moderna debe cumplir con unos requisitos de seguridad, ya sea para su uso a través de internet o en las propias instalaciones de una empresa:

- Identificación de usuarios con un usuario/contraseña o uso de AD (Active Directory).
- Definición de usuarios, roles y permisos en todos los niveles de la aplicación, se debe realizar la comprobación tanto en la aplicación de usuario como en la parte servidora.
- No permitir acceder a partes de la aplicación o realizar ciertas acciones a usuarios según sus permisos.
- Se debe utilizar un token para evitar la suplantación de identidad de los usuarios.

Estos requisitos no serán incluidos en este proyecto para acotar el desarrollo de la aplicación y no superar el tiempo límite de desarrollo.

3.3 Identificación y análisis de soluciones posibles

Existen diferentes maneras de poder solucionar los requisitos de proyecto, los principales, aunque no únicos, son los siguientes:

- Desarrollo de una aplicación de escritorio Windows donde la parte lógica y visual sería una aplicación implementada en WPF o Windows form que se conectaría a una base de datos SQL.
    - Pros:
        - Fácil desarrollo.
        - Tecnologías utilizadas durante el grado por lo que el tiempo destinado a la aplicación sería prácticamente todo desarrollo.
        - Si solo hay un usuario se puede instalar la base de datos y la aplicación en un solo ordenador.
        - En caso de tener varios usuarios se puede alojar la base de datos en un equipo servidor en la red de trabajo de los equipos cliente.
        - Requiere un coste bajo de implantación.
    - Contras:
        - Dificultades para utilizar la aplicación desde fuera de la red de la oficina, en caso de teletrabajo requiere una conexión VPN que ralentiza el tiempo de respuesta de la aplicación.
        - Dificultades para escalar la aplicación en caso de tener diferentes ubicaciones desde las que se trabaja.
        - En caso de que la carga de trabajo o número de usuarios sea muy alta afectará mucho al rendimiento de la aplicación incluso llegando a ser insostenible.
        - En caso de querer modernizar la aplicación en un futuro (cambiar de lenguaje/tecnología) tendría que invertirse mucho tiempo en realizar la aplicación por completo desde cero siendo una inversión que muchas empresas no se pueden permitir.
        - La aplicación solo soporta sistemas operativos Windows, incluso solo ciertas versiones, dejando de lado otros sistemas operativos como Linux o Mac y dispositivos móviles.
- Desarrollo de una aplicación multi-plataforma basada en microservicios, servidor API implementado en .net que se conecta a una base de datos MongoDB, aplicación para windows en wpf, aplicación para android utilizando kotlin:
    - Pros:
        - Permite el uso de la aplicación en dispositivos móviles Android y escritorios Windows.

        - Sistema distribuido que permite la implantación en sistema “Cloud” o “On premises” que facilita el teletrabajo desde cualquier parte del mundo o el uso de la aplicación desde oficinas distribuidas en diferentes localizaciones.
        - Permite replicación de servicios para balancear la carga de peticiones.
        - Lenguajes de programación similares a los utilizados durante la carrera, requiere poca inversión en aprendizaje previo.
        - Permite reemplazar servicios del sistema por otros que realicen la misma función utilizando otros lenguajes/tecnologías sin ser un coste de desarrollo grande, de forma que sea más factible realizar poco a poco mejoras en la aplicación.
    - Contras:
        - Coste de desarrollo, requiere desarrollar 2 aplicaciones totalmente diferentes que se comporten de la misma manera, requiere doble implementación y doble mantenimiento.
        - Dificultad de desarrollo por coordinar todos los servicios, dependencias, tolerancia a fallos, …
        - Limite a solo Android y Windows, dejando de lado Linux y sistemas tanto de escritorio como móviles Apple, entre otros.
- Desarrollo de una aplicación web implementada en Angular basada en microservicios, servidor API implementado en .net conectado a base de datos MongoDB:
    - Pros:
        - Permite el uso de la aplicación desde cualquier dispositivo con conexión a internet utilizando cualquier sistema operativo que permita el uso de los navegadores más utilizado.
        - Igual que en la opción anterior, permite replicación de servicios, cambio de servicios sin afectar al sistema y es un sistema distribuido que permite su uso “Cloud” o “On premises”.
        - Utiliza una de las tecnologías más utilizadas en cada campo y requeridas en muchos puestos de trabajo, Angular para “Frontend”, .net para “Backend” y MongoDB como base de datos.
    - Contras:

- Tecnologías no utilizadas durante la carrera, requiere una inversión en aprendizaje.
- Dificultad de desarrollo por coordinar todos los servicios, dependencias, tolerancia a fallos, …
- Alta dependencia de sistemas “Cloud” para utilizarla de forma pública, requiere amplios conocimientos en arquitectura.

3.4 Solución propuesta

La primera opción, la aplicación de escritorio en Windows fue la idea inicial, que fuese un problema de programación, donde sea fácil implementar funcionalidades y visualmente para una exposición, en el mundo real sería un gran problema, así que se orientó el proyecto más en un proyecto de arquitectura.

La segunda opción era la directa candidata para suplir las necesidades sin tener que invertir tiempo extra en aprendizaje de lenguajes de programación, pero en el momento de empezar a implantar el proyecto y tener que realizar 2 aplicaciones diferentes desde cero, para luego solo poder llegar a tres cuartos de los dispositivos móviles [4] y ordenadores [5].

La opción adoptada finalmente es la tercera, requiere un coste elevado de aprendizaje y preparación, pero es la mejor forma de asentar buenas bases para hacer una aplicación competitiva y adaptada a los nuevos tiempos. Llegando a la mayoría de los dispositivos y supliendo las necesidades de hoy en día.

3.5 Plan de trabajo

El primer paso es diseñar la arquitectura a utilizar en la aplicación y en base a ese diseño crear un listado de tareas con su estimación:

Estimación horas desarrollo proyecto TFG Gestalert						
Servicio	**Descripción**	**Coste Estimado (h)**	**Coste Real (h)**	**Horas restantes estimadas**	**Horas restantes reales**	**Estado**
Documentación	Planificación arquitectura y modelos de datos a seguir en el proyecto	32	40	328	320	DONE
Documentación	Crear Tareas	3	3	325	317	DONE
Base de datos	Crear base de datos MongoDB en cloud y documentos requeridos	3	3	322	314	DONE
Azure	Configurar entorno azure	2	2	320	312	DONE
Azure	Crear script Terraform para la creación de recursos en Azure	3	2	317	310	DONE
Azure	Crear KeyVault para almacenar los secretos a utilizar	1	1	316	309	DONE
Nuget utilidades	Crear clase conexión con KeyVault de Azure	5	7	311	302	DONE
Nuget utilidades	Crear clase conexión con base de datos MongoDB	4	4	307	298	DONE
Nuget gestalert	Crear clases para modelo de datos y representación en base de datos	3	2	304	296	DONE
Nuget gestalert	Crear clases API para operaciones CRUD en los modelos de datos utilizados (task, category, user)	4	7	300	289	DONE

API	Crear proyecto .net con un controlador por cada modelo de datos	5	4	295	285	DONE
Nuget utilidades	Configurar nugets para poder utilizarlos de forma remota desde nuget.org o github	2	4	293	281	WON'T DO
Nuget utilidades	Crear nugets para utilizar de forma local desde el resto de servicios	2	2	291	279	DONE
Nuget utilidades	Crear conexión con logger Azure AppInsights para centralizar loggers	5	5	286	274	DONE
API	Utilizar injección de dependencias para conectar con Keyvault y MongoDB	5	8	281	266	DONE
API	Realizar pruebas CRUD desde postman	3	2	278	264	DONE
API	Publicar aplicación en Azure utilizando Docker	3	3	275	261	WON'T DO
API	Publicar aplicación en Azure como web service	3	2	272	259	DONE
Frontend	Crear aplicación en Angular	8	8	264	251	DONE
Frontend	Mostrar lista de Tareas en web consumiendo API alojada en Azure	10	14	254	237	DONE
API	Crear sistema de Cliente Autogenerado de los controladores y clases utilizadas en API para Angular	6	20	248	217	DONE
Frontend	Utilizar cliente autogenerado para cargar listado de Tareas	3	6	245	211	DONE
Frontend	Publicar aplicación Angular en servidor Firebase	2	2	243	209	DONE

Nuget gestalert	Añadir funcionalidades especiales API tareas listado "MyDay" y "Outdated"	4	2	239	207	DONE
Nuget gestalert	Añadir funcionalidades especiales API tareas asignar/desasignar Tarea a Usuario,marcar completada	3	2	236	205	DONE
Frontend	Añadir Menu lateral y barra superior a la aplicación	6	8	230	197	DONE
Frontend	Modificar componentes para adaptarse a versión movil	4	10	226	187	DONE
Frontend	Extraer tabla tareas a componente independiente para hacer poder reutilizarla	2	10	224	177	DONE
Frontend	Añadir vista home con el listado de tareas de "MyDay" y "Outdated"	4	2	220	175	DONE
Nuget gestalert	Cambiar ordenación por defecto listados a fecha límite más cercana y luego prioridad	2	2	218	173	DONE
Cola datos	Crear cola RabbitMQ y hostear en Cloud AMQP	10	4	208	169	DONE
Nuget utilidades	Añadir conexión con RabbitMQ, consumo y publicación de mensajes	4	4	204	165	DONE
Simulator	Crear proyecto .net con un controlador que reciba las propiedades de una tarea como datos	3	3	201	162	DONE
Simulator	Enviar Json a cola RabbitMQ	3	3	198	159	DONE
Daemon	Crear proyecto .net con un sistema de CronJobs	2	2	196	157	DONE
Nuget utilidades	Añadir servicio de cronjob usando Quartz	2	6	194	151	WON'T DO

Daemon	Añadir cronjob de ejemplo utilizando HostedServices de Injección de dependencias	6	7	188	144	DONE
Daemon	Añadir cronjob para consumir mensajes de cola RabbitMQ y crear tareas con sus datos	3	5	185	139	DONE
Daemon	Añadir validador mensajes consumidos de la cola y registrando resultado en AppInsights	4	8	181	131	DONE
Nuget utilidades generales	Añadir servicio de envío de correos utilizando gmail	2	2	179	129	DONE
Gmail	Configurar cuenta para poder enviar correos desde aplicación con authentificación en 2 pasos	1	3	178	126	DONE
Daemon	Añadir cronjob para enviar diariamente informe a cada usuario de sus tareas para el día y las pendientes.	4	4	174	122	DONE
Daemon	Añadir cronjob que cada 30 minutos compruebe y registre el estado de la API de Azure en AppInisights	2	8	172	114	DONE
Frontend	Modificar tabla tareas para poder marcar o desmarcar tareas como completadas	4	8	168	106	DONE
Frontend	Modificar tabla tareas para poder asignar o desasignar tareas a usuarios	4	8	164	98	DONE
Nuget gestalert	Modificar respuesta de las API para que devuelvan un código y mensaje personalizado según la petición	8	12	156	86	DONE

API	Modificar controladores para que envíen la respuesta personalizadas al frontend	4	10	152	76	DONE
Frontend	Crear servicio para mostrar alertas en pantalla	2	3	150	73	DONE
Frontend	Añadir formulario para poder añadir nuevas tareas	4	12	146	61	DONE
Frontend	Añadir formulario para poder añadir nuevos usuarios	4	3	142	58	DONE
Frontend	Añadir formulario para poder añadir nuevas categorías	3	2	139	56	DONE
Frontend	Añadir ventana para poder ver las tareas en detalle	3	2	136	54	DONE
Frontend	Añadir pantalla para ver infomación usuario	3	2	133	52	DONE
Frontend	Añadir pantalla login/registro/recuperar contraseña/log out	20	20	113	32	DONE
Frontend	Mejoras visuales estilos y mejoras modo responsive para móvil y diferentes tamaños de aplicación	20	20	93	12	DONE
Frontend	Modificar tablas para poder recargar datos, ocultar columnas y priorizar ocultación automática.	10	12	83	0	DONE
Frontend	Editar ventana detalle para poder editar tareas.	10	0	73	0	TO DO
User	Crear servicio en .net para gestionar usuarios y tokens	3	0	70	0	TO DO
User	Crear opciones de login que devuelva un token, revise que no esté caducado y devuelva estado.	30	0	40	0	TO DO
Frontend	Conectar pantallas usuario con nuevo servicio user	30	0	10	0	TO DO
API	Eliminar controlador de usuarios a nuevo servicio	5	0	5	0	TO DO

Tabla 1: Horas estimadas para desarrollo del proyecto desglosado por tareas y servicios

Como se puede ver en el listado de tareas hay algunas tareas (marcadas en azul con estado en pendiente) que no han podido ser realizadas por la estimación realizada al principio del proyecto, resultado ser inferior a lo estimado por la complejidad de las tecnologías y entornos utilizados.

3.6 Presupuesto

Pese a ser una aplicación alojada en un entorno Cloud, por la parte de infraestructura ha habido un gasto inferior a 1 céntimo.

La parte web, está alojada en un servidor gratuito de Google (Firebase) [6]. Para el servidor API, está alojado como API pública en un servidor de Microsoft (Azure) [7], que permite su uso gratuito durante 1 hora al día mientras esté recibiendo peticiones, suficiente para el uso durante su desarrollo y exposición. El servidor de la base de datos, alojado en el servidor de MongoDB Atlas [8], es un servidor de uso compartido gratuito, por lo que tampoco supone ningún coste. La cola de RabbitMQ [9] está alojada también en un servidor gratuito de CloudAMQP [10]. Para los servicios restantes, requieren el uso continuado por lo que solo se ejecutan de forma local para evitar costes durante el desarrollo.

En el ámbito de personal, deberíamos tener en cuenta las 360 horas de técnico utilizadas para el desarrollo de esta aplicación.

4. Diseño de la solución

4.1 Arquitectura del sistema

El diseño de la aplicación actual se ha fundamentado en el uso de varios servicios que estén alojados en diferentes Clouds para simular un entorno real donde cada componente puede estar gestionado por empresas, servidores y entornos diferentes. También una parte importante es reducir el coste de desarrollo al mínimo posible utilizando entornos que permitan el uso gratuito de sus infraestructuras.

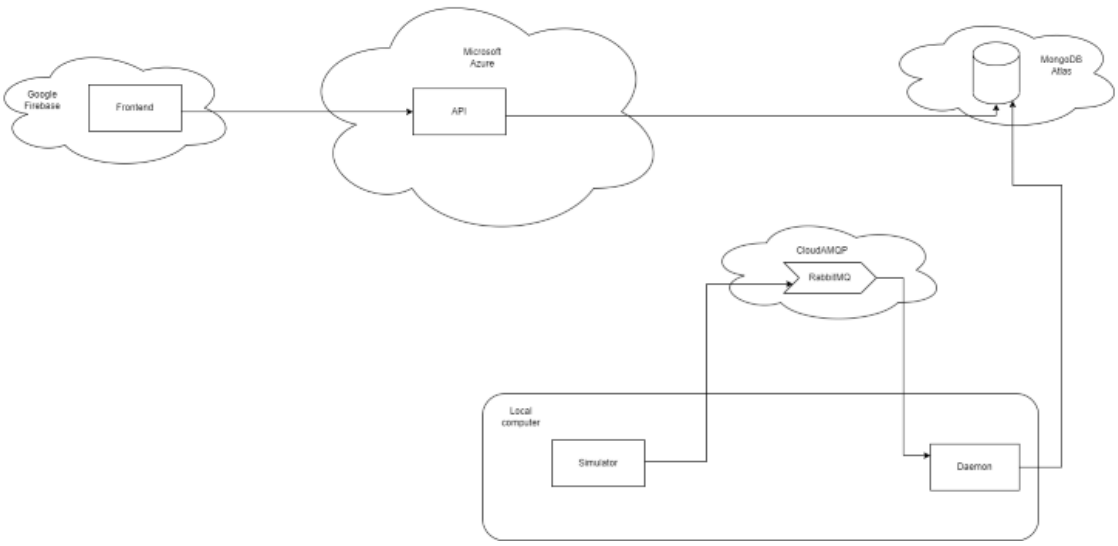

Ilustración 1: Arquitectura de microservicios

Es importante comentar, que tanto el servicio de “Simulator” como el “Daemon” están ejecutados en la máquina local para realizar pruebas, no es necesario que estén en la misma red ni que tenga que ser un equipo local, puede estar alojado en cualquier contenedor usando Docker [11], Kubernetes,etc.

El servicio de Frontend está alojado en el Cloud de Google, Firebase, que nos proporciona la URL de acceso a la aplicación, en este caso https://gestalert-591fd.web.app/.

El servicio API alojado en el Cloud de Microsoft, Azure, utiliza una url para el acceso a la API pública, en este caso https://gestalert-api.azurewebsites.net/

La base de datos MongoDB alojada en Atlas, se conecta utilizando la url configurada en el Keyvault de Azure, que se lee desde todos los servicios.

La cola de RabbitMQ, está alojada en el servidor CloudAMQP, que de igual forma que la URL de la base de datos, está configurada en nuestro KeyVault.

4.2 Diseño Detallado

Esta aplicación basada en microservicios tiene los siguientes repositorios, que durante el desarrollo se ha utilizado Github [12] como control de versiones, uno para cada servicio.

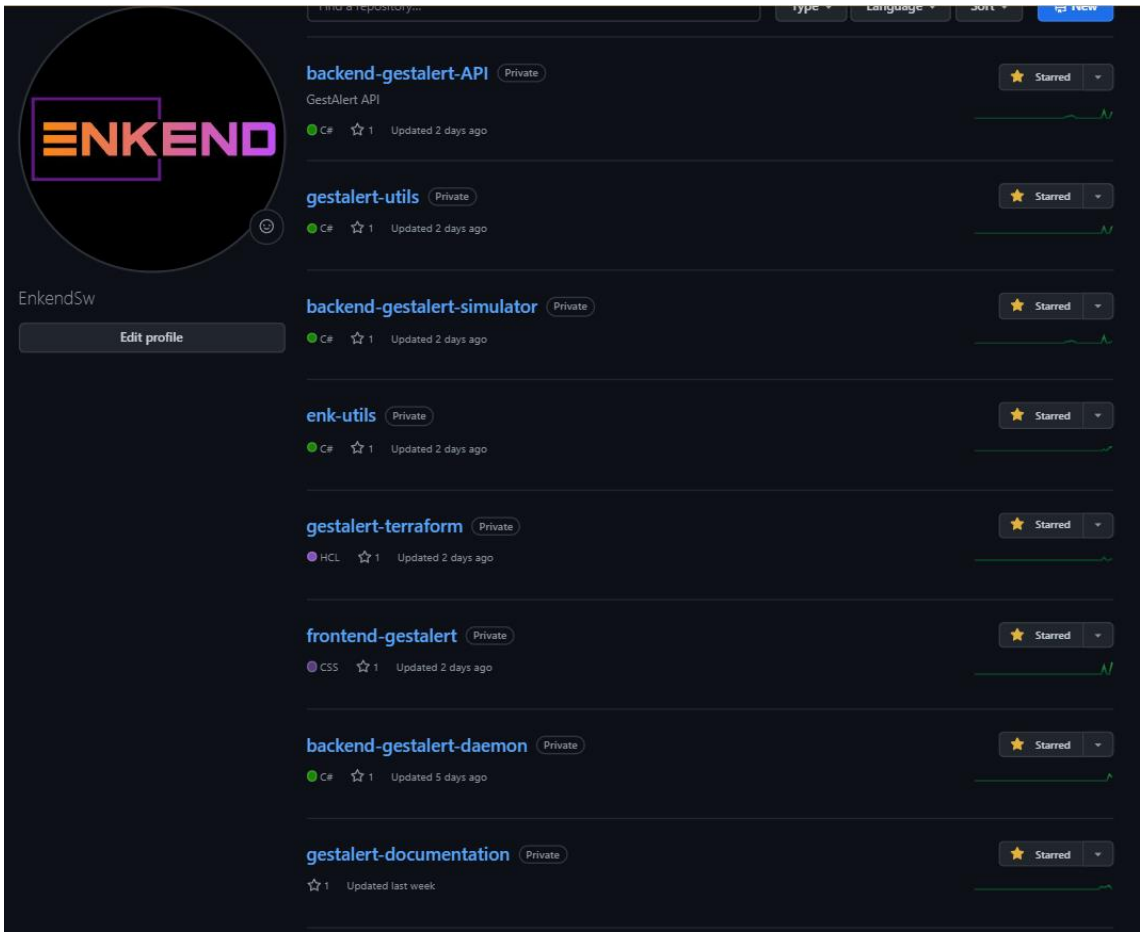

Ilustración 2: Control de versiones para los repositorios

Para evitar la duplicidad de código, la lógica común de la aplicación está centralizada en 2 paquetes de nuget [13], uno específico para esta aplicación y otro genérico que puedo reutilizar para otros proyectos. Todos los proyectos desarrollados en .net (API, Simulator y Daemon) los requieren.

Los proyectos de .net siguen una organización de directorios similar para facilitar el desarrollo.

Nuget Gestalert

Podemos encontrar las definiciones de los modelos de datos, la representación de los datos en la base de datos y los datos a enviar al Frontend. También las APIs de los 3 modelos de datos con sus funcionalidades a utilizar desde el resto de los servicios, con sus respectivas interfaces para en caso de necesitar cambiar el tipo de base de datos, bastar con cambiar la clase que implementa dicha interfaz mediante inyección de dependencias, para el servicio principal no es relevante que tipo de base de datos se utiliza.

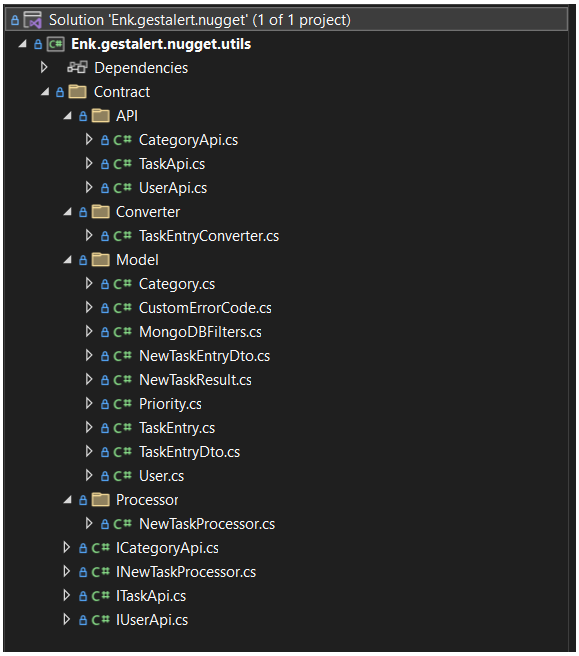

Ilustración 3: Estructura del proyecto de nuget gestalert

Nuget Genérico

Este paquete no depende de esta aplicación, aunque su completa implementación fue realizada para este proyecto, se diferencian 3 nugets diferentes (Cloud, Contract y Services).

El de Cloud, contiene las funcionalidades para utilizar la telemetría de AppInsights [14] de Azure que se utiliza desde todos los servicios en .net para que desde un solo lugar podamos ver los registros y si hay algún error de todas las instancias, estén alojadas en el sistema que sea y con la cantidad de instancias que tengan. También implementa las funcionalidades de recuperar los secretos de nuestro KeyVault para parametrizar todas las aplicaciones y que no tengamos datos sensibles en el código fuente del programa. De esta forma con solo cambiar el valor de la URL de la base de datos en KeyVault, todos los servicios que conecten con ella utilizarán el nuevo valor al iniciar.

En el apartado Contract, tenemos todas las definiciones genéricas de las interfaces de los servicios a utilizar, de forma que todas las clases que utilicen estos servicios usen siempre la interfaz, para que cambiando la clase que la implementa utilizando inyección de dependencias, podemos cambiar la base de datos, servicios que utilizamos, etc. También algunos modelos de objetos que se utilizan desde el resto de los servicios incluyendo el nuget especifico de esta aplicación.

El último nuget dentro de este paquete sería el de los servicios, donde tenemos las implementaciones de 3 servicios distintos, Quartz para los cronjobs, pese a que finalmente no se utilizó esta implementación, servicio de Gmail para el envío de correos y el de RabbitMQ para las colas. En caso de optar por diferentes implementaciones como por ejemplo utilizar Outlook en lugar de Gmail o ServiceBus de Azure en lugar de RabbitMQ, bastaría con implementar la interfaz existente en este nuget y cambiar la clase a utilizar en los servicios dependientes mediante inyección de dependencias.

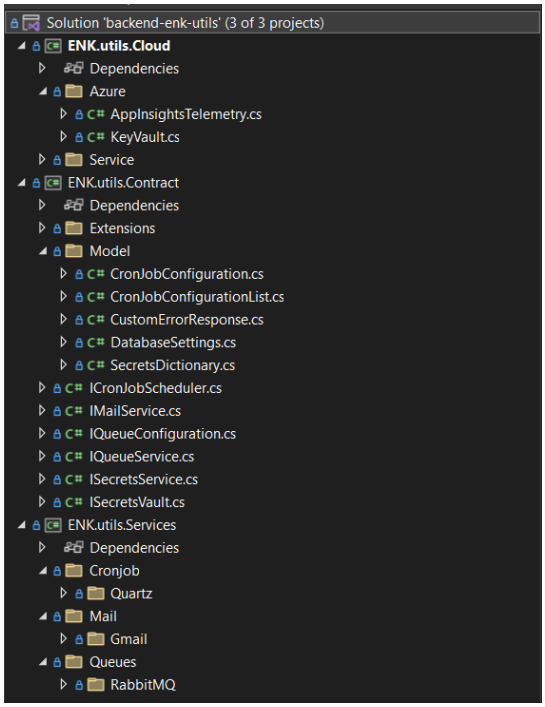

Ilustración 4: Estructura del proyecto de nuget utilidades

Por ejemplo, en el proyecto de la API, podemos ver la dependencia con dichos Paquetes de nuget.

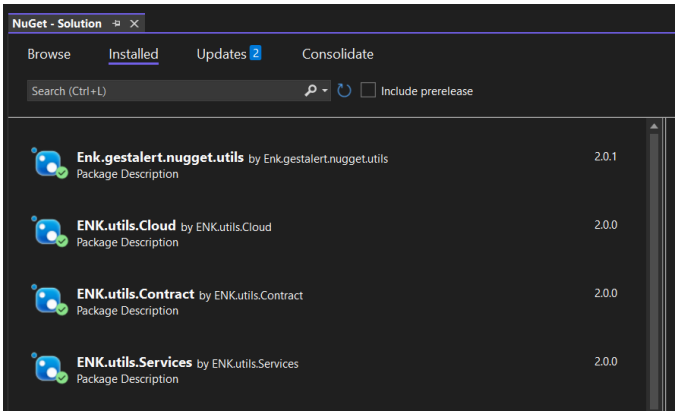

Ilustración 5: Uso de nugets desde otros servicios

Realizar una petición desde la web

Al entrar en la aplicación, para visualizar por ejemplo las tareas, accedemos a la Url https://gestalert-591fd.web.app/#/list-tasks, esto dirige a el servidor Cloud de Firebase donde cargará nuestra web. Al iniciar esta página, hará una petición a nuestra API alojada en Azure.

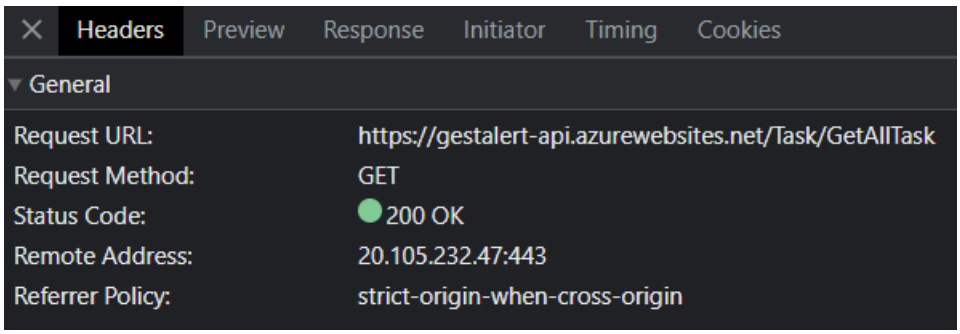

Ilustración 6: Respuesta del servidor ante la petición de la aplicación

Para utilizar estas llamadas utilizamos un cliente Autogenerado con la utilidad de NSwagStudio que permite exportar el swagger generado por nuestra API en .net a una clase en typescript para ser utilizada desde nuestro proyecto de Frontend. Esto facilita mucho la implementación web, de forma que el desarrollador no tiene que escribir ningún tipo de llamada manual, ni crear el modelo de los objetos a utilizar, viene todo en el cliente autogenerado. Una llamada a la API es tan simple como suscribirse al resultado de esta.

```
private loadTasks() {
  this.apiService
    .taskGetAllTask()
    .subscribe((response) => this.taskListSubject.next(response));
}
```

Ilustración 7: Ejemplo de llamada al servidor

En caso de requerir una respuesta personalizada, la implementación se complica un poco ya que requiere “pipes” para no lanzar excepciones y mostrar mensajes emergentes en su lugar como está por ejemplo implementado en la creación de usuario.

Cuando la API recibe la petición, va al controlador que requiera, este conecta con la API implementada por el “nuget-gestalert”, que a su vez utiliza el nuget genérico para poder hacer peticiones a la base de datos.

```
[HttpPost("CreateTask", Name = nameof(CreateTask))]
1 reference
public async Task<ActionResult> CreateTask([FromBody] NewTaskEntryDto newItem)
{
    var response = await _service.CreateAsync(newItem);

    return response.StatusCode == HttpStatusCode.OK
        ? Ok(response)
        : BadRequest(response);
}
```

Ilustración 8: Ejemplo de implementación funcionalidad API

Esta base de datos está alojada en un servidor de MongoDB Atlas, que permite las conexiones desde los diferentes servicios utilizando las variables de nuestro KeyVault [15], donde está parametrizada la URL con el usuario y contraseña, además del nombre de la base de datos a utilizar, que permite que está conexión sea efectiva. Para el uso de este KeyVault requiere variables de entorno para poder autenticar los datos del usuario y garantizar el acceso seguro a estos datos sensibles.

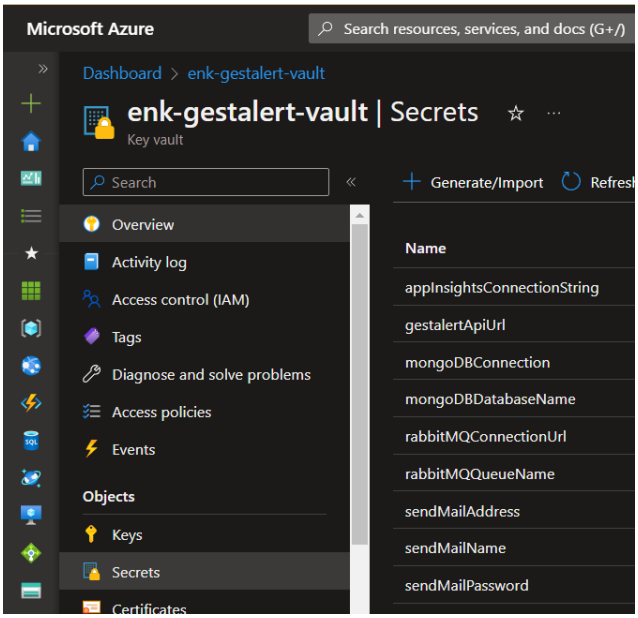

Ilustración 9: KeyVault de Azure

En caso de que esta API no esté en funcionamiento, el usuario podrá ver un mensaje emergente donde le indicará que no se ha podido acceder a la API. Desde el portal de Azure podremos ver un análisis de su estado gracias al servicio Daemon que cada 30 minutos escribe si está o no disponible dicha API y poder investigar irregularidades en base al porcentaje de disponibilidad y la duración media de respuestas. También podemos generar gráficas con estos datos para ver el histórico o en tiempo real.

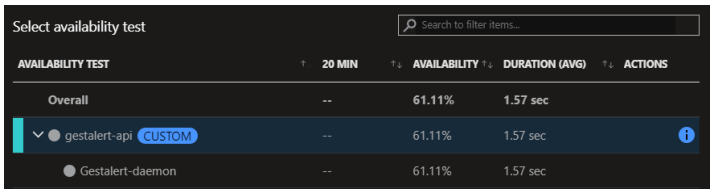

Ilustración 10:Resultado tests de disponibilidad de servicios

Podemos visualizar también los logs que escriben todos los servicios en AppInsights, donde podemos filtrar por día/hora, servicio que ha escrito el log, descripción, nivel de severidad, etc.

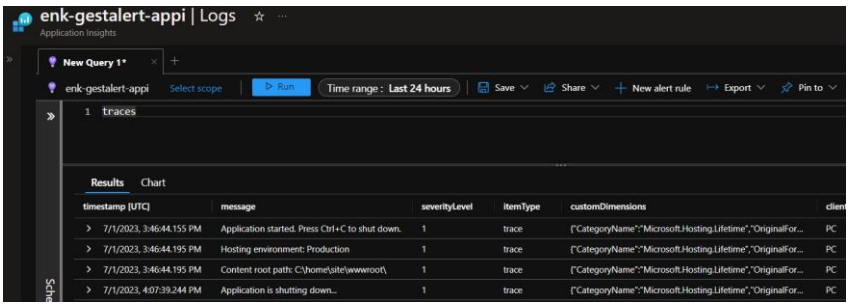

Ilustración 11: Registros centralizados de servicios en AppInsights

En caso de que necesitemos que un usuario sin conocimientos de informática pueda monitorizar estos datos, podemos utilizar un “Workbook” [16] creado para visualizar estos datos con tablas y gráficas. En este caso podemos ver el número de errores que se repiten, el mensaje de error y la fecha y hora de última repetición. También podemos filtrar por nivel de severidad y un filtro de fechas.

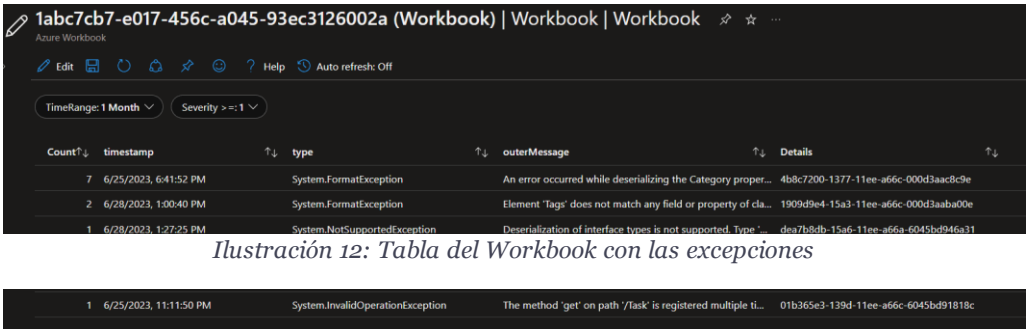

Ilustración 12: Tabla del Workbook con las excepciones

Uso de la cola de RabbitMQ

El proyecto “Simulator” está creado con la única función de simular los mensajes que enviaría una aplicación o cliente externo a nuestra cola de RabbitMQ, para eso genera un JSON con los datos de una tarea y lo envía a la cola.

Utilizada la interfaz web de swagger podemos realizar estos envíos. En este simulador no hay ningún tipo de validación, ya que no es servicio para utilizar fuera del entorno de desarrollo, la validación se realizará al hacer el consumo de la cola mediante el servicio Daemon.

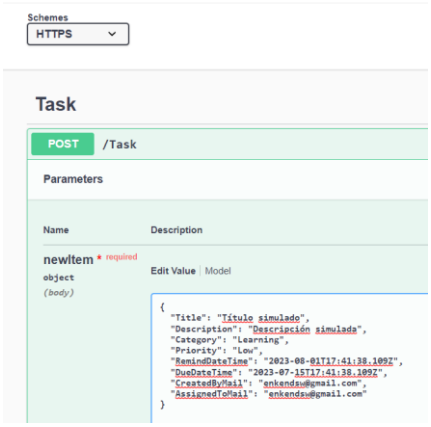

Ilustración 13: Interface de Swagger

Desde el portal de RabbitMQ podemos ver que se ha creado un nuevo mensaje en nuestra cola.

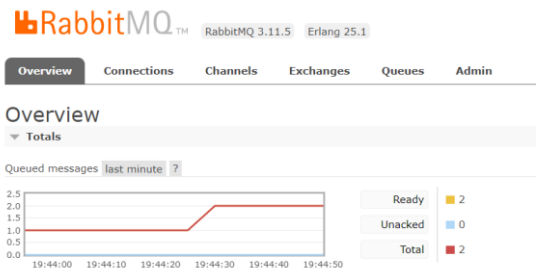

Ilustración 14: Gráfica de mensajes en cola de RabbitMQ

Desde el servicio Daemon, en concreto en el cronjob “StreamingAnalytics”, consume los mensajes de esta cola cada 30 segundos, aunque tarda menos de 1 segundo en procesar cada mensaje, podríamos reducir este tiempo sin problema. Este Cronjob sería uno de los primeros candidatos a separar a un servicio independiente para poder tener varias instancias en caso de que la cola se llene más rápido de lo que podemos consumirla.

En caso que los datos de la tarea superen la validación, se creará en nuestra base de datos una nueva tarea, utilizando la implementacíon de API del nuget de la aplicación

La validación de compone de los siguientes puntos:

- Usuario de creación existente
- Usuario de asignación existente en caso de que exista
- Categoría existente
- Prioridad existente
- Validación de tipos

En caso de fallar alguno de estos puntos, no se creará la tarea pero si que registrará en AppInsights un log de warning para tener en cuenta que ha fallado de la validación y que datos tenía.

En caso de no poder procesar el mensaje por un error interno, dicho mensaje se devolvería a la cola de RabbitMQ enviandolo a la cola de DeadLetter para ser reprocesado.

4.3 Tecnologías Utilizadas

La principal herramienta utilizada para el conteo de las tareas ha sido Microsoft To Do, donde agrupaba las tareas por servicio, y me organizaba los días para ver que tareas hacer en cada momento.

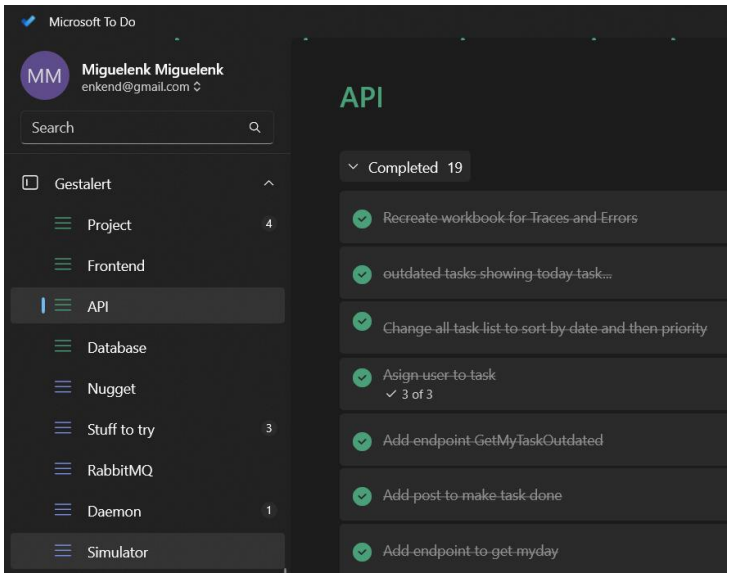

Ilustración 15: Microsoft To Do con las tareas por servicio

Para la redacción de este documento he utilizado Microsoft Office Word y para los diagramas draw.io. Toda la documentación y repositorios los he alojado en Github para el control de versionado y evitar la pérdida de información, siempre en repositorios privados.

Para los diferentes servicios:

Desarrollados utilizando Visual Studio 2022, en C# .net 6.0.

Las librerías se utilizan todas desde el nuget “Enk-utils” por lo que no tienen dependencia directa los servicios, pero las pongo debajo del servicio que las utiliza igualmente.

- Enk-utils
    - Librerías utilizadas desde todos los servicios.
        - Azure Identity y Security
        - Microsoft Dependency Injection, Hosting, Logging y Optional

  Backend-gestalert-api
    - Librerías utilizadas:
        - MongoDB
    - Utilidades extra:
        - NswagStudio (Generación de cliente autogenerado)
        - Publicación de la aplicación en Azure desde VS2022
- Backend-gestalert-daemon
    - Librerías utilizadas:
        - RabbitMQ
        - MongoDB
- Backend-gestalert-simulator
    - Librerías utilizadas:
        - RabbitMQ
- Gestalert-utils
    - Librerías utilizadas:
        - Newtonsoft JSON

Desarrollado utilizando Visual Studio Code, utilizando Angular.

- Frontend-gestalert
    - Librerías utilizadas:
        - DevExtreme
    - Utilidades extra:
        - Dependencias de Firebase para la publicación

5. Desarrollo de la solución propuesta

5.1 Primeros pasos

El mayor problema de este proyecto y la mayoría hoy en día es que sin una buena preparación y planificación previa, se perderá demasiado tiempo una vez se empiece a desarrollar, incluso tener que rehacer parte del programa por imposibilidad de adaptarse a ciertas necesidades.

En mi caso, lo más complicado fue como comunicar todos los servicios, tanto la parte web, como los servicios de Backend como la base de datos, eran retos muy interesantes, pero siempre estaba la posibilidad que fuese demasiado complicado como para poder ser implementado en un proyecto de estas características.

La primera semana dedicada al proyecto fue centrada en realizar esquemas de la arquitectura a emplear en el proyecto, de forma que se tengan claras las tecnologías y posibilidades a utilizar.

5.2 Problemas y dificultades

Uno de los mayores problemas es la estimación de tiempo, en el trabajo que se realiza en la empresa, normalmente, eres parte de un equipo, donde cada uno de los miembros suele estar especializado en algún apartado concreto (desarrollador de Frontend o Backend, arquitecto, diseñador gráfico,) donde en las tareas a realizar suelen ser centradas en la funcionalidad a implementar. Esto me hizo que la estimación de tiempo sea normalmente por debajo de lo utilizado realmente, ya que no es solo tiempo de desarrollo, también es tiempo de preparación previa, como utilizar la arquitectura actual, como desarrollarlo en el Backend y como se mostrará en el Frontend.

Al ser un proyecto basado en microservicios, hace que se dificulte su implementación, no están todos los elementos de la aplicación en un solo proyecto. Para hacer un cambio simple en los elementos a mostrar en la web, es necesario

modificar al menos 3 proyectos, generar paquetes de nuget, generar el cliente de la API y publicar tanto los servicios de Backend como Frontend.

Para evitar duplicidad de código entre servicios, se utilizan los paquetes de nuget, que originalmente la idea era publicarlos en NuGet o Github para poder usarlos de forma remota, esto incrementa demasiado el tiempo de espera y desarrollo de cada funcionalidad por lo que se descartó esta opción.

Todos los proyectos de Backend utilizan la inyección de dependencias de Microsoft para poder modular la implementación del servicio, de forma que los componentes no dependan de la implementación, solo utilizan una interfaz, que, al inicio del programa, seleccionamos que clase implementa la interfaz. Esto hace bastante más complicado el desarrollo, pero a la larga hace el sistema más tolerante a cambios.

Para los cronjobs a utilizar en el servicio Daemon, la primera opción, fue utilizar Quartz, una librería que permite crearlos de forma fácil, el mayor problema es su implementación utilizando inyección de dependencias, es necesario realizar muchos cambios por lo que se descartó utilizar esta tecnología.

En el servicio API, se integró en un Docker, para poder ser publicado en un contenedor, el problema es que para poder publicar esta imagen suponía un coste en Azure, por lo que se descartó esta opción durante el desarrollo.

5.3 Decisiones tomadas

Un parte del proyecto que no ha podido ser desarrollada es la gestión de los usuarios a nivel de login, tokens, roles y permisos, ya que, en el punto de empezar a desarrollar esta funcionalidad, habíamos superado las horas disponibles de desarrollo para este proyecto. Esta funcionalidad será implementada pero fuera de esta entrega.

Para disminuir el tiempo de espera entre los cambios en los paquetes de nuget y su publicación online, se generaron solo los paquetes de forma local para disminuir el tiempo de espera entre publicación y construcción de dichos paquetes. Este modelo solo funciona porque es un desarrollo individual, en caso de ser parte de un equipo, este modelo es insostenible por lo que hay que alojar dichos paquetes en una red

compartida o algún repositorio de internet, aunque conlleve algo más de tiempo de espera.

Para el servicio de Daemon, los cronjobs se desarrollaron utilizando Hosted Services que se ejecutaban repetidas veces en base a la parametrización, de forma que son fácilmente utilizables con la inyección de dependencias a diferencia de Quartz.

6. Implantación

6.1 Despliegue en la nube

- Frontend: Aplicación web
    - Desplegado en la nube de Firebase de forma manual, utilizando la línea de comando desde visual studio code.
    - Para realizar este despliegue, primero se hace un build del proyecto de Angular, generando unos ficheros para ser alojados en el servidor, de forma que no tiene un tamaño tan elevado como el proyecto real y protege la implementación del mismo.
- Backend: Servicio API
    - Desplegado en la nube de Azure de forma manual, utilizando la utilidad Publish de Visual Studio 2022
    - Para realizar este despliegue, primero hay que generar los recursos a utilizar en Azure, para ello se dispone de un script de Terraform para generar los recursos.
- Base de Datos: MongoDB
    - Desplegado en la nube de MongoDB Atlas, creada base de datos y documentos de forma manual no necesita cambios en el despliegue de nuevas versiones.
- Cola de Datos: RabbitMQ
    - Desplegado en la nube de CloudAMQP de forma manual, creación de cuenta, colas y enrutamiento. No necesita cambios al desplegar nuevas versiones del resto de servicios.

- Servicios Extra:
    - Daemon y Simulator Services: Ejecutados de forma local, aunque la mejor opción sería realizar un despliegue en la nube cuando se utilice la aplicación en entorno de producción.

6.2 Despliegue en local

Todos los servicios pueden ser ejecutados de forma local, bastaría con dirigir las conexiones al entorno local en lugar del entorno Cloud.

Podemos hacer una imagen de Docker para todos ellos, tanto los servicios de Frontend y Backend, la base de datos de MongoDB y la cola de RabbitMQ. Esto tiene un consumo bastante elevado de los recursos del ordenador, pero con un equipo adecuado sería posible.

Esto nos permite poder utilizar esta aplicación en una red local de una empresa en caso de que se requiera un despliegue “On premises”.

6.3 Despliegue automatizado

Los despliegues utilizados actualmente son altamente ineficientes, han sido utilizado únicamente por no requerir ningún coste de alojamiento o gestiones. En caso de ser desarrollado por una empresa, todo debería ser desplegando, utilizando CI/CD (continuous integration and continuous deployment) de forma que al hacer un commit en la rama adecuada por ejemplo o utilizar algún tag en el repositorio, se realice un despliegue de todos los servicios en la nube.

En caso de utilizar este proyecto en una empresa, utilizaría un despliegue de todos los servicios en la nube de Azure utilizando Kubernetes para orquestación. Utilizaría los repositorios de Azure DevOps para el control de versiones junto con las Pipelines que permiten implementar el CI/CD.

7. Pruebas

7.1 Pruebas de primeras versiones

Al ser una aplicación web, no requiere instalación por lo que, enviando un enlace a la web, pueden realizar pruebas fácilmente, en este caso me ayudaron amigos y familiares a probar las primeras versiones.

Uno de los puntos en común de la mayoría de los usuarios es que la versión móvil no se adaptaba bien, era simplemente una versión en miniatura de la versión de escritorio por lo que fue una de las tareas añadidas a posteriori para mejorar la visualización. Después de realizar este cambio el grado de aceptación de la aplicación en su versión móvil fue mucho más positiva.

Otro tema importante que se solucionó prácticamente al final del desarrollo fue los colores utilizados en la interfaz gráfica, cambiando los estilos y la apariencia visual de la aplicación para ser más agradable visualmente. Este cambio fue bien recibido por la mayoría de los usuarios, aunque no por todos, al final la solución sería crear varios estilos y que el usuario pueda cambiar entre ellos para intentar contentar al mayor porcentaje de usuarios.

El último tema que se modificó en base a las pruebas realizadas por los usuarios fue la trata de errores, mostrar mensajes al usuario de la situación actual de la aplicación, había varias quejas de mensajes poco claros de porque no podían realizar ciertas acciones. Esto llevo a implementar mensajes desde la API al Frontend para mostrar al usuario por ejemplo porque no puede crear una tarea, que campos fallan. Además, se limitó la posibilidad de estos fallos intentando limitar al máximo los campos de texto libre y utilizando más los desplegables de opciones y componentes que produzcan el formato esperado.

7.2 Control de carga y disponibilidad

Se realizaron pruebas de carga para ver cómo se comportaba el sistema, por ejemplo, se enviaron miles de mensajes de forma simultánea a la cola de RabbitMQ, que se fueron incluyendo en la cola de forma escalonada y procesados de forma regular por el servicio de Daemon a los pocos minutos. Esto nos muestra igualmente

la necesitad de separar el servicio de StreamingAnalytics del Daemon para poder realizar replicación y que en caso de que sean millones de mensajes, tener varios consumidores para que el tiempo de espera entre la publicación de un mensaje y el procesamiento de este, no sea mayor a 5 minutos.

En las pruebas de disponibilidad iniciales se realizaron pruebas de disponibilidad de la API, apagando el servicio o retrasando la respuesta y viendo cómo reacciona la web y que información envía al usuario. Gracias a esta prueba vimos que al apagar el servicio no muestra ningún mensaje de error y simplemente no carga datos, aunque el servicio vuelva a estar disponible posteriormente. Esto provocó cambios en la forma de esperar y leer los datos, incluyendo subscriptores para los mensajes a recibir desde la API para poder mostrarlos en cuanto el servicio esté disponible y un servicio de mensajes Toast para informar al usuario del estado de la petición.

También se hicieron pruebas desactivando la base de datos temporalmente, viendo que se muestra un mensaje controlado de error en la web que no ha podido conectar con la base de datos.

8. Conclusiones

8.1 Problemas encontrados

Uno de los mayores problemas y dificultades encontrados es que realizar una aplicación que siga las necesidades actuales es muy complejo para una sola persona, requiere conocimientos muy elevados en demasiados campos, lo que explica porque hoy en día tenemos equipos de desarrolladores variados para que cada uno pueda suplir unas necesidades concretas del proyecto.

Intentar desarrollar una aplicación en la nube que no implique ningún coste, es complejo ya que la mayoría de Clouds dan un soporte limitado gratuito, por esto los equipos de desarrollos siempre tienen un presupuesto para estos gastos.

8.2 Errores cometidos

Uno de los errores principales es la estimación inferior a lo esperado, que ha dejado servicios importantes como el de usuarios fuera del proyecto. Al ser el primer proyecto basado en microservicios que realizo de forma individual desde cero, intenté sobreestimar la duración de las tareas, aunque no fue suficiente.

Un error que me gustaría subsanar cuando continue el desarrollo de este proyecto fuera de esta entrega es añadir test unitarios y de integración para garantizar el correcto funcionamiento de los componentes, ya sea en los servicios de Frontend como de Backend.

Viendo el proyecto en perspectiva, si en lugar de realizar un proyecto de arquitectura, fuese un proyecto de programación como era la idea inicial, podría haber tenido menos necesidad de emplear tiempo en aprendizaje y podría haber mostrado muchas más funcionalidades en la presentación y parecer una aplicación más completa. El problema con esto es que a la larga sería una aplicación solo para realizar esta entrega mientras que el proyecto realizado va a ser continuado e incluso utilizarlo en empresas reales ya que tiene unas buenas bases asentadas, aunque no muestre tanta funcionalidad visual.

8.3 Conocimientos aprendidos

Los principales y mayores conocimientos aprendidos son en el ámbito de arquitectura, comunicación entre servicios, despliegue Cloud y dependencias. Son campos que no utilicé durante los estudios realizados que realmente me han hecho especializarme en Arquitectura en mi trabajo actual.

Las bases de datos no relacionales son una tecnología muy demandada por lo que pesa a nunca haberlas utilizado durante la carrera, era un punto importante para hacer este proyecto más interesante.

Otro punto importante es el desarrollo web, que no había utilizado nunca durante los estudios, solo algún desarrollo básico en HTML, nunca un entorno complejo y moderno como Angular.

También es importante nombrar las capacidades mejoradas al experimentar la toma de decisiones, importancia de la planificación y estimación de tiempo de

desarrollo, ya que recae sobre ti todo el peso de hacerlo de forma correcta para que el proyecto salga adelante.

8.4 Relación con los estudios cursados

Al ser un proyecto de programación, las asignaturas de programación cobran especial importancia, aunque tampoco hay que dejar de lado el resto de las asignaturas.

Las asignaturas de Tecnologías de Sistemas de Información en la Red y Concurrencia y Sistemas Distribuidos fueron de gran ayuda como base para la comunicación de colas y entre servicios. Pese a que la asignatura de Bases de Datos cursada se centre más en uso de SQL, la mayoría de los conceptos aprendidos sobre datos, fueron de gran ayuda para utilizar la base de datos no relacional.

De la especialización de Ingeniería del software, fueron útiles la mayoría de las asignaturas, en especial Diseño de Software, Análisis y especificación de requisitos para la preparación previa a empezar el desarrollo.

8.5 Cumplimiento de los objetivos establecidos

Las tecnologías principales utilizadas en el proyecto han sido .net framework utilizando C# como lenguaje de programación para los microservicios de backend y angular como framework con Typescript como lenguaje utilizado para la aplicación web. Desplegando todos los servicios en entornos en la nube, tanto del proveedor Microsoft (Azure) como Google (Firebase) y utilizando como base de datos MongoDB, siendo esta una base de datos no relacional. Estas 4 tecnologías son altamente demandadas en muchos puestos de trabajo como desarrollador o arquitecto de software por su alta flexibilidad y eficiencia.

Con el uso de la aplicación desarrollada, se puede facilitar y agilizar mucho el trabajo del usuario, evitando el uso excesivo de papel y disminuyendo la posibilidad de errores y el tiempo necesario para introducir o buscar datos en comparativa con el uso tradicional de papel.

Con la arquitectura utilizada, se garantiza la mantenibilidad gracias a la distribución de las implementaciones para evitar duplicidades y escalabilidad de la aplicación ya sea en la nube o en una red local. Esto nos permite suplir las

necesidades de las aplicaciones multiplataforma, siendo capaz de desplegar los diferentes servicios en cualquier tipo de servidor y ejecutar la aplicación prácticamente en cualquier dispositivo. Con estos puntos suplidos, se ha creado una buena base para desarrollar una aplicación potente utilizada por miles de usuario cumpliendo las necesidades actuales, permitiendo utilizar este software para crear un startup alrededor de su comercialización y desarrollo o que una empresa de software ya establecida lo añada a sus aplicaciones comercializadas y mantenidas.

9. Trabajos futuros

9.1 Tareas pendientes por realizar

Algunas de las tareas pensadas inicialmente a incluir en esta entrega no han sido realizadas por superar el tiempo disponible para el desarrollo de este incluyen la edición de tareas desde la interfaz gráfica y todos los servicios relacionados con usuarios: autenticación, token de sesión, roles, permisos.

Creación de test unitarios y de integración para garantizar la funcionalidad de los servicios de Backend y que, en caso de nuevas implementaciones y cambios en el futuro, se siga manteniendo la funcionalidad originalmente implementada.

Creación de test de visibilidades en los componentes de Frontend para asegurar el correcto funcionamiento pese a los cambios a realizar en un futuro.

9.2 Ampliaciones y mejoras

Sería recomendable realizar diversas modificaciones para el despliegue en producción. Principalmente alojar todos los servicios a excepción de la base de datos en Azure, utilizando Docker para los contenedores de AKS (Azure Kubernetes Service). Teniendo una o varias instancias de cada servicio dentro de un Nodo de Kubernetes permitiendo al sistema AKS balancear la carga utilizando replicación, ya sea del servicio Web + API o del servicio StreamingAnalytics en caso de sobrecargas de la cola.

Crearíamos una red virtual para el acceso desde fuera y utilizando la utilidad Ingress de Kubernetes, comunicaríamos con el Frontend de nuestro clúster de nodos, quedando el resto de los servicios solo accesibles desde la red interna local de AKS lo que aumentaría mucho la seguridad y flexibilidad del sistema.

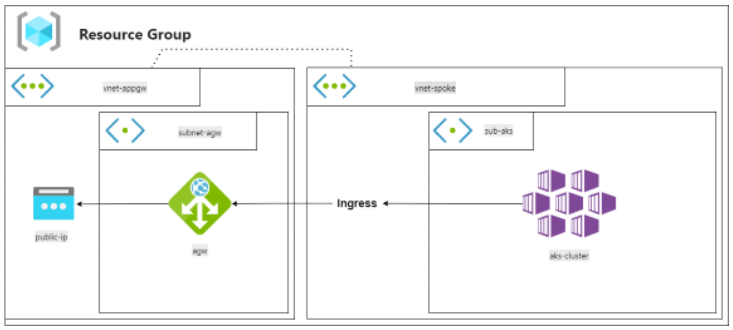

Ilustración 16: Arquitectura a utilizar en un futuro

10. Referencias

10.1 Anexo

Siglas utilizadas:

- ERP (Enterprise Resource Planning) Software de gestión para empresas
- ODS (Objetivos de Desarrollo Sostenible)
- SaaS (Software-as-a-Service) Aplicación en la nube que permite el uso de aplicaciones en internet bajo demanda.
- AD (Active Directory) Conjunto de servicios que permiten conectar a los usuarios con los recursos o aplicaciones en red que necesiten.
- CI/CD (continuous integration and continuous deployment): Integración y despliegue continuo.
- AKS (Azure Kubernetes Service)

Conceptos clave:

- On premises: En ordenadores físicos dentro de los edificios de la empresa.
- Cloud: En la nube, ordenadores físicos de un proveedor externo (Azure, AWS…).
- Frontend: En software, parte del desarrollo del lado del cliente.
- Backend: En software, parte del desarrollo del parte del servidor.

11. Bibliografía

[1] «How to get started with CloudAMQP,» . Available: https://www.cloudamqp.com/docs/index.html.

[2] «Una breve guía para ayudar a su oficina a dejar de usar papel,» . Available: https://geekflare.com/es/paperless-office-software/.

[3] «La evolución de las herramientas de gestión,» . Available: https://revistabyte.es/tema-de-portada-byte-ti/evolucion-herramientas-de-gestion/.

[4] «¿Qué es la gestión de tareas y por qué la usan los líderes?,» . Available: https://www.joemoliner.com/blog/gestion-de-tareas/.

[5] «Android Statistics (2023),» . Available: https://www.businessofapps.com/data/android-statistics/.

[6] «Global market share held by operating systems for desktop PCs, from January 2013 to July 2023,» . Available: https://www.statista.com/statistics/218089/global-market-share-of-windows-7/.

[7] «Firebase,» . Available: https://console.firebase.google.com/u/0/.

[8] «Start in the cloud with Azure,» . Available: https://azure.microsoft.com/en-us/free/.

[9] «MongoDB Documentation,» . Available: https://www.mongodb.com/docs/.

[10] «RabbitMQ Documentation,» . Available: https://www.rabbitmq.com/documentation.html.

[11] «Docker documentation,» . Available: https://docs.docker.com/.

[12] «Github Documentation,» . Available: https://docs.github.com/en.

[13] . Available: https://learn.microsoft.com/en-us/nuget/quickstart/create-and-publish-a-package-using-visual-studio?tabs=netcore-cli.

[14] «Azure monitor,» . Available: https://azure.microsoft.com/en-us/products/monitor/.

[15] «Create keyvault azure for web app,» . Available: https://learn.microsoft.com/es-es/azure/key-vault/general/tutorial-net-create-vault-azure-web-app.

[16] Azure Workbooks Available: https://learn.microsoft.com/en-us/azure/azure-monitor/visualize/workbooks-overview.

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades**.	X			
ODS 11. **Ciudades y comunidades sostenibles**.				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**	X			
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Uno de los principales y más fáciles ODS que podemos relacionar con este proyecto es el número 15, Vida de ecosistemas terrestres, con la implantación de una aplicación como la desarrollada para esta entrega. Podemos reducir sustancialmente el uso de papel por las empresas que la empleen, de forma que registren las tareas y datos en la aplicación en lugar de utilizar papel y con ello reducir la tala de árboles.

Otro de los puntos importantes es el objetivo 10, Reducción de las desigualdades, debido a que, con el uso de la aplicación, ya sea en ordenador, móvil o tablet, podemos permitir a personas con diversidad funcional que tendrían dificultades para realizar documentaciones a papel, poder realizar estas funciones más fácilmente gracias a este proyecto.

El último punto importante, sería el punto 9, Industria, innovación e infraestructuras, ya que, gracias al uso de esta aplicación, las empresas pueden gestionar mucho más fácilmente sus datos, sin tener que depender de documentación escrita a papel que se degrada y es difícil de buscar. Con los filtros y buscadores que ofrece nuestra aplicación, podemos agilizar estas funciones para mejorar el funcionamiento de la empresa.