Universidad de Alcalá
Escuela Politécnica Superior

Máster Universitario en Desarrollo Ágil para la Web.

Trabajo Fin de Máster

Gestor de tareas para procesos Automatización Robótica de Procesos o RPA

ESCUELA POLITECNICA SUPERIOR

Autor: David García Ruiz

Tutor: José María Gutiérrez Martínez

2022

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Máster Universitario en Desarrollo Ágil para la Web.

Trabajo Fin de Máster

Gestor de tareas para procesos Automatización Robótica de Procesos o RPA

Autor: David García Ruiz

Director: José María Gutiérrez Martínez

Tribunal:

Presidente: .....

Vocal 1º: .....

Vocal 2º: .....

Calificación: .....

Fecha: .....

A nuestros alumnos pasados, presentes y futuros...

"Empieza haciendo lo necesario, luego haz lo posible y de pronto empezarás a hacer lo imposible."
Francisco de Asís

Las base de todo conocimiento.....

"Lo importante no es saber, sino tener el teléfono del que sabe."
Les Luthiers

Agradecimientos

El ser durmiente dentro de mí se ha despertado. Yo soy el príncipe de todos los Saiyajin, una vez más... ¡He vuelto!

Vegeta. Dragon Ball Z.

Este trabajo es el fruto de muchas horas de trabajo, búsquedas de información y autoformación, así como un afán incondicional de búsqueda de conocimiento.

Quisiera hacer una mención especial Juan Ignacio García García e Ignacio García Ruiz por forzarme a querer seguir aprendiendo en el mundo de la tecnología.

Durante este desarrollo Alejandra Ramos Romero jamás me dejo rendirme y abandonar este proyecto con el fin de culminar este hito en mi carrera profesional.

A mi compañero de trabajo Victor Iniesta Plaza, que con sus ánimos y desánimos he llegado hasta aquí y culminar un año de sacrificios.

Finalmente, hay incontables contribuyentes gracias a sus conocimientos que fueron consultados mediante el todo poderoso Google. He intentado referenciar los más importantes en los fuentes de este documento, aunque seguro que he omitido alguno.

Me gustaría añadir que sin ninguno de ellos habría podido realizar este proyecto.

Resumen

Este documento ha sido generado como memoria del trabajo de fin de máster, en el cual se aplica los conocimientos aprendidos durante el curso para plasmar la manera de gestionar tareas desde diferentes herramientas del RPA (Robotic Process Automation, Automatización Robótica de Procesos) permitiendo realizar las tareas en conjunto desde diferentes herramientas o cada una de las partes de la tarea desde cada herramienta, de manera centralizada .

El origen de este proyecto aparece de la necesidad de una manera de gestionar tareas desde diferentes tecnologías o herramientas y resolverlas, permitiendo gestionar de manera unificada, cosa que no es sencilla en la actualidad.

El modelo de infraestructura completo sobre el que está estructurado el proyecto sigue la modelo básico de cualquier sistema con MVC (Modelo-Vista-Controlador):

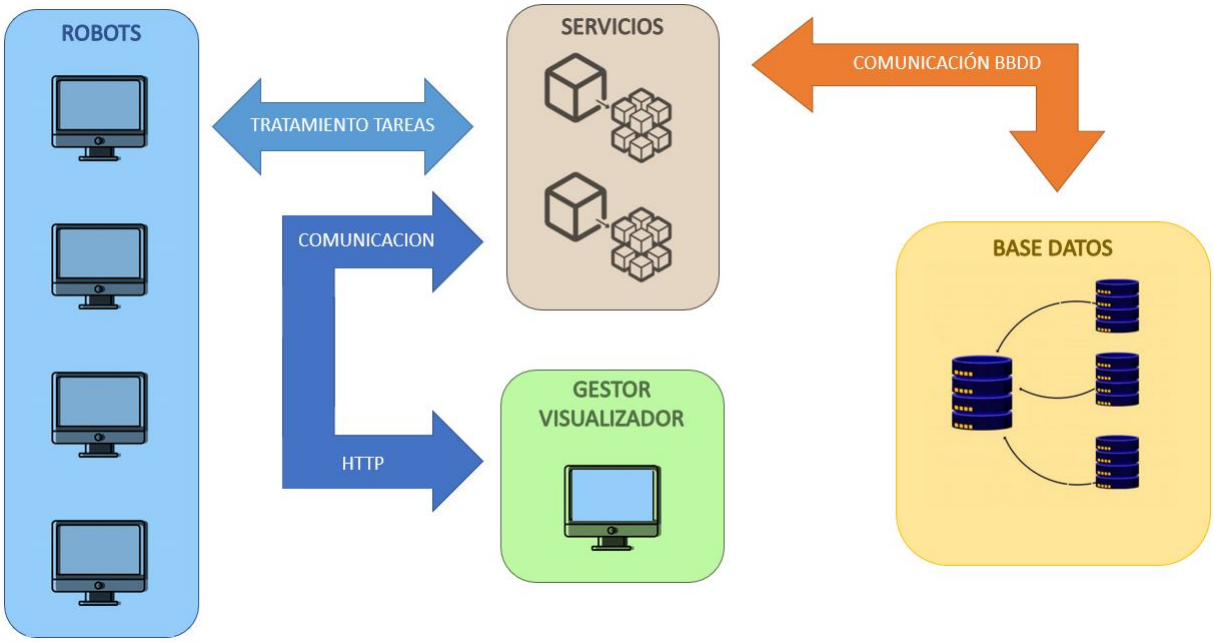


Figura 2: Esquema 1: Modelo de infraestructura para la gestión multi-plataforma de tareas formado por visualizador[1], servicios, robots[2] y base de datos [3].

Para la implementación de la solución será necesario una plataforma basada en una base de datos relacional (se escoge MySQL) y un servidor (en el que puede estar la misma base de datos) para hacer funcionar los servicios y visualizador.

Índice general

Índice de figuras

Índice de tablas

Índice de listados de código fuente

Capítulo 1

Introducción

En la era digital en la que nos encontramos las personas interactúan cada vez más con los robots, su capacidad de aprendizaje, la facilidad de integración y el deseo de evitar realizar tareas repetitivas han permito que surja la rama de trabajo RPA (Robotic Process Automation, Automatización Robótica de Procesos).
A comienzos de siglo empezó a aparecer esta tecnología con el fin de mejorar las condiciones de trabajo reaprovechando las capacidades de las personas y destinando las tareas más desagradables o repetivas a procesos automatizados.
El RPA ha ayudado a pequeñas y grandes empresas a automatizar tareas repetitivas como si fuera un humano el que realizará las acciones en un ordenador, con la ventaja de la fiabilidad de los datos introducidos son correctos, sin comprometer bases de datos al realizar las tareas con los mismos interfaces que un humano o similar, generalmente, con un mejor tiempo de respuesta y consiguiendo que las personas dedicadas a estas tareas puedan apoyar en otras que no sean automatizables.

En este proyecto se pretende que el diseño de un gestor de tareas pueda ser consumido por procesos RPA y un pequeño visualizador de estados que sea independiente del sistema de robotización escogido.

1.1 Estado actual del sistema

Actualmente en el mercado puedes encontrar algunos sistemas de gestión de tareas en colas de trabajo, ya sea median un sistema de gestión de colas de mensajes como puede ser un sistema Redis [18] o Kafka [19] o sistemas de gestión de colas con un visualizador complejo y no apto a las modificaciones del cliente según sus necesidades como podría ser Azure Queue Storage[1].
La mayoría de las herramientas de RPA (BluePrism, UiPath, Nice, Jacada, etc.) poseen su propio sistema de gestión de colas, a veces incluido en con el propio precio de las licencias o como un sub-servicio aparte, pero que no se puede integrar entre dos o más herramientas diferentes de RPA con facilidad.
Para la comunicación de estos mensajes o tareas se suele utilizar mensaje de texto con formato JSON[2] o

[1] El sistema Queue Storage de Azure[20] permite gestionar colas de tareas y visualizar su información, pero no da flexibilidad de visualización y son necesarios otros elementos para poder monitorizar que aumentan el costo.
[2] JSON (JavaScript Object Notation)[21] es un formato de intercambio de datos con un aspecto de árbol con referencia de clave-valor para cada uno de los parámetros.

Figura 1.1: Imagen de ámbitos de aplicación del RPA [4].

con formato [3]

1.2 Objetivos

Crear un sistema que permita la creación de un sistema de colas abstracto e independiente de cualquier tecnología que sea fácilmente configurable y con facilidad de adaptación a las necesidades del cliente con un pequeño desarrollo. Se crearán dos subsistemas independientes, un visualizador de los datos y del estado de los casos o tareas, y un sistema de gestión de tareas que evite que se realicen tareas por duplicado mediante un gestión de colas de trabajo. Estos sistemas serán homogéneos y no dependerán de una herramienta concreta de RPA para completar las tareas a realizar.

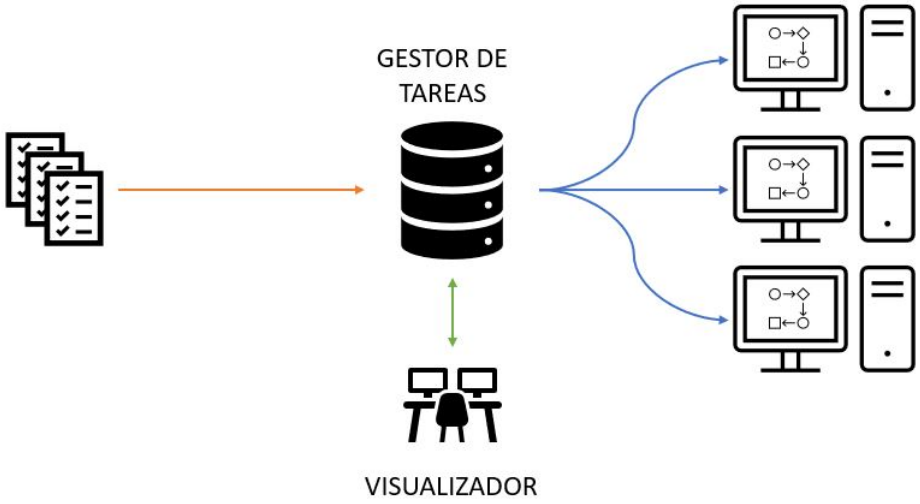


Figura 1.2: Manejo del vehículos autónomos a distancia.

[3] XML (Extensible Markup Language)[22] es un metalenguaje que permite el intercambio de información de manera estructura entre diferentes plataformas de manera abstracta y reutilizable.

1.3 Motivación y objetivos

La motivación de este proyecto fue la posibilidad de mejorar herramientas de integración del RPA, tecnología en puntera con la trabajo actualmente, permitiéndome utilizar los conocimientos aquí aprendidos para mejorar mis condiciones de trabajo.

Los objetivos principales de este proyecto:

1. Primer objetivo: Poder gestionar una cola de trabajo que evite tratamientos duplicados.
2. Segundo objetivo: Poder gestionar tareas, actualizarlas y cambiarles el estado.
3. Tercer objetivo: Poder analizar el porcentajes de resultados y las tareas pendientes.
4. Cuarto objetivo: Poder crear tareas manualmente desde el visualizador.
5. Quinto objetivo: Creación de un pluging para el uso en Blue Prism.
6. Sexto objetivo: Permita subida de documentos Poder extraer datos en CSV[4] para generar tareas.

1.4 Estructura del documento

Este documento presenta la siguiente estructura:

- Portada del proyecto.
- Dedicatorias.
- Agradecimientos.
- Resumen del proyecto.
- Índice de contenidos, índice de figuras, índice de tablas e índices adicionales.
- Introducción: breve introducción al proyecto y al tema a tratar.
- Estados de arte: una introducción sobre las tecnologías utilizadas
- Desarrollo de la aplicación
- Resultados de la solución
- Conclusiones y líneas futuras del proyecto.
- Apéndices con información de manuales de usuario e instalación y requerimientos para el desarrollo.
- Contraportada.

[4]CSV es una extensión clásica de compartir datos estructurados separando los valores en columnas.

Capítulo 2

Estudio teórico

...Encontré mil y una palabras que decir, pero esta singularidad de irracionalidad en la que vivimos, nos hizo cuerdos a los locos para enseñarnos que un mundo de ciegos el tuerto es el rey...

Anónimo

2.1 Introducción

En este capítulo se va a tratar todos los aspectos del estudio teórico y previo al desarrollo del proyecto.

El capítulo se estructura en los siguientes apartados:

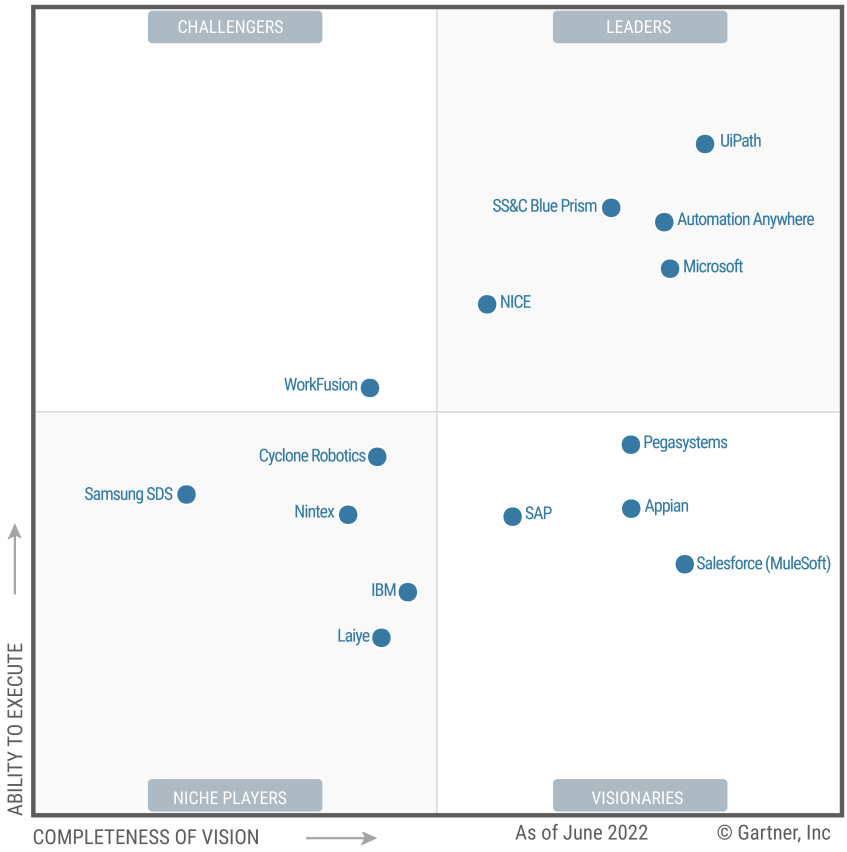
1. Herramientas de robotización y sistemas propios de gestión.
2. Gestores de colas.
3. Alternativas como sistemas de gestión de colas.
4. Técnicas utilizadas: define las metodologías a seguir para el desarrollo del proyecto.
5. Estado del Arte: define las tecnologías existentes a utilizar en el proyecto.
6. Conclusiones

2.2 Herramientas de robotización y sistemas propios de gestión.

La mayoría de las herramientas de robotización más modernas del mercado disponen de un sistema de gestión de colas de tareas, con las que realizar tareas repetitivas con gran facilidad y evitando duplicados. El único problema es que no poseen un sistema de gestión o portal web con el que poder crear casos o con una visualización personalizada que se adapte a las necesidades del cliente. Además cada gestor de colas es único para cada herramienta. Algunas de las herramientas que se encuentran actualmente como lideres del mercado son:

- Blue Prism
- UiPath

Figure 1: Magic Quadrant for Robotic Process Automation



Source: Gartner (July 2022)

Figura 2.1: Cuadrante de Gartner de Julio 2022 ofrecido por UiPath[7].

2.2.1 Blue Prism

Blue Prism se fundó en julio de 2001 como empresa de fabricación de software de automatización de procesos empresariales. Mediante el uso de los diferentes frameworks de Microsoft .Net comenzó a generar una aplicación capaz de automatizar casi cualquier plataforma como sistemas Mainframe[1], aplicaciones

[1]Definición de IBM[23]: Son aplicaciones que corren sobre computadoras de alto rendimiento con grandes cantidades de memoria y procesadores que procesan miles de millones de cálculos y transacciones simples en tiempo real. Suelen acceder a este tipo de aplicaciones mediante clientes pesados instalados en la maquina, los cuales suelen ser sistemas automatizables.

Windows Form, WPF (Windows Presentation Foundation), Java, Java Web, Web, entre otros. En la actualidad está mejorando sus capacidades para integrarse con sistemas Citrix[2] y su capacidad de reconocimiento de imágenes.

Durante muchos años ha sido líder en los procesos de automatización, en 2019 compro Thoughtonomy[3] y en la actualidad aún sigue siéndolo, hace poco , verano de 2022, fue comprada por SS&C[4].

Su sistema de procesamiento de colas de trabajo va incluido en el software pero para crear elementos tiene que realizarlo el propio software desde uno de los pasos que tiene su constructor de flujo, igual que para editarlo. Se gestiona desde un apartado de la aplicación llamado Control Room. Para la gestión y visualización actualmente solo se puede realizar mediante la aplicación nativa, aunque tienen pendiente construir en la versión Blue Prism 7.1 sea consumido como un servicio y permita conectarse e integrase para la visualización de la gestión únicamente. No permite la integración de su cola de trabajo con otras herramientas.

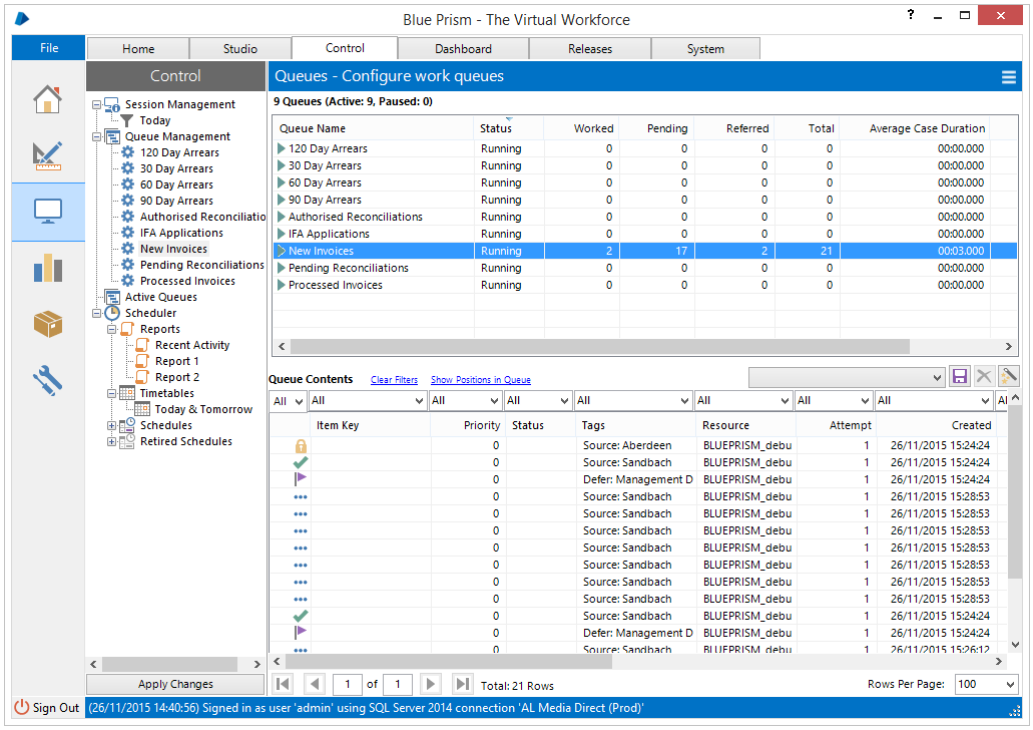


Figura 2.2: Aspecto de la Control Room de Blue Prism.

2.2.2 UiPath

UiPath se fundó en 2005 en Rumanía, creando la herramienta UiPath como herramienta de automatización de las tareas de front y back office[5] con el fin de mejorar la gestión, la relación de los clientes y aprovechar mejor los recursos empresariales.

UiPath utiliza tecnología de Screen Scraping[6] para integrar diferentes tipos de aplicaciones y generar automatizaciones.

[2]Las aplicaciones Citrix permiten interactuar remotamente con otras máquinas o software pero no son accesibles mediante Screen Scramping y la única manera es instalar un sistema que haga de pasarela para poder comunicarse con la aplicación Citrix o mediante reconocimiento de imagen por pantalla.

[3]Thoughtonomy: empresa dedicada a la virtualización de y software de IA (Inteligencia Artificial).

[4]SS&C: es una empresa estadounidense formada gracias a la compra de diferentes softwares y empresas de éxito.

[5]Front y back office: tareas de procesamiento y tareas en tiempo real que se realizan en la mayoría de empresas.

[6]Screen Scraping: es una tecnología que se permite integrar otras aplicaciones que no poseen un API (Application Programming Interface, Interfaz de Programacion de Aplicaciones) para consumirlas.

Tuvo un gran éxito inicial, pero carecía de un gestor de colas y tras ofrecer otro servicio basado en otorgar máquinas virtuales para la ejecución en un entorno Cloud propio, sacaron ,a su vez, un gestor Cloud llamado UiPath Orchestrator, que permitía hacer la gestión de colas de tareas y mostrar resultados, similar a otros entornos, pero sin la capacidad de personalización. En el año 2019-2020 sacaron la versión on-premise de este gestor para aquellas empresas que no querían que sus datos se encontrasen en la nube. El sistema de gestión de colas implica un coste adicional, y lo dota de una base de datos en la versión Cloud, donde almacenar la información pero de manera ofuscada, impidiendo crear APIs para consultarlo. Tampoco existe un API para poder consultar los datos.

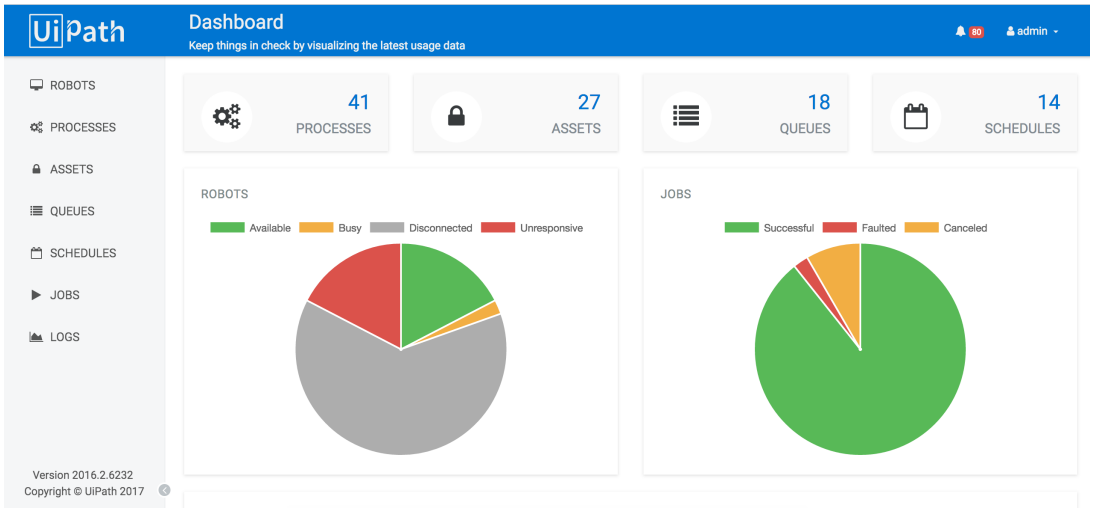


Figura 2.3: Gráfica de procesamiento de UiPath Orchestrator.

2.3 Gestores de colas

Los gestores de colas de tareas surgen como la idea de intentar paralelizar tareas entre diferente sistemas o procesadores, siguiendo algún tipo de ordenación, con el fin de resolver las diferentes tareas sin tener el inconveniente de que se duplique alguno de los procesos de manera indeseada y hacer un seguimiento controlado del mismo.

Figura 2.4: Montón de tareas repetitivas.

Algunos de los sistemas existentes en el mercado son:

- Azure Queue Storage
- IBM WebSphere MQ
- Apache Kafka

2.3.1 Azure Queue Storage

Desde la aparición de Microsoft Azure, la nube Cloud de Microsoft desde enero de 2010, han ido dotando de diferentes servicios que se usan de manera cada vez más frecuente en las empresas para competir con el resto de los grandes entornos Cloud en el mercado como AWS (Amazon Web Services), Google Cloud Platform o Alibaba Cloud.
Desde su aparición Queue Storage Azure en enero de 2014 el servicio a ido evolucionando hasta formar parte del área de servicios SAAS para Almacenamiento de información del entorno de Microsoft Azure.
Para la creación de una cola de trabajo primero tienes que asignarle disco físico o la creación de un disco dinámico que cobre por cantidad de almacenamiento (esta es la opción elegida normalmente, porque abarata costes). Posteriormente a ese disco se le asigna el servicio de Azure Storage Queue al cual se le asigna una IP privada para comunicarse con otros servicios de Microsoft Azure o mediante el servicio de IP pública para que pueda ser consumida desde el exterior. A esta herramienta se le puede, además, agregar un sistema de autentificación para proteger su uso.
El sistema se basa en una cola en la cual hay un emisor de tareas en el formato deseado (JSON, TXT, XML, Excel, etc.) para que sea solicitado desde uno o varios receptores. A este tipo de tareas se le denomina mensajes.

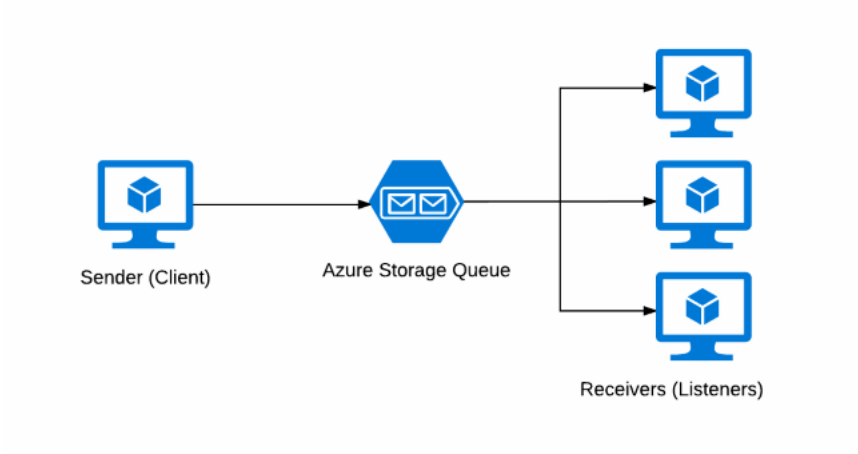


Figura 2.5: Repartidor de tareas Azure Queue Storage[8]: con un emisor o generador de mensajes, el gestor Azure Queue Storage y receptores o solicitantes de los mensajes.

2.3.2 IBM WebSphere MQ

IBM es una gran empresa de tecnología que se fundó entorno al año 1911. Durante su desarrollo como empresa consultora al rededor del diciembre de 1993 creo el sistema de IBM MQ que ha ido mejorando hasta ahora que lo ha convertido en un servicio web consumible desde cualquier entorno, denominado IBM WebSphere MQ[24].
Este producto permite la comunicación de mensajes entre aplicaciones independientes y potencialmente no concurrentes en un sistema distribuido y con las capacidad de aportar seguridad y alto rendimiento. Este servicio es de pago, y para obtenerlo es necesario disponer de una cuenta en el Cloud de servicios de IBM.

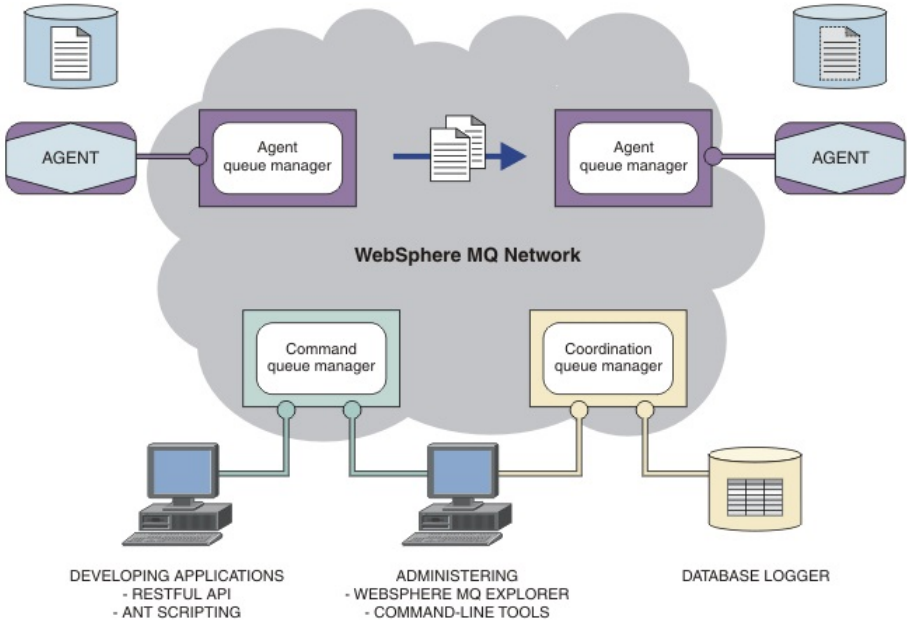


Figura 2.6: Diagrama de una topología simple de WebSphere MQ[9].

2.3.3 Apache Kafka

Apache Kafka[25] es un proyecto creado en noviembre de 2010 por LinkedIn como código abierto escrito en Java y Scala.
Este proyecto es capaz de gestionar de colas de eventos, posee una gran adaptación a la escalabilidad de máquinas que generan los eventos, como a las que los procesan con gran facilidad, funciona como un único servicio con capacidad de alta disponibilidad y almacenamiento de información de manera más segura y estable, con muy poca latencia y capaz de manipular un gran volumen de datos en tiempo real.
Es un sistema que permite el intercambio de mensajes con integración de múltiples lenguajes de programación y mediante mensajes síncronos o asíncronos.

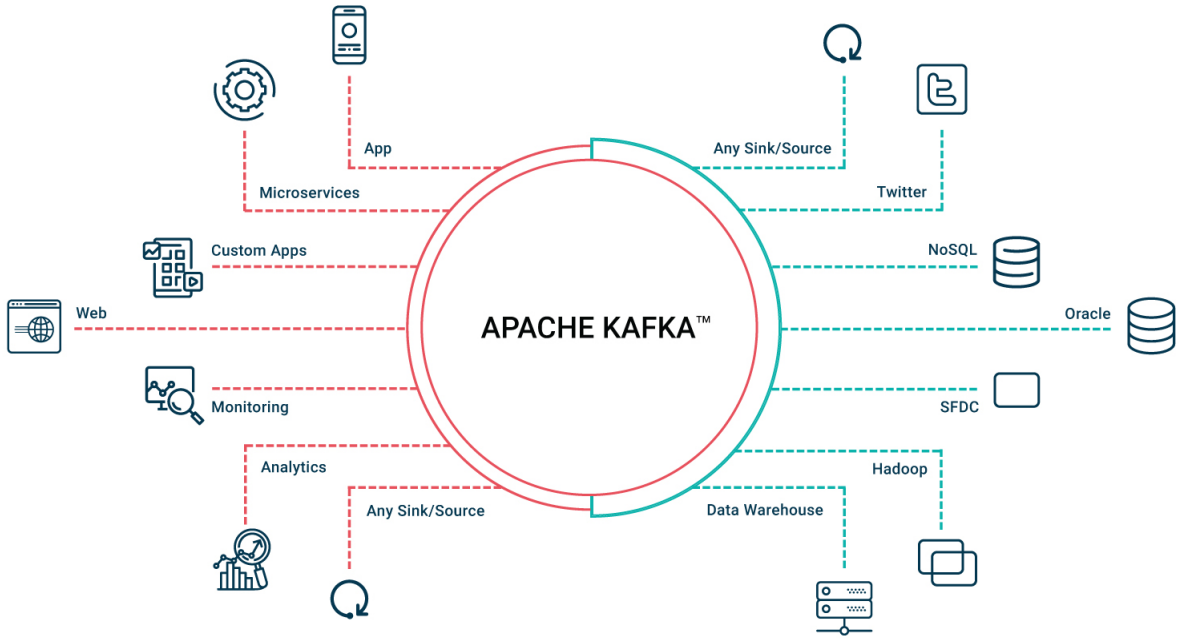


Figura 2.7: Imagen de interconexiones con Apache Kafka[10].

2.4 Alternativas como sistemas de gestión de colas

Algunas alternativas como sistemas de gestión de colas son las herramientas de seguimiento de incidencias y herramientas de BPM (Business Process Management, Gestion de procesos) que con ciertas configuraciones te permiten hacer un uso de gestión de tareas y colas pero con estados propios definidos. Algunos de estos sistemas son:

- Appian
- Jira

2.4.1 Appian

Es un software creado en 1999 para mejorar la gestión, los flujos de trabajo, la automatización y la extracción de procesos, posee un robot propio pero en la actualidad lo tiene muy en desuso y se aplica en muy pocos entornos debido a su dificultad de programación.
El sistema de gestión que dispone el sistema permite también se puede aplicar para la gestión de incidencias y cuentas con un gestor propio de colas de trabajo con las que procesas diferentes tareas.
La empresa ha focalizado su trabajo en el desarrollo de la aplicación de gestión que te permite cierto grado de personalización pero sigue obligando a que los flujos de trabajo estén definidos en su plataforma. Dispone de Plugins para integrarse con Blue Prism o UiPath por ejemplo.

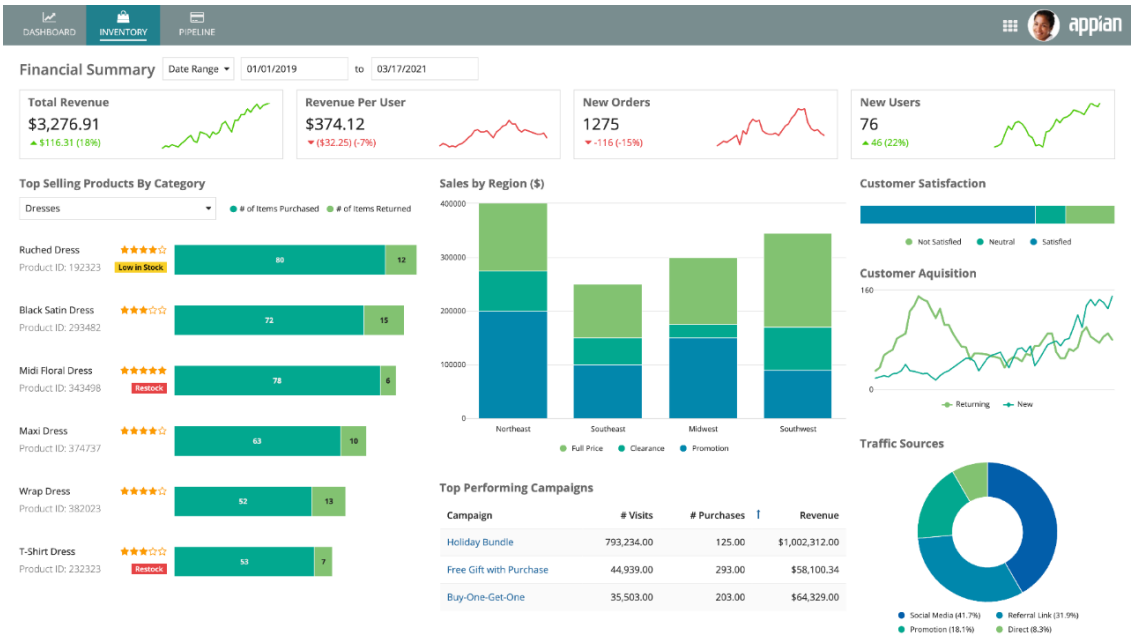


Figura 2.8: Diagrama de resultados en Appian[11]

2.4.2 Jira

Es un software creado en 2002 por Atlassian que se usa generalmente la gestión de proyecto y el seguimiento de errores e incidencias.
Dentro del MarcketPlace de Jira existe una extensión que convierte las issues[7] en un gestor de colas capaz de tratar de manera que cuando se asigna un caso a un usuario robot este pasa a bloquearse y que no pueda ser solicitado por otro evitando duplicados. El problema de este sistema es que no se puede controlar desde el propio flujo la cronología de estados del caso.
Jira es un software que se puede instalar tanto en un servidor Windows como Linux y cuenta con una pequeña versión gratuita y con una versión Cloud, su fuente de ingresos principal es el cobro por licencia y el uso de la plataforma Cloud.

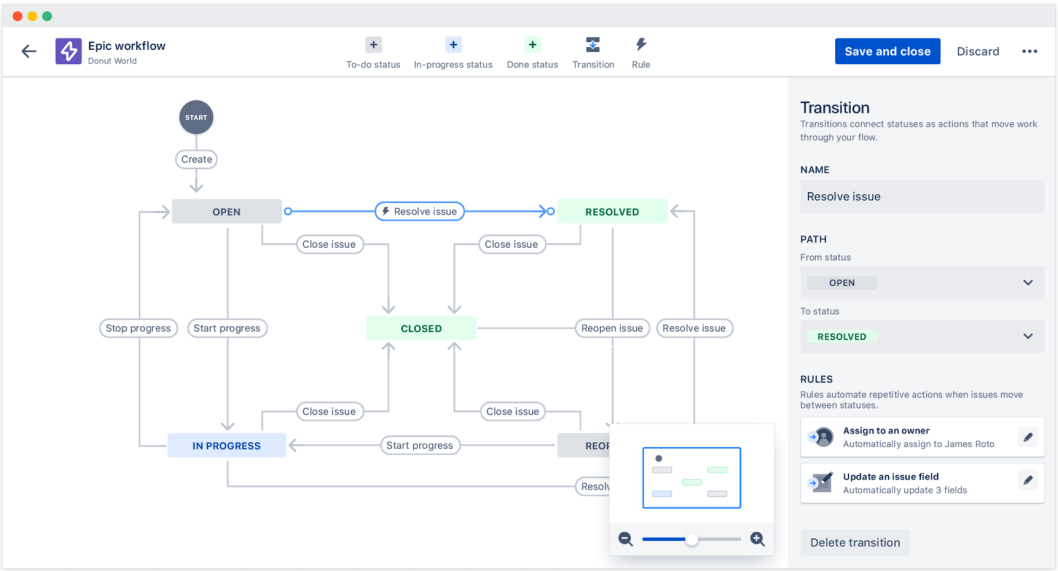


Figura 2.9: Editor de diagramas de estado de Jira[12]

[7]En Jira las solicitudes o peticiones reciben el nombre de ïssues"

2.5 Técnicas utilizadas

2.5.1 Metodología Agile

Se decide tomar una iniciativa de desarrollo ágil para poder aplicar todos los conocimientos aprendidos durante el curso y que facilite ver la estabilidad del sistema e ir realizando pruebas de los desarrollos. Esta metodología permite que con pequeñas modificaciones se pueda ir realizando pruebas hasta llegar a la idea final sin tener que replantear el proceso una vez finalizado.

Figura 2.10: Diagrama Metodología Agile.

2.5.2 Elementos Modularizables o Plugins

El desarrollo de la web se intentará crear por módulos que sean fácilmente editables. Teniendo la mayor bondad de toda aplicación es su fácil incorporar elementos que se puedan añadir y quitar sin que afecte al resto del sistema. Por eso cada representación se realiza en módulos permitiendo que se puedan representar cada una de las variables de manera independiente.

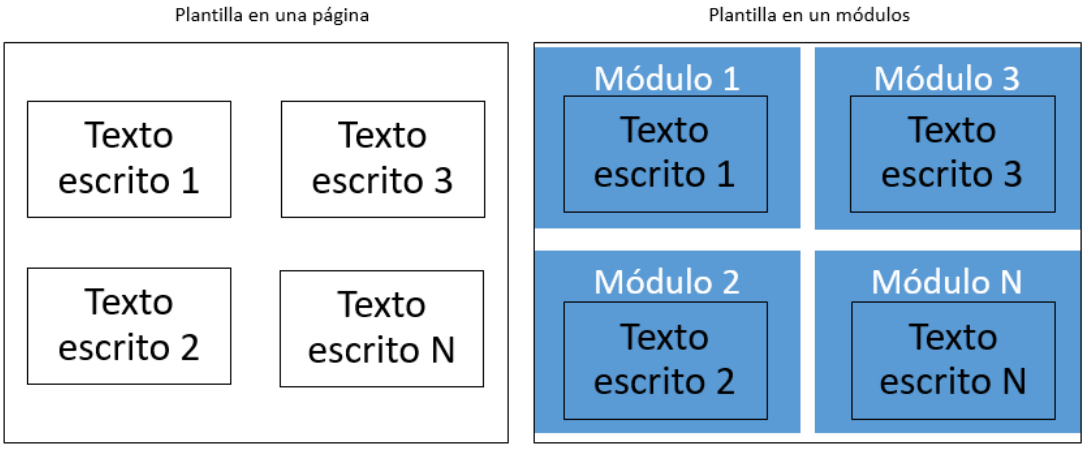


Figura 2.11: Comparativa desarrollo web modular.

Cada módulo se desarrolla como una página independiente permitiendo crear, eliminar o editar cada una de las partes sin interferir en el resto. Cada una de las dependencias de cada módulo se añaden en ese módulo (CSS, JavaScripts, etc.).

2.5.3 Principio SOLID

El principio SOLID es un acrónimo formado por una serie de principios:

- SRP (Single Responsibility Principle, la noción de que un objeto solo debería tener una única razón para cambiar.)
- OCP (Open/closed Principle, la noción de que las debe estar permitido modificarse con facilidad y ser adaptable sin depender de una entidad.)
- LSP (Liskov Substitution Principle, la noción de que un objecto se reemplace con otro y no afecte al funcionamiento del programa.)
- ISP (Interface Segregation Principle, la noción de que las adaptaciones de interfaces cliente específicas son mejores que una interfaz de propósito general.)
- DIP (Dependency inversion Principle, la noción de que se debe depender de abstracciones, no depender de la implementación que se realizará.)

SOLID aporta este conjunto de principios con el fin de mejor la utilización de la programación orientada a objectos que usan los desarrolladores, permitiendo que escriban una gran cantidad de código en poco tiempo, de calidad, reduciendo el mantenimiento y la cantidad de miembros del equipo que conozcan ese software.

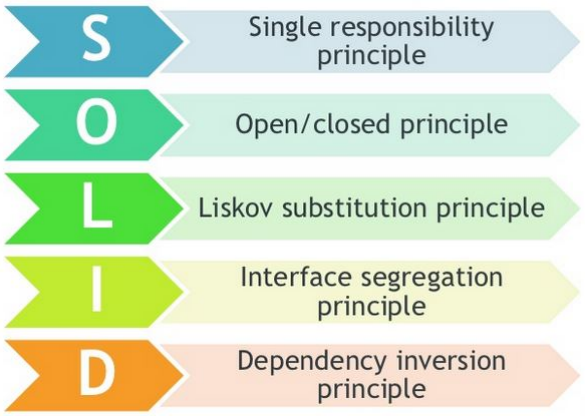


Figura 2.12: Acrónimos SOLID[13].

2.6 Estado del Arte

2.6.1 Lenguaje Java y Spring Boot

El lenguaje Java surgió a principios de 1995, aunque su creación comenzó en 1991. Fue desarrollado por Sun Microsystems con el fin el fin de dar nuevos enfoques y facilidades a los desarrolladores para la creación de aplicaciones.
Actualmente es uno de los lenguajes más utilizados, se aplica tanto en aplicaciones móviles como en servicios web, existen multitud de sistemas que funcionan con este lenguaje y existen algunas páginas web que precisan de tener instalado el intérprete para que se visualicen las web.
Es un lenguaje que soporta programación orientada a objecto, programación imperativa y de sentido de lenguaje estricto. Permitiendo el desarrollo bajo el principio de MVC (Modelo-Vista-Controlador).
Se ha elegido la versión 17, dado que la versión de Java 18 salió oficialmente en marzo del 2022.
Dentro de Java han ido apareciendo muchos Frameworks de desarrollo, la aplicada en este desarrollo, Spring Boot, surgió en 2002 para desarrollo de entornos web y contenedores.

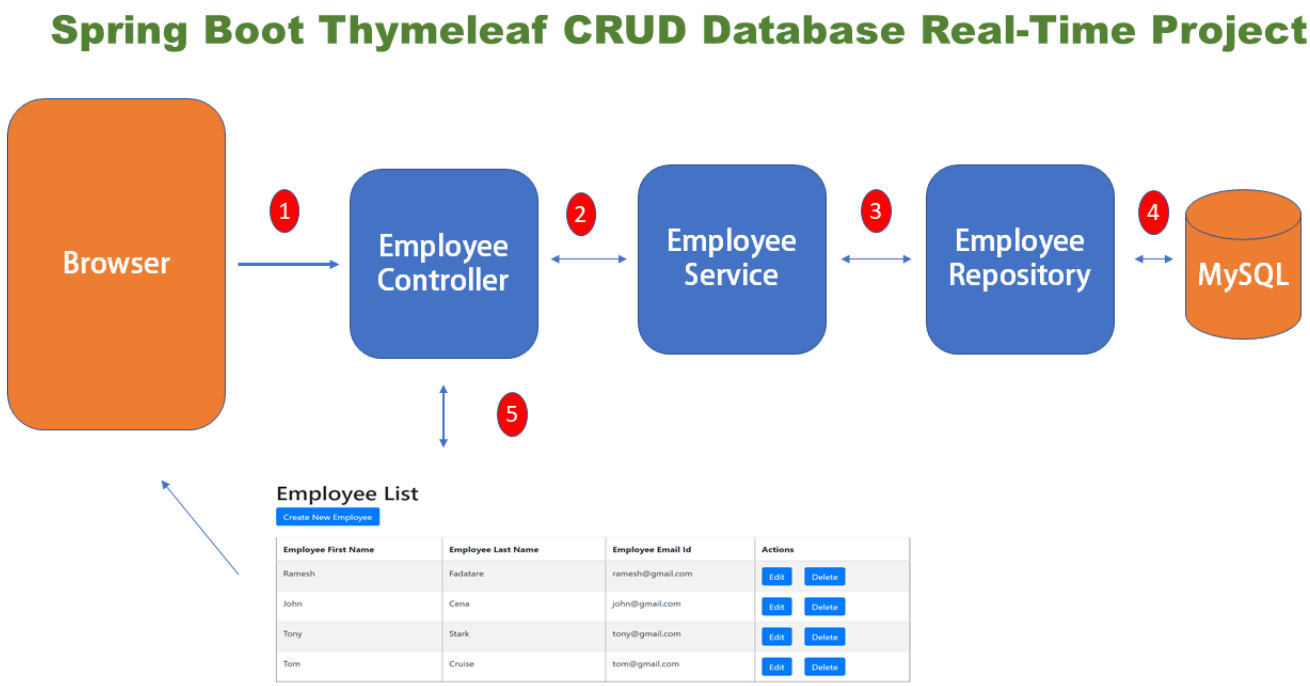


Figura 2.13: Flujo de arquitectura MVC de Spring Boot.

2.6.2 MySQL

Es un sistema de base de datos relacional desarrollado por *Oracle Corporation*. Es uno de los gestores más utilizados al ser gratuito, siendo muy fácil de usar, bastante rápida, diferentes capas de seguridad y capaz de funcionar con pocos requerimiento y utilizando su memoria de manera eficaz. Además tiene flexibilidad con Sistemas Linux y Windows. Fue escogida debido a estas cualidades y a su comportamiento eficiente en el trabajo de procesamiento de transacciones y su capacidad de consulta.

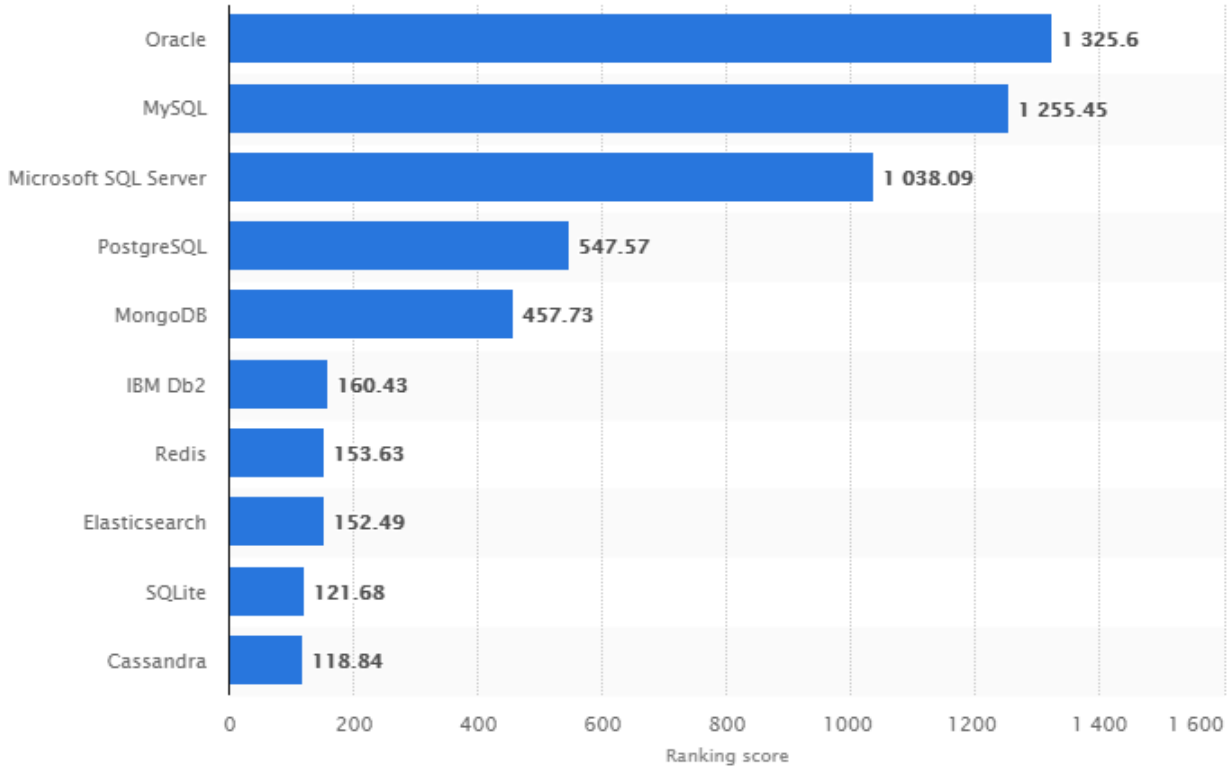


Figura 2.14: Estadísticas de popularización de base de datos sacadas de la página www.statista.com en diciembre 2020.

2.6.3 Gráficas HTML

Existen en la actualidad diferentes librerías para la implementación y visualización de gráficas en visualizaciones HTML.
Para la creación de gráficas en HTML se usará la librería Chart.js de software libre, para poder realizar visualizaciones de los datos de manera gráfica sin demasiada complejidad. Para ello se trabajará con dos versiones combinando ambas en una sola:

- Versión 2.9: Versión estable del proceso, con muchas visualizaciones diferentes como:
  - Gráficas de barras
  - Gráficas de línea
  - Gráficas de área
  - Gráficas de rosquilla
  - Gráficas de logarítmicas

Desde su web Chart.js se puede aprender con facilidad a utilizar dicha librería, además de contar con ejemplos y guías de aprendizaje.

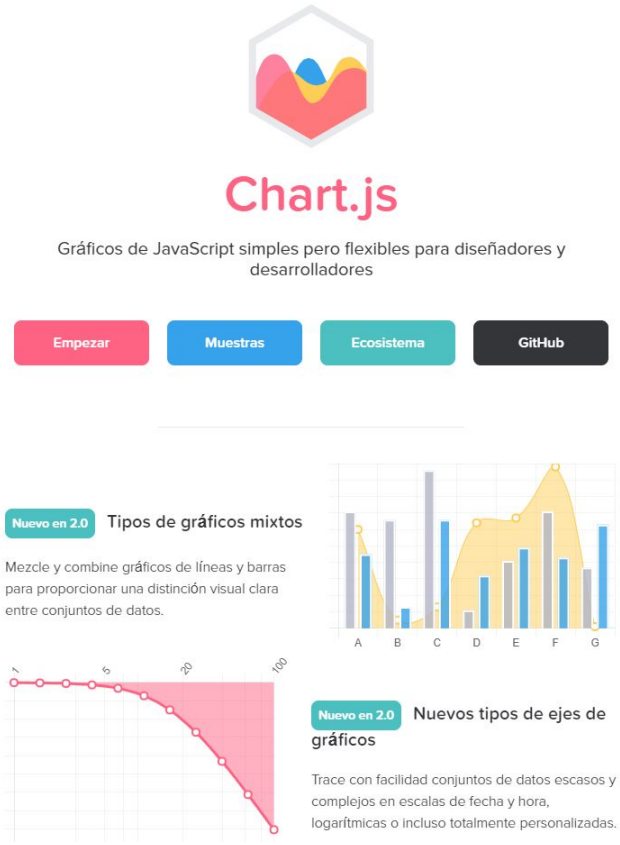


Figura 2.15: Página web Chart.js.

2.7 Conclusiones

La combinación de todas estas técnicas, el conocimiento aprendido de los elementos similares existentes en el mercado y las tecnologías que se utilizarán en el desarrollo, serán la parte fundamental del desarrollo de este proyecto.
Tener una herramienta que sea fácilmente adaptable al cliente suele ser difícil de encontrar y difícilmente se adapta a todas las necesidades.

Capítulo 3

Desarrollo

La vida es muy simple pero insistimos en hacerla complicada.

Confucio

3.1 Introducción

Al seguir una metodología Agile, se puede dividir el desarrollo en fases:

- Fase 1: Prerrequisitos
- Fase 2: Preparación del sistema
- Fase 3: Desarrollo tablas y procedimientos almacenados.
- Fase 4: Desarrollo Backend.
- Fase 5: Desarrollo Frontend
- Fase 6: Desarrollo Plugin Blue Prism

3.1.1 Planificación

El desarrollo del proyecto se va a realizar según una metodología ágil de Sprint por apartado o herramienta, en el cual las premisas principales son el desarrollo de trozos de código y la comprobación de funcionamiento con los desarrollos de apartados anteriores al implantarlo.
Para realizar este proyecto se divide en 4 Sprint y un 5 Sprint para documentaciones y revisión global del funcionamiento, cada Sprint será aproximadamente de 2 semanas.

Primer Sprint	Segundo Sprint	Tercer Sprint	Cuarto sprint	Quinto Sprint
Preparación del entorno	Creación del servicio gestor	Creación del visualizador	Creación de plugin de Blue Prism	Revisión del funcionamiento completo con set de pruebas completas
Desarrollo procedimientos almacenados en MySQL	Adaptar base de datos a los cambios.	Adaptación del servicio gestor y de la base de datos	Adaptación del servicio gestor y de la base de datos	Documentaciones.
	Revisión mediante Postman del funcionamiento del gestor	Revisión del funcionamiento del gestor y del visualizador con navegador web	Revisión del funcionamiento del gestor con el plugin y al poder interactuar simultáneamente con visualizador	

Figura 3.1: Planificación de los Sprint.

3.2 Prerrequisitos

3.2.1 Prerequisitos para el desarrollo

Para el funcionamiento y funcionamiento del sistema será necesario dispones de:

- Disponer de un sistema operativo Windows sobre el que pueda ejecutar el software a desarrollar.
- Los requisitos para el desarrollo del Backend son:
  - Java JDK 17 o superior.
  - Postman o similar.
  - Spring Boot 2.7.3.
  - Spring Boot Dev Tools.
  - Spring Web.
  - Spring Data JPA.
  - MySQl Driver.

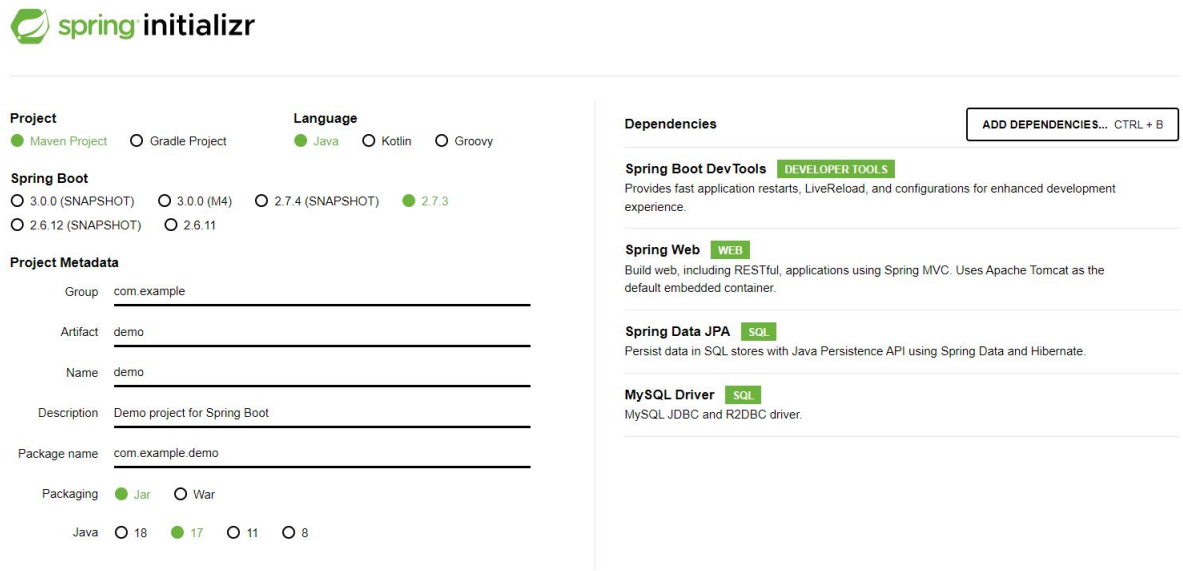


Figura 3.2: Imagen de Backend usando Spring Initializr[5].

- Los requisitos para el desarrollo del Frontend son:
  - Java JDK 17 o superior.
  - Navegador web.
  - Spring Boot 2.7.3.
  - Spring Boot Dev Tools.
  - Spring Web.
  - Spring Data JPA.
  - Thymeleaf.
  - Spring Security (Por si se desea crear perfiles y autentificación web).

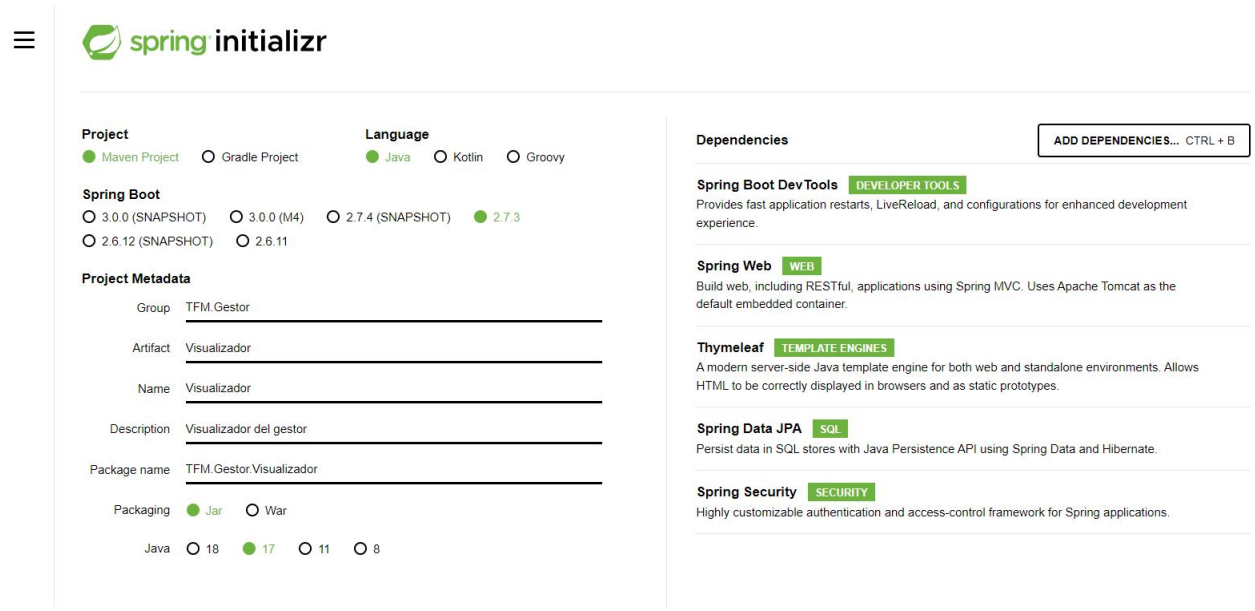


Figura 3.3: Imagen de Frontend usando Spring Initializr[5].

- Base de datos MySQL y MySQL WorkBench (o similar) con permisos de creación de tablas, procedimientos almacenados y permisos de lectura y escritura sobre la BBDD.
- Software de Blue Prism con licencia para poder realizar el desarrollo del plugin

3.2.2 Prerrequisitos del sistema

Para el funcionamiento y funcionamiento del sistema será necesario dispones de:

- Disponer de un sistema operativo Linux/Windows/Mac sobre el que pueda ejecutar el software que se ha desarrollado.
- Requisitos:
  - Java JRE 17 o superior.
  - Navegador web
- Base de datos MySQL con permisos de creación de tablas (o las tablas ya creadas) y permisos de lectura y escritura sobre la BBDD.
- Maquinas robot con el plugin de Blue Prism y software de Blue Prism instalado.

3.2.3 Otros Pre-Requisitos

La mayoría de las herramientas de RPA están desarrolladas para sistemas Windows, por lo tanto será necesario para poder ejecutar una versión de Windows 7 o superior y el software y la licencia asociada a los robots.

Figura 3.4: Imagen de escritorio de un Windows 10[6].

3.2.4 Compilación

Se necesitará una versión de Java JRE 17 o superior para ejecutar el software y una máquina con sistema Windows 7 o superior y con licencia de Blue Prism 6.5 o superior y el plugin instalado.

3.3 Preparación del sistema

Para preparar el sistema realizamos los siguientes pasos:

1. Instalación de Java JDK 17
2. Acceder a https://start.spring.io/[5] para crear los paquetes de las aplicaciones.
3. Instalación del entorno de desarrollo IntelliJ IDEA.
4. Instalación de la herramienta Postman.
5. Instalación de la herramienta MySQL WorkBench
6. Creación de la base de datos MySQL

Con estas premisas ya podemos iniciar el desarrollo, para ejecutar los proyecto

3.3.1 Creación de la base de datos MySQL

Se utiliza un contenedor Docker con el que invocar la creación de la bases de datos MySQL:

Listado 3.1: Comando Docker para crear contenedor MySQL

```
docker run -d -p 33061:3306 –name mysql57 -e MYSQL_ROOT_PASSWORD=secret mysql:5.7
–character-set-server=utf8mb4 -collation-server=utf8mb4_unicode_ci
```

3.4 Desarrollo tablas y procedimientos almacenados

Para la creación de este proyecto se utiliza el desarrollo "Model first"[1] basado en un 3 tablas para la estructuración de los procesos.

3.4.1 Creación de tablas

Para la creación de tablas se crea una estructura de dependencia entre las tabla de tareas y los datos de los casos teniendo la posibilidad de tener una relación de muchos a uno y una relación no necesariamente vinculada de la tabla de máquinas de muchos a uno.

Listado 3.2: Creación de tabla de datos.

```
CREATE TABLE `datos` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `valor1` decimal(20,4) NOT NULL,
  `valor2` decimal(20,4) NOT NULL,
  `operando` varchar(1) COLLATE utf8mb4_unicode_ci NOT NULL,
  `resultados` decimal(45,8) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=12 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
CREATE DEFINER=`root`@`%` TRIGGER `gestor`.`datos_AFTER_INSERT` AFTER INSERT ON `datos` FOR
    EACH ROW
BEGIN
  IF new.id IS NOT NULL THEN
    INSERT INTO gestor.tareas (tareas.iddatos) VALUES (new.id);
  END IF;
END
```

[1] Model first: es la construcción de un proceso MVC (Modelo-Vista-Controlador) con la estructuración primero de la base de datos y luego la construcción del software que lo aplica.

Listado 3.3: Creación de tabla de máquinas.

```
CREATE TABLE `maquinas` (
  `idmaquinas` int(11) NOT NULL AUTO_INCREMENT,
  `hastname` varchar(45) COLLATE utf8mb4_unicode_ci NOT NULL,
  `procedencia` varchar(45) COLLATE utf8mb4_unicode_ci NOT NULL,
  `estado` tinyint(4) NOT NULL DEFAULT '0',
  PRIMARY KEY (`idmaquinas`),
  UNIQUE KEY `idmaquinas_UNIQUE` (`idmaquinas`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Listado 3.4: Creación de tabla de tareas.

```
CREATE TABLE `tareas` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `iddatos` int(11) NOT NULL,
  `idmaquina` int(11) DEFAULT NULL,
  `creacion` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `actualizacion` datetime DEFAULT CURRENT_TIMESTAMP,
  `finalizacion` datetime DEFAULT NULL,
  `resultado` tinyint(4) DEFAULT NULL,
  `solicitado` tinyint(4) DEFAULT '0',
  `prioridad` int(10) unsigned NOT NULL DEFAULT '0',
  `error` varchar(45) COLLATE utf8mb4_unicode_ci DEFAULT '',
  PRIMARY KEY (`id`),
  KEY `datos_idx` (`iddatos`),
  KEY `maquinas_idx` (`idmaquina`),
  CONSTRAINT `datos` FOREIGN KEY (`iddatos`) REFERENCES `datos` (`id`) ON DELETE NO ACTION ON
      UPDATE NO ACTION,
  CONSTRAINT `maquinas` FOREIGN KEY (`idmaquina`) REFERENCES `maquinas` (`idmaquinas`) ON
      DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=20 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

3.4.2 Procedimientos almacenados

En este desarrollo los procedimientos almacenados son fundamentales dado que permiten que puedas simplificar funciones a crear desde la programación y permiten ser independientes del lenguaje de programación posterior para el desarrollo del gestor.
El procedimiento para solicitar casos permite solicitar un caso sin que otra petición interfiera y no se envíe por duplicado a dos máquinas.

Listado 3.5: Creación del procedimiento para solicitar casos.

```
DELIMITER $$
CREATE DEFINER=`root`@`%` PROCEDURE `solicitar_caso`(IN `maquina` int(11))
BEGIN
  START TRANSACTION;
    DROP table IF EXISTS  TablasTemp;
        CREATE temporary table TablasTemp (
      `idtarea` int(11),
      `idcaso` int(11),
      `valor1` decimal(20,4),
      `valor2` decimal(20,4),
      `operando` varchar(1),
      `resultados` decimal(45,8));
    SET @filtro = IF((SELECT COUNT(*) from gestor.tareas where tareas.solicitado = 0 AND
    tareas.resultado is null AND idmaquina=maquina) >= 1, maquina, 0);
    INSERT INTO TablasTemp (SELECT tareas.id AS idtarea,datos.* from gestor.tareas
      INNER JOIN gestor.datos ON tareas.iddatos = datos.id
        WHERE tareas.solicitado = 0 AND tareas.resultado is null AND tareas.finalizacion is
            null AND
          CASE when @filtro = 0
            THEN tareas.idmaquina is null
            ELSE tareas.idmaquina = maquina
          END
        ORDER BY tareas.prioridad DESC LIMIT 1);
        UPDATE tareas SET tareas.idmaquina = maquina,tareas.solicitado = 1
      WHERE tareas.id = (SELECT idtarea from TablasTemp LIMIT 1);
        SELECT * from TablasTemp;
        DROP table IF EXISTS  TablasTemp;
    COMMIT;
END$$
DELIMITER ;
```

El procedimiento para reiniciar un caso realiza un control que evita que la tarea se duplique.

Listado 3.6: Creación de tabla de tareas.

```
DELIMITER $$
CREATE DEFINER=`root`@`%` PROCEDURE `reiniciar_caso`(IN `idcaso` int(11))
BEGIN
  START TRANSACTION;
    SET @filtro = IF((SELECT COUNT(*) from gestor.tareas where tareas.finalizacion is null AND
      iddatos=idcaso) < 1, 1, 0);
    IF @filtro = 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error reiniciar caso';
    END IF;
    INSERT INTO tareas (iddatos) VALUES (idcaso);
    COMMIT;
END$$
DELIMITER ;
```

El procedimiento para guardar la respuesta simplifica la actualización en tablas.

Listado 3.7: Creación de tabla de tareas.

```
DELIMITER $$
CREATE DEFINER=`root`@`%` PROCEDURE `respuesta_robot`(
  IN `maquina` int,
  IN `intento` int,
  IN `resultado` bool,
    IN `reintento` bool,
    IN `mensaje` nvarchar(45))
BEGIN
  START TRANSACTION;
    SET @contador = (SELECT COUNT(*) FROM gestor.tareas WHERE tareas.idmaquina = maquina AND
        tareas.id = intento AND tareas.solicitado = 1 AND tareas.resultado is null);
    IF @contador = 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error finalizacion de caso no permitida';
    ELSE
      IF resultado THEN
        UPDATE gestor.tareas SET tareas.finalizacion = current_timestamp(), tareas.resultado =
            resultado WHERE tareas.id = intento;
      ELSE
        UPDATE gestor.tareas SET tareas.finalizacion = current_timestamp(), tareas.resultado =
            resultado, tareas.error = mensaje WHERE tareas.id = intento;
      END IF;
      IF reintento AND resultado is false THEN
        INSERT INTO gestor.tareas (iddatos,solicitado,prioridad) SELECT tareas.iddatos, 0,
            tareas.prioridad from gestor.tareas WHERE tareas.id = intento;
      END IF;
    END IF;
  COMMIT;
END$$
DELIMITER ;
```

3.5 Desarrollo Backend

La estructuración de la aplicación que realiza de backend realiza la gestión de la comunicación con la base de datos y como gestor de comunicación con los automatismos. Para tener el código organizado se sigue la estructura de:

- Modelos: creación de las clases y entidades a utilizar para el funcionamiento del proceso.
- DAO (Data Access Objects,objetos de acceso a datos): para la creación de las consultas a MySQL.
- Servicio: para preparar las consultas de los controladores
- Controlador: son los Endpoints para interactuar con la aplicación.

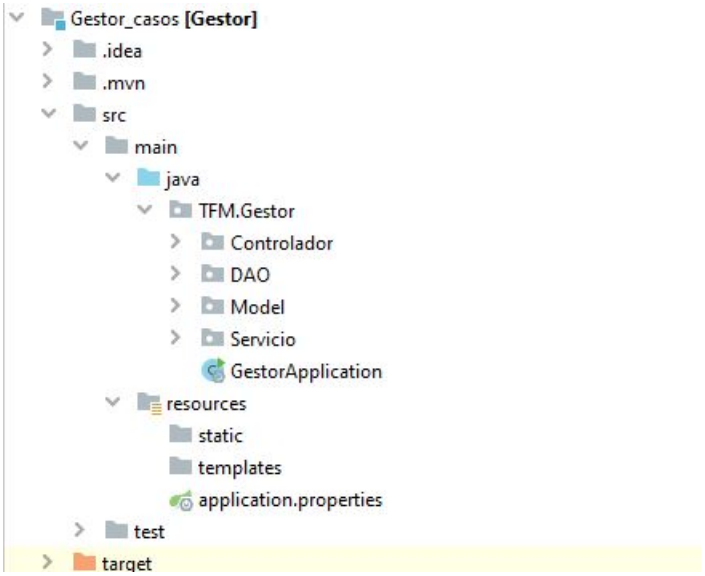


Figura 3.5: Estructura de paquetes del proyecto de backend.

3.5.1 Modelos

Se diseñan los modelos y entidades con el fin de manejar una estructura de datos cómoda para realizar las diferentes operaciones y que funcione correctamente la aplicación. Los modelos creados son:

- ActualizarCaso: se utiliza para actualizar un caso .
- ActualizarCasoRowMapperDAO: se utiliza para serializar la clase para actualizar datos y tareas.
- ActualizarTarea: se utiliza para actualizar un tareas y resultados de los datos.
- ActualizarTarea_Simple: versión reducida de ActualizarTarea.
- ContadorRowMapperDAO: se utiliza para realizar comprobaciones mediante contadores.
- DatosCompleted: se utiliza para realizar la acciones con la tabla de casos.
- DatosRowMapperDAO: se utiliza para serializar la clase datos.
- Maquinas: se utiliza para realizar la acciones con la tabla de máquinas.
- MaquinasRowMapperDAO: se utiliza para serializar la clase máquinas.

- MensajesSimple: se utiliza para indicar el tipo de mensaje de texto si es positivo o negativa la respuesta.
- nuevo_caso: Se usa para crear la estructura de datos recibida para la creación de casos en la tabla de datos.
- PorcentajesCasos: se utiliza para realizar la extracciones de los totales con la tabla de tareas y datos.
- PorcentajesRowMapperDAO: se utiliza para serializar la clase de porcentajes.
- Tareas:se utiliza para realizar la acciones con la tabla de tareas.
- TareasRowMapperDAO:se utiliza para serializar la clase tareas.

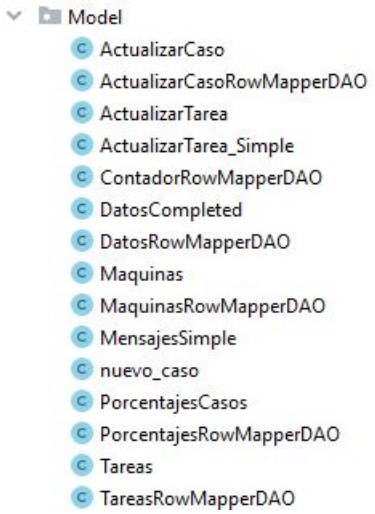


Figura 3.6: Clases del paquete de modelos.

Listado 3.8: Ejemplo clase: Modelo de mensajes simple.

```
package TFM.Gestor.Model;

public class MensajesSimple {
    public Boolean getResultado() {
        return resultado;
    }

    public void setResultado(Boolean resultado) {
        this.resultado = resultado;
    }

    public String getMensaje() {
        return mensaje;
    }

    public void setMensaje(String mensaje) {
        this.mensaje = mensaje;
    }

    Boolean resultado;
    String mensaje;

}
```

Listado 3.9: Ejemplo serializador: Serializador de la clase máquinas.

```
package TFM.Gestor.Model;

import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Component;

import java.sql.ResultSet;
import java.sql.SQLException;

@Component
public class MaquinasRowMapperDAO implements RowMapper<Maquinas> {
    @Override
    public Maquinas mapRow(ResultSet rs, int rowNum) throws SQLException {
        Maquinas maquina = new Maquinas();

        maquina.setIdmaquina(Integer.parseInt(rs.getString("idmaquinas")));
        maquina.setHastname(rs.getString("hastname"));
        maquina.setProcedencia(rs.getString("procedencia"));
        maquina.setEstado(rs.getBoolean("estado"));
        return maquina;
    }
}
```

3.5.2 DAO

Los objetos de acceso a datos de las bases de datos son fundamentales para los proyectos de MVC y por ello es necesario tener una construcción de cada las llamadas con una estructura y definición correctas.

Siempre se crea una interfaz o repositorios y una clase de interpretación.

- IDatosDAO e IDatosDAOImpl: son el interfaz y la clase para los datos de las tablas de datos y tareas que están directamente relacionadas.
- IMaquinasDAO e IMaquinasDAOImple: son el interfaz y la clase para los datos de la tabla de máquinas.

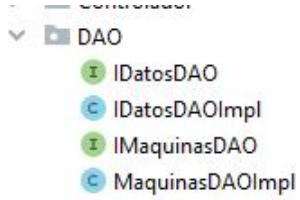


Figura 3.7: Interfaces y clases del paquete de DAO.

Listado 3.10: Ejemplo interfaz: Interfaz de máquinas.

```
1  package TFM.Gestor.DAO;
2
3  import TFM.Gestor.Model.DatosCompleted;
4  import TFM.Gestor.Model.Maquinas;
5
6  import java.util.List;
7
8  public interface IMaquinasDAO {
9      Maquinas buscar_maquina_id(Integer id);
10     Maquinas buscar_maquina(Integer id,String hastname);
11     Maquinas buscar_maquina_hp(Maquinas maquina);
12
13     Maquinas buscar_maquina_activa(Integer id,String hastname);
14     List<Maquinas> listado();
15     Maquinas crear_maquina(Maquinas maquina);
16     Maquinas cambiarEstado(Integer id, boolean estado);
17
18
19 }
```

Listado 3.11: Ejemplo clase: Clase de implementación del interfaz de IMaquinasDAO, parte 1.

```
1  package TFM.Gestor.DAO;
2
3  import TFM.Gestor.Model.DatosCompleted;
4  import TFM.Gestor.Model.DatosRowMapperDAO;
5  import TFM.Gestor.Model.Maquinas;
6  import TFM.Gestor.Model.MaquinasRowMapperDAO;
7  import org.springframework.beans.factory.annotation.Autowired;
8  import org.springframework.jdbc.core.JdbcTemplate;
9  import org.springframework.stereotype.Repository;
10
11 import java.util.ArrayList;
12 import java.util.List;
13
14 @Repository
15 public class MaquinasDAOImpl implements IMaquinasDAO{
16     @Autowired
17     private JdbcTemplate jdbcTemplate;
18     @Autowired
19     private MaquinasRowMapperDAO maquinasRowMapperDAO;
20
21     @Override
22     public Maquinas buscar_maquina(Integer id,String hastname){
23         Maquinas maquina = new Maquinas();
24         String query = String.format("SELECT idmaquinas,hastname,procedencia,estado from
               maquinas where idmaquinas = %d AND hastname='%s'",id,hastname);
25         final List<Maquinas> maquinas  = jdbcTemplate.query(query, maquinasRowMapperDAO);
26         try{
27             maquina=maquinas.get(0);
28         }
29         catch (Exception e){
30             maquina = new Maquinas();
31         }
32         return maquina;
33     }
34     @Override
35     public Maquinas buscar_maquina_id(Integer id){
36         String query = String.format("SELECT idmaquinas,hastname,procedencia,estado from
               maquinas where idmaquinas = %d",id);
37         final List<Maquinas> maquinas  = jdbcTemplate.query(query, maquinasRowMapperDAO);
38         Maquinas maquina = new Maquinas();
39         try{
40             maquina=maquinas.get(0);
41         }
42         catch (Exception e){
43             maquina = new Maquinas();
44         }
45         return maquina;
46     }
47     @Override
48     public Maquinas buscar_maquina_hp(Maquinas maquina){
49         String query = String.format("SELECT idmaquinas,hastname,procedencia,estado from
               maquinas where procedencia = '%s' AND hastname='%s'",maquina.getProcedencia(),
               maquina.getHastname());
50         final List<Maquinas> maquinas  = jdbcTemplate.query(query, maquinasRowMapperDAO);
51         try{
52             maquina=maquinas.get(0);
53         }
54         catch (Exception e){
55             maquina = new Maquinas();
```

Listado 3.12: Ejemplo clase: Clase de implementación del interfaz de IMaquinasDAO, parte 2.

```
1           }
2           return maquina;
3       }
4       @Override
5       public Maquinas buscar_maquina_activa(Integer id,String hastname){
6           String query = String.format("SELECT idmaquinas,hastname,procedencia,estado from
                maquinas where estado = 1 AND idmaquinas = %d AND hastname='%s'",id,hastname);
7           final List<Maquinas> maquinas  = jdbcTemplate.query(query, maquinasRowMapperDAO);
8           Maquinas maquina = new Maquinas();
9           try{
10              maquina=maquinas.get(0);
11          }
12          catch (Exception e){
13              maquina = new Maquinas();
14          }
15          return maquina;
16      }
17      @Override
18      public List<Maquinas> listado(){
19          String query = "SELECT sql_no_cache idmaquinas,hastname,procedencia,estado from
                maquinas";
20          final List<Maquinas> maquinas  = jdbcTemplate.query(query, maquinasRowMapperDAO);
21
22          return maquinas;
23      }
24      @Override
25      public Maquinas crear_maquina(Maquinas maquina){
26          jdbcTemplate.execute("SET autocommit = 0");
27          String query = "INSERT INTO maquinas (procedencia,hastname) VALUES (?,?)";
28          jdbcTemplate.update(query, maquina.getProcedencia(),maquina.getHastname());
29          jdbcTemplate.execute("COMMIT");
30          return buscar_maquina_hp(maquina);
31      }
32      @Override
33      public Maquinas cambiarEstado(Integer id, boolean estado){
34          jdbcTemplate.execute("SET autocommit = 0");
35          String query = "UPDATE maquinas SET estado = ? WHERE idmaquinas = ?";
36          jdbcTemplate.update(query,estado, id);
37          jdbcTemplate.execute("COMMIT");
38          return buscar_maquina_id(id);
39      }
40  }
```

3.5.3 Servicio

Las clases e interfaces del paquete de servicio hacen de pasarela para que los controladores pueden utilizar las clases del paquete DAO. En estos objetos se suelen realizar las lógicas de negocio.

- IDatosService e DatosServiceImpl: son el interfaz y la clase para construcción de los servicios que se consumirán desde los controladores.

Figura 3.8: Interfaces y clases del paquete de servicio.

Listado 3.13: Interfaz del servicio.

```
1  package TFM.Gestor.Servicio;
2
3  import TFM.Gestor.Model.*;
4
5  import java.math.BigDecimal;
6  import java.util.List;
7
8  public interface IDatosService {
9
10     List<DatosCompleted> listado_resultado();
11     Boolean insertar_valor(Double valor1, Double valor2, String operando);
12
13     MensajesSimple borrar_caso(Integer id);
14     MensajesSimple actualizar_estado_caso(ActualizarTarea tarea,Double resultado,Boolean
           revisar_maquina);
15     MensajesSimple actualizar_caso(ActualizarCaso caso,Boolean revisar_maquina);
16     ActualizarCaso solicitar_caso (String procedencia, String hastname);
17     Maquinas crear_maquina(Maquinas maquina);
18     Maquinas actualizar_maquina(Maquinas maquina);
19     Integer leer_intentos(Integer id);
20     MensajesSimple reiniciar_caso(Integer id);
21     MensajesSimple excepcionar_caso(Integer id,String mensajeError);
22     MensajesSimple desbloquear_caso(Integer id,Boolean revisar_maquina,ActualizarTarea tarea);
23     PorcentajesCasos porcentajesCasos();
24
25     List<Maquinas> listado_maquinas();
26     List<Tareas> listado_tareas();
27     MensajesSimple asingar_maquina(ActualizarTarea tarea);
28 }
```

Listado 3.14: Clase de implementación del interfaz como servicio, parte 1.

```
1  package TFM.Gestor.Servicio;
2
3  import TFM.Gestor.DAO.IDatosDAO;
4  import TFM.Gestor.DAO.IMaquinasDAO;
5  import TFM.Gestor.Model.*;
6  import org.springframework.beans.factory.annotation.Autowired;
7  import org.springframework.stereotype.Service;
8
9  import java.math.BigDecimal;
10 import java.util.List;
11
12 @Service
13 public class DatosServiceImpl implements IDatosService{
14
15     @Autowired
16     IDatosDAO datosDAO;
17     @Autowired
18     IMaquinasDAO maquinasDAO;
19     @Override
20     public List<DatosCompleted> listado_resultado(){
21         return datosDAO.listado_resultado();
22     }
23     @Override
24     public Boolean insertar_valor(Double valor1, Double valor2, String operando){
25         try {
26             datosDAO.insertar_valor(valor1, valor2, operando);
27             return true;
28         }
29         catch (Exception e){
30             return false;
31         }
32     }
33     @Override
34     public MensajesSimple borrar_caso(Integer id){
35         MensajesSimple mensaje = new MensajesSimple();
36         try{
37             datosDAO.borrar_caso(id);
38             mensaje.setResultado(true);
39             mensaje.setMensaje(String.format("Se borro el caso %d",id));
40         }
41         catch (Exception e){
42             mensaje.setResultado(false);
43             mensaje.setMensaje(String.format("No se borro el caso %d, %s",id,e.getMessage()));
44         }
45         return mensaje;
46     }
47     @Override
48     public MensajesSimple actualizar_estado_caso(ActualizarTarea tarea,Double resultado,
           Boolean revisar_maquina) {
49         MensajesSimple mensaje = new MensajesSimple();
50         try {
51             if(tarea.getIdTarea()==null){
52                 throw new Exception("No se indico tareas");
53             }
54             if(revisar_maquina){
```

Listado 3.15: Clase de implementación del interfaz como servicio, parte 2.

```
                Maquinas maquina = new Maquinas();
                maquina.setHastname(tarea.getHastname());
                maquina.setProcedencia(tarea.getProcedencia());
                maquina=maquinasDAO.buscar_maquina_hp(maquina);
                if(maquina == null){
                    throw new Exception("Esta mÃ¡quina no existe");
                }
                if(maquina.getEstado()==null){
                    throw new Exception("Esta mÃ¡quina no existe");
                }
                tarea.setIdmaquina(maquina.getIdmaquina());
            }
            datosDAO.actualizar_estado(tarea,resultado);
            mensaje.setResultado(true);
            mensaje.setMensaje(String.format("Se actualizo el caso %d", tarea.getIdTarea()));
        } catch (Exception e) {
            mensaje.setResultado(false);
            mensaje.setMensaje(String.format("No se actualizo el caso %d, %s", tarea.
                getIdTarea(), e.getMessage()));
        }
        return mensaje;
    }
    @Override
    public ActualizarCaso solicitar_caso (String procedencia, String hastname){
        ActualizarCaso caso = new ActualizarCaso();
        Maquinas maquina = new Maquinas();
        maquina.setProcedencia(procedencia);
        maquina.setHastname(hastname);
        try{
            maquina = maquinasDAO.buscar_maquina_hp(maquina);
            if(maquina == null){
                throw new Exception("Esta mÃ¡quina no existe");
            }if(maquina.getEstado() == null){
                throw new Exception("Esta mÃ¡quina no existe");
            }
            if(maquina.getEstado().equals(false)){
                throw new Exception("Esta mÃ¡quina no puede solicitar casos porque esta
                    desactivada");
            }
            caso = datosDAO.solicitar_caso(maquina.getIdmaquina(),hastname);
            caso.setIdmaquina(maquina.getIdmaquina());
            caso.setHastname(maquina.getHastname());
            caso.setMensaje("");
        } catch (Exception e){
            caso.setMensaje(String.format("%s",e.getMessage()));
        }
        return caso;
    }
    @Override
    public Maquinas crear_maquina(Maquinas maquina_origen){
        Maquinas maquina = maquinasDAO.buscar_maquina_hp(maquina_origen);
        if(maquina.getEstado() == null){
            maquina=maquinasDAO.crear_maquina(maquina_origen);
            maquina.setTexto("Esta maquina ya esta creada espere a que se active");
        }
        if(maquina.getEstado().equals(1)){
            maquina.setTexto("Esta maquina activa");
```

Listado 3.16: Clase de implementación del interfaz como servicio, parte 3.

```
        }
        else{
            maquina.setTexto("Esta maquina ya esta creada espere a que se active");
        }
        return maquina;
    }
    @Override
    public Maquinas actualizar_maquina(Maquinas maquina_origen) {
        Maquinas maquina = new Maquinas();
        try {
            maquina = maquinasDAO.buscar_maquina_id(maquina_origen.getIdmaquina());

            if (maquina.getEstado() == null) {
                maquina.setTexto("Esta maquina no existe");
            }
            if (maquina_origen.getEstado().equals(true)) {
                if (maquina.getEstado().equals(true)) {
                    maquina.setTexto("Esta maquina estÃ¡ activa");
                } else {
                    maquina=maquinasDAO.cambiarEstado(maquina_origen.getIdmaquina(),
                        maquina_origen.getEstado());
                    maquina.setTexto("Maquina activa");
                }
            } else {
                if (maquina.getEstado().equals(false)) {
                    maquina.setTexto("Esta maquina estÃ¡ desactiva");
                } else {
                    maquina=maquinasDAO.cambiarEstado(maquina_origen.getIdmaquina(),
                        maquina_origen.getEstado());
                    maquina.setTexto("Maquina desactiva");
                }
            }
        }
        catch (Exception e){
            maquina.setTexto("No se pudo actualizar el estado de la maquina");

        }
        return maquina;
    }

    @Override
    public Integer leer_intentos(Integer id){
        Integer cantidad = -1;
        try {
            cantidad=datosDAO.leer_intentos(id);
        }
        catch (Exception e){}
        return cantidad;
    }
    @Override
    public MensajesSimple reiniciar_caso(Integer id){
        MensajesSimple mensaje = new MensajesSimple();
        try{
            datosDAO.reintentas_caso(id);
            mensaje.setResultado(true);
            mensaje.setMensaje("Caso reiniciado");
        }
        catch (Exception e){
```

Listado 3.17: Clase de implementación del interfaz como servicio, parte 4.

```
            mensaje.setResultado(false);
            mensaje.setMensaje("El caso no se pudo reiniciar");
        }
        return mensaje;
    }
    @Override
    public MensajesSimple asingar_maquina(ActualizarTarea tarea){
        MensajesSimple mensaje = new MensajesSimple();
        try{
            datosDAO.asingar_maquina(tarea.getIdTarea(),tarea.getIdmaquina());
            mensaje.setResultado(true);
            mensaje.setMensaje("Se asigno maquina a la tarea");
        }
        catch (Exception e){
            mensaje.setResultado(false);
            mensaje.setMensaje(String.format("No se pudo asignar maquina a la tarea",e.
                getMessage()));
        }
        return mensaje;
    }
    @Override
    public MensajesSimple excepcionar_caso(Integer id,String mensajeError){
        MensajesSimple mensaje = new MensajesSimple();
        try{
            datosDAO.excepcionar_caso(id,mensajeError);
            mensaje.setResultado(true);
            mensaje.setMensaje("Caso excepcionado");
        }
        catch (Exception e){
            mensaje.setResultado(false);
            mensaje.setMensaje(String.format("El caso no se pudo excepcionar, %s",e.getMessage
                ()));
        }
        return mensaje;
    }
    @Override
    public MensajesSimple desbloquear_caso(Integer id,Boolean revisar_maquina,ActualizarTarea
        tarea){
        MensajesSimple mensaje = new MensajesSimple();

        try{
            if(revisar_maquina){

                Maquinas maquina = new Maquinas();
                maquina.setHastname(tarea.getHastname());
                maquina.setProcedencia(tarea.getProcedencia());
                maquina=maquinasDAO.buscar_maquina_hp(maquina);
                if(maquina == null){
                    throw new Exception("Esta mÃ¡quina no existe");
                }
                if(maquina.getEstado()==null){
                    throw new Exception("Esta mÃ¡quina no existe");
                }
                tarea.setIdmaquina(maquina.getIdmaquina());
                if(!datosDAO.relacion_caso_maquina(maquina.getIdmaquina(),id)){
                    throw new Exception("Esta mÃ¡quina no puede desbloquear esta caso");
                }
```

Listado 3.18: Clase de implementación del interfaz como servicio, parte 5.

```
            }
            datosDAO.desbloquear_caso(id);
            mensaje.setResultado(true);
            mensaje.setMensaje("Caso desbloqueado");
        }
        catch (Exception e){
            mensaje.setResultado(false);
            mensaje.setMensaje(String.format("El caso no se pudo desbloquear, %s",e.getMessage
                ()));
        }
        return mensaje;
    }
    @Override
    public MensajesSimple actualizar_caso(ActualizarCaso caso,Boolean revisar_maquina){
        MensajesSimple mensaje = new MensajesSimple();

        try{
            if(revisar_maquina){

                Maquinas maquina = new Maquinas();
                maquina.setHastname(caso.getHastname());
                maquina.setProcedencia(caso.getProcedencia());
                maquina=maquinasDAO.buscar_maquina_hp(maquina);
                if(maquina == null){
                    throw new Exception("Esta mÃ¡quina no existe");
                }
                if(maquina.getEstado()==null){
                    throw new Exception("Esta mÃ¡quina no existe");
                }
                caso.setIdmaquina(maquina.getIdmaquina());
                if(!datosDAO.relacion_caso_maquina(maquina.getIdmaquina(),caso.getIdtarea())){
                    throw new Exception("Esta mÃ¡quina no puede actualizar esta caso");
                }
            }
            else{
                caso.setIdtarea(datosDAO.get_ultimo_caso(caso.getId()));
            }
            datosDAO.actualizar_caso(caso);
            mensaje.setResultado(true);
            mensaje.setMensaje("Datos del caso actualizados");

        }
        catch (Exception e){
            mensaje.setResultado(false);
            mensaje.setMensaje(String.format("El caso no se pudo actualizar, %s",e.getMessage()
                ));
        }
        return mensaje;
    }
    @Override
    public PorcentajesCasos porcentajesCasos(){
        try{
            return datosDAO.porcentajes_casos();
        }
        catch (Exception e){
            return new PorcentajesCasos();
        }
```

Listado 3.19: Clase de implementación del interfaz como servicio, parte 6.

```
    }
    @Override
    public List<Maquinas> listado_maquinas(){
        return maquinasDAO.listado();
    }
    @Override
    public List<Tareas> listado_tareas(){
        return datosDAO.listado_tareas();
    }
}
```

3.5.4 Controlador

Las clases que se implementan es este paquete hacen de controladores para servicios Rest que reciben las peticiones HTTP (Hypertext Transfer Protocol,Protocolo de Transferencia de Hipertexto) y solicitan la respuesta al servicio y este al DAO.
Cada una de las llamadas o funciones tiene un URL diferente y cada controladores maneja un conjunto de llamadas según su base:

- GestionController: es el controlador de funciones para el visualizador únicamente. Con la base del path /gestion/*".
- DatosController: es el controlador de funciones para gestionar casos y tareas. Con la base del path /datos/*".
- GestionMaquinas: es el controlador de funciones para interactuar con las máquinas, generalmente esto lo utiliza el visualizador. Con la base del path /maquinas/*".

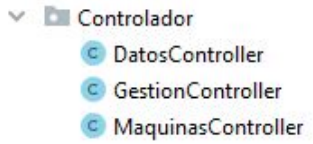


Figura 3.9: Interfaces y clases del paquete de control.

Listado 3.20: Clase del controlador rest para el endpoint de gestión.

```
package TFM.Gestor.Controlador;

import TFM.Gestor.Model.ActualizarCaso;
import TFM.Gestor.Model.ActualizarTarea;
import TFM.Gestor.Model.MensajesSimple;
import TFM.Gestor.Model.PorcentajesCasos;
import TFM.Gestor.Servicio.IDatosService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

@RestController
public class GestionController {
    @Autowired
    IDatosService iDatosService;

    @PostMapping("/gestion/datos/borrar")
    public MensajesSimple borrar_caso(@RequestBody Integer id){
        return iDatosService.borrar_caso(id);
    }
    @PostMapping("/gestion/excepcionarcaso")
    public MensajesSimple excepcionarcaso(@RequestBody ActualizarCaso caso){
        return iDatosService.excepcionar_caso(caso.getId(),caso.getMensaje());
    }
    @PostMapping("/gestion/reiniciarcaso")
    public MensajesSimple reiniciarcaso(@RequestBody ActualizarCaso caso){
        return iDatosService.reiniciar_caso(caso.getId());
    }
    @PostMapping("/gestion/desbloquear")
    public MensajesSimple desbloquearcaso(@RequestBody ActualizarCaso caso){
        return iDatosService.desbloquear_caso(caso.getId(),false,null);
    }

    @PostMapping("/gestion/actualizarcaso")
    public MensajesSimple actualizarcaso(@RequestBody ActualizarCaso caso){
        return iDatosService.actualizar_caso(caso,false);
    }

    @GetMapping("/gestion/porcentajes")
    public PorcentajesCasos porcentajes(){
        return iDatosService.porcentajesCasos();
    }




    @PostMapping("/gestion/asingarmaquina")
    public MensajesSimple asingarmaquina(@RequestBody ActualizarTarea tarea){
       return iDatosService.asingar_maquina(tarea);

    }
}
```

3.6 Desarrollo Frontend

La estructuración de la aplicación que realiza de frontend realiza la visualización de los datos solicitados al servicio de gestión.
Desde el visualizar se puede realizar la gestión de las tareas, crear casos y activar y desactivar las máquinas. Para ello se hace uso de la siguiente estructura de organización de objetos:

- Java - Modelos: creación de las clases a utilizar para el funcionamiento del proceso.
- Java - Servicio: para preparar las consultas y realizar las peticiones al servicio rest.
- Java - Controlador: son los Endpoints del servicio web.
- Resources - Templates: son las plantillas HTML que devuelve el controlador.
- Resources - Static: son objectos HTML estáticos como los JavaScript o los CSS.
- Aspecto páginas web: aspecto de las páginas con datos de prueba.

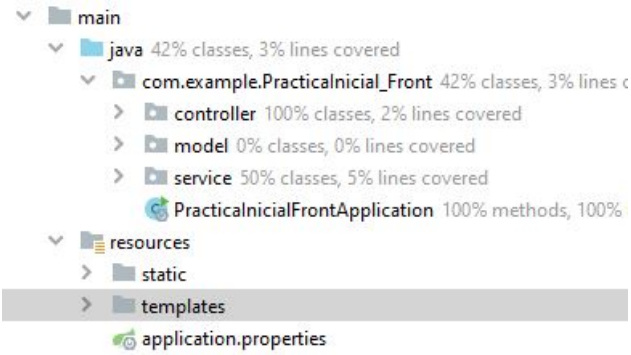


Figura 3.10: Estructura de paquetes del proyecto de frontend.

3.6.1 Java - Modelos

En el Front los modelos se usan para construir las estructuras de los datos de manera sencilla en las plantillas.
El listado de los modelos es:

- ActualizarCaso: se utiliza para la actualización de datos.
- ActualizarTarea: se utiliza para la gestión y manejo de las tareas de los casos.
- DatosCompleted: se utiliza para la visualización de datos.
- Maquinas: se utiliza para la visualización del listado de máquinas.
- MensajeSimple: se utiliza para la recepción de los mensajes de algunas operaciones.
- NuevoCaso: se utiliza para la creación de nuevos casos
- PorcentajeCasos: se utiliza para las gráficas y tablas de la página inicial.
- Tareas: se utiliza para visualizar el listado de tareas.

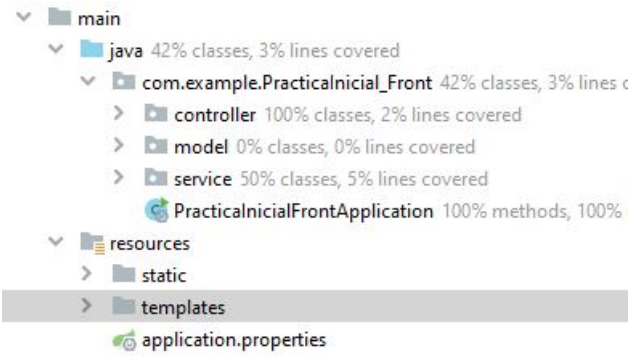


Figura 3.11: Estructura de paquetes del proyecto de frontend.

Listado 3.21: Clase del controlador para la creación de nuevos casos.

```
package com.example.PracticaInicial_Front.model;

public class NuevoCaso {
    public Double getValor1() {
        return valor1;
    }

    public void setValor1(Double valor1) {
        this.valor1 = valor1;
    }

    public Double getValor2() {
        return valor2;
    }

    public void setValor2(Double valor2) {
        this.valor2 = valor2;
    }

    public String getOperando() {
        return operando;
    }

    public void setOperando(String operando) {
        this.operando = operando;
    }

    private Double valor1;
    private Double valor2;
    private String operando;
}
```

3.6.2 Java - Servicio

Para la utilización de cada uno de los servicios expuestos en el gestor se utiliza un intérprete y clase de implementación que permite realizar la comunicación entre el servicio gestor y la carga sobre las plantillas que devuelve el controlador.

- IDatosService y DatosServiceImpl: se utilizan la interfaz y la clase para comunicarse con el servicio

con el path de "datos"que se va a consumir.

- IGestionService y GestionServiceImpl: se utilizan la interfaz y la clase para comunicarse con el servicio con el path de "gestion"que se va a consumir.
- IMaquinasService y MaquinasServiceImpl: se utilizan la interfaz y la clase para comunicarse con el servicio con el path de "maquinas"que se va a consumir.

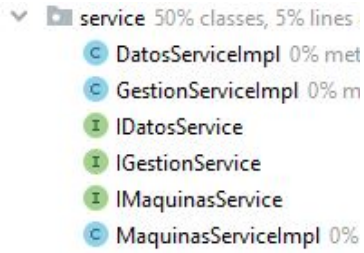


Figura 3.12: Estructura del paquete servicio del proyecto de frontend.

Listado 3.22: Interfaz del servicio para control de las máquinas.

```
1  package com.example.PracticaInicial_Front.service;
2
3  import com.example.PracticaInicial_Front.model.Maquinas;
4  import com.example.PracticaInicial_Front.model.PorcentajesCasos;
5
6  import java.util.List;
7
8  public interface IMaquinasService {
9      List<Maquinas> listadoMaquinas();
10     String cambiarEstado(Integer id,Boolean estado);
11 }
```

Listado 3.23: Clase del servicio para control de las máquinas.

```
1  package com.example.PracticaInicial_Front.service;
2
3  import com.example.PracticaInicial_Front.model.DatosCompleted;
4  import com.example.PracticaInicial_Front.model.Maquinas;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.beans.factory.annotation.Value;
7  import org.springframework.stereotype.Service;
8  import org.springframework.web.client.RestTemplate;
9
10 import java.util.Arrays;
11 import java.util.Comparator;
12 import java.util.List;
13
14 @Service
15 public class MaquinasServiceImpl implements IMaquinasService{
16     @Autowired
17     RestTemplate template;
18     @Value("${my.property.url}")
19     private String url;
20
21     private String base = "maquinas";
22
23     class SortbyId implements Comparator<DatosCompleted> {
24         public int compare(DatosCompleted a, DatosCompleted b)
25         {
26             return a.getId() - b.getId();
27         }
28     }
29     @Override
30     public List<Maquinas> listadoMaquinas(){
31         Maquinas[] maquinas = template.getForObject(url+base, Maquinas[].class);
32         return Arrays.asList(maquinas);
33     }
34     @Override
35     public String cambiarEstado(Integer id,Boolean estado){
36
37         Maquinas maquina = new Maquinas();
38         maquina.setEstado(estado);
39         maquina.setIdmaquina(id);
40         try {
41             maquina = template.postForObject(url+base+"/cambiarestado", maquina, Maquinas.
                   class);
42         }
43         catch (Exception e){
44             maquina.setTexto("Error al cambiar estado");
45         }
46
47         return maquina.getTexto();
48     }
49 }
```

3.6.3 Java - Controlador

Mediante las clases de controlador creadas en este paquete se expone los path que permiten la visualización e interactuación mediante una página web.

- clienteComun: se encarga de mostrar la pagina principal
- clienteDatos: se encarga del path que empieza por /datos/*.einteractúa con todo lo relacionado con los casos.
- clienteTareas: se encarga del path que empieza por /tareas/*"para poder visualizar y gestionar las tareas de los casos.
- clienteMaquinas: se encarga del path que empieza por /maquinas/*ƨ permite gestionar las máquinas.

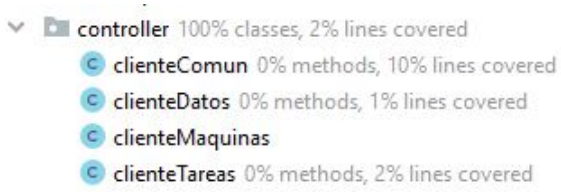


Figura 3.13: Estructura del paquete controladores del proyecto de frontend.

Listado 3.24: Clase del controlador que expone el enpoint principal.

```
package com.example.PracticaInicial_Front.controller;


import com.example.PracticaInicial_Front.model.PorcentajesCasos;
import com.example.PracticaInicial_Front.service.IGestionService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;


@Controller
@RequestMapping("/")
public class clienteComun {

    @Autowired
    IGestionService gestionService;


        @GetMapping(value={"/","home",""})
    public String home(Model model){
        Integer num_page = 4;
        PorcentajesCasos porcentajesCasos = new PorcentajesCasos();
        model.addAttribute("error", "");
        try{
            porcentajesCasos = gestionService.porcentajes();
        }
        catch (Exception e){
            model.addAttribute("error", e.getMessage());
        }

        model.addAttribute("porcentaje", porcentajesCasos);

        return "inicio";
    }
}
```

3.6.4 Resources - Templates

En las "templates"se crean las plantillas de HTML que utiliza el controlador para realizar las devoluciones de los controladores. Para la realización fue necesario consultar las librerías de Boostrap[26], los iconos de Boostrap[27], W3Schooll[28], Stackoverflow[29] y algunas funciones de Thymeleaf[30], y crear la estructura de carpetas:

- (Común): página principal, menú y mensajes de advertencias.
- Casos: se almacena la página principal de los casos y los módulos de este.
- Home: se almacenan los módulos de la página principal.
- Maquinas: se almacena la página principal que muestra el listado de páginas gracias a sus módulos.
- Tareas: se almacena la página principal de las tareas y los módulos necesarios para interactuar con ellas.

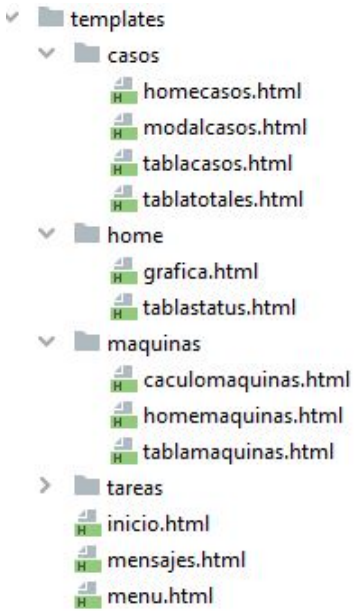


Figura 3.14: Estructura de la carpeta de templates del proyecto de frontend.

3.6.4.1 Desarrollo de módulos

Cuando se realizan páginas se realizan de manera estructurada en módulos, primero se prueba la página tal cual se desea crear y luego se divide en módulos y se agregan a la página principal.

Listado 3.25: Ejemplo de código de la plantilla Thymeleaf página de inicio.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Pagina principal</title>
    <link th:href="@{/css/bootstrap.min.css}" rel="stylesheet">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/3.9.1/chart.js" integrity="
        sha512-
        d6nObkPJgV791iTGuBoVC9Aa2iecqzJRE0Jiqvk85BhLHAPhWqkuBiQb1xz2jvuHNqHLYoN3ymPfpiB1o+Zgpw
        ==" crossorigin="anonymous" referrerpolicy="no-referrer"></script>
</head>
<body>
<nav th:insert="menu"></nav>
<h1 class="m-2 h1 text-primary d-flex justify-content-center">Visualizador del gestor de casos
    </h1>
<br>
<div class="container">
    <div class="row d-flex justify-content-center">
        <div class="col-6" th:insert="home/tablastatus"></div>
        <div class="col-6" th:insert="home/grafica"></div>
    </div>
</div>
</body>
</html>
```

Se utiliza la función "th:insert"para agregar los diferentes elementos, permitiendo tratar como si fueran módulos, que facilitan el desarrollo.

3.6.4.2 Desarrollo página principal

El desarrollo se comienza con la creación de la primera página web en la que se muestra una tabla y el resultado mediante una gráfica de resultados.

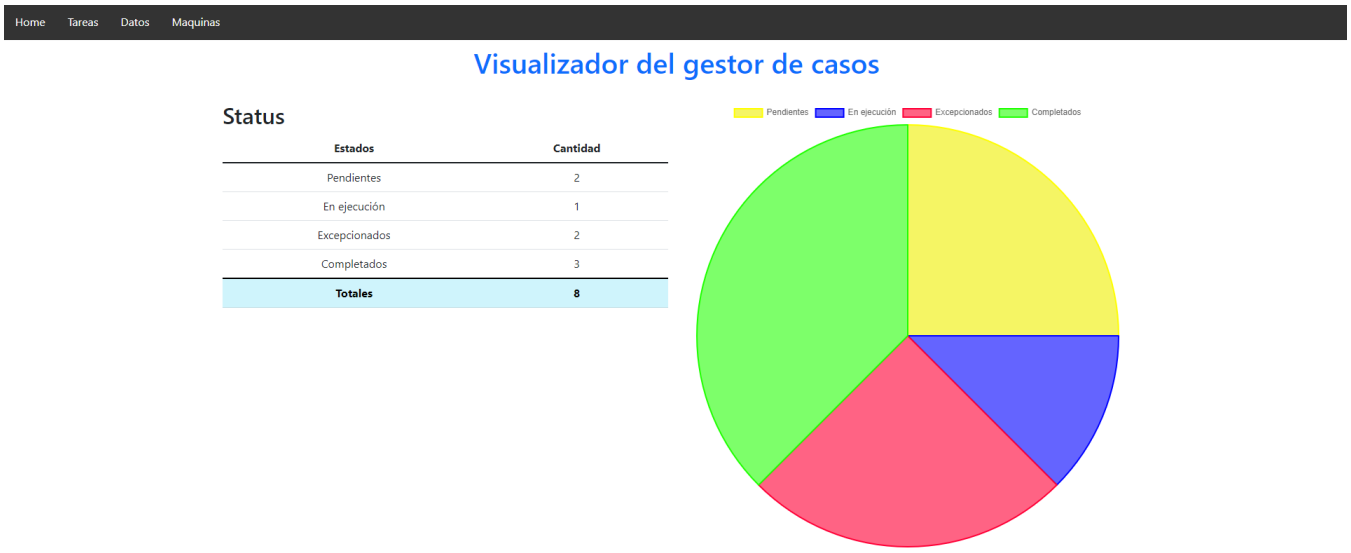


Figura 3.15: Página principal o de inicio de la aplicación.

3.6.4.3 Módulos de listado

Los módulos de las tablas se crean siguiendo el mismo patrón de una tabla en la que se muestran los datos y permite interactuar con cada una de las líneas de las tablas.

Figura 3.16: Página máquinas en las que interactuar según el listado.

Listado 3.26: Ejemplo de código de la plantilla Thymeleaf para listado de máquinas.

```
<!DOCTYPE html>
<html lang="es">
<head>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.9.1/font/
        bootstrap-icons.css">
</head>
<body>
<div>
    <div class="card">
        <h2 class="card-header">Maquinas robot</h2>
        <div class="card-body">
            <table class="table table-hover">
                <thead class="table-info text-center thead-light">
                <tr>
                    <th scope="col">Identificador</th>
                    <th scope="col">Hastname</th>
                    <th scope="col">Procedencia</th>
                    <th scope="col">Estado</th>
                </tr>
                </thead>
                <tbody>
                <tr th:each="maquina : ${maquinas}">
                    <td th:text="${maquina.idmaquina}"></td>
                    <td th:text="${maquina.hastname}"></td>
                    <td th:text="${maquina.procedencia}"></td>
                    <td><a><div class="custom-control custom-switch">
            <input type="checkbox" class="custom-control-input" th:id="|customSwitch${maquina.
                idmaquina}|"  th:checked="${maquina.estado}"
                 th:onchange="|myFunction(${maquina.idmaquina},${maquina.estado})|"/>
            <label th:if="${maquina.estado}" class="custom-control-label" th:for="|
                customSwitch${maquina.idmaquina}|">Activado</label>
            <label th:if="${maquina.estado} == false"class="custom-control-label" th:for="|
                customSwitch${maquina.idmaquina}|">Deactivado</label>
          </div></a></td>
                </tr>
                </tbody>
            </table>
            <script>
                function myFunction(val,estado) {
                    if(confirm('Â¿Estas seguro que deseas cambiar estado a la maquina '+val.
                        toString()+'?')){
                        window.location.href="../../../maquinas/cambiarestado/"+ val.toString
                            ()+"/"+estado.toString();
                    }}
            </script>
        </div>
    </div>
</div>
</body>
</html>
```

3.6.4.4 Módulos modales

Se crean modales para interactuar con ciertas acciones, para ello se crean módulos que abren ventanas modales emergentes que permiten realizar acciones.

Listado 3.27: Ejemplo de código de la modal para la creación de nuevos casos, parte 1.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
</head>
<body>
<div>
  <div>
    <button id="nuevo" type="button" class="btn btn-success btn-lg">Nuevo</button>
  </div>
  <!-- Modal -->
  <div class="modal fade" id="casosModal" tabindex="-1" role="dialog" aria-labelledby="
      exampleModalLabel" aria-hidden="true">
    <div class="modal-dialog" role="document">
      <div class="modal-content">
        <div class="modal-header">
            <h5 class="modal-title" id="exampleModalLabel">A&ntilde;adir nuevo caso</h5>
            <button type="button" class="close" data-dismiss="modal" id="closeModal" aria-
                label="Close">
                <span aria-hidden="true">&times;</span>
            </button>
        </div>
        <form action="/datos/nuevo" method="POST" >
        <div class="modal-body">
            <div id="mostrarid" class="form-group">
                <label for="idcaso">Id caso :<a id="idcasotext"></a></label>
                <input type="text" class="form-control" id="idcaso" name="idcaso" hidden>
            </div>
            <div class="form-group">
              <label for="valor1">Valor 1</label>
              <input type="number" class="form-control" step="0.01" id="valor1" name="valor1"
                  placeholder="Introduce el Valor 1">
            </div>
            <div class="form-group">
              <label for="valor2">Valor 2</label>
              <input type="number" class="form-control" step="0.01" id="valor2" name="valor2"
                  placeholder="Introduce el Valor 2">
            </div>
            <div class="form-group">
              <label for="operando">Operando</label>
              <input type="text" class="form-control" id="operando" name="operando"
                  placeholder="Introduce el Operando">
            </div>
            <div id="mostrarresultado" class="form-group">
              <label for="resultado">Resultado</label>
```

Listado 3.28: Ejemplo de código de la modal para la creación de nuevos casos, parte 2.

```
1                 <input type="number" class="form-control" step="0.0001" id="resultado" name="
                      resultado" placeholder="Introduce el resultado">
2               </div>
3               <div id="mostrarmensaje" class="form-group">
4                 <label for="mensaje">Mensaje error</label>
5                 <input type="text" class="form-control" id="mensaje" name="mensaje" placeholder=
                      "Introduce el mensaje">
6               </div>
7             </div>
8             <div class="modal-footer">
9               <button type="button" class="btn btn-secondary" data-dismiss="modal" id="
                    buttonClose">Cerrar</button>
10              <input type="submit" id="inputsubmit" class="btn btn-primary" value="Crear" />
11            </div>
12          </form>
13        </div>
14      </div>
15    </div>
16    <script type="text/javascript">
17      $(document).ready(() => {
18        $("#nuevo").click(function () {
19          $('#inputsubmit').val("Crear");
20          $('#exampleModalLabel')[0].innerText=("A&ntilde;adir nuevos datos");
21          $('#mostrarid').hide();
22          $('#mostrarresultado').hide();
23          $('#mostrarmensaje').hide();
24
25          $('#operando').val("");
26          $('#valor1').val("");
27          $('#valor2').val("");
28          $('#casosModal form').attr("action","/datos/nuevo");
29          $('#casosModal').modal('show');
30        });
31
32        $(".editar_caso").click(function () {
33          $('#inputsubmit').val("Editar");
34          $('#mostrarid').show();
35          $('#mostrarresultado').show();
36          $('#mostrarmensaje').show();
37          $('#casosModal form').attr("action","/datos/editar");
38          $('#exampleModalLabel')[0].innerText=("Editar datos del caso "+$(this).parent().parent
                ().parent().parent().parent().find(".campo_id")[0].innerText);
39          $('#idcaso').val($(this).parent().parent().parent().parent().parent().find(".campo_id"
                )[0].innerText);
40          $('#idcasotext')[0].innerText=($(this).parent().parent().parent().parent().parent().
                find(".campo_id")[0].innerText);
41          $('#operando').val($(this).parent().parent().parent().parent().parent().find(".
                campo_operando")[0].innerText);
42          $('#valor1').val($(this).parent().parent().parent().parent().parent().find(".
                campo_valor1")[0].innerText);
43          $('#valor2').val($(this).parent().parent().parent().parent().parent().find(".
                campo_valor2")[0].innerText);
44          $('#resultado').val($(this).parent().parent().parent().parent().parent().find(".
                campo_resultado")[0].innerText);
45          $('#mensaje').val($(this).parent().parent().parent().parent().parent().find(".
                campo_mensaje")[0].innerText);
```

Listado 3.29: Ejemplo de código de la modal para la creación de nuevos casos, parte 3.

```
1
2           $('#casosModal').modal('show');
3         });
4         $("#buttonClose").click(function () {
5           $('#casosModal').modal('hide');
6         });
7         $("#closeModal").click(function () {
8           $('#casosModal').modal('hide');
9         });
10      });
11    </script>
12  </div>
13  </body>
14  </html>
```

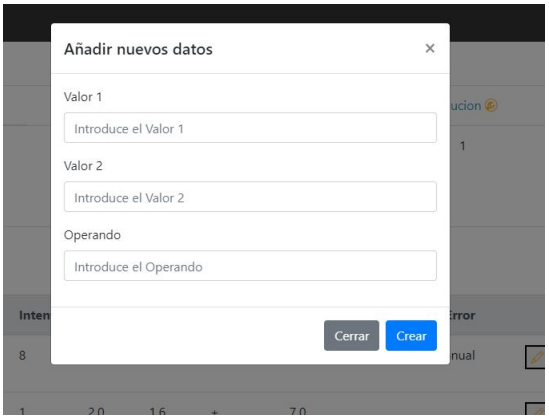


Figura 3.17: Aspecto del modal de creación de nuevos casos.

3.6.5 Resources - Static

Conjunto de librerías JavaScript y ficheros CSS (Cascading Style Sheets, hojas de estilo en cascada) para aplicar de estilos a las páginas web.

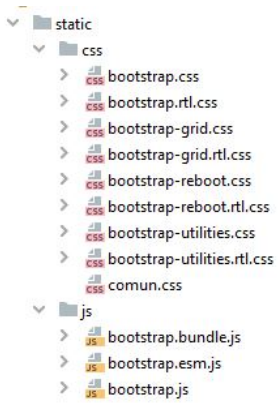


Figura 3.18: Estructura de carpetas con ficheros JavaScript y archivos CSS de estilos.

3.7 Desarrollo Plugin Blue Prism

Blue Prism es un software diseñado en .Net que permite automatizar procesos. Desde el gestor del sistema de Blue Prism se pueden configurar plugins u objectos en modo servicio y puedan ser utilizados por los procesos como un objeto más.
Para la creación de estos servicios Blue Prism cuenta con un configurador que hay que adaptar para que permita escribir el código en VB o C#(el elegido).

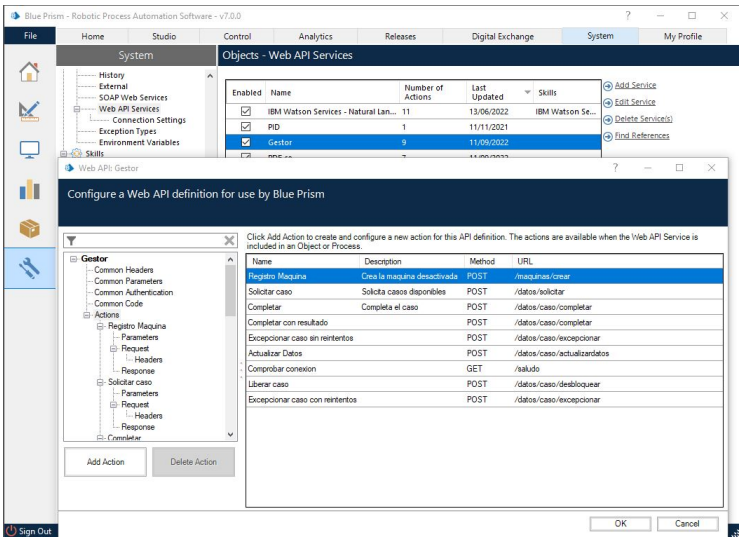


Figura 3.19: Configuración del plugin de servicio gestor en Blue Prism.

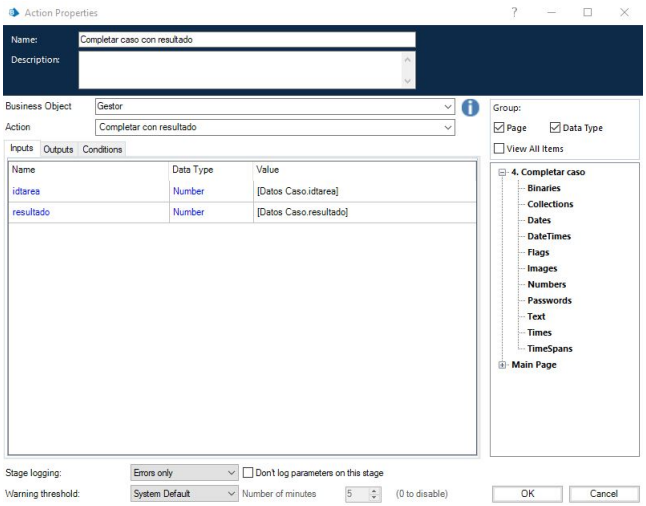


Figura 3.20: Aspecto de acción completar con resultados del plugin.

3.8 Conclusiones

Los desarrollos de este entorno web ha sufrido adaptaciones de código según se realizaban los desarrollos para conseguir adaptar al completo la comunicación entre el gestor, el visualizar y el plugin. Al ser un desarrollo ágil y hacer una revisión en cada implantación permitía ir comprobando los resultados.
La conexión del gestor con la base de datos fue sencilla tras buscar la manera de realizar consultas SQL de procedimientos almacenados.
La parte de adaptar y comunicar llamadas entre Blue Prism y el servicio gestor fue fundamental para comprobar el funcionamiento correcto del gestor de tareas y enviaba tareas disponibles de manera correcta, dado que el evitar tareas duplicadas era una de las partes fundamentales del proyecto.
Durante el proceso de adaptar el visualizador y el gestor fue una adaptación un poco más rápida al compartir lenguaje de programación y disponer de las librerías creadas en el gestor, siendo reutilizadas en el visualizador.

Capítulo 4

Resultados

Los grandes resultados requieren grandes ambiciones

Heráclito

4.1 Introducción

En este capítulo se introducirán los resultados más relevantes del trabajo.

La estructura del capítulo es:

- Entorno Experimental
  - Base de datos utilizadas
  - Resultados del gestor web.
  - Resultados visualizador Web
  - Respuestas del plugin de Blue Prism
- Flujo de vida de un caso
  - Creación del caso.
  - Asignación del casos o excepcionar manualmente.
  - Solicitud de casos.
  - Actualización del caso.
  - Resultados de ejecución
  - Reintentos
  - Otras funciones
- Conclusiones

4.2 Entorno experimental

El entorno puede realizarse sobre cualquier maquina en la que esté disponible un sistema Linux/Windows o Mac o una virtualización de estos y algún tipo de sistema que consuma los casos y los resuelva como una máquina robot. Para la utilización de un entorno es necesario disponer todos los puntos descritos en el Aparado 1.3.2. También es necesario crear la base de datos.

4.2.1 Bases de datos utilizadas

Para la creación de la base de datos es necesario crear las diferentes tablas, triggers y procedimientos almacenados para conseguir tener todo el entorno en correcto funcionamiento, tal como se indica en el apartado3.5.2.

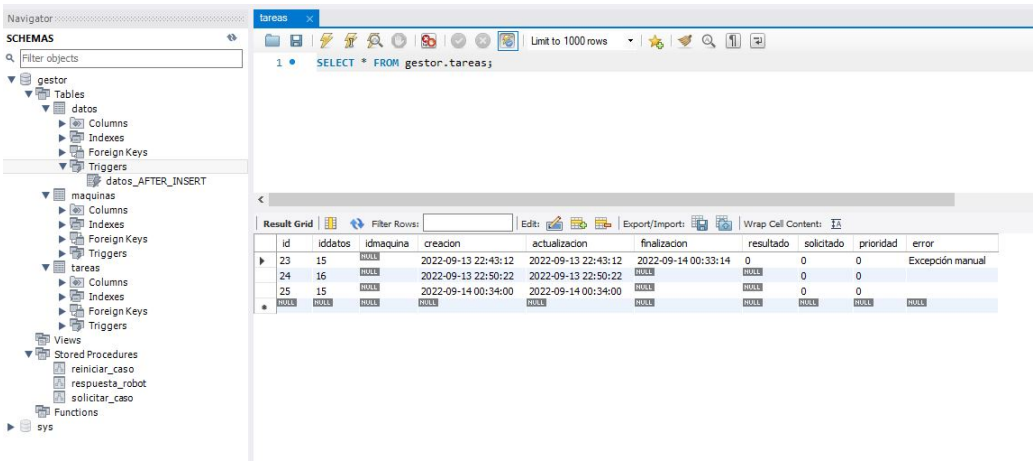


Figura 4.1: Esquema de la base de datos en MySQL y algunos datos de la tabla del tareas.

4.2.2 Resultados Web

Tras iniciar el entorno web ya se con el depurando o como servicio, el aspecto final de cada una de las páginas es:

- Página principal: donde se mostrará la tabla de casos totales y una gráfica del total de cada tipo de caso.

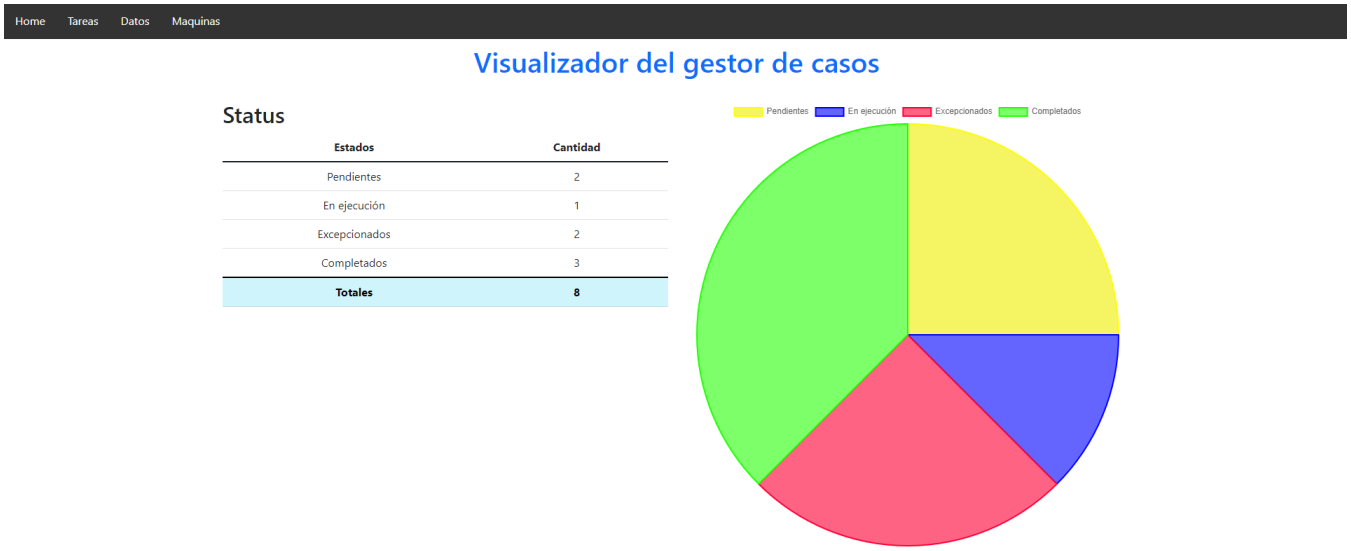


Figura 4.2: Página principal o de inicio de la aplicación.

- Página de tareas: un tabla en la parte superior con los totales de las tareas y abajo un listado individual de cada tareas.

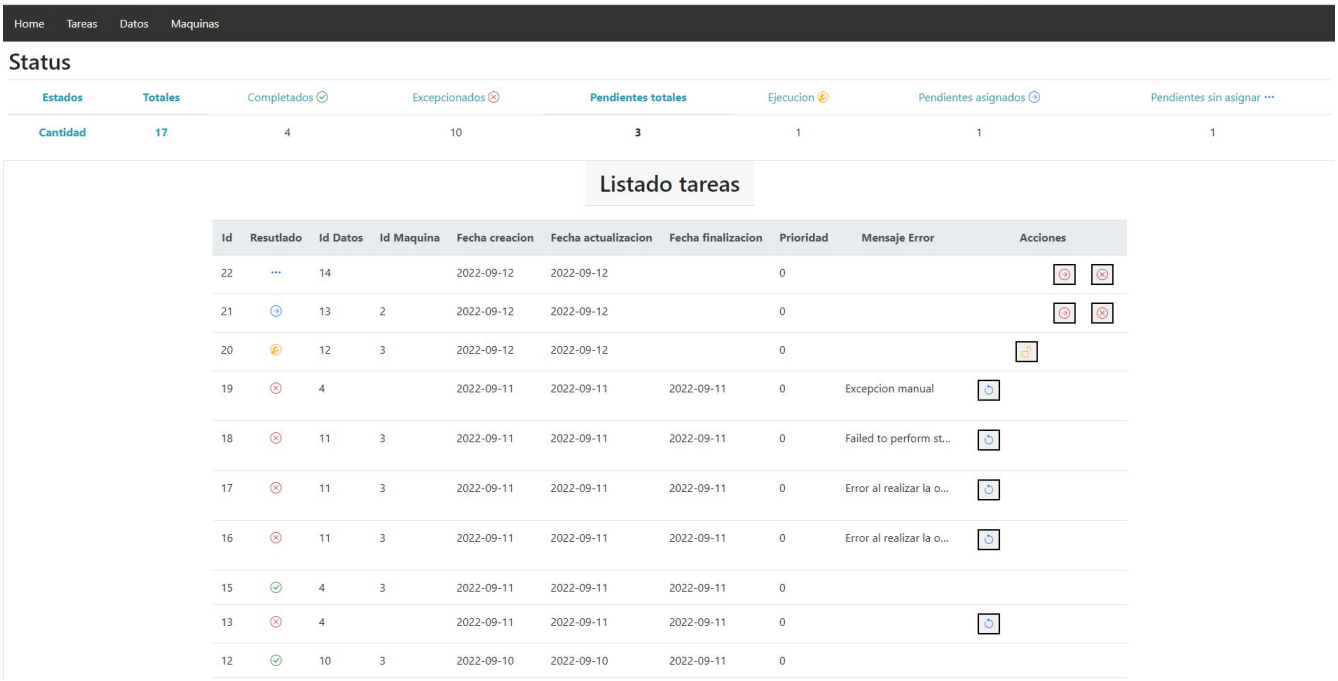


Figura 4.3: Página de tareas.

- Página de datos: una tabla de total de casos y el resultado o estado de cada uno. Debajo aparece un listado de cada uno de los casos y la opción de crear nuevos casos.

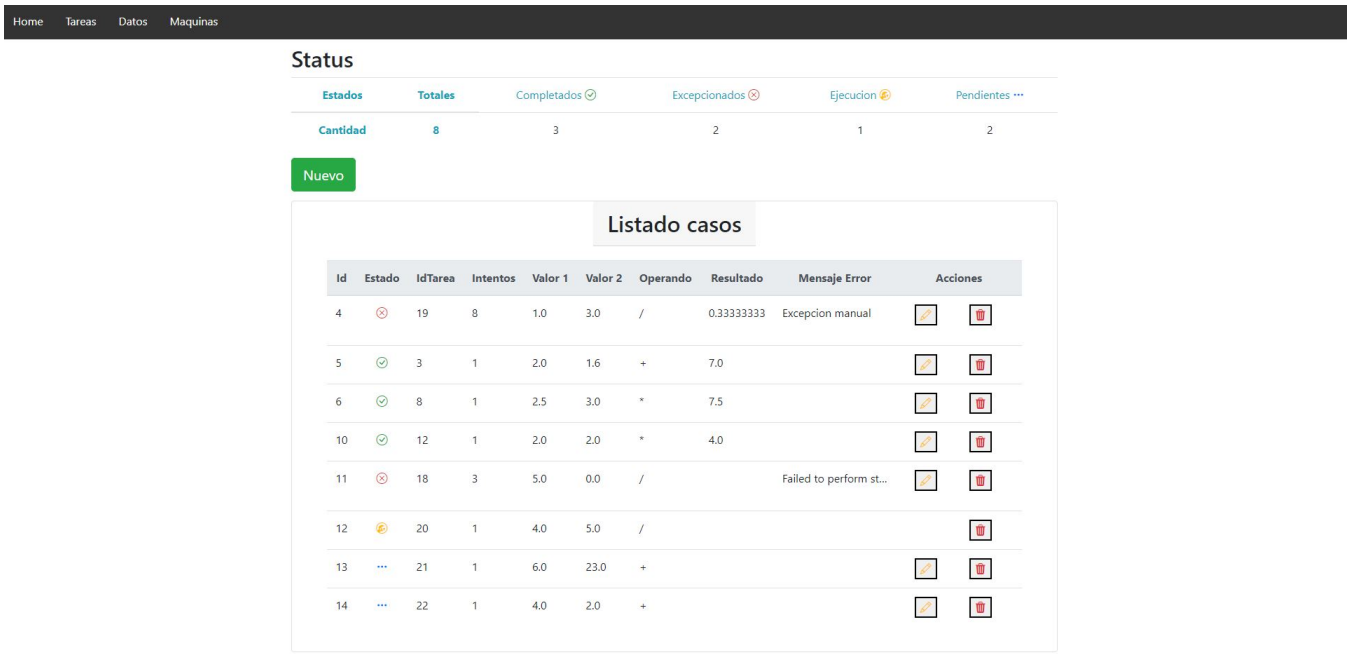


Figura 4.4: Página de datos.

- Página de máquinas: Aparece un resumen del estado de las máquinas y un listado de máquinas indicando su estado.

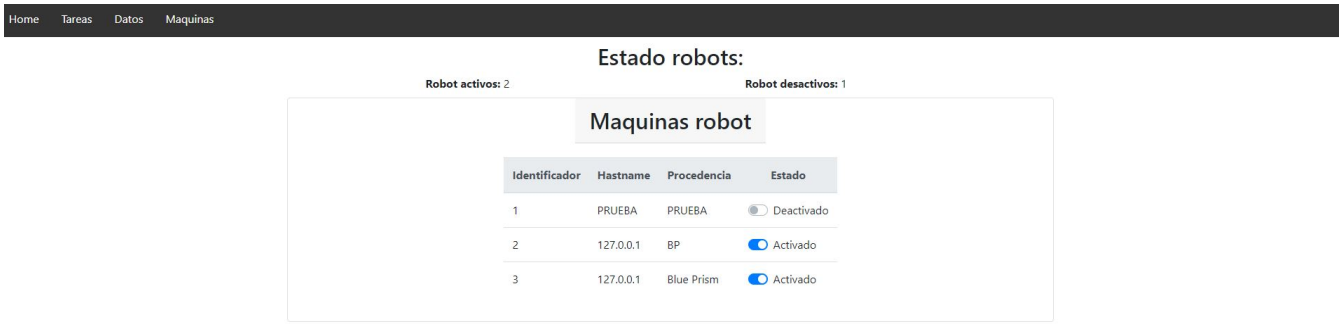


Figura 4.5: Página de máquinas.

4.2.3 Resultado del gestor

Gracias al canal gestor y los procedimientos almacenado podemos garantizar que comparar las tareas no se duplican y es capaz de gestionar los casos completos de manera completa junto con algún sistema que los automatice y resuelva. Gracias al gestor garantizamos seguridad, fiabilidad y velocidad permitiendo realizar tareas en paralelo si existen más de un sistema que realice las tareas.

4.3 Flujo de vida de un caso

4.3.1 Creación del caso

Para generar un caso se puede realizar mediante una llamada al servicio gestor pasando los parámetros de:

- Valor1: será el valor del primer operando.
- Valor2: será el valor del segundo operando.
- Operando: es el signo de realización de la operación. Las opciones son +, -, /, * o X.

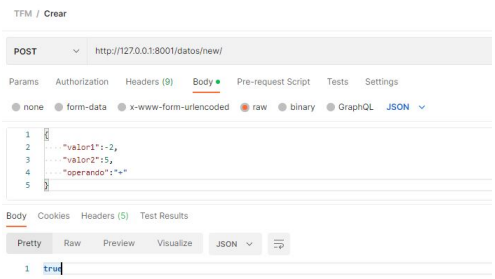


Figura 4.6: Llamada para creación de caso mediante Postman.

O si no realizando los accediendo al menú de datos, pulsando el botón de nuevo, se rellenan los campos y después al botón crear para generar el nuevo caso.

Figura 4.7: Creación mediante el visualizador.

Y como resultado es la creación de los dos casos.

Figura 4.8: Listado de casos desde el visualizador.

4.3.2 Asignación de casos o excepcionar manualmente

Una alternativa a previa a que el caso sea solicitado es la posibilidad de asignar una maquina concreta desde el menú de tareas, para que realice el caso como primera opción.

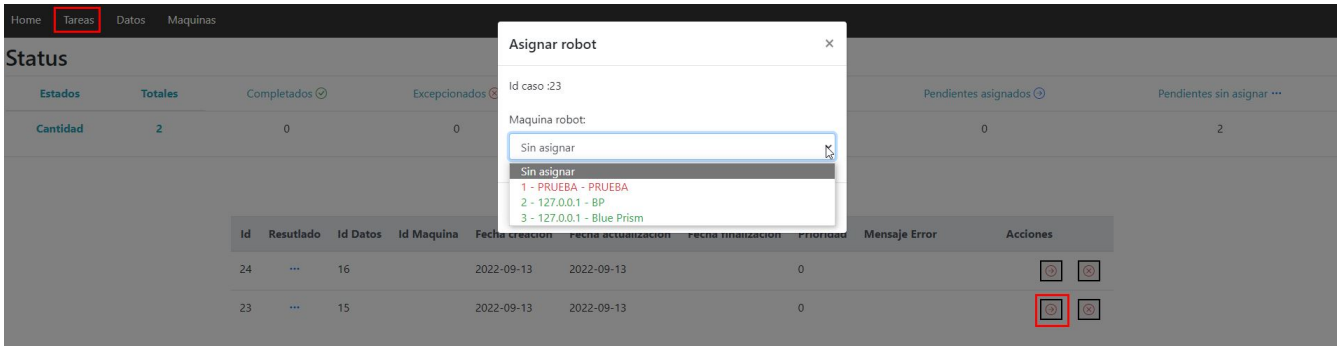


Figura 4.9: Asignar robot a un caso.

En verde se pueden ver las opciones disponibles y en rojo los robots desactivados, desde esta acción se puede modificar el robot asignado para la tarea hasta que el caso sea solicitado. Desde esta ventana se puede generar una excepción manual desde el portal que impide la ejecución del caso.

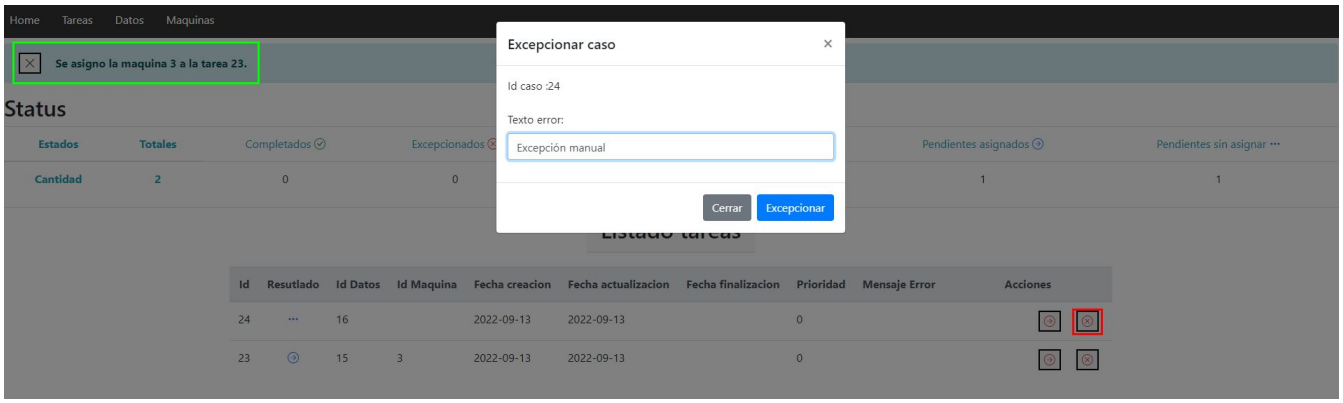


Figura 4.10: Excepcionar una caso y visualizar el mensaje de notificación de la asignación de un robot.

4.3.3 Solicitud de casos

Para realizar solicitudes de casos se puede realizar mediante consumo de servicio o desde el plugin de Blue Prism.

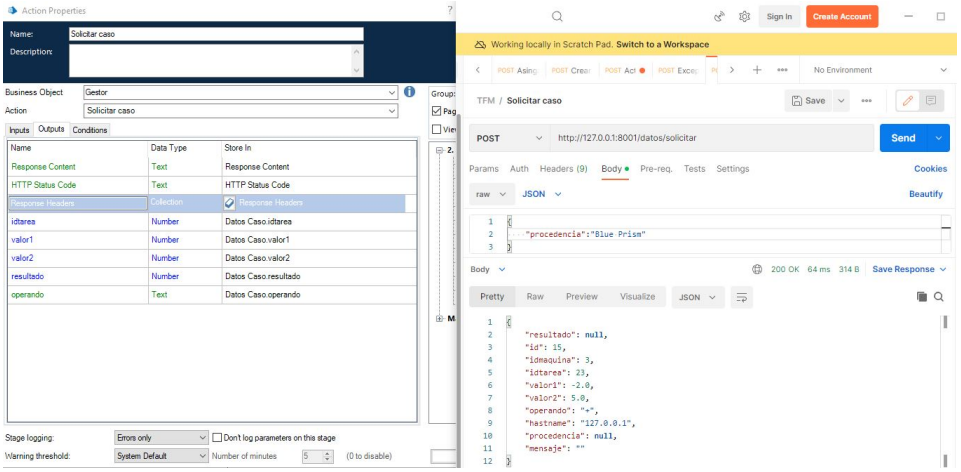


Figura 4.11: Solicitud de caso mediante Postman y ejemplo de funcionamiento del output de la llamada.

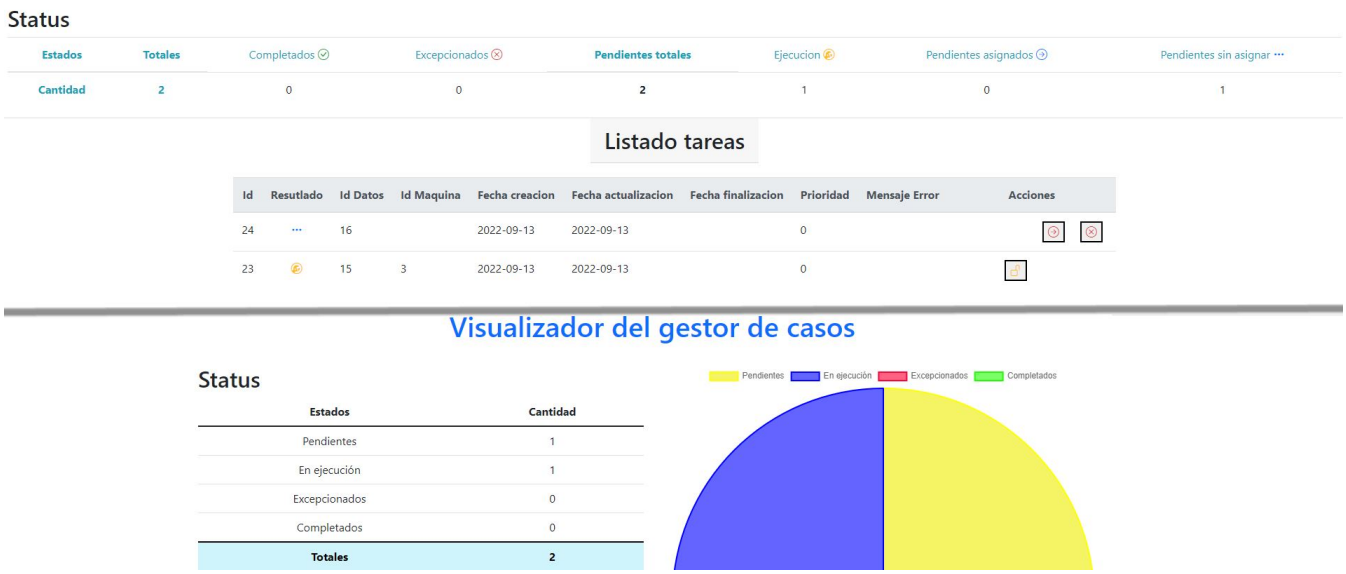


Figura 4.12: Resultado de la llamada la pestaña de tareas y en la página principal.

4.3.4 Actualización del caso

Se pueden actualizar los valores desde el portal editando un caso , siempre que no esté solicitado en ese momento o desde Postman[1] o desde el plugin aun estando en solicitado.

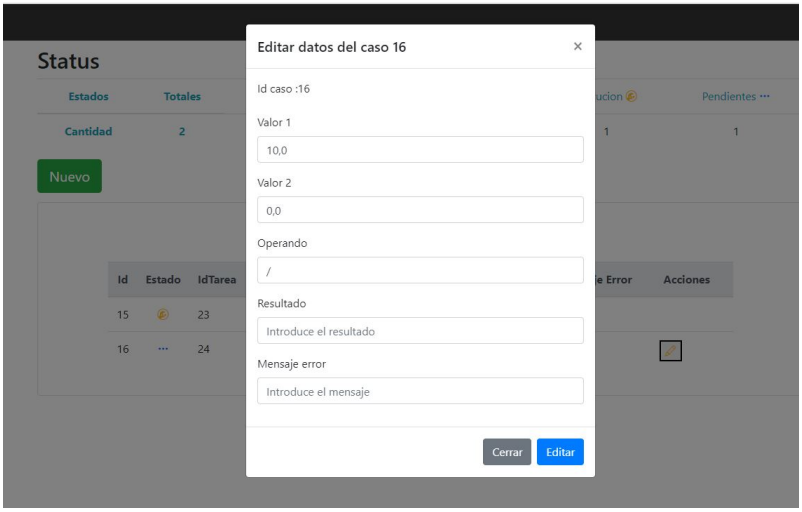


Figura 4.13: Editar o actualizar valores del caso.

[1] "procedencia":Es necesario indicar la procedencia que identifica el tipo de robot que es según el robot que lo ejecute.

4.3.5 Resultados de ejecución

Existen 4 tipos de resultados que generar desde el plugin:

1. Completar: Indica que el caso se ha resuelto completamente.

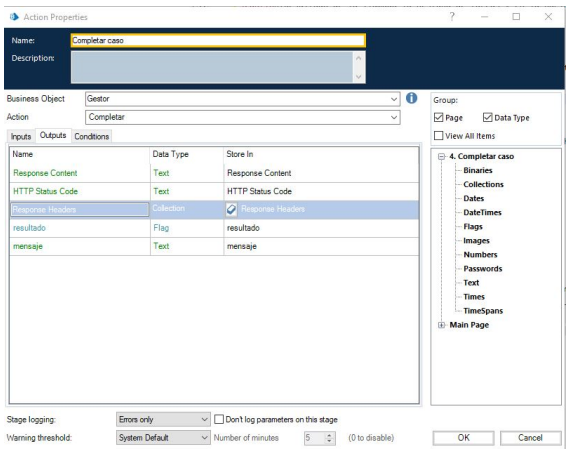


Figura 4.14: Valores de salida de llamada completar desde Blue Prism.

2. Completar con valor de resultado: Indica que el caso se ha resuelto completamente y además permite agregar. Posee el campo extra de resultado respecto de la otra llamada

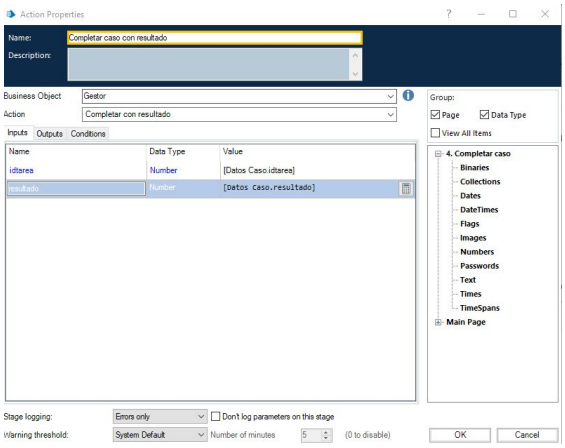


Figura 4.15: Valores de entrada de llamada completar con resultado desde Blue Prism.

3. Excepcionar con reintento y mensaje de error: permite excepcionar el caso y reintentarlo si tiene menos de 3 intentos, guardando el mensaje de error.

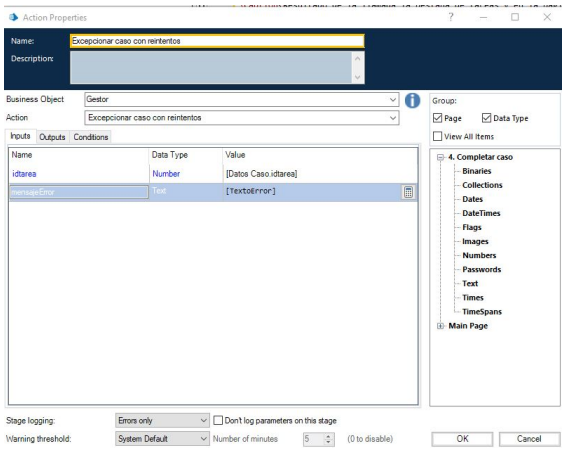


Figura 4.16: Valores de entrada de excepcionar con reintentos.

4. Excepcionar sin reintento y con mensaje de error:permite excepcionar el caso y lo marca para que se no se reintente, guardando el mensaje de error.

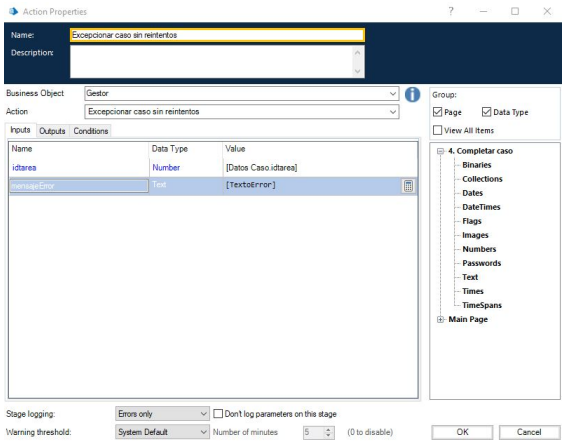


Figura 4.17: Valores de entrada de excepcionar sin reintentos.

También se puede realizar la actualización del estado de la tarea mediante el servicio indicando los campos, numero de tarea, si el resultado es correcto, que permite actualizar el resultado, o no, en cuyo caso permite reintentar el caso y además puede guardar el mensaje de error.

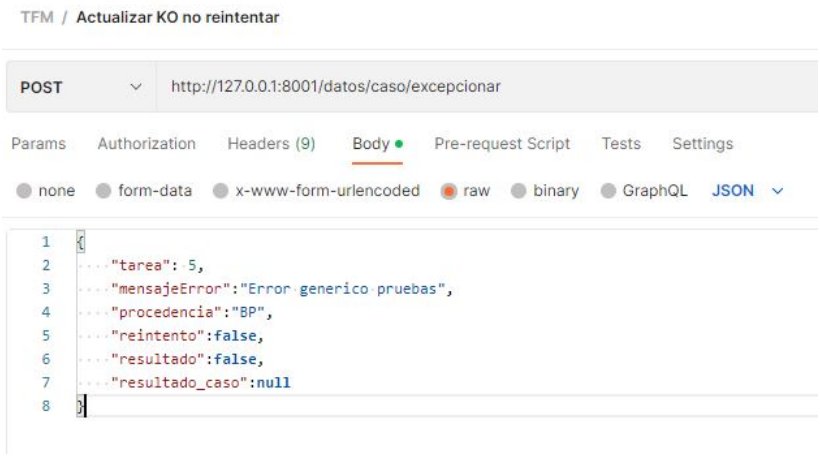


Figura 4.18: Ejemplo de estructura del JSON de una llamada de actualizar estado de la tarea.

4.3.6 Reintentos

Tanto si el caso se ha reintentado de manera manual desde el portal o se ha realizado mediante el tipo de excepción al generar una tarea duplicada en cuanto a id pero mantiene la relación de datos.

Figura 4.19: Aspecto de una tarea reintentada y remarcado el identificador de reintento manual.

4.3.7 Otras funciones

- Desbloquear: se puede hacer desde el visualizar o desde la máquina robot y permite desbloquear el caso para que sea cogido por ese o por cualquier otro robot.
- Activar/Desactivar maquinas: mediante el visualizador se puede activar o desactivar una maquina consiguiendo que esta pueda solicitar casos.

4.4 Conclusiones

El funcionamiento demuestra la cantidad de usos que puede tener y la fiabilidad del desarrollo garantiza que no existe ningún inconveniente en el uso en entornos productivos.
Al comparar el desarrollo con los gestores existentes en el mercado demuestra como una adecuada personalización permite una gestión más detallada y eficiente. Además, se adapta a las necesidades manteniendo la funcionabilidad.

Capítulo 5

Conclusiones y líneas futuras

Rem tene, verba sequentur (Si dominas el tema, las palabras vendrán solas).

Catón el Viejo

En este apartado se resumen las conclusiones obtenidas y se proponen futuras líneas de investigación que se deriven del trabajo.

La estructura del capítulo es:

- Conclusiones generales
- Líneas futuras

5.1 Conclusiones Generales

El proyecto me sirve como comienzo para un proyecto mayor y para comprender que los gestores de colas no son sistemas tan simples como parecen al principio dado que hay que tener muchas variables en cuenta a la hora de conseguir que se realicen tareas sin duplicar al ser solicitado de manera simultánea por dos o más máquinas. He ampliado mis conocimientos de consultas específicas, procedimientos almacenados y son las bases de datos MySQL. Y he sentado las bases para futuros proyectos de mi carrera en la que la personalización del software suele ser primordial y necesaria en sistemas de RPA.

5.2 Líneas futuras

Las líneas futuras del mejoras de este desarrollo tienen diferentes enfoques,

- Permitir gestionar más de una cola diferente.
- Permitir aplazar casos en el tiempo, identificadores y estados del caso.
- Permitir carga desde CSV o ficheros Excel.
- Aplicación móvil.
- Implementar seguridad.

- Ampliación a otros sectores que necesiten gestión de eventos.
- Pulir el funcionamiento para manejar grandes volúmenes de datos.
- Aplicar paginaciones y filtros en las visualizaciones.
- Mejoras visuales según las necesidades del cliente
- Separar el gestor en partes

5.2.1 Permitir gestionar más de una cola diferente

Para que el sistema sea robusto podría ser interesante que se pudieran crear las colas desde el mismo portal indicando el tipo de campos a generar y consiguiendo flexibilidad al diseño y escalabilidad a más de un automatismo con un único gestor.

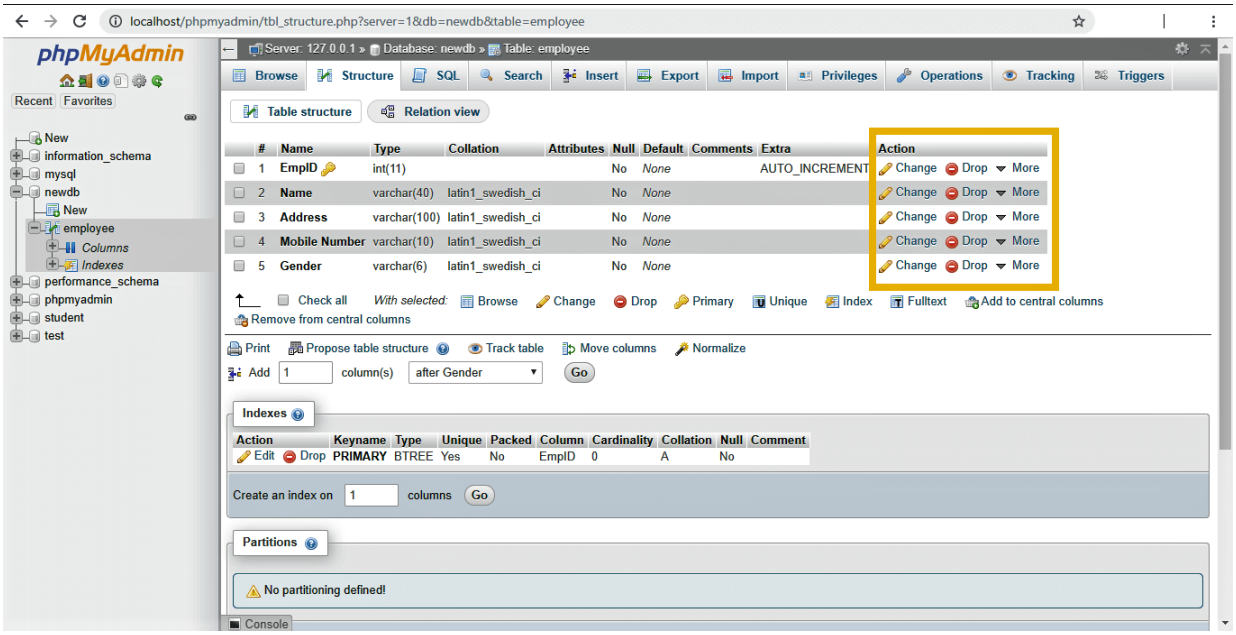


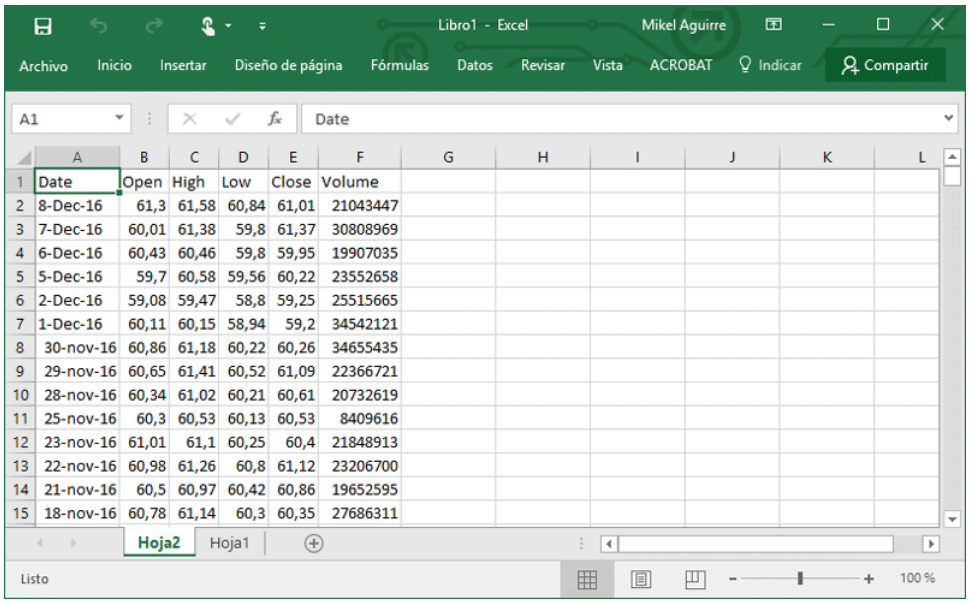
Figura 5.1: Ejemplo creación de tablas con de Php-MyAdmin[14].

5.2.2 Permitir aplazar casos en el tiempo, identificadores y estados del caso

Permitir que se asignen estados y búsquedas rápidas mediante identificadores o tag ayuda a filtrar y buscar y conocer la situación de una caso. Al permitir aplazar en el tiempo los casos consigues que algunas intervenciones asíncronas puedan realizar las operaciones y mientras aprovechar esos momentos por otra tarea en la maquina.

5.2.3 Permitir carga desde CSV o ficheros Excel

Muchísimos sistemas permiten la extracción de datos mediante CSV o ficheros Excel siendo lo más usado por los asistentes de administración en su día a día. Por ello que se permita carga ese Excel desde el portal sería de gran utilidad.



	A	B	C	D	E	F
1	Date	Open	High	Low	Close	Volume
2	8-Dec-16	61,3	61,58	60,84	61,01	21043447
3	7-Dec-16	60,01	61,38	59,8	61,37	30808969
4	6-Dec-16	60,43	60,46	59,8	59,95	19907035
5	5-Dec-16	59,7	60,58	59,56	60,22	23552658
6	2-Dec-16	59,08	59,47	58,8	59,25	25515665
7	1-Dec-16	60,11	60,15	58,94	59,2	34542121
8	30-nov-16	60,86	61,18	60,22	60,26	34655435
9	29-nov-16	60,65	61,41	60,52	61,09	22366721
10	28-nov-16	60,34	61,02	60,21	60,61	20732619
11	25-nov-16	60,3	60,53	60,13	60,53	8409616
12	23-nov-16	61,01	61,1	60,25	60,4	21848913
13	22-nov-16	60,98	61,26	60,8	61,12	23206700
14	21-nov-16	60,5	60,97	60,42	60,86	19652595
15	18-nov-16	60,78	61,14	60,3	60,35	27686311

Figura 5.2: Fichero Excel de ejemplo[15].

5.2.4 Aplicación móvil

En la actualidad todos las personas incluso trabajando utilizamos muchísimo el móvil, se podría crear algún tipo de aplicación a la que conectar el robot y generase alertas según el tipo de usuario desease o que pudiera conocer el estado de las diferentes colas de trabajo.

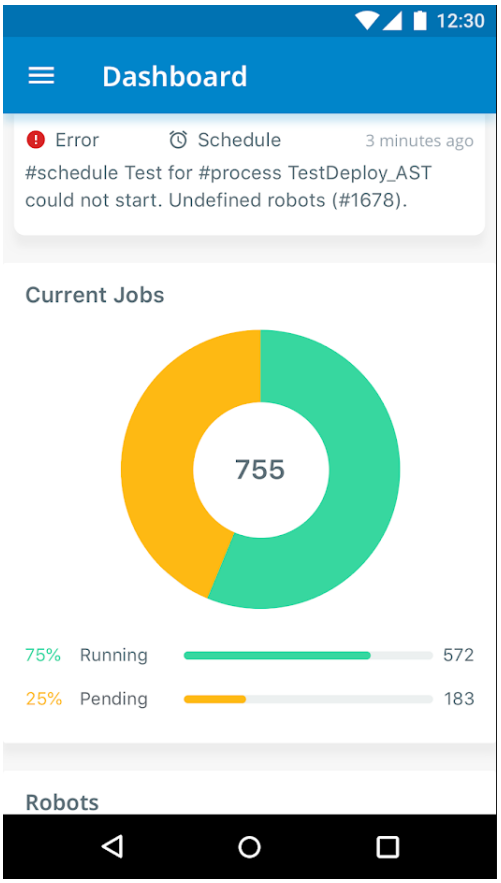


Figura 5.3: Gráfica UiPath desde móvil Android.[16].

5.2.5 Implementar seguridad

La autentificación y los roles son algo fundamental en el sector empresarial, por ello es recomendable aplicar alguna tipo de seguridad para controlar el acceso a la aplicación.

5.2.6 Ampliación a otros sectores que necesiten gestión de eventos

Se puede utilizar para repartir tareas a tramitar de manera manual por las personas de tal manera que se pueda evitar el uso de un fichero Excel al realizar tareas administrativas no automatizables.

5.2.7 Pulir el funcionamiento para manejar grandes volúmenes de datos

Se puede realizar mejoras de visualización y del funcionamiento utilizando sistemas de memorias temporales para datos antiguos que no han sufrido cambios y consiguiendo una optimización de la realización de los cálculos. También se puede generar una comunicación reactiva entre el gestor y el visualizador que permita ver cambios en tiempo real.

5.2.8 Aplicar paginaciones y filtros en las visualizaciones

En las tablas de datos el uso de paginación es muy recomendable dado que agiliza las búsquedas y el poder realizar filtros te permite localizar los datos con mayor facilidad.

Books: 1-10 / 18 | Page 1 of 2 | ? Clear

Book		Price	Delivery Items		
Title	Author	$	In store	Shipping	Publication Date
Boris Godunov	Alexandr Pushkin	7.15	☑	1 Hour	01/01/1999
The Rainmaker	John Grisham	7.99	☐	2 Days	12/01/2001
The Green Mile	Stephen King	11.10	☑	24 Hours	01/01/1992
Misery	Stephen King	7.70	☑	na	01/01/2003
The Dark Half	Stephen King	0	☑	2 Days	10/30/1999
The Partner	John Grisham	12.99	☐	24 Hours	01/01/2005

Figura 5.4: Filtros sobre una tabla HTML[17]

5.2.9 Mejoras visuales según las necesidades del cliente

La adaptación a los colores empresariales y a las necesidades del cliente es fundamental para tener un buen contacto con el cliente y conseguir que la aplicación tenga un mejor impacto y sea acorde al resto de aplicativos existentes dentro de la empresa.

5.2.10 Separar el gestor en partes

Para conseguir un mayor rendimiento de la aplicación se podrían separar las funcionabilidades utilizadas por los automatismos y las funcionabilidades utilizadas por el visualizador web.

Bibliografía

[1] "Imagen de base de datos." https://www.thinkartha.com/services/data-migration-services/.

[2] "Imagen de ordenador." https://www.vecteezy.com/vector-art/652938-computer-icon-image.

[3] "Imagen de servicios." https://educosta.dev/blog/what-is-microservice/.

[4] "Información sobre las perspectivas del rpa en 2020." https://www.helpsystems.com/fr/blog/rpa-perspectives-pour-2020.

[5] "Página web que genera las bases de un proyecto en spring." https://start.spring.io/.

[6] "Imagen de escritorio de windows 10." https://www.educaciontrespuntocero.com/tecnologia/.

[7] "Cuadrante de gartner en página oficial de uipath." https://www.uipath.com/es/resources/automation-analyst-reports/gartner-magic-quadrant-robotic-process-automation.

[8] "Imagen de modelo de azure queue storage." https://thegreenerman.medium.com/azure-storage-queues-and-azure-functions-storage-triggers-9d1d111ff1c0.

[9] "Diagrama de una topología simple de websphere mq." https://www.ibm.com/docs/en/ibm-mq/7.5?topic=websphere-mq-managed-file-transfer.

[10] "Imagen de interconexiones con apache kafka." https://axual.com/what-is-apache-kafka/.

[11] "Imagen del diagrama de resultados en appian." https://appian.com/es-es/why-appian/about.html.

[12] "Imagen del editor de diagramas de estado de jira." https://www.atlassian.com/es/software/jira/whats-new/core-experiences.

[13] "Imagen definiciones solid." https://yosoydani.com/principios-solid/.

[14] "Página de apoyo al desarrollador yolearnonline." https://www.yolearnonline.com/php/phpmyadmin.html.

[15] "Página de tutorias y tecnología al desarrollador geeknetic." https://www.geeknetic.es/Noticia/11610/Como-abrir-correctamente-un-archivo-CSV-en-Excel.html.

[16] "Página de busqueda de aplicaciones moviles getapp." https://www.getapp.cl/software/121647/uipath-robotic-process-automation.

[17] "Tutorias de solvetic de cómo crear filtros en tablas html." https://www.solvetic.com/tutoriales/article/852-filtrar-y-gestionar-tablas-html-con-jquery/.

[18] "Página con la información para la implementación y utilización de redis." https://redis.io/docs.

[19] "Página con la información para la implementación y utilización de kafka." https://kafka.apache.org/documentation/#.

[20] "Página con la información sobre azure queue storage." https://azure.microsoft.com/es-es/services/storage/queues/ y https://docs.microsoft.com/es-es/azure/storage/queues/storage-queues-introduction.

[21] "Página con la información sobre json: Javascript object notation." https://es.wikipedia.org/wiki/JSON.

[22] "Página con la información sobre xml: Extensible markup language." https://es.wikipedia.org/wiki/Extensible_Markup_Language.

[23] "Definición de mainframe por ibm." https://www.ibm.com/es-es/topics/mainframe.

[24] "Página con la información para la implementación y utilización de ibm webshere mq." https://www.ibm.com/docs/es/ibm-mq/7.5?topic=ssfksj-7-5-0-com-ibm-mq-pro-doc-q001010--htm.

[25] "Página con la información para la implementación y utilización de kafka desde python/django," https://kafka.apache.org/documentation/# y https://pypi.org/project/kafka-python/.

[26] "Página documentación de bootstrap." https://getbootstrap.com/docs/ y https://www.educba.com/bootstrap-toggle-button/.

[27] "Página botones para bootstrap." https://icons.getbootstrap.com/.

[28] "Página documentación de w3schools." https://www.w3schools.com/.

[29] "Página de apoyo al desarrollador stackoverflow." https://stackoverflow.com/.

[30] "Página documentación de thymeleaf." https://o7planning.org/12373/thymeleaf-th-class-th-classappend-th-style-th-styleappend.

[31] "Página de documentación del app service de azure." https://docs.microsoft.com/es-es/azure/app-service/deploy-staging-slots.

[32] "Información sobre gnu/linux en wikipedia," http://es.wikipedia.org/wiki/GNU/Linux [Último acceso 1/noviembre/2013].

[33] "Imagen de la control room de blue prism." https://bpdocs.blueprism.com/bp-7-0/en-us/control-queues.html.

[34] "Imagen de listado de tareas repetitivas." https://www.freepik.es/fotos-premium/pila-primer-plano-monton-papel-papeleo-informe-o-documento-impreso-escritorio-oficina-apilar_8949938.htm.

[35] "Página sobre procesador de texto notepath++," https://notepad-plus-plus.org/.

[36] "Página w3 schools resuelve muchas de las consultas sobre python, html, css y javascript," https://www.w3schools.com/.

[37] "Página para creacion de velocimetro," https://canvas-gauges.com/.

[38] "Página para gráficos estadisticos," https://www.chartjs.org/.

Apéndice A

Manual de usuario

Solo sé que no se nada...

Sócrates

A.1 Introducción

El manual de usuario de este aplicativo es bastante sencillo porque cuenta con poca interacción. La navegación por la web es bastante intuitiva, no tiene grandes complejidades de manejo. El manual de usuario se divide en:

- Visualizador web:
  - Visualización datos generales
  - Visualización del listado y funciones en tareas
  - Visualización del listado y funciones en datos
  - Visualización del listado y operación con las máquinas robot
  - Modales y alertas
    - Nuevo caso
    - Modal excepción manual
    - Modal asignar robot
    - Editar caso
    - Alerta de eventos
    - Confirmación de cambios estado
- Plugin Blue Prism
  - Actualizar datos
  - Completar
  - Completar con resultado
  - Comprobar conexión
  - Excepcionar caso con reintentos

- Excepcionar caso sin reintentos
- Liberar caso
- Registro máquina
- Solicitar caso

A.2 Visualizador web

A.2.1 Visualización básica

La página principal muestra el resultado de los casos totales en cada uno de sus posibles estado y una gráfica circular con los porcentajes de cada uno.

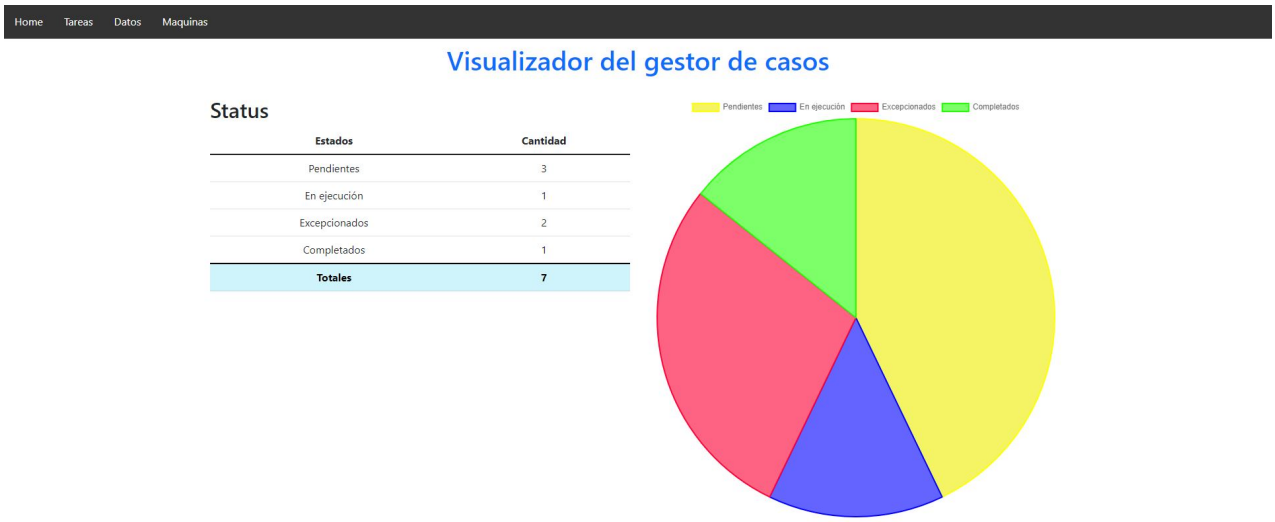


Figura A.1: Página de inicio o página home.

A.2.2 Visualización del listado y funciones en tareas

En el apartado de tareas se puede visualizar un resumen en la parte superior y en orden inverso de ID para facilitar los casos pendientes el listado de tareas completo. Desde el visualizador se pueden realizar las siguientes tareas. Según el estado en el que se encuentre el caso se pueden realizar diferentes acciones.

- Con el caso completado no se pueden realizar ninguna acción.
- Con el caso excepcionado solo se puede reintentar siempre que no haya otra tarea del mismo caso en pendientes, asignado o en ejecución.
- En ejecución solo se puede desbloquear el caso.
- En pendiente el caso se puede asignar un robot o desasignarlo y excepcionarlo sin reintentos con un mensaje de error.

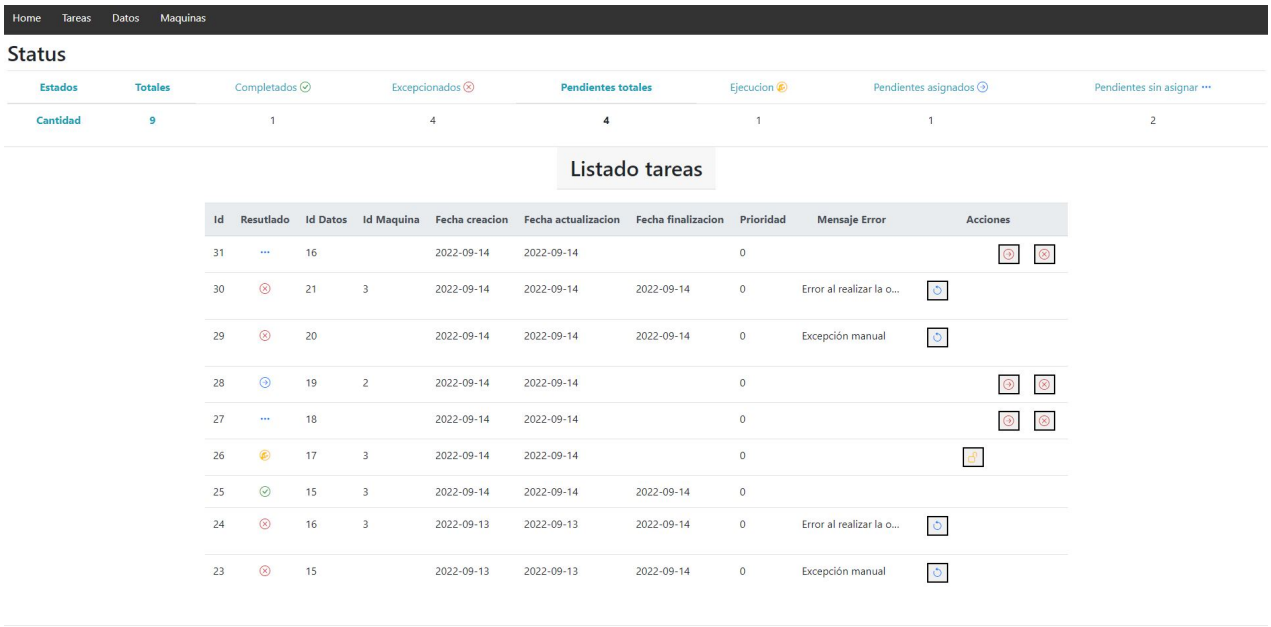


Figura A.2: Página de tareas.

A.2.3 Visualización del listado y funciones en datos

Desde esta ventana se pueden crear casos y visualizar los totales y el listado de casos.
Desde este apartado se pueden borrar los casos que están excepcionados o completados. Siempre se permite editar el caso, pudiendo asignar un mensaje de error o un resultado al caso.
Mientras el caso este en ejecución no se puede editar pero mientras esta pendientes si se puede.

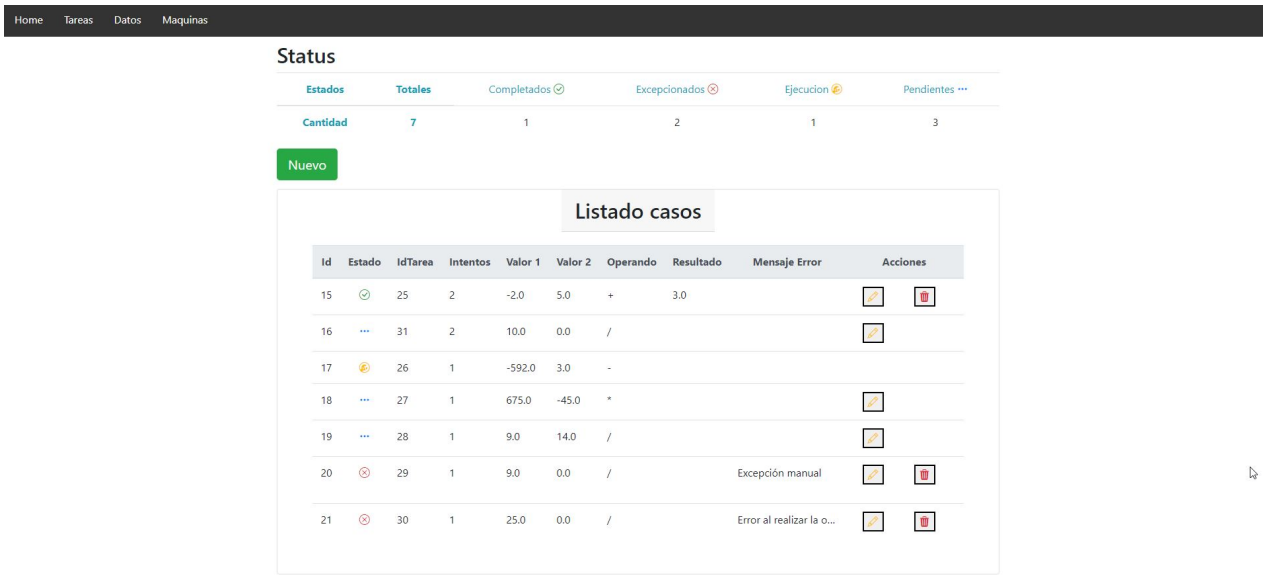


Figura A.3: Página de datos.

A.2.4 Visualización del listado y operación con las máquinas robot

Desde la pestaña de máquinas se puede visualizar la cantidad de robots existentes.
Desde esta ventana se pueden activar y desactivar las máquinas. Aparece una alerta de cambio de estado

de la máquina y además si la máquina tenía asignado casos cuando la desactivas los casos se desasignan para que puedan ser cogidos por otra máquina.

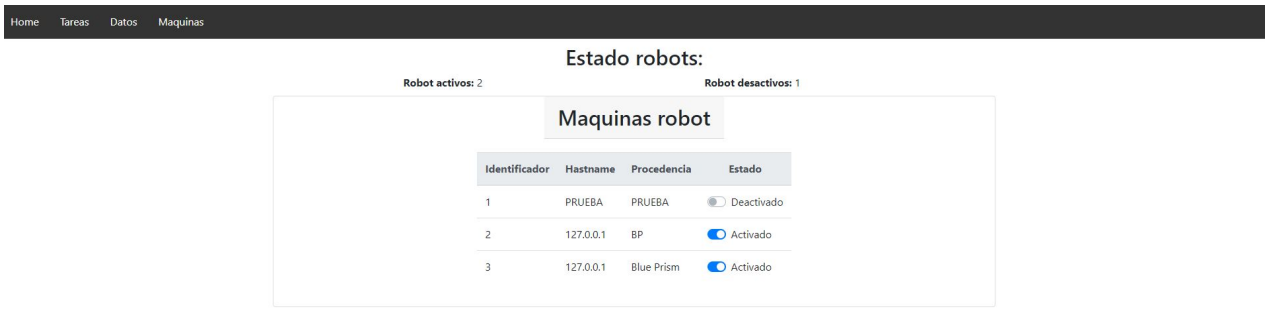


Figura A.4: Página de máquinas.

A.2.5 Modales y alertas

Los modales y alertas se utilizan para interactuar con los diferentes tareas, casos o máquinas.

A.2.5.1 Nuevo caso

Este modal aparece cuando pulsamos el botón de nuevo en la ventana de datos.

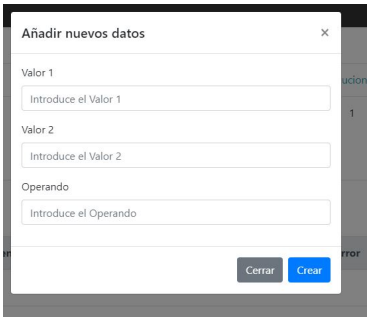


Figura A.5: Modal de creación de casos.

A.2.5.2 Modal excepción manual

Este modal aparece cuando pulsamos el botón de excepcionar en la ventana de tareas y permite bloquear un caso como excepción.

Figura A.6: Modal de excepcionar caso.

A.2.5.3 Modal asignar robot

Este modal aparece cuando pulsamos el botón de asignar robot en la ventana de tareas y nos permite asignar una máquina concreta o desasignar una máquina concreta.

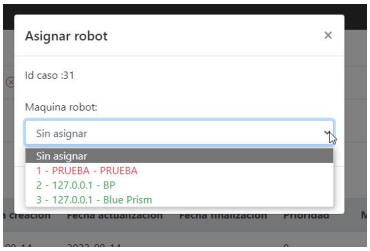


Figura A.7: Modal de asignar un robot.

A.2.5.4 Editar caso

Este modal permite editar los datos de un caso desde la ventana de datos.

Figura A.8: Modal de editar un caso.

A.2.5.5 Alerta de eventos

Cuando se interviene con un caso, tarea o maquina sale un mensaje que notifica el resultado de la operación.

Figura A.9: Mensajes de información de ejemplo de una edición de un caso.

A.2.5.6 Confirmación de cambios estado

Cuando se realiza desea cambiar el estado de una máquina se genera un mensaje de confirmación para que se realice el cambio.

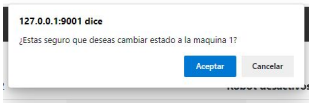


Figura A.10: Mensajes de confirmación del cambio de estado de una máquinas.

A.3 Plugin Blue Prism

El conjunto de eventos disponible a ejecutar es:

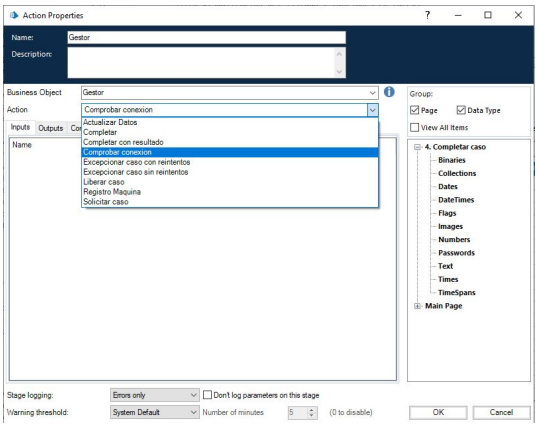


Figura A.11: Listado de acciones de Blue Prism.

Todas las acciones tienen como respuesta los campos:

Campo	Tipo	Grupo
Response Headers	colección	ouptut
HTTP Status Code	texto	ouptut
Response Content	texto	ouptut

Tabla A.1: Campos output comunes

A.3.1 Actualizar datos

Desde este evento se pueden actualizar los datos del caso.

Campo	Tipo	Grupo
idtarea	numero	input
resultado	numero	input
valor1	numero	input
valor2	numero	input
operando	texto	input
resultado	flag	ouptut
mensaje	texto	ouptut

Tabla A.2: Campos exclusivos de completar tarea

A.3.2 Completar

Desde este evento se pueden completar el caso.

Campo	Tipo	Grupo
idtarea	numero	input
resultado	numero	input
resultado	flag	ouptut
mensaje	texto	ouptut

Tabla A.3: Campos exclusivos de completar tarea con resultados

A.3.3 Completar con resultado

Desde este evento se pueden completar el caso guardando además el resultado de la operación.

Campo	Tipo	Grupo
idtarea	numero	input
resultado	numero	input
resultado	flag	ouptut
mensaje	texto	ouptut

Tabla A.4: Campos exclusivos de actualizar datos con resultados

A.3.4 Comprobar conexión

Intervención para realizar comprobaciones de saludo con el servicio y comprobar que está disponible.

A.3.5 Excepcionar caso con reintentos y sin reintentos

Permite excepcionar una tarea o un caso completamente si no ha reintentos, aunque se marque que se reintente si se llega a el máximo de reintentos (3 en la definición del servicio) se excepciona el caso.

Campo	Tipo	Grupo
idtarea	numero	input
mensajeError	texto	input
resultado	flag	ouptut
mensaje	texto	ouptut

Tabla A.5: Campos exclusivos de excepcionar tareas y casos

A.3.6 Liberar caso

El servicio permite liberar un caso cogido por el robot.

Campo	Tipo	Grupo
idtarea	numero	input
resultado	flag	ouptut
mensaje	texto	ouptut

Tabla A.6: Campos exclusivos de liberar datos

A.3.7 Registro máquina

Los robot deben registrar, inicialmente se registran sin permisos de ejecución.

Campo	Tipo	Grupo
mensaje	texto	ouptut
idmaquina	texto	ouptut
hastname	texto	ouptut
procedencia	texto	ouptut
estado	flag	ouptut

Tabla A.7: Campos exclusivos de liberar datos

A.3.8 Solicitar caso

Desde este evento se pueden actualizar los datos del caso.

Campo	Tipo	Grupo
idtarea	numero	ouptut
resultado	numero	ouptut
valor1	numero	ouptut
valor2	numero	ouptut
operando	texto	ouptut
resultado	flag	ouptut

Tabla A.8: Campos exclusivos de completar tarea

Apéndice B

Manual de instalación

B.1 Introducción

El manual de instalación describe la manera de instalar las herramientas necesarias para el funcionamiento del proceso. El manual de instalación se divide en:

- Instalación básica del software del gestor y visualizador
- Instalación del plugin de Blue Prism
- Despliegue en Azure Cloud

B.2 Instalación básica del software del gestor y visualizador

**Se necesita de un servidor dedicado sobre el que ejecutar las aplicaciones.*
Para poder ejecutar la aplicaciones en el servidor se necesita:

- Aplicaciones:
  - Instalar la versión JRE de Java 17
  - Instalar Maven
  - Instalar los drivers para entablar comunicación con las bases de datos MySQL
  - Disponer de puertos o enpoint diferentes para el gestor y el visualizador web.
- Para la creación de la bases de datos es necesario tener conexión a la misma previamente a la ejecución del proceso.
- Para poder visualizar los resultados se necesita de un gestor que permita realizar llamadas a los servicios, Postman o similar.
- Para visualizar la página es necesario disponer de un visualizador web como Chrome, Firefox o Microsoft Edge

B.3 Instalación en Plugin

Para realizar la instalación del plugin es necesario disponer de una licencia activa de Blue Prism y permisos para importar el software.

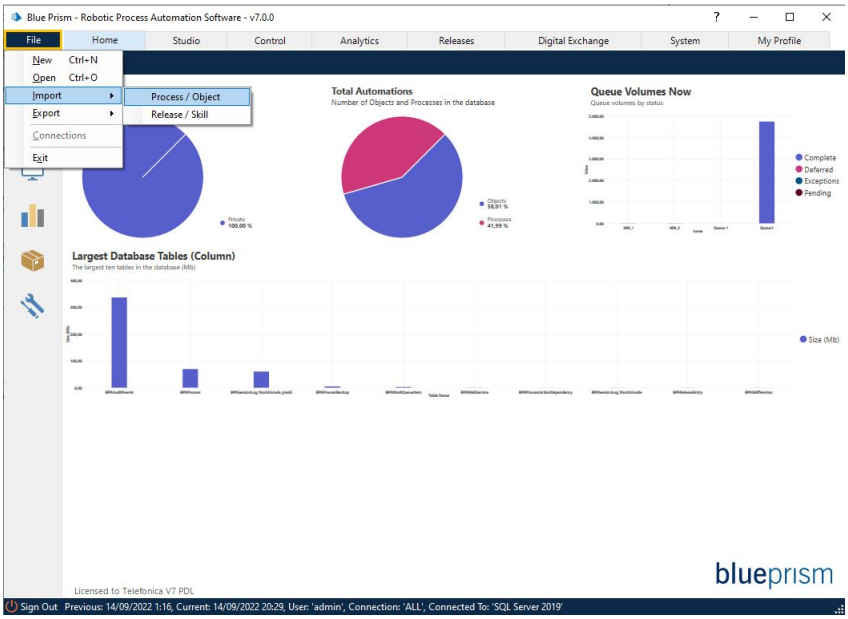


Figura B.1: Importar objectos a Blue Prism.

B.4 Despliegue en Azure Cloud

Para la instalación en Azure existe la posibilidad de desplegar automáticamente en la parte de App Service[31], desde un repositorio o subir los ficheros al contenedor que ejecutará la aplicación.

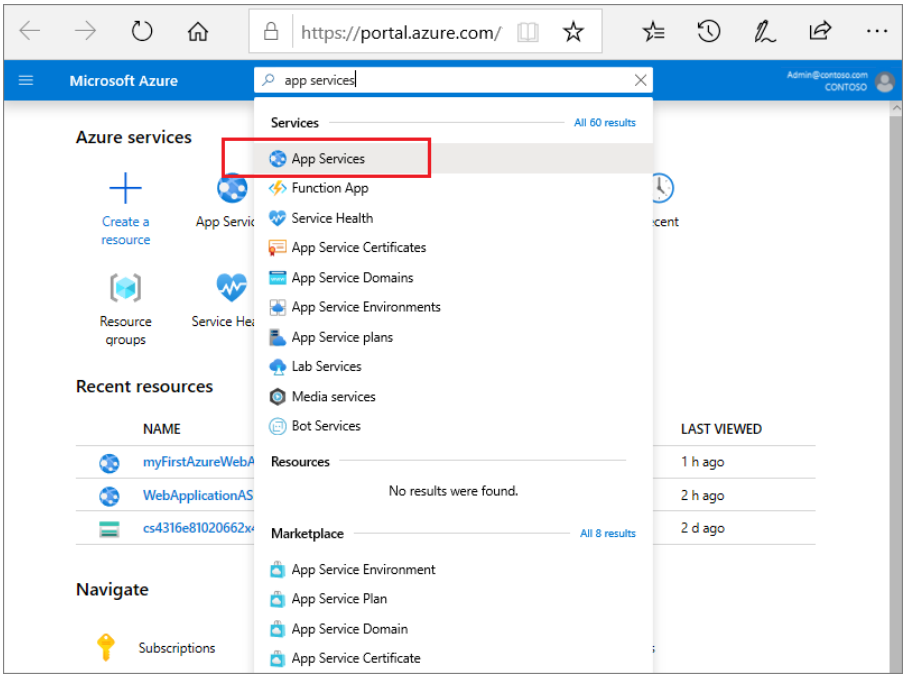


Figura B.2: Crear un App Services en el portal de Azure.

Apéndice C

Herramientas y recursos

Las herramientas necesarias para la elaboración del proyecto han sido:

- PC compatible
- Sistema operativo GNU/Linux[32], Windows o Mac
- IntelliJ Comunity o Pro.
- Interprete JDK de Java 17
- WorkBench MySQL
- Docker para crear un contenedor de MySQL
- Licencia de Blue Prism y software para desarrollar el plugin sobre sistema Windows

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá