CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Cuadro de mando para la monitorización de una aplicación bancaria

Autor: Raquel Santana García

Director: Fernando Pérez Costoya

MADRID, JUNIO 2019

Agradecimientos

En primer lugar, me gustaría dar lar gracias a mi familia por aguantarme estos cuatros años en los peores momentos, en los cuales me han dado los ánimos para continuar con este reto.

También quiero agradecérselo a mi amiga Paola, la cual me lleva sufriéndome durante casi 10 años, y me ha obligado muchas veces a salir de casa para que me tomase un respiro.

Agradecer a mi compañero Alberto, que me ha ayudado y aguantado durante todo el desarrollo del proyecto.

Agradecer a mi compañero Jorge, el cual ha sufrido lo mismo que yo y ha entendido mi agobio.

Por último, agradecer a una persona que conozco desde hace pocos meses y me ha aguantado en este ultimo empujón y no ha huido.

Contenido

Tabla de ilustraciones

1. RESUMEN DEL TRABAJO

Hoy en día las aplicaciones sufren fallos que producen que la aplicación deje de funcionar correctamente, por lo tanto, esto requiere una monitorización de las aplicaciones para que en caso de fallo se pueda detectar el problema de manera rápida y sencilla.

El presente trabajo de fin de grado tiene como objetivo el desarrollo de un sistema que permita la monitorización tanto del sistema interno una aplicación como las llamadas a los servicios externos, para así tener una información global del estado de la aplicación y poder conocer en caso de fallo de la aplicación cuál ha sido el problema.

El sistema a desarrollar será el encargado de monitorizar los procesos internos de una aplicación como las llamadas a servicios externos.

También se desarrollará un cuadro de mando web, en el cual se mostrará toda la información de la aplicación, tanto procesos internos como externos.

2. SUMMARY

Nowadays applications suffer failures that cause the application to stop working correctly, therefore, this requires monitoring of the applications so that in case of failure the problem can be detected quickly and easily.

The purpose of this final degree project is to develop a system that allows the monitoring both the internal system of an application and calls to external services, to have global information of the status of the application and to know in case of failure of the application which has been the problem.

The system to be developed will be responsible for monitoring the internal processes of an application such as calls to external services.

A web control panel will also be developed, in which all the information of the application will be shown, both internal and external processes.

3. INTRODUCCIÓN

Este proyecto surge ante la necesidad de la empresa, en la que actualmente trabajo, de un sistema que monitorice las diferentes partes que constituyen la aplicación que desarrolla mi empresa.

La necesidad de un sistema que monitorice todas estas partes se debe a que la aplicación consta de llamadas a servicios externos, los cuales en ocasiones no funcionan correctamente o se produce alguna caída del sistema lo que produce fallos en la aplicación, también se debe en ocasiones a problemas internos de la propia aplicación.

El objetivo de este sistema es monitorizar tanto el sistema interno de la propia aplicación como las llamadas a los servicios externos, para así tener en un mismo sistema toda la información agrupada y en caso de fallo, saber en un menor periodo de tiempo si el fallo se ha debido a un fallo interno de la propia aplicación o del servicio externo. En el caso de que el fallo sea interno de la aplicación el sistema mostrará con información detallada a qué se ha debido dicho fallo.

Las tecnologías que se han utilizado en este proyecto han sido las mismas que se utilizan en la aplicación tanto en la parte de *backend* como de *frontend*, esto ha facilitado la integración del servicio para monitorizar la aplicación.

Algunas de las decisiones que he tomado durante el desarrollo de este proyecto han estado relacionadas con el diseño y la estructura. Las decisiones relacionadas con el diseño han sido la manera en la que se va a mostrar la información y cuál va a ser la información que se va a mostrar. Las decisiones relacionadas con la estructura han sido en la parte de *backend* a la hora de estructurar todo el proyecto y la monitorización de cada uno de los sistemas.

4. OBJETIVOS

El objetivo principal de este trabajo consiste en la implementación de un cuadro de mando que permita monitorizar una aplicación del ámbito bancario orientada a la reducción del uso de papel a la hora de la firma de los documentos para las diferentes operaciones del cliente del banco (contrataciones, pagos, cancelaciones, etc.).

Para cumplir el objetivo principal del trabajo, se ha divido dicho trabajo en distintas tareas, las cuales serán necesarias para el desarrollo del mismo:

- Investigación de las tecnologías de desarrollo de interfaces de usuario.
- Diseño de la arquitectura del sistema.
- Diseño y desarrollo de la interfaz de usuario.
- Investigación de los servicios de monitorización
- Análisis, diseño e implementación de uso de monitorización.

5. TRABAJOS PREVIOS

Al tratarse de un cuadro de monitorización web, es necesario el desarrollo de la parte de *frontend* como de *backend*, por lo tanto, es necesario el uso de varios lenguajes para el desarrollo de ambas partes.

5.1. Lenguajes utilizados

En este apartado se explicarán los diferentes lenguajes utilizados durante el desarrollo del proyecto y algunas de sus características.

5.1.1. Java

Java[1] es un lenguaje de programación de propósito general, concurrente, orientado a objetos, el cual fue diseñado para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo, esto significa que el código que es ejecutado en una plataforma no tiene que ser recompilado para poder ejecutarse en otra.

Figura 1. Logotipo de Java.

5.1.2. HTML

HTML (*HyperText Markup Language*)[2] es el elemento de construcción más básico en una página web. Su uso principal consiste en la creación y representación visual de una página web. HTML solo determina el contenido de la página web, no su funcionalidad. Son necesarias otras tecnologías distintas de HTML para describir la apariencia de una página web como CSS o su funcionalidad como JavaScript.

Figura 2. Logotipo de HTML.

5.1.3. CSS

CSS[3] es el lenguaje utilizado para describir la apariencia de documentos HTML o XML, esto incluye varios lenguajes que se basan en XML, algunos de ellos son XHTML o SVG. CSS describe cómo debe caracterizarse el elemento estructurado en la pantalla.

Figura 3. Logotipo de CSS.

5.1.4. JavaScript

JavaScript[4] es un lenguaje de programación interpretado. Se define como un lenguaje orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado cliente, implementado como una parte de un navegador web. JavaScript permite mejoras en la interfaz de usuario y páginas web dinámicas.

Figura 4. Logotipo de JavaScript.

6. DESARROLLO DEL PROYECTO

6.1. Requisitos

El cuadro de monitorización web tendrá que cumplir los siguientes requisitos:

- Mostrar el número de paquetes que se encuentran en cada estado y su porcentaje respecto al total.
- El color del porcentaje de paquetes de un determinado estado debe cambiar en el caso de que dicho porcentaje no se encuentre dentro de unos valores estables.
- Respecto a los paquetes que se encuentran en error mostrar una tabla con información más detallada de dichos paquetes.
- Debe enviar un correo de notificación en el caso de que los porcentajes de los paquetes no se encuentren dentro de los valores estables.
- Mostrar en un semáforo el estado de los servicios externos de la aplicación y los tiempos de respuesta correspondientes a cada servicio.
- El color del semáforo debe cambiar si los tiempos de respuesta de los sistemas externos son elevados.
- Debe enviar un correo de notificación en el caso de que se produzca la caída de alguno de los sistemas externos.

De esta lista de requisitos se han implementado la mayoría de ellos, quedando los que no se han podido implementar para futuras mejoras.

6.2. Sistemas monitorizados

6.2.1. Alnova

Alnova es una solución de banca central que introduce la banca de alto rendimiento. Alnova es un sistema desarrollado por la empresa *Alnova Technologies Corporation*[5], esta empresa se encarga de desarrollar y comercializar licencias de soluciones tecnológicas con el fin de satisfacer las necesidades del sector financiero, que incluyen desde la consolidación y centralización de todos los procesos y sistemas, hasta la automatización del back-office, el desarrollo de los sistemas de CRM, la banca por Internet y la distribución multicanal.

Alnova consta de las siguientes características:

1. Nuevo modelo centrado en el cliente.
2. Mejora de las capacidades de venta cruzada.
3. Adapta y facilita la evolución bancaria central.
4. Solución modular para una transformación completa o dirigida.
5. Solución en tiempo real y parametrizada para eficiencias de procesamiento.
6. Capacidad de adaptarse de manera rápida y eficiente a los cambios por demandas del cliente.
7. Modelo distribuido multi-entidad y multi-estado.

El *frontend* de Alnova consiste en una plataforma de sucursal permitiendo realizar el servicio y las ventas a través de la solución *frontend* de Nacar.

El procesamiento del *Batch* de Alnova se realiza a través de una secuencia de Jobs controlados por la herramienta de automatización de la empresa Control-M.

6.2.2. Docusign

Docusign[6] es un Software de *Digital Signature Software* creado por la empresa Docusign. Docusign reemplaza la impresión, fax, escaneado de documentos en papel para realizar transacciones comerciales.

Docusign consta de las siguientes características[7]:

- Compatibilidad completa con archivos. Docusign es compatible con prácticamente todos los tipos de archivos de la mayoría de las aplicaciones para garantizar que todos los documentos importantes sean enviados con firma. Además, Docusign reconoce documentos en PDF y marca automáticamente los campos de relleno para que los firmantes introduzcan sus datos.
- Conversión de formulario PDF. Al cargar un PDF, Docusign automáticamente reconoce y convierte los campos PDF en campos para rellenar por el firmante. Esta función ahorra tiempo en la preparación de documentos de suscripción.
- Más de 20 marcas y campos estándar y personalizados. Permite orientar a los clientes para que firmen y marquen las iniciales en los lugares correctos. Permite utilizar marcas estándar para recopilar firmas, iniciales, nombres, títulos, nombres de empresa y otra información relevante. También se pueden modificar las marcas para propósitos específicos y guardarlas como marcas personalizadas para su uso futuro.
- Integración de almacenamiento en la nube. Permite recuperar documentos de servicios de almacenamiento en nube ampliamente usados.
- Marcas automáticas. Permite anclar marcas y campos en determinadas líneas del texto para que, cuando los coloque en un documento, aparezcan automáticamente en el lugar correcto. Aunque el documento cambie, las marcas de anclaje acompañan el texto.
- PowerForms. PowerForms permite gestionar documentos bajo demanda de autoservicio para su firma. Esto ayuda a eliminar el tiempo de preparación de documentos y transfiere fácilmente los datos recopilados a las aplicaciones existentes.
- Comentarios. Los comentarios permiten que el remitente y los destinatarios de un sobre intercambien notas dentro de un documento directamente de las experiencias de suscripción Web y de dispositivos móviles de Docusign. La característica ofrece notificaciones en tiempo real.

Figura 5. Logotipo de Docusign.

6.2.3. Docpath

Este sistema es un software documental desarrollado por la empresa Docpath. Docpath[8] permite la generación de documentos en toda la red de una oficina de forma fácil y eficiente.

Las principales ventajas[9] de Docpath para sus procesos documentales se dividen en dos áreas: beneficios globales y técnicos.

Los beneficios globales son:

- Mejora importante de las comunicaciones con clientes y terceros, mediante TransPromo, generación y distribución de documentos online y una gran variedad de canales de distribución.
- Unificación de los documentos reduciendo el número de modelos, ya que cada plantilla diseñada permite generar automáticamente diferentes documentos para cada cliente.
- Solución multiplataforma.
- Una única solución para producción en *batch*, entornos Web, etc.
- Control de impresión con perfiles de ahorro de tóner.
- Integración con diferentes gestores documentales.
- Integración con los principales ERPs (*Enterprise Resource Planning*) del mercado.
- Posibilidad de renting para licencias de software Docpath.
- Flexibilidad en todos los procesos, todas las soluciones de Docpath son escalables y pueden expandirse para adaptar a los requisitos específicos del negocio.

Figura 6. Logotipo de Docpath.

6.3. Tecnologías utilizadas

Al tratarse de un cuadro de monitorización web, es necesario el desarrollo *frontend* como *backend*, por lo tanto, será necesario el uso de tecnologías de ambas partes. Las tecnologías que se describirán a continuación son las mismas tecnologías que se utilizan en la aplicación a monitorizar, esto ha permitido poder integrar el servicio de monitorización de una manera más sencilla.

6.3.1. Tecnologías *frontend*

6.3.1.1. AngularJS 1.3

AngularJs[10] es un *framework* de Javascript de código abierto, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo principal es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC)

en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles. La versión 1.3[11] de Angular añade nuevas funciones, las cuales son:

- ngAria: nuevo módulo que ayuda a que los componentes personalizados en Angular sean más accesibles por defecto.
- ngMessages: nueva directiva que simplifica la escritura y coordinación del feedback sobre la validez del formulario.
- ngModelOptions: directiva que facilita la personalización de los modelos enlazados.
- Strict DI: opción para encontrar lugares en la aplicación que no se minimicen debido al uso de sintaxis short-hand DI.

Figura 7. Logotipo de AngularJS.

6.3.1.2. Bootstrap 3.3

Bootstrap[12] es un *framework* desarrollado por Twitter, cuyo objetivo es facilitar el diseño web. Permite crear de forma sencilla webs que se ajusten a cualquier dispositivo y tamaño de pantalla, manteniendo la misma calidad de la web. A diferencia de muchos *frameworks* web, solo se ocupa del desarrollo *frontend*. La versión 3.3[13] de Bootstrap tiene algunos aspectos a destacar, los cuales son:

- Se agregó un conjunto de nuevas variables *Less* para una personalización más fácil.
- Se eliminaron los cambios recientes de la barra de progreso para porcentajes bajos.
- Se eliminaron todas las instancias de translate3d a medida que mejoraron el rendimiento de *repaint*, pero también se agregaron varios errores de navegación cruzados.
- Se han añadido *transforms* para mejorar el rendimiento del carrusel de navegadores modernos.
- Actualizado Normalize.css y sus estilos de impresión H5BP a sus últimas versiones.
- Mejora en la accesibilidad para *navs, panels, tooltips, buttons* y más.
- Resolución de errores de documentación y JavaScript.

Figura 8. Logotipo de Bootstrap.

6.3.1.3. NodeJS 8

Node.js[14] es un entorno Javascript de lado del servidor, basado en eventos. Node ejecuta javascript utilizando el motor V8. Aprovechado el motor V8 permite a Node proporcionar un entorno de ejecución del lado del servidor que compila y ejecuta javascript a velocidades increíbles.

La versión 8 de Node.js[15] se suministra con V8 5.8, una actualización significativa del tiempo de ejecución de JavaScript que incluye mejoras importantes en el rendimiento y el desarrollo frente a las APIs.

Figura 9. Logotipo de NodeJS.

6.3.2. Tecnologías *backend*

6.3.2.1. Spring core 3.0.7

Spring Core[16] es un *framework* que proporciona un completo modelo de programación y configuración para aplicaciones empresariales modernas basadas en Java, en cualquier tipo de plataforma de implementación.

Un elemento clave de Spring es el soporte de infraestructura a nivel de aplicación: Spring se enfoca en la "tubería" de las aplicaciones empresariales para que los equipos puedan enfocarse en la lógica de negocios a nivel de la aplicación, sin vínculos innecesarios con entornos de implementación específicos.

Las características más importantes de Spring Core son:

- Tecnologías básicas: inserción de dependencias, eventos, recursos, i18n, validación, enlace de datos, conversión de tipos, SpEl, AOP.

- Pruebas: objetos ficticios, TestContext *framework*, Spring MVC Test, WebTestClient.
- Acceso a datos: transacciones, soporte DAO, JDBC, ORM, Marshalling XML.
- Spring MVC and Spring WebFlux web frameworks.
- Integración: comunicación remota, JMS, JCA, JMX, email, tareas, programación, cache.
- Lenguajes: Kotlin, Groovy, lenguajes dinámicos.

Figura 10. Logotipo de Spring Core.

6.3.2.2. Spring jdbc 4.3.5

Spring JDBC[17] facilita la implementación de repositorios basados en JDBC. Este módulo trata sobre el soporte mejorado para capas de acceso a datos basadas en JDBC. Facilita la creación de aplicaciones impulsadas por Spring que utilizan tecnologías de acceso a datos.

Spring JDBC pretende ser conceptualmente fácil. Para lograr esto, NO ofrece almacenamiento en caché, carga diferida o muchas otras características de JPA. Esto hace que Spring JDBC sea un ORM simple y limitado.

Las características más importantes de Spring JDBC son:

- Operaciones CRUD para agregados simples con NamingStrategy personalizable.
- Soporte para anotaciones @Query.
- Soporte para consultas MyBatis.
- Eventos.
- Configuración de repositorios basada en JavaConfig introduciendo @EnableJdbcRepositories.

Figura 11. Logotipo de Spring JDBC.

6.3.2.3. Httpcore-4.2.3

HttpCore[18] es un conjunto de componentes de transporte HTTP de bajo nivel que se pueden usar para crear servicios personalizados de cliente y servidor HTTP con una huella mínima.

HttpCore permite dos modelos de entrada/salida basado en la entrada/salida clásica de Java y el modelo entrada/salida de bloqueo de eventos basado en Java NIO.

HttpCore tiene las siguientes características[19] de soporte:

- Soporte HTTP/1.1 y HTTP/2.
- Compatibilidad con *async/await* para peticiones HTTP no bloqueantes.
- Tiempos de espera estrictos en todas partes por defecto.
- 100% cobertura de prueba.

6.3.2.4. Oracle Sql Developer 18.2.0.183

Oracle Sql Developer[20] es un entorno de desarrollo integrado y gratuito que simplifica el desarrollo y la administración de Oracle Database tanto en implementaciones tradicionales como en la nube. SQL Developer ofrece un desarrollo completo de sus aplicaciones PL / SQL, una hoja de trabajo para ejecutar consultas y scripts, una consola DBA para administrar la base de datos, una interfaz de informes, una solución completa de modelado de datos y una plataforma de migración para mover bases de datos de terceros a Oracle.

En la versión 18.2 se añadieron nuevas características[21], las cuales son:

- Permite importar ficheros Excel más rápido.
- Compatibilidad con sintaxis Oracle JOIN.

Figura 12. Logotip de SQL Developer.

6.3.2.5. Oracle 12c

Oracle Database[22] 12c es un sistema de gestión de bases de datos relacionales, disponible para un gran número de plataformas.

Oracle 12c se comercializa en tres gamas:

- Edición Estándar. Esta edición incluye todas las funcionalidades básicas que permiten implementar aplicaciones cliente-servidor para un grupo de trabajo o

departamento de una empresa. En esta edición se limita a servidores o clústers de servidores, con una capacidad máxima de cuatro procesadores.

- Edición Entreprise. Esta edición se destina principalmente a las aplicaciones criticas de una empresa y ofrece funcionalidades adicionales, de manera estándar.
- Edición Personal. Esta edición está destinada únicamente a plataformas Windows.

7. ESTRUCTURA DE LA APLICACIÓN

7.1. Estructura *backend*.

La arquitectura *backend* de la aplicación se basa en un proyecto con tecnologías Java.

La estructura del proyecto se divide en las siguientes partes:

- Librerías: conjunto de programas que serán utilizados dentro de las transacciones a la hora de realizar llamadas a sistemas externos a la aplicación.
- Transacciones: conjunto de programas que se utilizarán a la hora de realizar el proceso de firma de un documento.

La estructura de una transacción es la siguiente:

- src/main/java: conjunto de ficheros .java en los cuales se encuentra todo el desarrollo de la funcionalidad de la transacción.
- src/resources: conjunto de ficheros en los cuales se encontrarán ficheros de configuración, por ejemplo, las conexiones a las bases de datos o ficheros .xml con la configuración de los parámetros de entrada y de salida de la transacción. También se pueden encontrar en esta carpeta ficheros en los cuales se encuentran las consultas a la base de datos.
- pom.xml: fichero en el que se encuentran todas las dependencias de la transacción, tanto llamadas a las librerías de la aplicación como librerías externas.
- target: directorio con los ficheros .jar de la transacción una vez se ha compilado

7.2. Estructura *frontend*.

La arquitectura *frontend* de la aplicación se basa en un proyecto con tecnologías NodeJS y el *framework* AngularJS en la versión 1.3.

Dentro de las dependencias generales se pueden extraer diferentes librerías:

- Gulp: automatización en el proceso de construcción de la aplicación: generación e inyección de los ficheros estáticos, construcción y despliegue de los ficheros en las carpetas build y empaquetado de la aplicación.
- Karma: desarrollo y ejecución del *testing.*
- Socket.io: gestión de las peticiones contra los diferentes servicios REST de la aplicación.
- Express: servidor web para el despliegue de la aplicación.
- Mongoose: comunicación con la base de datos MongoDB con toda la configuración, cacheado de elementos, etc.
- ESLint: validador del formato y estructura del código.

La estructura del proyecto sería:

- build: directorio donde se construye la aplicación.
- client: directorio donde se encuentra todo el código de la aplicación. Está dividido por submódulos:

    - assets: directorio con los ficheros estáticos de la aplicación (imágenes, fuentes, etc.)
    - lib: directorio con las librerías de las que depende el proyecto (angular, Bootstrap, etc.)
    - modules: directorio con el código de la aplicación.
    - views: directorio con las templates globales
- config: directorio con los ficheros de configuración del servidor con diferentes propiedades dependiendo del entorno
- node_modules: directorio con la instalación de todas las dependencias del proyecto.
- server: directorio con el código asociado al servidor que arranca la aplicación
- unit_tests: directorio con los test a ejecutar por Karma.
- bower.json: fichero con las dependencias del proyecto enfocadas al front
- karma.conf.js y karma.configuration.js: ficheros de Karma con la definición de propiedades para ejecutar los test.
- package.json : fichero de configuración global del proyecto donde se define el proyecto en sí, las dependencias, comandos para arrancar la aplicación, ...
- server.js: punto de partida para la gestión y arranque del servidor.

7.3. Estructura de la base de datos.

La base de datos de la aplicación consta de un conjunto de tablas, cada tabla contiene una información determinada, la cual será utilizada en los procesos internos de la aplicación.

En el desarrollo *backend* ha sido necesario la consulta a una de estas tablas, la cual tiene los siguientes campos:

- Cod_internal: ID interno para identificar cada operación.
- Cod_external: ID externo de un sistema remoto de firma, por ejemplo, Docusign
- Cod_signature_system: Sistema que gestiona el proceso de firma.
- Cod_channel: Canal que invoca el proceso de firma.
- Cod_primary_participant: Participante primario en el proceso de firma.
- Cod_primary_account: Cuenta primaria en el proceso de firma.
- Cod_signature_type: Tipo de firma.
- Cod_product_type: Tipo de producto.
- Qtl_expiration: Número de días que el paquete estará activo.
- Cod_urn: URN que utiliza Alnova para identificar cada operación.
- Cod_status: Estado actual del paquete.
- Tim_initial: Timestamp que indica cuando se ha generado el paquete.
- Tim_last_mod: Timestamp que indica la última modificación del paquete.
- Cod_initial_user: Usuario que ha generado el paquete.
- Cod_last_mod_user: Usuario que ha generado la última modificación en el paquete.

- Cod_last_mod_ip: IP de la última modificación.
- Cod_access_code: Código de acceso genérico para el paquete.
- Cod_participant_type: Sistema de origen del participante.
- Cod_access_code_review_docs: Código de acceso a Docusign para revisar los documentos.

De todos los campos que contiene la tabla, se han utilizado cod_internal, cod_signature_system, cod_status y tim_last_mod. Todos estos campos son necesarios para saber el número de paquetes y la fecha en la que se encuentran en cada estado.

8. RESULTADOS

Para la monitorización de todos los sistemas ha sido necesario el desarrollo de la parte *backend* como el *frontend*. Para cada uno de los sistemas se explicará el desarrollo de cada parte.

En la parte de *backend* existe un método principal, el cual recibe dos parámetros el primero es la opción del sistema del cual queremos obtener su estado, este parámetro es obligatorio debido a que dependiendo de la opción que reciba realizará un proceso u otro. El segundo parámetro es de tipo JSON, este parámetro no es obligatorio por lo tanto estará vacío salvo que queramos obtener información adicional del sistema.

El método principal devolverá un *String* con la información correspondiente de cada sistema.

En la parte de *frontend* existe una función que se llama cada vez que se carga la página, este método llama a su vez a otras funciones las cuales devuelven la información correspondiente de cada sistema.

8.1. Monitorización de paquetes.

En la parte *backend* el método principal recibe como primer parámetro la opción "*packagesStatus*" y el segundo parámetro es la fecha de la cual queremos obtener los datos de los paquetes. El método principal llamada al método correspondiente para obtener la información de los paquetes.

El método que obtiene la información de los paquetes sigue el siguiente proceso para obtener la información:

1. Obtiene la fecha correspondiente de la cual queremos obtener la información.
2. Después de obtener la fecha, ejecuta unas consultas a la base de datos, estas consultas nos devolverán el número de paquetes totales y los que se encuentran en cada uno de los paquetes en la fecha indicada.
3. A continuación, calculará el porcentaje de paquetes en cada uno de los estados.
4. Todos estos valores se asignan a una variable JSON, la cual en la parte *frontend* nos servirá para obtener los datos.
5. Por último, se convierten todos estos datos en un *String* el cual tiene JsonProperty que son cada uno de los datos de los paquetes.

En la parte *frontend*, toda esta información se encuentra en un componente de interfaz de usuario de tipo *card* con toda la información correspondiente de los paquetes. En este componente de tipo *card* se encuentra un menú desplegable con los tiempos para los que queremos obtener el número de paquetes. Las opciones son:

- 1 minuto.
- 5 minutos.
- 15 minutos.
- 30 minutos.

- 1 hora.
- 6 horas.

Existe una función que realiza la petición a la parte de *backend* para obtener la función, en el caso de que la petición haya ido bien se obtienen los datos los cuales mostraremos en pantalla. Para cada uno de los estados de los paquetes se han establecido unos rangos y dependiendo de dichos rangos el color del porcentaje cambiará de un color a otro.

El resultado que se muestra en la pantalla es el siguiente:

Figura 13. Resultado información de paquetes.

8.2. Monitorización de Alnova.

En la parte *backend* el método principal recibe como parámetro la opción "*alnovaStatus*". El método principal llamada al método correspondiente para obtener la información del estado de Alnova.

La función que obtiene el estado de Alnova realiza el siguiente proceso:

1. Realiza una consulta a la base de datos de la cual obtenemos dos campos que serán necesarios para saber si Alnova funciona correctamente.
2. Con estos campos realiza la llamada a un proceso que llama a Alnova, antes de realizar dicha llamada obtenemos el tiempo en ese momento.
3. Después de realizar la llamada obtenemos otra vez el tiempo actual, con estos dos tiempos se obtiene el tiempo de respuesta.
4. Una vez que el proceso ha terminado, comprobamos el resultado que nos ha devuelto, en el caso de que haya funcionado correctamente, asignaremos el valor true a la variable *alive*. A continuación, asignaremos los valores del tiempo de respuesta y la variable *alive* a una variable JSON, esta variable nos servirá en la parte *frontend* para obtener los datos.
5. Por último, se convierten todos estos datos en un *String* el cual tiene JsonProperty que son los valores de la variable *alive* y el tiempo de respuesta.

En la parte *frontend* esta información se encuentra en un componente de interfaz de usuario de tipo card, la cual contiene un semáforo con el tiempo de respuesta en su interior. Este semáforo tendrá un color determinado dependiendo de si el sistema esta funcionando correctamente y del tiempo de respuesta.

El semáforo será de color verde o naranja si el sistema funciona correctamente, el color verde o naranja varia dependiendo del tiempo de respuesta del sistema. El semáforo será rojo en el caso de que el sistema no funcione correctamente.

En el caso de que el sistema funcione correctamente y el tiempo de respuesta esté dentro de un rango, se mostrará así:

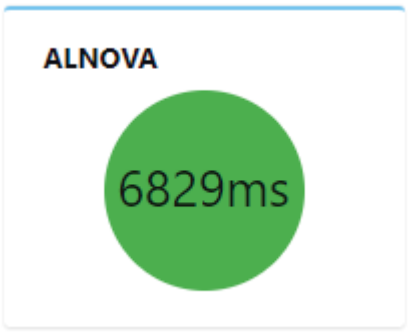

Figura 14. Ejemplo 1 monitorización Alnova.

En el caso de que el sistema funcione correctamente y el tiempo de respuesta sea superior a un rango determinado, se mostrará así:

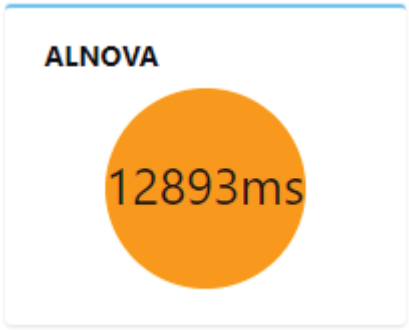

Figura 15. Ejemplo 2 monitorización Alnova.

En el caso de que el sistema no funcione, se mostrará así:

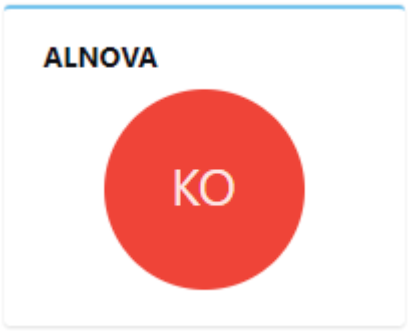

Figura 16. Ejemplo 3 monitorización Alnova.

8.3. Monitorización de Docusign.

En la parte *backend* el método principal recibe como primer parámetro la opción "*docusignStatus*", en este caso no pasamos el segundo parámetro. El método principal llamada al método correspondiente para obtener la información del estado de Docusign.

La función que obtiene el estado de Docusign realiza el siguiente proceso:

1. Realiza la conexión con una URL de Docusign, antes de crear la conexión se obtiene el tiempo actual.
2. Después de crear la conexión, comprobamos que se ha creado correctamente, en el caso de que se haya creado correctamente, se obtiene el tiempo actual en ese momento. Con el tiempo que hemos obtenido anteriormente y el tiempo actual, se obtiene el tiempo de respuesta.
3. Una vez que el proceso ha terminado, comprobamos el resultado que nos ha devuelto, en el caso de que haya funcionado correctamente, asignaremos el valor true a la variable *alive*. A continuación, asignaremos los valores del tiempo de respuesta y la variable *alive* a una variable JSON, esta variable nos servirá en la parte *frontend* para obtener los datos.
4. Por último, se convierten todos estos datos en un *String* el cual tiene JsonProperty que son los valores de la variable *alive* y el tiempo de respuesta.

En la parte *frontend* esta información se encuentra en un componente de interfaz de usuario de tipo card, la cual contiene un semáforo con el tiempo de respuesta en su interior. Este semáforo tendrá un color determinado dependiendo de si el sistema está funcionando correctamente y del tiempo de respuesta.

El semáforo será de color verde o naranja si el sistema funciona correctamente, el color verde o naranja varía dependiendo del tiempo de respuesta del sistema. El semáforo será rojo en el caso de que el sistema no funcione correctamente.

En el caso de que el sistema funcione correctamente y el tiempo de respuesta esté dentro de un rango, se mostrará así:

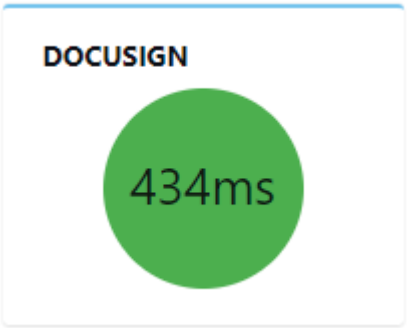

Figura 17. Ejemplo 1 monitorización Docusign.

En el caso de que el sistema funcione correctamente y el tiempo de respuesta sea superior a un rango determinado, se mostrará así:

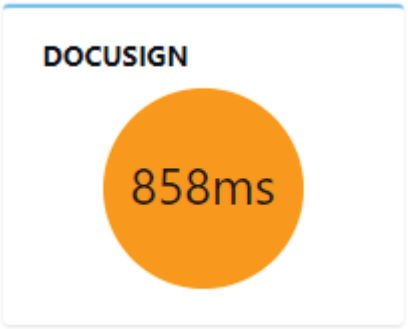

Figura 18. Ejemplo 2 monitorización Docusign.

En el caso de que el sistema no funcione, se mostrará así:

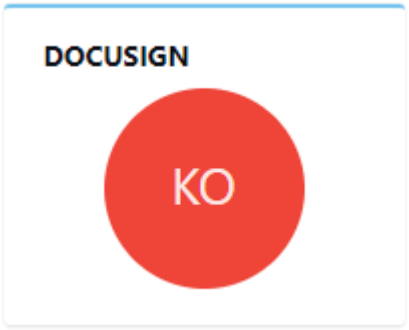

Figura 19. Ejemplo 3 monitorización Docusign.

8.4. Monitorización de Docpath.

En la parte *backend* el método principal recibe como primer parámetro la opción "*docpathStatus*", en este caso no pasamos el segundo parámetro. El método principal llamada al método correspondiente para obtener la información del estado de Docpath.

La función que obtiene el estado de Docpath realiza el siguiente proceso:

1. Se crea un servicio de Docpath, el cual nos proporciona un método *isAlive*, este método devuelve un parámetro de tipo *boolean* que nos indica si el sistema está funcionando correctamente o no.
2. Antes de llamar al servicio se obtiene el tiempo actual.
3. Se realiza la llamada al método isAlive, a continuación, se obtiene el tiempo actual. Con el tiempo obtenido antes de realizar la llamada al servicio isAlive, obtenemos el tiempo de respuesta.

4. Una vez que el proceso ha terminado, comprobamos el resultado que nos ha devuelto, en el caso de que haya funcionado correctamente, asignaremos el valor true a la variable *alive*. A continuación, asignaremos los valores del tiempo de respuesta y la variable *alive* a una variable JSON, esta variable nos servirá en la parte *frontend* para obtener los datos.
5. Por último, se convierten todos estos datos en un *String* el cual tiene JsonProperty que son los valores de la variable *alive* y el tiempo de respuesta.

En la parte *frontend* esta información se encuentra en un componente de interfaz de usuario de tipo card, la cual contiene un semáforo con el tiempo de respuesta en su interior. Este semáforo tendrá un color determinado dependiendo de si el sistema está funcionando correctamente y del tiempo de respuesta.

El semáforo será de color verde o naranja si el sistema funciona correctamente, el color verde o naranja varía dependiendo del tiempo de respuesta del sistema. El semáforo será rojo en el caso de que el sistema no funcione correctamente.

En el caso de que el sistema funcione correctamente y el tiempo de respuesta esté dentro de un rango, se mostrará así:

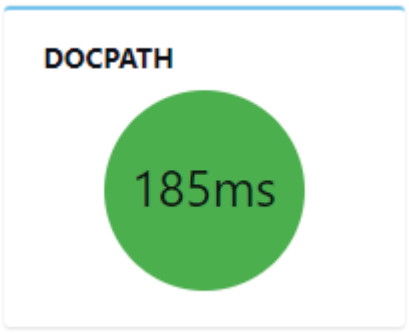

Figura 20. Ejemplo 1 monitorización Docpath.

En el caso de que el sistema funcione correctamente y el tiempo de respuesta sea superior a un rango determinado, se mostrará así:

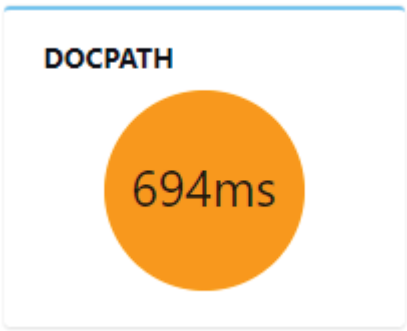

Figura 21. Ejemplo 2 monitorización Docpath.

En el caso de que el sistema no funcione, se mostrará así:

Figura 22. Ejemplo 3 monitorización Docpath.

El conjunto de todos los sistemas monitorizados se mostrará así:

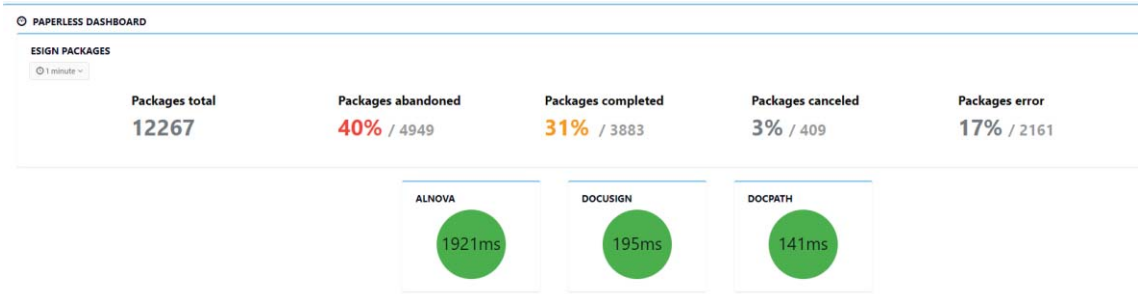

Figura 23. Resultado monitorización de todos los sistemas.

9. CONCLUSIONES

El resultado final de este proyecto ha sido el esperado. La creación de un cuadro de mando para la monitorización de una aplicación bancaria en la vida real.

El requisito principal de este proyecto se ha cumplido, este requisito era la monitorización de todos los sistemas que se encuentran dentro de la aplicación.

La primera dificultad encontrada ha sido el desarrollo *frontend* del cuadro de mando, debido a que anteriormente no había utilizado las tecnologías *frontend*. Se han dado algunas dificultades a la hora de monitorizar algunos de los sistemas de la aplicación, al tener cada sistema una arquitectura distinta, era necesario primero entender su arquitectura para así poder monitorizarlo.

Respecto a los puntos fuertes de este proyecto, ha sido la adquisición de nuevos conocimientos en distintas tecnologías, y el reto de desarrollar este proyecto y la satisfacción por el resultado final.

10. FUTURAS MEJORAS

Una vez finalizado este proyecto, se proponen las siguientes mejoras a realizar:

- Respecto a la monitorización de paquetes, en los paquetes en estado error, implementar un botón que realice una llamada a la base de datos y te muestre en detalle la información de cada paquete que ha quedado en error.
- En el caso de que se produzca un fallo en la aplicación y los números de los paquetes no se encuentren dentro de los valores establecidos, se envié un correo de notificación.
- En el caso de que alguno de los sistemas externos no funcione correctamente, enviar un correo de notificación.

11. BIBLIOGRAFÍA

[1] “Java (lenguaje de programación) - Wikipedia, la enciclopedia libre” [En línea]:

https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)

[2] “HTML | MDN” [En línea]:

https://developer.mozilla.org/es/docs/Web/HTML

[3] “CSS | MDN” [En línea]:

https://developer.mozilla.org/es/docs/Web/CSS

[4] “JavaScript - Wikipedia, la enciclopedia libre” [En línea]:

https://es.wikipedia.org/wiki/JavaScript

[5] “Alnova, la nueva compañía de Anderser Consulting” [En línea]:

http://www.computing.es/mercado-ti/noticias/1000668046401/alnova-compania-andersen-consulting.1.html

[6] “appvizer.es – La mejor selección de software” [En línea]:

https://www.appvizer.es/colaboracion/firma-digital/docusign

[7] “Funciones | Docusign” [En línea]:

https://www.docusign.com.br/produtos/docusign/funcionalidades

[8] “Docpath: Compañía de software documental” [En línea]:

https://www.docpath.com/company-document-output-management-and-customer-communications-management/?lang=es

[9] “Ventajas de software documental de Docpath” [En línea]:

https://www.docpath.com/advantages-document-software/?lang=es

[10] “AngularJS - Wikipedia, la enciclopedia libre” [En línea]:

https://es.wikipedia.org/wiki/AngularJS

[11] “AngularJS: AngularJS 1.3.0-superluminal-nudge” [En línea]:

https://blog.angularjs.org/2014/10/angularjs-130-superluminal-nudge.html

[12] “Qué es Bootstrap y cuáles son sus ventajas” [En línea]:

https://puntoabierto.net/blog/que-es-bootstrap-y-cuales-son-sus-ventajas

[13] “Bootstrap 3.3.0 released | Bootstrap Blog” [En línea];

https://blog.getbootstrap.com/2014/10/29/bootstrap-3-3-0-released/

[14] “Node.js: ¿Qué es y para qué sirve NodeJS” [En línea]:

https://www.netconsulting.es/blog/nodejs/

[15] “Node v8.0.0 (Current) | Node.js” [En línea]:

https://nodejs.org/es/blog/release/v8.0.0/

[16] “Spring Framework” [En línea]:

https://spring.io/projects/spring-framework

[17] “Spring Data JDBC” [En línea]:

https://spring.io/projects/spring-data-jdbc

[18] “Apache HttpComponents – HttpComponents HttpCore Overview” [En línea]:

http://hc.apache.org/httpcomponents-core-ga/index.html

[19] “httpcore - PyPI” [En línea]:

https://pypi.org/project/httpcore/

[20] “SQL Developer” [En línea]:

https://www.oracle.com/database/technologies/appdev/sql-developer.html

[21] “SQL Developer New Features 18.1” [En línea]:

https://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/sqldev-newfeatures-v181-4423630.html

[22] “Oracle 12c Administración – Presentación de Oracle Database 12c” [En línea]:

https://www.ediciones-eni.com/open/mediabook.aspx?idR=527a2879ea41bdf8f374d7427e413f0b

Este documento esta firmado por

Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
Fecha/Hora	Mon Jun 03 10:07:19 CEST 2019
Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
Numero de Serie	630
Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)