Análisis, diseño y desarrollo de un sistema de información que permite el ingreso y consolidación de datos del modelo de entendimiento de pruebas MoEP 360°, para la toma de decisiones operacionales de la empresa Choucair Testing S.A.

Deibison Steven Grajales Palacio

Informe de práctica como requisito para optar al título de Ingeniero de Sistemas

Asesores:
Gabriel Darío Uribe Guerra, Matemático
Bladimir Molina Ganoa, Arquitecto de Automatización

Universidad de Antioquia
Facultad de Ingeniería
Ingeniería de Sistemas
Medellín, Colombia
2022

TABLA DE CONTENIDO

RESUMEN 8
ABSTRACT 9
I. INTRODUCCIÓN 10
II. OBJETIVOS 12
A. Objetivo general 12
B. Objetivos específicos 12
III. MARCO TEÓRICO 14
IV. METODOLOGÍA 16
A. Análisis 17
B. Diseño de solución 17
C. Capacitación 18
D. Implementación 18
E. Pruebas 19
V. RESULTADOS 20
Del lado del FrontEnd 21
Del lado del BackEnd 21
Api Gateway y Load Balancer 21
Eureka Server 22
Microservicios 22
Documentación de microservicios 23
Pruebas 24
Base de datos 24
Visualización de interfaces 25

VI. CONCLUSIONES 30

VII. RECOMENDACIONES.....32

REFERENCIAS 34

LISTA DE TABLAS

TABLA I: TECNOLOGÍAS PARA EL DESARROLLO DEL SISTEMA 15
TABLA II: DESCRIPCIÓN DE TIPOS DE PRUEBA EN LOS MICROSERVICIOS 19
TABLA III: DETALLE DE LOS MICROSERVICIOS IMPLEMENTADOS 23

LISTA DE FIGURAS

Fig. 1. Fases iterativas del desarrollo 17
Fig. 2. Pirámide de testing para microservicios sugerida por Martin Fowler. 19
Fig. 3. Arquitectura MoEP 360° basada en microservicios. 21
Fig. 4. Ejemplo de documentación en Swagger referente al CRUD de clientes 24
Fig. 5. Lista de clientes registrados en el MoEP360°. 25
Fig. 6. Lista de herramientas aplicadas a un cliente seleccionado 26
Fig. 7. Avance general de ejecución de la herramienta seleccionada y muestra de porcentajes del progreso obtenido para cada pilar. 26
Fig. 8. Lista de componentes asociados a un módulo seleccionado. 27
Fig. 9. Registro de análisis asociado a un componente 28
Fig. 10. Visualización de diferentes análisis asociados a un componente 29

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

BCT	Business Centric Testing
I+D+I Finance.	Investigación, Desarrollo, Innovación Finance
MoEP360°	Modelo de entendimiento de pruebas 360°
UEN Finance.	Unidad Estratégica de Negocio Finance
MVP	Minimum Viable Product
API	Interfaz de programación de aplicaciones
CRM	Customer Relationship Management
ERP	Enterprise Resource Planning
TI	Tecnología de información
CRUD	Create, Read, Update, Delete
MVC	Model, View, Controller
UI	Interfaz de usuario
E2E	End to End

RESUMEN

El modelo de entendimiento de pruebas 360° de la empresa CHOUCAIR TESTING S.A, emergió como una herramienta habilitadora de valor y de apoyo a los procesos de la compañía con un enfoque particular en sus clientes. Sin embargo, es manejada y gestionada de forma poco práctica y convencional en Excel, razón por la cual han aparecido dificultades en la gestión y toma decisiones estratégicas de la compañía. Es por este motivo que el presente trabajo busca dar solución con la sistematización del MoEP360°, que permita la administración de usuarios, ingreso y visualización de la información y facilidad en la toma de decisiones a los gerentes de la empresa. La sistematización abarca y soporta cuatro aspectos fundamentales como flexibilidad, reutilización, escalamiento e integración con otras tecnologías o herramientas. En base a estos requisitos, se construyó un sistema basado en una arquitectura de microservicios, en la cual, por medio de desarrollos iterativos e incrementales se generaron entregas funcionales de mínimos productos. Cómo resultado, se desarrolló un sistema web integrado de información, documentado y capaz de soportar los procesos que maneja la herramienta MoEP360°.

Palabras clave — **Arquitectura de microservicios, Aplicación web, API Rest, Swagger, SQL Server, Lombok, Spring Boot, React, Java.**

ABSTRACT

The 360° testing understanding model of the company CHOUCAIR TESTING S.A, emerged as a value enabling tool to support the company's processes with a particular focus on its customers. However, it is handled and managed in an impractical and conventional way in Excel, reason for which difficulties have appeared in the management and strategic decision making of the company. It is for this motive that the present work seeks to give solution with the systematization of MoEP360°, that allows the administration of users, entry and visualization of the information and facility in the decision making to the managers of the company. The systematization encompasses and supports four fundamental aspects such as flexibility, reusability, scalability and integration with other technologies or tools. Based on these requirements, a system was built based on a microservices architecture, in which, through iterative and incremental developments, functional deliveries of minimum viable products were generated. As a result, an integrated web-based information system was developed, documented and capable of supporting the processes managed by the MoEP360° tool.

Keywords — **Microservices architecture, Web Application, API Rest, Swagger, SQL Server, Lombok, Spring Boot, React, Java.**

I. INTRODUCCIÓN

A diario muchas organizaciones buscan ser generadores de valor apoyados en sus modelos de negocio digitales, es por eso por lo que en el creciente auge de la transformación digital, Choucair Testing S.A, una empresa de pruebas de software centrada en el negocio (BCT) [1], enfrenta el desafío de que sus clientes buscan ser relevantes en el mercado, lo que provoca que sea necesario tomar decisiones en base a estrategias que funcionen en pro de materializar la promesa de valor a sus clientes [2]. Así pues, el área I+D+I Finance la cual nace desde la operación TI de los clientes Choucair, con miras de entender las necesidades de sus clientes y cómo están articulados desde una perspectiva de valor operacional, construyó el MoEP 360°, como una de las herramientas habilitadoras de soluciones de valor para tomar decisiones y plantear estrategias que permitan encaminar la operación tanto de la UEN Finance como de sus clientes [3].

El modelo de entendimiento de pruebas (MoEP 360°), si bien permite la toma decisiones y se ha convertido en una herramienta de valor para la compañía, utiliza información que está en constante crecimiento y cambio, todo el proceso de diligenciamiento de datos, búsqueda de información, creación de tablas, gráficos e informes que se manejan en la herramienta, se realizan de forma poco práctica y convencional en la herramienta Excel utilizada a su vez como base de datos. La forma de operar y trabajar la herramienta, ha provocado dificultades en la toma de decisiones para los gerentes de la compañía, ya que el ingreso y gestión de la información son tareas complejas que requieren bastante tiempo y esfuerzo, propensas a errores y difíciles de mantener con el tiempo. De igual manera, el área I+D+I Finance maneja otras herramientas las cuales a pesar de estar relacionadas con el MoEP 360°, son difíciles integrarlas y escalarlas.

Con el fin de dar solución a los problemas anteriores, este proyecto analiza, diseña, implementa y ejecuta una solución basada en software que permite la digitalización y gestión de datos concernientes a la herramienta MoEP 360° del área I+D+I Finance y brinda la posibilidad de integrar otras herramientas relacionadas del área. La sistematización está basada en una arquitectura de microservicios y utiliza tecnologías como React Js, Spring Boot, Java, Spring Cloud Gateway, API Rest, SQL Server, entre otros. El trabajo se realizó en base a entendimientos del negocio y las necesidades, investigaciones y con desarrollos iterativos e incrementales, en las

cuales se realizaban entregas de informes parciales, actividades semanales y reuniones constantes de alineación y retroalimentación con el cliente, los cuales tenían como objetivos validar y verificar lo construido y que cumplieran con los requisitos y necesidades del negocio.

II. OBJETIVOS

A. Objetivo general

Analizar, diseñar e implementar el Frontend y el Backend de un sistema de información, que permita a los gerentes de servicio, gerentes de la Unidad Estratégica de Negocios y los gerentes de operaciones, ingresar, visualizar y consolidar la información que soporta el MoEP 360°.

B. Objetivos específicos

- Analizar, diseñar y documentar la arquitectura basada en microservicios para el Backend del sistema, en donde se especifiquen los componentes principales, la comunicación y los servicios.

- Definir, diseñar y documentar el Frontend del sistema, que permita escalamiento, flexibilidad y adaptación con otras tecnologías futuras.

- Especificar y documentar URLs o Endpoints de las operaciones que serán expuestas por el API RestFul de los diversos microservicios, con el fin de obtener un entendimiento mayor de lo necesario para el consumo del lado del cliente.

- Establecer, diseñar e implementar las entidades básicas del sistema de información del MoEP360° para cada uno de los microservicios.

- Implementar las APIs RestFul haciendo uso de la arquitectura de microservicios, las cuales permitan realizar operaciones de almacenamiento, edición, visualización y eliminación de la información del MoEP360° para cada uno de los microservicios.

- Crear interfaces básicas del lado del FrontEnd con React para el consumo de los distintos servicios expuestos por los microservicios y permitan el ingreso y visualización de la información del MoEP360°.

- Interconectar las interfaces creadas del Frontend con el Backend.

- Realizar procesos de verificación y validación de forma iterativa de lo construido tanto a nivel Backend como Frontend con el cliente.

III. MARCO TEÓRICO

Las aplicaciones con una arquitectura monolítica han resultado en varios casos de uso en el mundo, tales como: software de equipos médicos, programas de escritorio, procesadores de texto o incluso sistemas más completos como los clásicos CRM o ERP. Sin embargo, con el paso del tiempo se han generado problemas al escalar, testear y realizar una depuración de errores en este tipo de aplicaciones, ya que los cambios constantes de los modelos de negocio originan que se tenga que modificar toda la aplicación, haciendo que sea más compleja de mantener [4]. Además, si la aplicación con una arquitectura monolítica falla en un pequeño punto, todo el sistema deja de funcionar, quedando totalmente inoperable. También, existe una fuerte dependencia al Stack tecnológico, ya que como todo el software es una sola pieza, implica tener que utilizar el mismo Stack tecnológico para absolutamente todo y como consecuencia limita la posibilidad de aprovechar diferentes lenguajes de programación, usar distintas tecnologías de almacenamiento y explotar las capacidades y destrezas del equipo de desarrollo que tengan en diferentes tecnologías.

Debido a las limitaciones y dificultades de las arquitecturas monolíticas, surgieron nuevos modelos de negocio mucho más estables ante cambios y eficientes, como las arquitecturas basadas en microservicios, que plantean la construcción de conjuntos pequeños de servicios independientes, los cuales se comunican entre sí a través de peticiones HTTP y se ejecutan de forma autónoma, siendo escalables y fáciles de mantener [5]. Además, pueden ser implementados en diferentes tecnologías de forma independiente, según sea el caso que mejor se ajuste a los requerimientos y alinea al equipo de desarrollo en pequeños grupos que trabajan enfocados en diferentes partes del código, logrando mayor claridad en el desarrollo y mejor productividad [6]. Pero no solo lo anterior, los microservicios permiten realizar despliegues más rápidos a producción sin afectar el resto de componentes, ante una falla el sistema puede seguir operando (resilientes), son fáciles de probar y la médula espinal de la arquitectura de microservicios, tienen la ventaja de ser reusables, ya que al tener pequeños componentes que realizan una única tarea, es decir; componentes altamente cohesivos, se facilita la reutilización por otras aplicaciones o microservicios.

En la **TABLA I** se detallan a nivel general las tecnologías necesarias y su definiciones para la sistematización del MoEP360°.

TABLA I:
TECNOLOGÍAS PARA EL DESARROLLO DEL SISTEMA

Tecnología/Lenguaje	Definición
React	Es una biblioteca de JavaScript para construir interfaces de usuario [7].
Javascript	JavaScript (abreviado comúnmente JS) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript, utilizado principalmente para crear páginas web dinámicas. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente del lado del cliente, implementado como parte de un navegador web y permite mejoras en la interfaz de usuario y páginas web dinámicas y JavaScript del lado del servidor (Server-side JavaScript o SSJS) [8].
Spring Boot	Spring Boot facilita la creación y el grado de producción de aplicaciones independientes basadas en Spring [9].
Hibernate	Hibernate ORM permite a los desarrolladores escribir más fácilmente aplicaciones cuyos datos sobreviven al proceso de la aplicación. Como marco de mapeo de objetos/relaciones (ORM), Hibernate se preocupa por la persistencia de los datos tal como se aplica a las bases de datos relacionales (a través de JDBC) [10].
Swagger	Swagger es un framework de código abierto para diseñar y describir API, que permite el desarrollo en todo el ciclo de vida de la API, desde el diseño y la documentación hasta la prueba y la implementación [11].
SQL Server	SQL Server es el sistema de base de datos profesional de Microsoft. Contiene una variedad de características y herramientas que se pueden utilizar para desarrollar y administrar bases de datos y soluciones de todo tipo basadas en ellas [12].

Nota: Definición general de las tecnologías para la construcción del sistema

IV. METODOLOGÍA

Con el fin de lograr los objetivos planteados, inicialmente fue necesario conocer a detalle la herramienta MoEP360° desarrollada en Excel, para esto se realizaron diferentes reuniones de explicación y entendimiento del negocio por parte del asesor externo de los diferentes flujos, estructura, elementos, procesos, reglas de negocio y restricciones de la herramienta. Una vez se comprendió el funcionamiento básico y las necesidades que se buscaban solventar en la herramienta, se hicieron reuniones de contextualización y alineación referente a las tecnologías, frameworks, infraestructura y bases de datos utilizados en la compañía, específicamente con el área de TI de Choucair Testing S.A, esto con la intención de evitar en el futuro barreras referentes a políticas empresariales que no permitieran la integración de la sistematización del MoEP360° a lo manejado en la organización.

Teniendo en cuenta lo anterior, se pasó por una fase de análisis y diseño, en donde se propuso la implementación de una arquitectura basada en microservicios con Spring Boot del lado del Backend y del lado del FrontEnd React, que soportara y permitiera el almacenamiento y gestión de los datos concernientes al MoEP360°. Con la aprobación de la propuesta y un panorama claro de lo que se deseaba construir, se iniciaron fases iterativas de análisis, diseño, capacitación, implementación y pruebas (**Fig. 1**), en la cual cada semana se realizaban entregas de informes respecto a las actividades desarrolladas para la sistematización y cada 15 días pequeños fragmentos funcionales, incrementales y de valor del sistema, los cuales permitieron realizar cambios de acuerdo a retroalimentaciones hechas por el cliente en cada iteración y así evitar la construcción de funcionalidades no deseadas, además de reuniones constantes con el equipo, que permitieron identificar avances, mejoras y dificultades en el desarrollo.

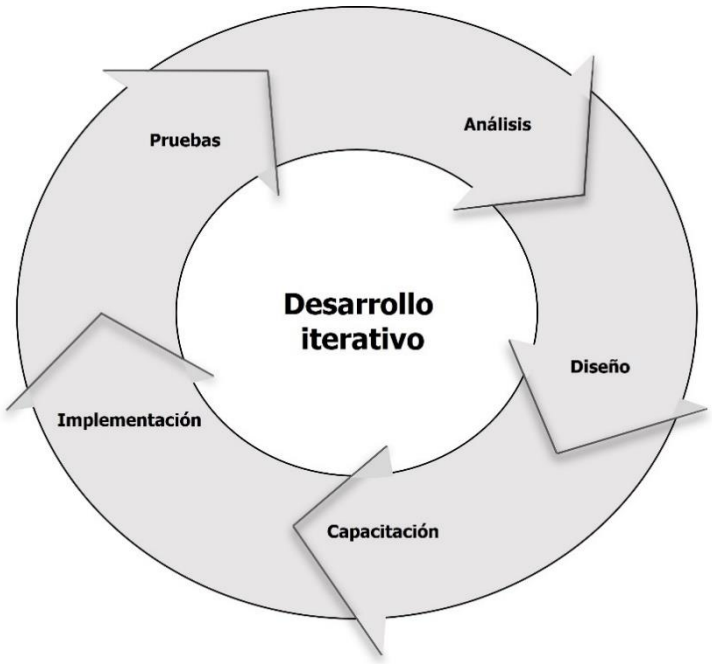

Fig. 1. Fases iterativas de desarrollo

A. Análisis

- Se realizó un entendimiento del modelo de negocio, el flujo y el marco de trabajo de la empresa, para identificar las necesidades fundamentales y lograr una correcta estructuración de los requisitos y necesidades.

- Análisis y comprensión de cada uno de los elementos que forman parte del Moep360°.

- Realización y presentación de Mockups en etapas tempranas y avanzadas del desarrollo con el fin de validar y verificar lo que se construiría y así obtener un mejor acercamiento de lo deseado por el cliente.

- Revisión y desglose de tareas y actividades necesarias para la sistematización del MoEP360°.

B. Diseño de solución

- Investigación y estudio de las tecnologías que debían ser empleadas para la implementación de acuerdo con los requerimientos del sistema identificados.

- Investigación, estudio de tendencias y arquitecturas más utilizadas en entornos empresariales en el desarrollo de aplicaciones para la sistematización del MoEP360°.

- Propuesta de arquitectura más acorde a las necesidades del área y la compañía.

- Diseño de arquitectura Backend para cada uno de los microservicios y tecnologías.

- Diseño de arquitectura y modularización del FrontEnd

C. Capacitación

- Fundamentación, investigación y aumento de destrezas necesarias para la construcción del FrontEnd (React Js, redux, router, Bootstrap y demás librerías necesarias para la creación de interfaces y el consumo de los distintos servicios del API Rest).

- Capacitación y documentación en arquitectura de microservicios y todo lo relacionado al Backend (Spring Boot, Spring Data JPA - Hibernate, Spring IoC, Java, API RestFul, Swagger, SQL Server, Spring Cloud Gateway, Eureka Server, Eureka Discovery, Eureka Client, Feign, entre otros frameworks y tecnologías).

D. Implementación

- Construcción del FrontEnd.

- Construcción del BackEnd, el objetivo de esta fase fue la implementación, configuración y comunicación de cada uno de los microservicios, conforme se adaptaba y se escalaba la arquitectura en cada iteración.

E. Pruebas

Validación y verificación de la implementación realizada, adaptando algunas estrategias de pruebas que se manejan en aplicaciones monolíticas y considerando otras como pruebas de componentes (**Fig. 2**) que se aplican en los microservicios [13].

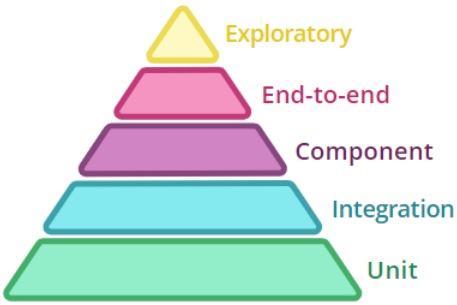

Fig. 2. Pirámide de testing para microservicios sugerida por Martin Fowler.

Nota: Fuente: https://martinfowler.com/articles/microservice-testing/#conclusion-test-pyramid

En la **TABLA II** se describen los diferentes tipos de pruebas de acuerdo a la pirámide de testing en base a una arquitectura de microservicios.

TABLA II:
DESCRIPCIÓN DE TIPOS DE PRUEBA EN LOS MICROSERVICIOS

Tipos de prueba	Descripción
Pruebas Unitarias	El objetivo de esta fase es testear cada componente o unidad de software individual e independiente sin considerar otras partes de la aplicación [14].
Pruebas de integración	Se enfoca en probar si los componentes funciona bien después de integrarlos [14] tales como pruebas de API y bases de datos.
Pruebas de componentes	Encargado de realizar pruebas sobre un componente de forma aislada en la cual se simulan eventos y se intercambian mensajes entre componentes.
Pruebas End to End	Las pruebas E2E consiste en probar todo el sistema, de inicio a fin, verificando que cumpla con los requerimientos identificados en etapas previas.
Pruebas Exploratoritas	Las pruebas exploratorias permiten al equipo aprender sobre el sistema y educar y mejorar sus pruebas automatizadas. Este tipo de pruebas exploran manualmente el sistema de maneras que no se han considerado como parte de las pruebas programadas [13].

Nota: Definición para cada tipo de prueba según la pirámide definida por Martin Fowler

V. RESULTADOS

En esta sección se detallan los elementos, las actividades que se realizaron durante el proceso de desarrollo y los resultados que se obtuvieron de las fases mencionadas anteriormente.

Se consultó información de múltiples fuentes bibliográficas referente a la implementación de arquitecturas basadas en microservicios, con el fin de establecer y obtener una contextualización de los elementos fundamentales que debían ser considerados para el sistema a desarrollar y que cumpliera con los objetivos planteados, estas fuentes de bibliográficas fueron: videos, libros, páginas web, artículos, trabajos de grado, documentos de investigación, entre otros. Cómo resultado se obtuvieron documentos de investigación y documentación del sistema que permitirán disminuir la curva de aprendizaje y el tiempo al momento de tratar de incorporar nuevos integrantes en el equipo y servirá como guía para la realización de cambios en el sistema.

Conforme se pasó por las diferentes fases de desarrollo, se propusieron cambios a la base de datos del MoEP360°, la cual fue diseñada y construida en SQL Server mucho antes de que se comenzara con el desarrollo del sistema, primordialmente se buscó tener una base de datos más optimizada y eficiente, disminuir su complejidad, vulnerabilidad y eliminar campos innecesarios y redundantes.

La arquitectura que mejor se adaptó a las necesidades del sistema, se muestra en la **Fig. 3.** Aquí se representa a nivel general el alcance que se tuvo en el desarrollo de la práctica, en el cual se logró cumplir con los objetivos planteados, construir un sistema integrado que maneja y permite la consolidación de los datos concernientes a la herramienta MoEP360°. Sin embargo, es importante aclarar que ciertos elementos que se encuentran en la arquitectura por motivos de prioridades, tiempo y alcance del proyecto no se desarrollaron, pero se dejan expresados para futuros desarrollos. Los resultados obtenidos más concretamente se mencionan a continuación:

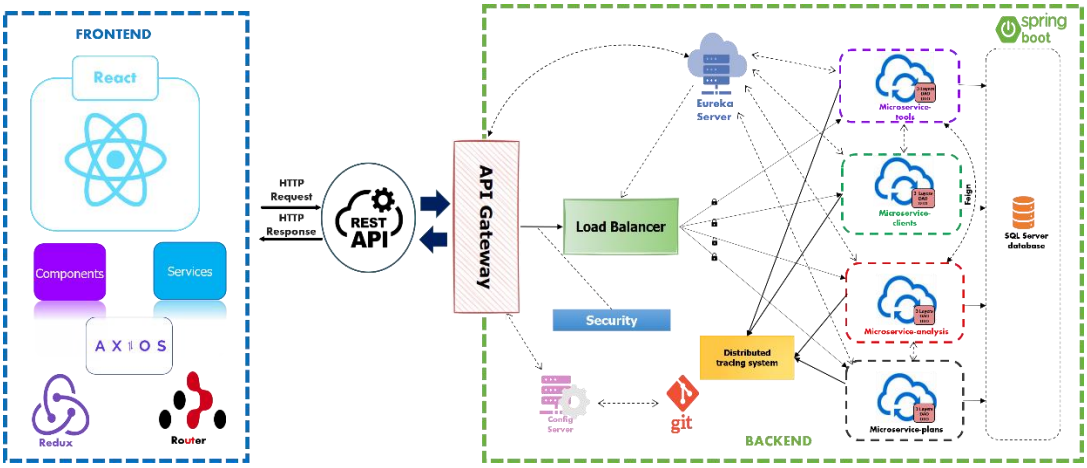

Fig. 3. Arquitectura MoEP 360° basada en microservicios.

Del lado del FrontEnd

El desarrollo fue en ReactJS para la creación de interfaces de usuario y el consumo de datos a través de los diferentes endpoints del API Rest, compuesto por cada uno de los microservicios. Se utilizaron librerías como Bootstrap, Bulma, Font-Awesome para un diseño intuitivo y responsivo, librerías de React para la creación de barras de progreso y de Steps que debía contener el sistema. De igual manera, fue necesario el uso de librerías para la navegación entre los diferentes componentes y páginas del sistema. Luego, fue imprescindible Redux para la administración del estado de la interfaz de usuario y para los llamados al API Rest que se implementaron mediante Axios, se aplicaron librerías de Redux Middleware, en este caso se usó Redux Saga, lo que facilitó la gestión y el manejo de los Side Effects de la aplicación.

Del lado del BackEnd

Api Gateway y Load Balancer

El microservicio Spring Cloud Gateway se desarrolló para la centralización de las llamadas a los demás microservicios, como compuerta permitió controlar que recursos o servicios de la infraestructura estaban disponibles. Así, solo se exponían los recursos necesarios y se ocultaban los que no. También, facilitó el enrutamiento de forma dinámica para el acceso a cada microservicio a través de filtros y se redirigieron las llamadas a los microservicios configurados

(*microservicio-análisis*, *microservicio-herramientas*, *microservicio-clientes*, *microservicio-planes*). Además, el microservicio Spring Cloud Gateway internamente permitía un balanceo de cargas en el cliente de forma automática, el cual atiende una demanda creciente de peticiones en casos donde la aplicación requiera un escalamiento y se agreguen múltiples instancias o en casos de que falle una instancia y sea necesario el uso de otras instancias para atender correctamente las peticiones, sin necesidad de reiniciar los microservicios o cambiar la configuración para agregar un nuevo servidor al balanceador. Por defecto utiliza Ribbon para la selección de la mejor instancia disponible y así evitar usar instancias con fallos.

Eureka Server

Servidor desarrollado para el registro de las direcciones a los microservicios que componen la aplicación y redireccionar las peticiones hacia estos a través de una URL genérica a la dirección del servidor en el que se encuentra dicho microservicio. Este contenedor de microservicios registra el nombre, la ubicación física, IP y el dominio o puerto de cada microservicio, lo que permitió que los microservicios se pudieran comunicar mediante Feign simplemente haciendo referencia al identificador (el nombre del microservicio) para obtención de datos y desacoplar el código de la aplicación lo máximo posible.

Microservicios

En la siguiente tabla (**TABLA III**) se describen cada uno de los microservicios desarrollados para el funcionamiento de la plataforma, los cuales se realizaron en el lenguaje de programación Java con el framework Spring Boot y son distribuidos en una base de datos compartida.

TABLA III:
DETALLE DE LOS MICROSERVICIOS IMPLEMENTADOS

Microservicio	Tecnología y Composición	Descripción
Análisis	Api Rest, Java 1.8, Spring Boot, Spring Mvc, Lombok, MapStruct, Eureka Client, SQL Server, Swagger.	Este microservicio soporta las peticiones referentes a los análisis, ítems y rúbricas de la aplicación, maneja el CRUD (Create, Read, Update, Delete) para análisis, ítems y rúbricas
Herramientas	Api Rest, Java 1.8, Spring Boot, Spring Mvc, Lombok, Eureka Client, SQL Server, Swagger	Este servicio soporta las peticiones referentes a las herramientas, pilares, módulos y componentes de la aplicación, maneja el CRUD (Create, Read, Update, Delete) para herramientas, pilares, módulos y componentes
Clientes	Api Rest, Java 1.8, Spring Boot, Spring Mvc, Lombok, MapStruct, Eureka Client, SQL Server, Swagger	Este microservicio soporta las peticiones referentes a los clientes, sedes y UNegocio de la aplicación, maneja el CRUD (Create, Read, Update, Delete) para clientes, sedes y UNegocio.
Planes de acción	Api Rest, Java 1.8, Spring Boot, Spring Mvc, Lombok, MapStruct, Eureka Client, SQL Server, Swagger	Este microservicio soporta las peticiones referentes a los planes de acción, riesgos, indicadores etapas, actividades y responsables, maneja las operaciones CRUD para la gestión y consolidación de los datos.

Nota: Descripción de tecnologías y estructura de cada uno de los microservicios que engloban la lógica de la aplicación.

Documentación

Cada microservicio tiene diversas API encargadas de realizar diferentes operaciones (CRUD). Es por esta razón que se incorporó a cada microservicio la herramienta Swagger para la documentación de los diferentes servicios expuestos por el API, ordenar cada uno de los métodos (GET, PUT, POST, DELETE), categorizar las operaciones y expandir cada uno de los métodos con el objetivo de interactuar con el API Rest mediante la UI de Swagger y obtener una visión clara de cómo responde la API a los parámetros. Por ejemplo, en la siguiente imagen (**Fig. 4**) se visualiza la documentación de las operaciones básicas tales como: consulta, creación edición y eliminación de clientes.

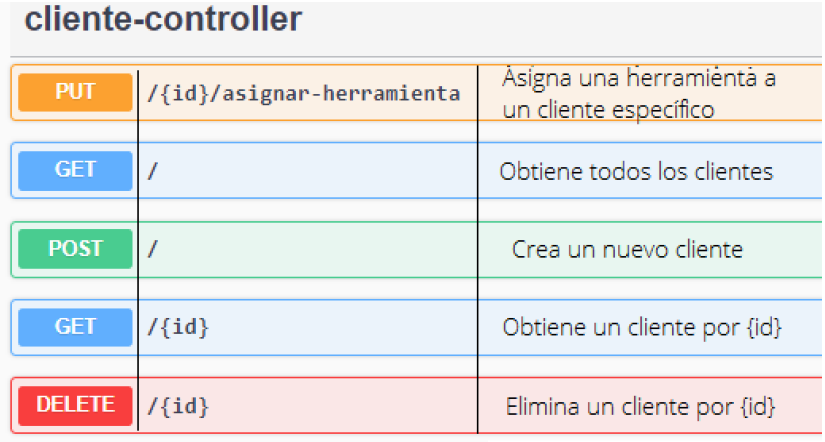

Fig. 4. Ejemplo de documentación en Swagger referente al CRUD de clientes

Pruebas

Durante el desarrollo de las funcionalidades para cada microservicio se aplicaron pruebas unitarias, esto con el fin de que pequeñas unidades del sistema no terminen afectando comportamientos futuros al escalar el aplicativo y mitigar posibles daños al realizar integraciones o cambios. La herramienta utilizada fue JUnit en Java, la cual permitió validar y verificar el comportamiento de los métodos de forma aislada y cumplir con lo requerido.

Para las pruebas de integración se utilizó Swagger, en la cual se validó para cada API desarrollada de los distintos microservicios las funciones de agregar, listar, actualizar, eliminar, buscar y se verificaron algunas otras como la asignación de herramientas a clientes, módulos a pilares, cálculos de avances, entre otros.

En las pruebas de componentes se simularon eventos e intercambios de mensajes entre los diferentes componentes de forma asíncrona, mediante RabbitMQ, con motivo de comprobar la correcta comunicación entre los microservicios.

Base de datos

Capa desarrollada a través del ORM con el framework Hibernate, que permite la persistencia de los datos en la base de datos relacional SQL Server. Así, todos los microservicios

acceden a la misma base de datos de forma compartida, en donde se separan los datos en diferentes esquemas para cada uno de los microservicios y las relaciones son manejadas de forma distribuida mediante comunicación entre microservicios independientes (Feign). Sin embargo, existe la posibilidad de realizar una migración en el futuro a otros tipos de gestores de base de datos para cada microservicio.

Visualización de interfaces

En las posteriores secciones se muestran algunas imágenes de ejecución del sistema, las cuales por políticas empresariales de la empresa solo se muestra de forma general la navegabilidad entre componentes, ingreso y visualización de la información y solo algunos procesos del aplicativo.

En la siguiente imagen (Fig. 5. Lista de clientes registrados en el MoEP360°.**Fig. 5**) se lista todos los clientes registrados en la herramienta MoEP360°, aquí se visualiza la información general para cada uno de los clientes, como dirección y UEN a la que pertenece. Esta interfaz es el home de la aplicación, en la cual, para poder visualizar toda la información como los pilares, avances, módulos, componentes, entre otros elementos de la aplicación, debe seleccionarse previamente uno de los clientes.

Fig. 5. Lista de clientes registrados en el MoEP360°.

Una vez es seleccionado uno de los clientes, el sistema cargará y listará todas las herramientas las cuales están siendo aplicadas a ese cliente en particular (**Fig. 6).** Además, se muestra en la parte izquierda superior al lado del logo, el cliente que fue seleccionado.

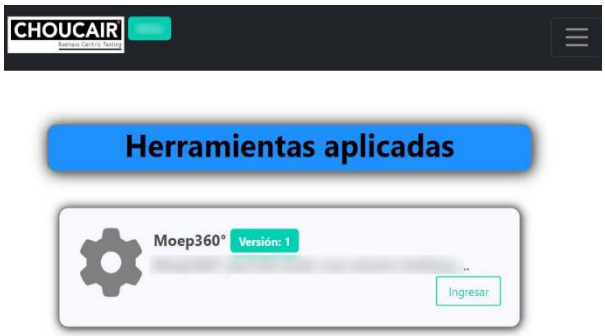

Fig. 6. Lista de herramientas aplicadas a un cliente seleccionado

Al ingresar a una de las herramientas aplicadas, el sistema cargará toda la información referente a dicha herramienta (**Fig. 7**), información clave que permite a los gerentes tomar decisiones en base a porcentajes y avances que se han tenido en el sistema.

Fig. 7. Avance general de ejecución de la herramienta seleccionada y muestra de porcentajes del progreso obtenido para cada pilar.

Para cada uno de los pilares listados anteriormente (**Fig. 7**), es posible seleccionar uno de los pilares y mostrar información más detallada acerca de los módulos asociados a ese pilar. En la

siguiente imagen (**Fig. 8**), se listan todos los componentes asociados a un módulo en particular, con información respecto a los avances y porcentajes del trabajo realizado para cada componente.

Fig. 8. Lista de componentes asociados a un módulo seleccionado.

El formulario de registro de un análisis se muestra en la siguiente imagen (**Fig. 9**), de esta manera es posible ingresar información de un análisis asociándolo de acuerdo a un componente seleccionado.

Fig. 9. Registro de análisis asociado a un componente

En la siguiente imagen (**Fig. 10**) se visualiza la información ingresada de los correspondientes análisis asociados a un componente, información que es de vital importancia para una correcta alineación en la toma de decisiones por parte de los gerentes.

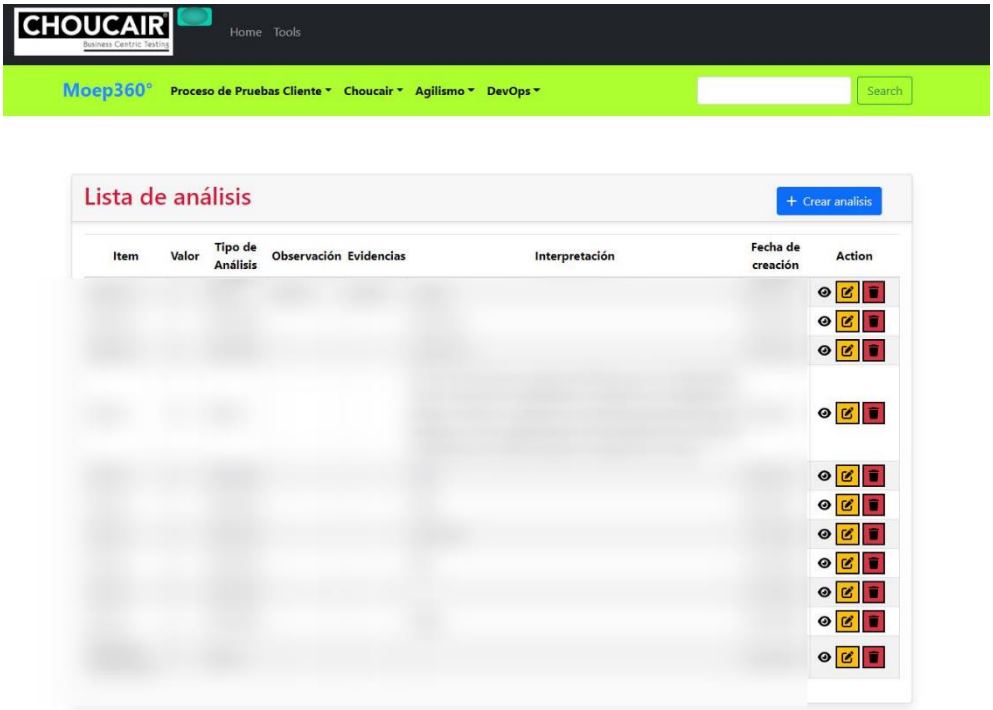

Fig. 10. Visualización de diferentes análisis asociados a un componente

VI. CONCLUSIONES

En el proceso de consulta e investigación se llegó a la conclusión de que el principal objetivo de las arquitecturas basadas en microservicios es el desacoplar lo máximo posible todos los componentes, mediante la división de problemas complejos en bloques funcionales e independientes entre sí, fomentando la reutilización y asignación de responsabilidades únicas, aspectos claves que facilitaron el desarrollo, ya que a medida que se escalaba o se agregaban nuevas funcionalidades en etapas avanzadas del desarrollo conforme lo requería el cliente, estas se podían incorporar y adaptar fácilmente a lo ya construido.

La elección de una arquitectura basada en microservicios fue clave para satisfacer los requerimientos y necesidades del cliente. Esta arquitectura fue pensada de manera que, si en un futuro hay escenarios donde se requiere flexibilidad, mayor capacidad para soportar gran cantidad de usuarios, escalamiento de la aplicación e integración con otras tecnologías o herramientas, esto pueda ser más sencillo y no requiera de tanto esfuerzo por el equipo de desarrollo. Por lo anterior, se destaca la importancia de las reuniones constantes con los interesados e involucrados en cada una de las fases, las cuales brindaron un buen entendimiento del negocio y una correcta alineación a lo que se quería.

El análisis, diseño e implementación de la herramienta MoEP360° en un sistema web integral de información, se pudo llevar a cabo de forma satisfactoria, el cual permite el ingreso, visualización y consolidado de datos, además de una mejora en la toma decisiones para los gerentes de servicio, operación y UEN de la compañía.

Al pasar por una fase de capacitación previo a la fase de implementación permitió tener un panorama más claro de los microservicios y tecnologías de la arquitectura que debían ser desarrollados, esto fomentó un desarrollo más fluido y sin tantas dificultades en la implementación. Sin embargo, existieron algunos desafíos al tratar de implementar diversas tecnologías, principalmente por la curva de aprendizaje y más aún cuando muchas de ellas eran nuevas para el equipo, todo esto se tradujo en un gran tiempo invertido adquiriendo conocimientos y destrezas para la implementación.

La generación de documentación necesaria a medida que se desarrollaban y se creaban los diferentes microservicios, se facilitó enormemente con la herramienta “SWAGGER”, gracias a esta herramienta era poco el tiempo dedicado en documentación y mucho mayor el tiempo en desarrollo.

Las pruebas son una parte fundamental en el desarrollo de cualquier sistema, como principales ventajas obtenidas, se resalta menos costos al momento de agregar nuevas funcionalidades, detección de errores no considerados en fases previas, mejora en la calidad del código y una menor deuda técnica en correcciones de bugs y errores

VII. RECOMENDACIONES

Aunque los objetivos del proyecto se alcanzaron, se detectan importantes actividades futuras con los que deberá contar el sistema:

- **Creación de seguridad basada en roles:** Debido al alcance del proyecto este tema no se desarrolló a profundidad ya que a petición del cliente era primordial otros requisitos. Por lo tanto, se sugiere la implementación de seguridad basada en tokens con JWT, OAuth2 y Spring Security para impedir el acceso a usuarios no autorizados
- **Creación de interfaces administrativas:** Se propone el diseño y construcción de interfaces de acuerdo con el rol del usuario.
- **Implementación de librerías como Chart:** Se sugiere la implementación de librerías que permitan mostrar la información del sistema en una interfaz tipo Dashboard, que permita visualizar mediante gráficos todo lo relacionado a la herramienta MoEP360°
- **Tolerancia a fallas con Hystrix u otras librerías:** Se sugiere implementar el patrón Circuit Breaker, que permita describir una estrategia contra fallas en cascada en los diferentes niveles de la aplicación. Así, Hystrix permitirá definir caminos alternativos en casos donde ocurra fallos en los llamados a los microservicios.
- **Validaciones en el FrontEnd y Backend:** Debido al tiempo y al tamaño en la construcción del sistema que se tuvo en el proyecto, se sugiere mejorar las validaciones en las interfaces de ingreso de información y demás funcionalidades del sistema tanto en el FrontEnd como en el Backend.
- **Traza distribuida:** Se sugiere el desarrollo de un sistema distribuido que permita unificar los logs en un solo punto y se agrupen por ejecución, ya que una sola llamada a un microservicio repercute en la llamada de otros microservicios, lo que implica la necesidad de ir sacando uno a uno el log de cada microservicio, que luego unirlos se vuelve una tarea complicada.
- **Realización de prueba de estrés:** Se sugiere la aplicación de pruebas de estrés para la aplicación y comprobar sus tiempos de respuesta, detectar mejoras y posibles optimizaciones en el rendimiento.

- **Creación de pruebas en el Frontend:** Se sugiere implementar pruebas en el frontend que por motivos de tiempo y alcance no se alcanzaron a cubrir

REFERENCIAS

[1] «Choucair Testing». https://www.choucairtesting.com/

[2] «Servicios – Choucair Testing». https://www.choucairtesting.com/servicio/

[3] «I+D+i Finance». https://choucairtesting.sharepoint.com/sites/IDiFinance

[4] D. A. Barrios Contreras, «Arquitectura de microservicios», *Tecnol Investig Acad. TIA*, vol. 6, n.º 1, pp. 36-46, mar. 2018.

[5] P. Quevedo-Avila, M. G. Zhindón-Mora, y A. S. Quevedo-Sacoto, «Arquitectura de microservicios para compras en línea: caso de uso "ala orden"», *Polo Conoc.*, vol. 5, n.º 1, pp. 151-162, nov. 2020, doi: 10.23857/pc.v5i1.1884.

[6] S. Newman, «Building Microservices», *OReilly Media Inc*, 2015.

[7] «React – Una biblioteca de JavaScript para construir interfaces de usuario». https://es.reactjs.org/ (accedido 17 de marzo de 2022).

[8] C. E. G. Ramírez, R. O. Robles, y B. A. Celis, «Desarrollo de aplicaciones web utilizando JavaScript». Mexico, 2020.

[9] «Spring Boot». https://spring.io/projects/spring-boot#overview (accedido 9 de julio de 2022).

[10] «Your relational data. Objectively. - Hibernate ORM», *Hibernate*. https://hibernate.org/orm/ (accedido 9 de julio de 2022).

[11] «Swagger 101 | SwaggerHub Documentation». https://support.smartbear.com/swaggerhub/docs/tutorials/writing-swagger-definitions.html (accedido 18 de marzo de 2022).

[12] M. Pérez, *Microsoft SQL Server 2008 R2. Motor de base de datos y administración*. RC Libros, 2011.

[13] M. Fowler, «Testing Strategies in a Microservice Architecture», *martinfowler.com*. https://martinfowler.com/articles/microservice-testing/#conclusion-test-pyramid

[14] A. de Camargo, I. Salvadori, R. dos S. Mello, y F. Siqueira, «An architecture to automate performance tests on microservices», en *Proceedings of the 18th International Conference on*

Information Integration and Web-based Applications and Services, Singapore Singapore, nov. 2016, pp. 422-429. doi: 10.1145/3011141.3011179.