UNIVERSIDAD
NACIONAL
DE COLOMBIA

Descomposición de componentes front-end de tipo web mediante estrategias de desacoplamiento en arquitecturas de microservicios

Cristian Camilo Orjuela Velandia

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, D. C., Colombia
2022

Descomposición de componentes front-end de tipo web mediante estrategias de desacoplamiento en arquitecturas de microservicios

Cristian Camilo Orjuela Velandia

Trabajo final presentado como requisito parcial para optar al título de:
Magíster en Ingeniería - Ingeniería de Sistemas y Computación

Dirigido por:
Jeisson Andrés Vergara Vargas, M.Sc.

Línea de investigación:
Arquitectura de Software
Grupo de Investigación:
ColSWE - Grupo de Investigación en Ingeniería de Software

Universidad Nacional de Colombia
Facultad de Ingeniería, Departamento de Ingeniería de Sistemas e Industrial
Bogotá, D. C., Colombia
2022

Dedicado a la vida, que me puso en el lugar donde estoy ahora y que incentivó mi interés por el mundo del software.

Agradecimientos

Las palabras pueden ser cortas cuando uno trae a colación un agradecimiento, sin embargo, quiero hacer explícito mi reconocimiento a todas las personas que hicieron posible la consecución de este trabajo final, pues dedicaron parte de su tiempo de vida a la realización de un proyecto personal que termina siendo muy representativo. En ese sentido, quiero darle las gracias a mi familia, a mis amigos, a mis profesores, a los autores bibliográficos mencionados, y a conocidos que de alguna manera u otra pusieron su grano de arena en esta aspiración científica.

Asimismo, quiero agradecer de manera particular al alma máter que me abrió las puertas para el fomento del conocimiento, ya que esto constituyó parte de la educación crítica e intelectual con la que cuento hoy en día. El hecho de haber recorrido la Universidad Nacional de Colombia no solo forjó gran parte de mi formación académica, que es fundamental para la construcción de sociedad y de país, sino que también respaldó la ejecución de este proceso investigativo.

Finalmente, quedo muy agradecido con mi profesor y director, Jeisson Andrés Vergara Vargas, por haberle dado cabida a un proyecto que nació simplemente de intereses subjetivos y que después se convirtió en una realidad como consecuencia de un esfuerzo mutuo.

Título en español

Descomposición de componentes front-end de tipo web mediante estrategias de desacoplamiento en arquitecturas de microservicios.

Title in English

Web-Type Front-End Components Decomposition using Decoupling Strategies in Microservices Architectures

Resumen

La fabricación de software es un proceso que ha tomado mucha importancia en la industria desde hace algunos años, puesto que la gran mayoría de sistemas de información han ido migrando su proceso operacional a mecanismos tecnológicos. Sin embargo, el crecimiento acelerado de estas estructuras ha provocado inconvenientes en toda su fase de desarrollo. Por este motivo, resulta fundamental encontrar estrategias de descomposición de software, particularmente en los componentes de Front-End, que ayuden a mitigar estos problemas y que mejoren las características inherentes de las aplicaciones. En consecuencia, el siguiente trabajo final tiene como objetivo aplicar una estrategia de desacoplamiento basada en el patrón arquitectónico de microservicios con el fin de realizar una evaluación de la cohesión entre sus componentes resultantes.

Para esto, se presenta un contexto general sobre la descomposición arquitectónica y los componentes de software, así como una definición de conceptos que permiten comprender la idea, luego se detallan algunos mecanismos que se han empleado para descomponer componentes de software, se presenta la implementación de referencia que se construyó basada en una de las industrias que más utiliza el software en la actualidad, como lo es el área financiera, y finalmente, se aplica el método de descomposición junto con su respectivo análisis.

Palabras Clave: Arquitectura de Software, Sistema de Software, Front-End, Descomposición Arquitectónica, Arquitectura de Microservicios, Cohesión, Microfrontend

Abstract

Software building is a process that has become a significant concern in the industry for some years since most information systems have been migrating their operational process to technological mechanisms. Nevertheless, the exponential growth of these structures has brought some issues throughout their development stage. For this reason, it is important to find software decomposition strategies, especially in Front-End components, that help us to mitigate these problems and improve the intrinsic characteristics of applications. Therefore, the following thesis aims to apply a decoupling strategy based on the microservices architectural pattern in order to perform an evaluation of the cohesion between its resulting components.

To make this possible, a general context about the architectural decomposition and software components is presented, as well as a definition of concepts that allow understanding the idea. Then, some tools that have been used to decompose software components are detailed, in addition to the process of building the reference implementation, which is based on one of the industries that most uses software, such as the financial sector. To conclude, the decomposition method is applied along with its corresponding analysis.

Keywords: Software Architecture, Software System, Front-End, Architectural Decomposition, Microservices Architecture, Cohesion, Microfrontend

Este Trabajo Final de Maestría fue calificado en Septiembre de 2022 por el siguiente evaluador:

Henry Roberto Umaña Acosta, M.Sc.
Profesor Facultad de Ingeniería
Universidad Nacional de Colombia

Contenidos

Agradecimientos **vii**

Resumen **x**

Lista de Figuras **xvi**

1 Introducción **1**
1.1 Introducción al problema 3
1.2 Objetivos 4
1.2.1 Objetivo General 4
1.2.2 Objetivos Específicos 4
1.3 Metodología 5
1.3.1 Análisis de arquitecturas de microservicios 5
1.3.2 Implementación de referencia 5
1.3.3 Estudio general de la implementación de referencia 6
1.3.4 Diseño de la estrategia de desacoplamiento 6
1.3.5 Proceso de evaluación de la cohesión 6

2 Marco Teórico **7**
2.1 Arquitectura de Software 7
2.2 Componentes de Software 8
2.2.1 Componentes de procesamiento y tratamiento de datos 11
2.2.2 Componentes de almacenamiento de información 11
2.2.3 Componentes basados en la interacción del usuario 12
2.3 Arquitectura de Microservicios 13
2.4 Descomposición Arquitectónica 16
2.5 Cohesión 17
2.6 Acoplamiento 18

3 Trabajos Relacionados **20**

4 Implementación de referencia **25**
4.1 Descripción estructural 26

4.2 Vistas Arquitectónicas 26
4.2.1 Vista de Capas 26
4.2.2 Vista de Componentes y Conectores 27
4.2.3 Vista del Modelo de Datos 28
4.2.4 Vista de Descomposición Funcional 30

5 Modelo de descomposición de arquitectura 32
5.1 Estrategia de descomposición 32
5.2 Modelo resultante 35
5.3 Integración del modelo resultante 36
5.3.1 Formulación de escenarios arquitectónicos 37
5.3.2 Definición del escenario final 40

6 Evaluación del modelo 43
6.1 Métrica implementada 43
6.2 Selección del flujo funcional de información 44
6.3 Diagrama de estados del flujo seleccionado 46
6.4 Consideraciones previas 47
6.5 Aplicación de la estrategia 48

7 Conclusiones y Trabajos Futuros 53
7.1 Conclusiones 53
7.2 Trabajos Futuros 54

Bibliografía 55

Lista de Figuras

1-1 Ejemplo de la representación gráfica de una arquitectura de tres capas. 2
1-2 Fases metodológicas del proceso de investigación. 5

2-1 Ejemplo de una vista de componentes y conectores en un sistema de software. . 9

4-1 Vista de capas de la implementación de referencia. 27
4-2 Vista de componentes y conectores. 28
4-3 Vista parcial del modelo de datos. 29
4-4 Vista de descomposición funcional. 30
4-5 Ejemplo de una de las páginas de la implementación de referencia. 31

5-1 Análisis de las técnicas utilizadas para integrar los diferentes dominios. 33
5-2 Análisis de las tecnologías para el contenedor de la aplicación. 35
5-3 Representación del modelo de descomposición para el componente de Front-End. 36
5-4 Escenario arquitectónico que representa la relación MicroFrontend-Microservicio. 38
5-5 Escenario arquitectónico que representa la relación API Gateway-API Shell. . . . 39
5-6 Escenario arquitectónico que representa la relación Microservicio-API Shell. . . 40
5-7 Posibles integraciones con base en los escenarios arquitectónicos planteados. . . 41
5-8 Representación de la integración del modelo de descomposición y la implementación de referencia. 42

6-1 Analogía entre el diseño detallado y el diseño arquitectónico. 44
6-2 Diagrama de estado para la creación de solicitudes. 46
6-3 Tabla de actividades del diagrama de estado. 47
6-4 Identificación de relaciones entre componentes y datos. 48
6-5 Árbol de subconjuntos con los pesos correspondientes. 50

1 Introducción

Desde hace algún tiempo en el mundo de la ingeniería de software se han venido buscando alternativas y estrategias de diseño que permitan que el software tenga una mejor estructura, un mejor comportamiento y que particularmente, sea un sistema sólido [62]. Esta búsqueda constante por mejorar gran parte del sistema tecnológico ha llevado a descubrir diferentes mecanismos que, en la práctica, han mejorado la interacción que tienen los sistemas de software con el cliente y con su entorno en general [14].

Uno de los principales ejercicios que ayudó a enriquecer el proceso de desarrollo en la construcción del software fue la implementación de una arquitectura definida, ya que esta práctica no solo representa un sistema abstracto de los requerimientos funcionales y no funcionales de las aplicaciones [43], sino que también apoya el proceso de toma de decisiones durante toda la fase de diseño del software, gracias al profundo vínculo que tiene con el comportamiento, la estructura y la evolución de los sistemas, tal como resalta Joao Franco [23].

En ese sentido, el uso de la arquitectura hizo que los sistemas de software empezaran a presentar mejoras en el rendimiento, en la distribución de los elementos, en el desarrollo [74], en el diseño e incluso en las formas de trabajo, definiendo así un hito importante para esta rama de las ingenierías [56]. De hecho, durante todo este proceso de adaptación surgieron múltiples esquemas que se han establecido como un parámetro guía para la evolución de los sistemas tecnológicos, y que además, se convirtieron en la hoja de ruta para la fabricación del software que se visualiza hoy en día [14]. Por ejemplo, una de las principales referencias arquitectónicas que se tienen en cuenta en la actualidad es el patrón arquitectónico de capas, un patrón en donde es común que los sistemas de software se organicen en segmentos funcionales que tienen permitido implementar propiedades de capas superiores.

De acuerdo con Guerraoui y Frølund [28], este patrón además de haber sentado una base teórica respecto a la forma en la que se dividen lógica y físicamente los sistemas de software, también revolucionó las prácticas de implementación que se realizaban en períodos anteriores, pues esta segmentación permitió que existiera una separación clara entre los distintos módulos de software. Algunos de los segmentos que más se utilizan en la modernidad son: La capa de Presentación, que abarca la interfaz de exposición de los datos hacia los usuarios, la capa de Lógica de Negocios, que considera la interacción, el manejo y el procesamiento de la información, y finalmente, la capa de Acceso a Datos, que gestiona el flujo, persistencia y almacenamiento de

los datos [39]. Una ilustración visual de lo que podría ser una segmentación para un sistema de software que abarca las tres capas mencionadas anteriormente es expuesto en la figura **1-2**.

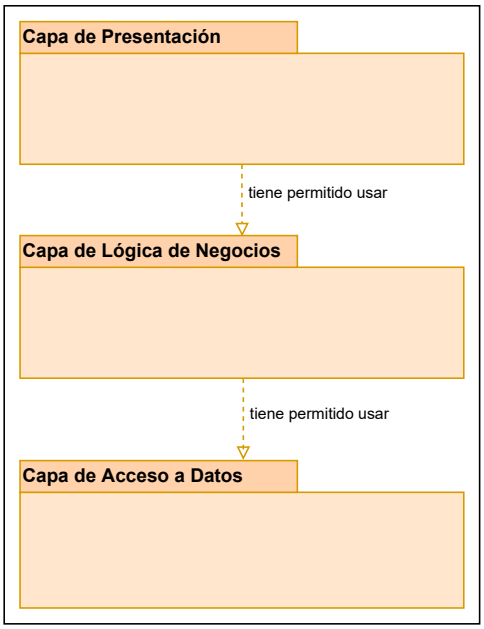

Figura 1-1: Ejemplo de la representación gráfica de una arquitectura de tres capas.

De igual manera, el uso de este modelo de capas generó un punto de inflexión en el desarrollo de software respecto a la forma en la que se estructuraba, puesto que la construcción de los sistemas en diferentes partes del mundo se empezó a dirigir hacia la fabricación de dos grandes áreas. Por un lado, las interfaces de usuario como uno de los módulos encargados únicamente de la parte visual de las aplicaciones y por el otro, el módulo de la lógica de negocios, que incluía una variedad de dominios para el procesamiento de la información [45].

Sin embargo, los sistemas de software tuvieron un crecimiento adicional debido al requerimiento generalizado de crear y migrar mecanismos manuales a nuevos sistemas tecnológicos [62]. Por este motivo, tanto el componente de Back-End como el componente de Front-End tuvieron que buscar soluciones para mitigar el incremento de los requisitos del software, adoptando nuevas tecnologías, procesos innovadores de desarrollo [3], renovadas metodologías de trabajo [51] y mucho más determinante, nuevas arquitecturas de software [55]. Como consecuencia de lo anterior, surgió el patrón arquitectónico de microservicios, que en su nacimiento como modelo se encargó de proponer nuevas soluciones a las dificultades asociadas a la capa de lógica de negocios y a la capa de persistencia de los datos [31].

En este patrón, cada uno de los elementos de la lógica de negocios es llevado a módulos específicos de desarrollo donde se ejecutan funciones y procedimientos de forma particular e inde-

pendiente [58]. Esta separación de responsabilidades en los componentes llevó a que muchos sectores vieran con buenos ojos la adopción de este modelo debido a la autonomía de aquellos módulos especializados, y ya con un proceso de evolución a sus espaldas, este método arquitectónico se transformó en una de las referencias a usar en el mercado [58]. Tal fue su crecimiento que se estableció de manera importante en el mundo de la investigación gracias a las facilidades que brinda para cierto tipo de aplicaciones [19], y en consecuencia, fue tomado como un elemento a seguir por diversas plataformas de tecnología debido a las características principales que lo definen, convirtiéndolo por último, en una fuente de análisis para el campo de la academia por su enfoque distribuido [48].

Por su parte, en el componente del Front-End, el problema de resolver los dilemas del crecimiento exabrupto del software es un poco más difícil dado que no se han aplicado patrones o estilos arquitectónicos determinantes que definan una estructura particular para el componente, y que integren de manera armoniosa cada uno de sus módulos [31]. En virtud de ello, la forma en la que se han tratado de remediar los inconvenientes de construcción de software ha sido adaptando tecnologías nuevas e independientes que se utilizan en la fabricación propia del software tales como librerías auxiliares y frameworks de desarrollo. Gracias a ciertas extensiones y complementos, estas herramientas han facilitado el proceso de construcción de las interfaces de usuario, pero no han solucionado detalles de fondo como temas de escalabilidad, autonomía, rendimiento en sistemas complejos [22], tiempos oportunos de respuesta o agilidad en los procesos de codificación para plataformas mayúsculas [2].

1.1 Introducción al problema

En la actualidad, los componentes de Front-End del software han tenido un incremento acelerado debido al auge de las plataformas tecnológicas [25], a la expansión global de las redes sociales, al uso desmedido de las aplicaciones móviles, y en general, a la transformación digital que ha sufrido el mundo por razones como la pandemia. Este crecimiento además de propiciar la exploración de alternativas que permitan mitigar los desafíos emergentes en el componente [22], también ha hecho que los sistemas actuales busquen la forma de simular las prácticas, métodos y tecnologías que se usan en otros componentes de software para no quedar rezagados con respecto a sus semejantes [66].

Algunas de las causas que han ayudado a describir las limitaciones que tiene este componente son descritas a continuación:

a. Crecimiento acelerado de los sistemas de software y en particular, de plataformas que requieren interfaces de usuario dinámicas y funcionales [30] [73].

b. Estancamiento de las actuales tecnologías y frameworks de desarrollo [74].

c. Falta de complementos funcionales para dar cumplimiento total de los requerimientos e interacción constante con el usuario final [50].

d. Falta de consideraciones arquitectónicas que permitan una evolución pragmática de los componentes de software actuales [72].

Ahora bien, por la repercusión de estas razones dentro del mundo del desarrollo de software, es necesario encontrar estrategias de descomposición para los componentes de Front-End, especialmente para los componentes de tipo web, con el objetivo de encontrar tanto en la academia como en la industria, componentes de Front-End elásticos, separados funcionalmente, con piezas integrales y completas, desacoplados, resistentes a cambios y escalamientos [26]. Así pues, la situación ideal para estos casos sería encontrar métodos definidos que permitan realizar el desacoplamiento lógico y físico de los componentes Front-End con el propósito de aumentar la calidad del software y optimizar las diferentes perspectivas arquitectónicas de los sistemas.

1.2 Objetivos

1.2.1 Objetivo General

Aplicar estrategias de desacoplamiento del patrón arquitectónico de microservicios en componentes Front-End de tipo web, con el propósito de evaluar la cohesión entre sus componentes.

1.2.2 Objetivos Específicos

1. Identificar los aspectos de desacoplamiento más relevantes en el patrón arquitectónico de microservicios.

2. Realizar una implementación de referencia (sistema de software - caso de estudio) siguiendo los principios de una arquitectura de microservicios con un único componente de Front-End.

3. Analizar los componentes y módulos funcionales de la arquitectura diseñada en la implementación de referencia.

4. Diseñar una estrategia de desacoplamiento para el componente de Front-End de tipo web mediante los aspectos de desacoplamiento identificados en microservicios.

5. Evaluar la cohesión de los componentes de Front-End resultantes a partir de los principios de cohesión en el patrón arquitectónico de microservicios.

1.3 Metodología

El proceso de construcción de este proyecto tiene como punto de referencia un enfoque investigativo desarrollado por Micah Godbolt, cuya finalidad es estudiar las arquitecturas de sitios web escalables a partir de un análisis cualitativo de ciertos pilares en el componente de Front-End, teniendo en cuenta la práctica e indagación experimental [27]. Esta aplicación metodológica contempla algunas fases de construcción de sistemas de software que son comprendidas en el proceso de estudio de Harman y Jones, en donde se establecen actividades críticas para la fabricación de cualquier sistema de software como lo son la planificación, el análisis, el diseño, la implementación, las pruebas, el despliegue o el mantenimiento [30]. Estas actividades son tomadas en consideración para definir un conjunto de etapas de trabajo que describen el desarrollo de la investigación, y que están alineadas de forma directa con los objetivos específicos mencionados en la sección anterior.

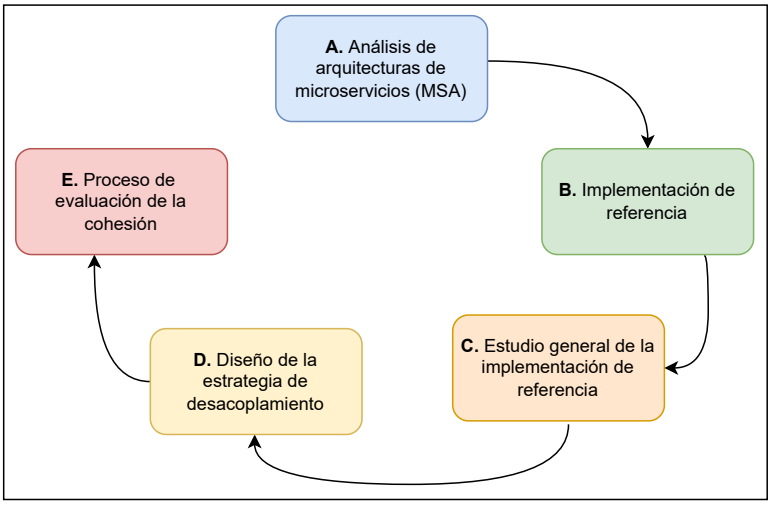

Figura 1-2: Fases metodológicas del proceso de investigación.

1.3.1 Análisis de arquitecturas de microservicios

Esta fase metodológica contempla el estudio detallado de las propiedades y aspectos más importantes del patrón arquitectónico de microservicios con el fin de tener una base teórica y conceptual lo suficientemente amplia para la consecución del proyecto.

1.3.2 Implementación de referencia

Esta etapa está relacionada a todo el proceso de construcción de la implementación de referencia de acuerdo con los principios de arquitectura encontrados en la fase anterior, y teniendo como punto de partida un solo componente de Front-End para la capa de presentación.

1.3.3 Estudio general de la implementación de referencia

En esta fase se realiza una exploración sobre los elementos, propiedades, relaciones, interacciones, funcionalidades y flujos de datos resultantes de la implementación desarrollada, con el propósito de determinar los parámetros y aristas que se deben considerar para proponer cualquier tipo de estrategia de descomposición.

1.3.4 Diseño de la estrategia de desacoplamiento

El propósito de esta etapa de la metodología es tener en cuenta cada una de las consideraciones de las fases previas para llevar a cabo un proceso de diseño e implementación de una estrategia de desacoplamiento que tenga presente tanto los requisitos funcionales del sistema como los principios arquitectónicos del patrón de microservicios en aras de construir un prototipo funcional y descompuesto.

1.3.5 Proceso de evaluación de la cohesión

En esta etapa se realiza una evaluación de la cohesión del modelo resultante tomando como base uno de los flujos funcionales que más interacciones tiene dentro del sistema. Allí, se establece una métrica análoga que permite medir la cohesión basada en términos arquitectónicos y que define una escala particular para el método de evaluación.

2 Marco Teórico

Como parte fundamental del desarrollo de este trabajo, es necesario conceptualizar de manera profunda la idea que se quiere desarrollar a través de la definición semántica de los términos utilizados. Por esta razón, varios de los conceptos empleados en esta investigación son definidos según fuentes bibliográficas o referencias académicas. Inicialmente, se debe tener en cuenta que el ámbito principal sobre el cual se adelanta la investigación es la arquitectura de software, una práctica que ha sido incluida dentro de la rama de ingeniería del software desde hace ya varios años [14], y que ha tenido una extensa acogida en el mundo académico y de la industria del software.

2.1 Arquitectura de Software

De acuerdo con Taylor, la arquitectura de software es un conjunto definido de las principales decisiones de diseño que se deben de tomar para construir y desarrollar un buen sistema de software [71]. Cuando se hace mención a un buen sistema de software, se hace referencia a que cada una de estas decisiones repercuta en las relaciones entre los componentes, permita definir reglas claras de cómo los elementos arquitectónicos deben estar organizados para cumplir una labor particular [58], brinde una idea de cómo deben ser los comportamientos de cada uno de los procesos incluidos, delimite las decisiones relacionadas con la interacción de los elementos y que finalmente, determine los detalles de implementación de toda la solución para que estos cumplan a cabalidad con las especificaciones del usuario [71].

De igual forma, estas decisiones deben ir en búsqueda de mejorar la estructura y la composición de los elementos del sistema a través de una correcta interpretación de los requisitos funcionales y no funcionales [45], puesto que cuando se nombra el concepto de estructura, se consideran cada una de las piezas que determinan el sistema y la forma en la que estos elementos se integran para formar un conjunto sólido de software. Por consiguiente, vale la pena resaltar que la arquitectura de un sistema también se puede definir en términos de sus componentes como la estructura o las estructuras del sistema de software que comprenden elementos, propiedades visibles de estos elementos, las relaciones que permiten su comunicación y sus principios de diseño y evolución [59]. En esta definición se encapsulan dos conceptos claves, el término de estructura y el término de elementos, dejando claro que estos son pilares fundamentales para la

noción de una arquitectura.

Por otro lado, se encuentra que la arquitectura está ligada netamente con el sistema de software, ya que para contar con un conjunto definido de atributos, se debe contemplar el panorama completo de sus posibles interacciones y los aportes de esta interacción dentro del sistema; idea que está alineada al concepto de que la arquitectura de software permite realizar la abstracción de un sistema mediante sus componentes, sus propiedades y las relaciones entre estos [65] [71].

Paralelamente, considerando que la arquitectura de un sistema de software está vinculada con la integración apropiada de elementos, es oportuno traer a colación las diversas partes que configuran esta estructura. En este caso, para comprender en una mejor medida el objetivo sobre el cual se apunta, es necesario saber qué es un componente de software.

2.2 Componentes de Software

Según Clements, los componentes de software son el conjunto de todos aquellos elementos que tienen presencia en tiempo de ejecución dentro de los sistemas computacionales tal como los procesos, los objetos, los clientes, los servidores, los almacenes de datos, etcétera. Por lo tanto, cuando se habla de un componente de software se precisa en aquellas unidades que conforman un sistema bajo ciertos lineamientos arquitectónicos que se suelen establecer con un objetivo específico. Estas piezas de software tienen un nivel de abstracción mucho mayor que cualquier elemento funcional dentro del conjunto, dado que son vistos como módulos que exponen una conducta relacional, más allá del desempeño de sus labores asignadas [16].

Generalmente, los componentes de software brindan una serie de servicios para el resto de elementos del sistema que permiten que el conjunto trabaje de forma integral, por lo tanto, cuando una de estas partes es removida, alguna funcionalidad o fragmento de software deja de trabajar [59]. Asimismo, cuando se construye un componente de software de la mano de una arquitectura definida, primero se intenta especificar una colección de responsabilidades que este elemento debe cumplir, y segundo, se establece una serie de tareas que se deben satisfacer sin importar la naturaleza del componente. De manera similar, la arquitectura de software define el comportamiento e interacción del componente dentro del sistema, pues parte de las configuraciones preliminares vienen de la mano con el diseño definido en las etapas introductorias de la construcción [40]. Por esta razón, son estos lineamientos los que determinan los límites a los cuales puede llegar un componente en términos de funcionalidad, desempeño, detalles de implementación, y relación con las otras piezas del conjunto.

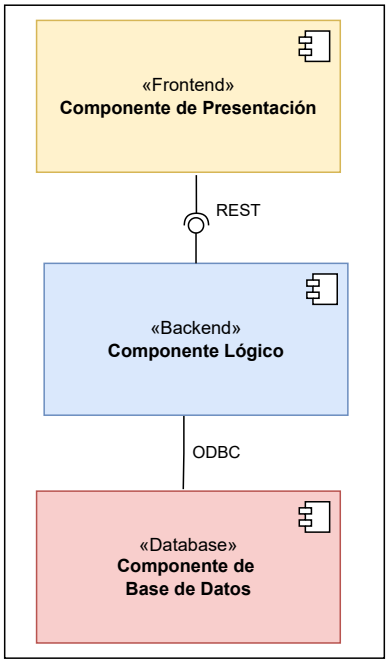

Figura 2-1: Ejemplo de una vista de componentes y conectores en un sistema de software.

Dependiendo el tipo de componente de software que se esté construyendo, se presentan una gama particular de ventajas, así como un grupo específico de riesgos o inconvenientes. Esto se da ya que el componente como tal puede contener una serie de dependencias que le permiten cumplir con sus funciones, pero que a la misma vez lo imposibilitan de exponer una característica adicional al sistema [17]. De la misma forma, es conveniente resaltar que los componentes de software tienen una gran variedad de propiedades que los distinguen de otros elementos de los sistemas por el enfoque especial que tienen una vez son construidos [9] [44]. Algunas de las propiedades más importantes de estas unidades son:

i. **Rendimiento en el manejo de los datos**: Independiente del tipo de componente que sea, si es construido particularmente para gestionar información o tiene alguna función especial, los componentes de software deben tener en cuenta la forma y los métodos utilizados para procesar los datos puesto que esta porción de software debe proveer una respuesta ágil a cualquier tipo de llamado [21].

ii. **Acceso por uno o diferentes puntos de comunicación**: Una de las propiedades fundamentales dentro de estas piezas es la relación que tienen con el resto de los miembros del conjunto de software, y es por esta razón, que deben brindar de manera clara ciertos puntos de acceso para que las diversas entidades del sistema puedan comunicarse de manera precisa. Usualmente, la forma en la que se realiza la conexión es a través de los puertos de red del componente [59].

iii. **Cumplimiento de un conjunto predefinido de funciones**: Los componentes de software deben estar construidos de manera tal que solo se encarguen de un conjunto predefinido de funciones para que esto no implique cambios en tiempo de ejecución o que se identifiquen fallos por realizar tareas que no tenían asignadas desde un inicio. El hecho de llevar a cabo más de un propósito hace que exista dependencia entre otras unidades o que se haya hecho una mala abstracción desde la fase de diseño [31].

Como parte del proceso de concepción de los componentes, es imprescindible conocer los mecanismos con los cuales se construyen estos grandes módulos. Por esto vale resaltar que la construcción de componentes de software tiene un historial bastante amplio debido a que desde hace varios años se inició este proceso. Particularmente, el surgimiento de estos elementos de software se dio cuando se establecieron ciertos métodos que brindaban servicios básicos al usuario [51], y a partir de allí, la construcción y el desarrollo del software en general, mejoró sus formas y mecanismos de construcción estableciendo metodologías ágiles de desarrollo, tecnologías adaptables a los requerimientos, estilos de codificación, patrones de diseño y considerando las condiciones funcionales de los sistemas [67][68].

Una de las principales estrategias para la fabricación de los componentes de software es la elaboración de modelos en notaciones formales tales como lenguajes y gramáticas de dominio específico. Por ejemplo, la forma *Backus-Naur Form* y las transformaciones modelo a modelo, han permitido diseñar en gran medida los módulos específicos del software y algunas interfaces de usuario [21]. Otro de los mecanismos que se utilizan en la actualidad es la separación y asignación única de responsabilidades basada en la descomposición de tareas. Esta distinción de funcionalidades está representada como un grafo de responsabilidades únicas donde cada módulo cumple una función específica y hay enlaces de conexión que permiten establecer los datos o conectores sobre los cuales se comunica el sistema [76].

Análogamente sucede lo mismo con los tipos de componentes de software, que a pesar de que estén diseñados con diversos propósitos, y las áreas para los que se utilizan sean completamente distintas, hay un abanico amplio de componentes que se implementan en todas las industrias [69]. En concreto, cuando un componente es construido de manera flexible o tiene requerimientos que en cierta medida son estándar, puede funcionar tranquilamente para diferentes segmentos que comparten un dominio. Esta cualidad es destacable en el mundo del software ya que la idea de que algunos componentes sean adaptables a diversos tipos de escenarios es un valor agregado que evita escribir código desde cero. En la actualidad, el software es aprovechado en varios sectores económicos, lo cual aumenta el espectro de los tipos de componentes que existen [26]. Algunos de ellos son:

a. Componentes de software para el control y la telemetría de satélites.

b. Componentes de escritorio para el acceso a información privilegiada.

c. Componentes basados en la interacción del usuario.

d. Componentes de software hechos a la medida.

e. Componentes de procesamiento y tratamiento de datos.

f. Componentes de almacenamiento de información.

g. Componentes de software basados en el paradigma de cliente-servidor.

h. Componentes específicos que cumplen un solo propósito.

De manera adicional, vale la pena destacar que los componentes de software tienen a su vez diversas categorías que los agrupan, y que pueden estar enlazadas directamente con los tipos enunciados arriba. Para esta ocasión, el tratamiento que se quiere realizar está basado en los componentes de tipo web, que son todos aquellos módulos de software que trabajan bajo el sistema WWW a través de Internet [53], y que pueden estar basados en el procesamiento de información o en la relación con el usuario. Dentro de este conjunto hay tres componentes que se utilizan de manera enfática dentro de la investigación. Estos están asociados a la forma en la que se diseñó la arquitectura del sistema desde un inicio, y cumplen con labores particulares dentro del conjunto con la finalidad de estructurar la implementación de referencia.

2.2.1 Componentes de procesamiento y tratamiento de datos

Estos tipos de componentes, en algunas ocasiones llamados componentes de *Back-End*, son los encargados de manejar y gestionar toda la lógica de negocios dentro de un sistema de software. Allí, se llevan a cabo procesos de manipulación y tratamiento de la información con el fin de brindar un conjunto de servicios que proporcionan los datos necesarios para cumplir con funcionalidades específicas [39]. Estos están incluidos dentro de los componentes que procesan y tratan datos porque precisamente su función dentro de los esquemas de software es recibir datos de entrada, realizar una transformación interna y exponer otro tipo de información que es útil para el resto de elementos del sistema [11].

2.2.2 Componentes de almacenamiento de información

Los componentes de almacenamiento tienen un enfoque un poco más particular que el resto de los componentes de software mencionados, ya que estos se encargan de preservar la información que es enviada desde otros componentes. Aquí no hay ningún tipo de manipulación más allá de una conversión de texto, fechas o datos numéricos [47] [53].

2.2.3 Componentes basados en la interacción del usuario

Dentro de este gran rango de componentes, se encuentran los componentes de *Front-End*, que son los componentes de software que se presentan de cara al usuario. Son todas aquellas interfaces visuales o mecánicas que tienen contacto directo con el usuario final a través una pantalla, un receptor o un actuador [27]. Puntualmente, los componentes de Front-End son todas aquellas interfaces, plataformas, medios interactivos o elementos gráficos que posibilitan la visualización y exponen un conjunto de datos o de información hacia el usuario [1].

Algunas de las propiedades de los componentes de Front-End comprenden tanto características inherentes a la construcción del sistema, como cualidades que se utilizan exclusivamente en este tipo de componentes, ejemplo de ello es el tipo de contacto que tiene el usuario sobre la interfaz [27]. Por otro lado, existen algunas propiedades que están directamente relacionadas con el diseño y desarrollo del componente, como el correcto desempeño, la operación eficaz de la interfaz o ciertas consideraciones no funcionales contempladas en la fase de análisis del sistema [9].

Así como lo menciona Sherif Elshafei, los componentes encargados de gestionar la interacción con el usuario deben considerar el rendimiento asociado al manejo, administración y representación de los datos, disminuyendo la cantidad de lógica procesable y los procedimientos asociados a ellos [21]. Asimismo, es necesario que estos elementos consideren factores que están asociados más a la faceta funcional de las interfaces [50], tal como lo pueden ser:

a. El alineamiento de las herramientas visuales para la interfaz de usuario.

b. Las fuentes o recursos legibles sobre la interfaz.

c. La división clara de las etiquetas de texto.

d. La forma estética y el diseño de la interfaz.

e. La visualización ligera de los íconos.

En la actualidad, la gran mayoría de interfaces que se encuentran en el mercado son prototipos para aplicaciones web [26] y esquemas para aplicaciones móviles [33]. Estos sistemas tienen un gran espectro de uso y un amplio desarrollo en ambientes comerciales, sistemas empresariales y plataformas transaccionales. Sin embargo, la industria en general también se ha migrado a procesos computacionales involucrando dentro de su currículo prototipos hechos a la medida que permiten facilitar las operaciones de las compañías y automatizar funciones que en tiempos anteriores se realizaban de forma manual [12].

Otro grupo importante que se debe tener en cuenta dentro de la clasificación de estos componentes son los prototipos generados para con un contexto particular, o lo que en la actualidad

se le denomina *aplicaciones de dominio específico*. Estos esquemas han ido aumentando la cantidad productiva del mercado debido a su nivel de granularidad y especificidad [54]. Es importante resaltar que estos son los elementos objetivo sobre los cuales se va a desarrollar la investigación, ya que desde una perspectiva arquitectónica tienen un potencial muy prometedor, y claramente merecen el esfuerzo de conceder una mirada particular.

2.3 Arquitectura de Microservicios

Otro de los conceptos clave dentro de este proceso de investigación es la arquitectura de microservicios, un estilo arquitectónico que se fundamenta en la distribución funcional del software a través de pequeños componentes, llamados *microservicios*, que cuentan con un conjunto de atributos y propiedades definidas enfocados en resolver un problema de dominio específico. Este término de microservicios llegó acompañado de la aparición de los sistemas distribuidos que surgió alrededor del año 2004, con el propósito de contar con varios puntos de procesamiento de la información de manera autónoma sin perder la esencia misma del sistema, y la noción de independencia [48].

Además, en este tipo de arquitecturas aparte de conservar el paradigma de distribución, se intenta que la colección de los pequeños servicios cumpla con los estándares establecidos en las decisiones de diseño para que la estructura del sistema trabaje de forma adecuada, y satisfaga los requisitos del modelo [19]. Así que la idea principal de este estilo es la fragmentación del software en servicios minúsculos que generalmente agrupan un dominio particular o un conjunto de funcionalidades que cumplen un único fin [75].

En este punto, vale la pena mencionar que la noción de microservicio viene de la idea semántica de un servicio de software, sin embargo, apunta hacia una pieza minúscula que tiene unos límites definidos debido a su dimensión. Esta pieza es autónoma en cada una de las fases de su construcción (análisis, diseño, implementación, testeo, mantenimiento, etc.), pero debe garantizar que trabaja de forma conjunta para cumplir con los requisitos del sistema [47]. Uno de los dilemas que a veces sufren este tipo de arquitecturas es la definición del tamaño del microservicio, pues al ser útil para una gran cantidad de dominios, la magnitud corresponde más a una especificación subjetiva del mismo, que una determinación formal por parte de la literatura.

En sus inicios, la arquitectura de microservicios brindó unos fundamentos teóricos que sirvieron de base para la práctica y la implementación de este tipo de estructuras, y con el tiempo, más propiedades se fueron agregando a la descripción vertebral de la arquitectura para contar con propiedades más sólidas y contundentes [31]. Algunas de ellas son expuestas a continuación:

a. **Componentes enfocados en realizar de manera completa y eficaz una única característica:** La concepción principal de este estilo se fundamenta en la idea de construir unidades

de servicio individuales que estén diseñadas para cumplir con un conjunto limitado de funciones, que no sobrepasen las fronteras de implementación definidas para evitar así una superposición de tareas entre diferentes componentes [48].

b. **Autonomía:** Esta propiedad parte de la premisa de que cada una de estas piezas de software son una entidad separada que poco o nada tiene que ver con el resto de componentes del sistema. Aquí, el desarrollo de la unidad de servicio puede realizarse de manera independiente al resto de elementos del sistema, y para su construcción solo necesita información exclusiva del dominio, así como las reglas de implementación definidas en las decisiones de diseño [48].

c. **Modularidad:** Este es un principio fundamental del patrón que enuncia que los sistemas deben descomponerse en términos de *módulos* para que puedan ser desarrollados, entendidos y liberados por diferentes personas. La idea detrás de la modularidad es que se especifiquen pequeños módulos donde se condense la funcionalidad general del sistema [58].

d. **Paradigma de distribución:** Una característica intrínseca de este tipo de modelo indica que las arquitecturas deben procurar la descentralización con el objetivo de no tener puntos comunes de fallo o fuertes dependencias sobre un único componente. El pilar de este patrón es tratar de desacoplar de la manera más granular posible los dominios del sistema con el fin de pretender entidades libres más fáciles de inspeccionar [48].

e. **Comunicación independiente:** Gracias a algunas de las características mencionadas anteriormente, es destacable que los microservicios pueden tener protocolos de comunicación totalmente independientes del resto de los componentes del sistema.

Evidentemente, la aplicación de este patrón dentro de un modelo nuevo o ya existente debe ser evaluado en la etapa de análisis y diseño de fabricación del software, pues no todos los entornos digitales resisten la migración a sistemas distribuidos como la arquitectura de microservicios. Sin embargo, la puesta en marcha de este tipo de arquitecturas dentro de un sistema de software tiene una gran cantidad de beneficios que pueden ser aprovechados dependiendo el tipo y la intención del sistema. Así pues, algunas de las ventajas que trae su implementación son enunciadas enseguida:

i. **Heterogeneidad tecnológica:** Debido a que cada una de las unidades de servicio son componentes individuales, cabe la posibilidad de realizar la construcción de estos elementos de manera aislada mediante una tecnología independiente. Así pues, la disposición de un *stack* de tecnología específico para cada situación brinda la posibilidad de escoger indistintamente un lenguaje de programación, una herramienta, una librería o un protocolo. Esta pluralidad no solo es aplicable a la etapa de fabricación, sino también a una potencial migración o actualización. Esto no solo representa un beneficio pragmático dependiendo las

funciones del servicio, sino que también garantiza el desacoplamiento de dependencias que se podían presentar en componentes monolíticos anteriores [48].

ii. **Pruebas autónomas:** Además de que es más sencillo hacer pruebas sobre componentes de software más pequeños a través de mecanismos de pruebas unitarias o funcionales, estos elementos tienen la capacidad de ser probados de manera individual sin que algún proceso interno deba ser ejecutado en la mitad [57].

iii. **Resiliencia:** Otra ventaja que tiene la aplicación de esta arquitectura, es la resiliencia que existe detrás de su modelo, es decir que esta propiedad está inmersa dentro de los microservicios en la medida en que si una parte del sistema presenta un inconveniente técnico asociado al desarrollo o a su ejecución en sí (un servicio, un conector o un módulo), el sistema tiene la oportunidad de recuperarse de ese fallo sin perjudicar ninguna de las funciones del sistema. Este factor es muy importante respecto a otro tipo de patrones ya que cuando algo falla en estas estructuras, el error no suele afectar todo el conjunto, no se presentan indisponibilidades del servicio ni tampoco períodos extensos de mantenimiento [48].

iv. **Reducción de la complejidad:** El hecho de contar con unidades pequeñas con comportamientos predecibles hace que se simplifique la comprensión lógica y funcional de estas piezas de software. Del mismo modo, su carácter minúsculo concede la chance de eliminar posibles dependencias computacionales cuando se realizan migraciones o cambios en el código, proporcionar mecanismos sencillos de comunicación y facilitar la interacción con el resto de componentes del sistema [57].

v. **Facilidad en el despliegue:** Con el uso de los microservicios, la aplicación, reversión o actualización de cualquier magnitud no representa un reto en la fase de despliegue, ya que como el componente es autónomo y está aislado del resto de partes del conjunto en su entorno de ejecución, el proceso de puesta en marcha se realiza de forma independiente sin afectar otros componentes y sin crear efectos colaterales. Esta ventaja también va en búsqueda de que el sistema tenga una respuesta inmediata a los cambios y no gaste largos períodos de tiempo en fases que no lo requieren [48].

vi. **Escalamiento:** En este enfoque, los servicios que necesiten un aumento en su capacidad de respuesta o una mejora en su rendimiento pueden ser escalados de manera individual o conjunta según sea el caso; situación completamente diferente a las arquitecturas monolíticas, ya que allí se debe escalar todo el sistema incluso cuando algunos componentes no lo requieren. Adicionalmente, la ventaja de realizar escalamiento en un sistema distribuido radica en la forma en la que se manejan los costos o los mecanismos de aprovisionamiento, pues están controlados bajo la demanda del microservicio y así no hay solo un ahorro de costos para el sistema en general, sino que también las mismas métricas de rendimiento dan estadísticas de mejora, de retroalimentación o de cambio [48].

vii. **Alineación organizacional:** Uno de los factores por los cuales surgió la arquitectura de microservicios desde un lado no tan técnico es por la forma en la cual se organizan los equipos de trabajo para completar una tarea. Frecuentemente cuando un sistema de software de una magnitud medianamente compleja es monolítico, y aparte tiene una gran cantidad de personas trabajando detrás, ese sistema colapsa en algún punto de su construcción porque se realizan mezclas de funcionalidades o no hay una separación clara de las responsabilidades. Incluso, existiendo una división precisa de tareas, los segmentos compartidos terminan teniendo conflictos porque alguno de los miembros del equipo agrega cosas diferentes a las de otro incluso sobre los mismos archivos.

   De este modo, la distribución del trabajo en muchos equipos de desarrollo además de ser una tarea difícil también puede llegar a ser un inconveniente que se presenta en el día a día de la construcción del software y en estas condiciones lo mejor es evitar cruzar obligaciones. Por lo cual, la construcción de pequeños servicios con una responsabilidad única ayuda en la productividad del trabajo en la medida en que los equipos con menos integrantes suelen ser más compactos y rendidores, pues hay una mejor distribución de cada uno de los elementos del sistema y se garantiza la disposición de piezas libres de conflictos debido a la sincronización del trabajo [48].

viii. **Evolución y extensibilidad:** Gracias a que la mayoría de las propiedades de este sistema surgieron como respuesta a las dificultades que tenían otras arquitecturas, se incorporaron las mejores características de otros patrones en pro de contar con un modelo sólido y robusto. Esto se puede evidenciar en los procesos simplificados de entrega continua, los *pipelines* de desarrollo, la comprensión de componentes desacoplados o las unidades distribuidas de despliegue. Por esta razón, la evolución y mantenimiento de este tipo de distribuciones es un proceso más ágil como consecuencia del volumen mismo del microservicio [57].

ix. **Libertad en bases de datos:** Debido a que los servicios están bajamente acoplados, y se comunican en algunos casos a través de topologías de APIs, una forma de aportar en el tema del bajo acoplamiento es haciendo que cada uno de los componentes de servicio tenga una o varias bases de datos independientes [58].

2.4 Descomposición Arquitectónica

Ahora bien, teniendo en cuenta que la definición de la arquitectura de software se relaciona con la composición estructurada de elementos dentro de los sistemas de software, es necesario considerar que hay un proceso inverso donde cada una de las piezas que componen el sistema se separa sin perder la integridad de la información o el contenido natural del sistema. Generalmente, este proceso de desacoplamiento es nombrado en la literatura como *descomposición*

arquitectónica [41].

De acuerdo con Abdulaziz Alkhalid, la descomposición arquitectónica reúne todas aquellas estrategias, prácticas o modelos que permiten dividir sistemas de software en partes más pequeñas con el fin de tener mejores procesos de construcción, desarrollo, integración, comunicación y evolución [6]. Generalmente, la distribución de estas estructuras constituye un conjunto de entidades y relaciones que tienen una asociación fuerte entre sí [5]. Asimismo, cuando los componentes o elementos que son divididos tienen un proceso de comunicación definido y están descompuestos de tal forma que hay una coherencia general del software, la integración de estos constituye un sistema de software compuesto [46].

2.5 Cohesión

Uno de los principios que mejor describen las buenas prácticas dentro de la ingeniería del software es la *Alta Cohesión*, un concepto que apunta a que todas las funciones que tengan intereses en común dentro de un componente de software estén dentro del mismo dominio, con el objetivo de disponer de piezas con responsabilidades exclusivas. De igual forma, este principio hace referencia a la medición que estima el cumplimiento de los objetivos establecidos en la fase de diseño por parte de un módulo de software [36].

En consecuencia, contar con alta cohesión dentro de una arquitectura termina siendo provechoso para el sistema porque procura que todas las características relacionadas estén dentro del mismo conjunto, y esto es una ayuda fundamental en procesos de modificación, actualización o eliminación funcional [8]. Adicionalmente, gracias a que el término de cohesión destaca la interacción de los elementos de un módulo de software en términos funcionales, es posible evaluar algunas decisiones arquitectónicas en materia de desacoplamiento.

Otro aspecto fundamental en esta propiedad es que la cohesión está soportada por el principio de responsabilidad única, lo que garantiza que las implementaciones con alta cohesión siempre van a estar bien definidas porque tienen un propósito singular de cambio [49]. Al mismo tiempo, el hecho de tener cohesión en un sistema no solo hace que se facilite el proceso de mantenimiento, evolución y pruebas, sino que también su independencia funcional permite reducir la complejidad del sistema ya que los dominios están correctamente separados y no son dependientes uno del otro [38].

De este modo, se logra ver la relación que tiene la alta cohesión con el patrón arquitectónico de microservicios, pues el desacoplamiento de las diferentes unidades de servicio debe estar soportado en un ejercicio responsable de separación según el dominio y los requisitos del sistema [48]. Algunas de las propiedades que se deben considerar en la arquitectura de microservicios

desde el punto de vista de la cohesión son las siguientes:

a. Los microservicios son unidades autónomas de servicio que son desarrolladas con una única responsabilidad y trabajan de forma integrada para cumplir un solo objetivo, por ende, tienen la tarea de realizar de manera completa una característica dentro del ecosistema de software sin sobrepasar los límites de implementación definidos en las decisiones de diseño [58].

b. El hecho de que un microservicio pueda ser construido por un grupo de trabajo independiente, hace que se redunde en cohesión e independencia organizacional puesto que cada grupo de trabajo se encarga de una sola funcionalidad sin depender de otros elementos estructurales [4].

c. Si un microservicio comprende dos características o dominios diferentes, el mejor enfoque es implementar cada característica de manera independiente de tal forma que no se acoplen [48].

d. Una forma de continuar y aportar en el tema del bajo acoplamiento en una pieza de software, es sugiriendo que cada uno de los componentes de servicio tenga su propia base de datos, sin ser esto una regla universal [58].

e. La construcción de los microservicios está contenida bajo las restricciones del negocio, es decir, no debe hacer ni más ni menos de lo que se necesita debido a que se pueden fomentar fracciones de código prescindibles [19].

f. Un microservicio no debe exponer todo su funcionamiento a diferentes entidades de un sistema de software si no es un componente altamente conectado, pues aumenta la posibilidad de generar dependencias e incrementar su acoplamiento [48].

g. La modificación o actualización de un microservicio debe ser totalmente independiente a otra funcionalidad con el fin de que el nivel de acoplamiento sea reducido y los componentes de software no sufran cambios por agentes externos [58].

2.6 Acoplamiento

Otro de los pilares fundamentales que se pueden incluir como parte de los atributos de calidad del software es el *Bajo Acoplamiento*, un principio que está relacionado con el grado de dependencia que tiene un componente de software respecto a otro. En efecto, este fundamento apunta a que el nivel de subordinación que tiene cualquier elemento del sistema sobre otro realmente no represente un vínculo crítico para la estructura del conjunto [42], dado que contar con relaciones estrechas puede generar inconvenientes en los procesos de diseño y construcción del software.

Adicionalmente, la idea detrás de este precepto es que las piezas de software no tengan ninguna atadura o que la relación existente entre ellas sea verdaderamente ínfima, con el objetivo de poder realizar cambios o modificaciones sin llegar a alterar alguna función u operación dentro de otro componente [52]. Naturalmente la comunicación y colaboración entre las diferentes partes del sistema es esencial para el desempeño de cualquier funcionalidad, pero en este proceso de análisis también se debe considerar el grado de complejidad que surge de relacionar una entidad con otra [35] [52].

De forma similar, el principio de bajo acoplamiento procura evitar los efectos colaterales de cualquier modificación dentro de los componentes del sistema en la medida en que la relación existente entre una y otra característica se da en términos de integración de los datos, pero no a través de un intercambio de métodos o funciones [63]. Este mecanismo de integración de información generalmente es una interfaz que brinda la posibilidad de conectar diversas piezas de software de tal forma que solo exista un único punto de enlace para acceder a los detalles compartidos. De esta manera, se aíslan de manera estricta las especificaciones internas de implementación del componente y se procura tener un nivel de acoplamiento bajo en la medida de lo posible [32].

3 Trabajos Relacionados

Parte primordial de la elaboración de este trabajo se basa en la descripción de aquellos acercamientos, aproximaciones o alternativas de solución que se han proporcionado para clarificar la idea que se tiene en mente. Por este motivo, es preciso traer a colación aquellas estrategias de descomposición que se han realizado tanto en la industria como en la academia para consolidar el planteamiento en desarrollo.

En primer lugar, es necesario resaltar que durante el análisis realizado sobre la literatura, no se lograron identificar estrategias formales de descomposición específicamente para los componentes de Front-End, pero sí existe una recopilación de algunos artículos científicos en donde se realiza el desacoplamiento de partes funcionales del sistema, tanto en componentes de Back-End como en aplicaciones *Full-Stack* en general.

En efecto, algunas técnicas que pueden servir como una referencia teórica y una guía experimental para realizar el proceso de descomposición arquitectónica son las siguientes:

- La primera es una *técnica de descomposición* implementada a través de un método construido en el año 2013, que se fundamenta en el uso y análisis de palabras clave. Esta herramienta que tiene acogida en varios campos de la informática, es un mecanismo que se basa en algoritmos de agrupamiento de acuerdo con un análisis de responsabilidades. Allí, los métodos computacionales logran identificar las funcionalidades particulares que tienen los sistemas y las métricas operacionales que deben cumplir, con el objetivo de establecer una estructura de elementos integrales.

  Este método que emplean los autores es una muy buena referencia comparada con las implementaciones que se van a presentar en seguida, puesto que no solo se identifican los requerimientos funcionales en el sistema, sino que también se separan los módulos o componentes en determinados conjuntos con algún tipo de interés. Sin embargo, este procedimiento realiza una separación rígida de los elementos sin tener en cuenta la integridad de los datos y las funciones, es decir, que al realizar la separación de un módulo puede considerar funcionalidades de otras secciones cercanas sin contemplar la operación particular del elemento [6].

  Esta omisión puede tener ciertas consecuencias en términos de integridad o coherencia de

la información, porque las funciones hacen parte fundamental del contenido estructural del software [18]. Pese a esto, un punto muy importante que puede ser tomado como referencia para la aplicación del método, es la técnica de *clustering*, que hace énfasis en la selección de los componentes del sistema de forma abstracta y considera que los elementos deben quedar ubicados como un conjunto integral con puntos de interés [5].

- Otro mecanismo de descomposición de los sistemas de software encontrado dentro de la revisión bibliográfica es el *modelado funcional de la arquitectura*, cuya elaboración se realizó en el año 2014. Durante la revisión del modelo se encontró que hay una descripción profunda de las funciones prioritarias del sistema, y en particular, de aquellos núcleos de implementación que tiene el sistema. En ese sentido, el modelo implementado concibe todos los requerimientos del sistema con el fin de establecer un lenguaje de descripción de arquitectura (ADL). La finalidad de realizar la construcción del lenguaje es tener en cuenta las consideraciones mencionadas, además de una métrica estándar para la estructuración de arquitecturas predefinidas [64].

  Adicionalmente, este lenguaje soporta la desagregación de los elementos, las modificaciones sobre los componentes, y una estructura particular de relaciones mediante las cuales se comparte información o llamados a servicios. En ese aspecto, el lenguaje resiste la adición de atributos de calidad sobre el sistema y realiza una retroalimentación continua de los atributos no funcionales. A pesar del gran alcance que tiene la elaboración de este lenguaje, las aproximaciones específicas que se han logrado hacer se sostienen en la idea de dividir los módulos del Back-End sin considerar los resultados de lo que puede ser el proceso de distribuir los elementos del componente de Front-End. Esto reduce el espectro de consideración puesto que hay un enfoque remarcado en dividir de forma granular los servicios de la capa de lógica de negocios sin considerar los demás elementos del sistema [64].

- Por otro lado, una técnica arquitectónica alternativa es la distribución de componentes basados en arquitecturas de software dinámicas. Este modelo fue un desarrollo implementado en el año 2015 donde se planteó un lenguaje de descripción de arquitecturas extenso, sin embargo, este lenguaje no está implementado de forma estándar, sino que concibe ciertos parámetros que permiten una modificación particular de elementos que conforman el sistema [13].

  Lo anterior da pie para mencionar que las arquitecturas dinámicas son todas aquellas estructuras elásticas que no necesariamente están constituidas bajo procesos o métodos específicos, pero que soportan sistemas de software flexibles y son resistentes a modifi-

caciones o cambios funcionales. Bajo esa misma perspectiva, las arquitecturas dinámicas constituyen un gran marco de referencia para los sistemas de software actuales, sin embargo, las consecuencias que tienen estas estructuras no han sido evaluadas en los atributos de calidad del software como el rendimiento, la comunicación o la interoperabilidad.

Esta estrategia no deja de ser un aporte importante dentro del marco de investigación pues concibe una composición distribuida en el sistema. Por ejemplo, en esta muestra se toma como referencia el módulo de usuarios donde se realiza un escalamiento de peticiones por cada una de las solicitudes requeridas y de forma adicional, el lenguaje permite el análisis funcional de los componentes de acuerdo con su uso e interacción con otras piezas de software [13].

- Un mecanismo adicional para la descomposición de componentes de software, es un patrón estructural producido en el año 2016, donde se propone un modelo arquitectónico de componentes que considera condiciones y restricciones específicas de un sistema. El desarrollo de este modelo está configurado como la extensión de un trabajo realizado previamente donde se especifica la descripción formal de la arquitectura en términos de cada una de las piezas que integran el sistema, todo esto se lleva a cabo mediante la definición de un lenguaje denominado CLACS. La construcción y evolución de este modelo evoca la idea de usar el paradigma del desarrollo de software basado en componentes con la finalidad de promover de manera más exacta la descripción de la arquitectura [72].

  Este paradigma no solo genera una descomposición de los módulos de forma funcional, sino que también genera una métrica de comunicación entre los componentes teniendo en cuenta las restricciones definidas en la etapa inicial del diseño. En ese aspecto, utilizar el paradigma como un mecanismo de construcción de software no solo genera una referencia de comunicación e interacción entre los componentes, sino que también produce un modelo arquitectónico que puede ser utilizado como guía en otros sistemas.

  Durante el análisis de este trabajo se obtiene que el fundamento primordial para poder realizar la descripción de la arquitectura son las restricciones topológicas, físicas, funcionales y no funcionales del sistema. Por esta razón, los módulos abarcan todo el contenido del problema cuando se realiza un proceso puntual de despliegue. Por último, es necesario mencionar que la aplicación del modelo utilizado es probada en un sistema particular de software con un catálogo de restricciones tomadas de varios sistemas [72].

Finalmente, vale la pena resaltar que en la figura **3-1** se presentan de manera cronológica los detalles principales de las estrategias de descomposición descritas anteriormente.

Método	Descripción	Año	Autor Principal	Análisis
Descomposición de la arquitectura usando un algoritmo particular de vecinos cercanos	Implementación de un algoritmo de descomposición de la arquitectura de software teniendo en cuenta los elementos cercanos del sistema.	2013	Abdulaziz Alkhalid	En su proceso de descomposición no tiene en consideración elementos importantes del desacoplamiento como las relaciones entre elementos o los datos que se pueden compartir entre los componentes.
Método de modelado funcional de la arquitectura	Método para modelar la arquitectura funcional de un sistema de software como un sistema simplificado que permita las interacciones rápidas entre los componentes de software. Está enfocado en un uso fácil y con permanente comunicación.	2014	Wilbert Seele	Las consideraciones especiales que se tienen en cuenta para el proceso de desacoplamiento solo están diseñadas para la separación de los componentes del Back-End, es decir, que no vinculan integraciones o componentes de la capa de presentación.
Arquitecturas de software que permiten dinamismo	Modelo de arquitectura que permite un dinamismo comportamental en cada uno de los elementos de software que componen el sistema.	2015	Everton Cavalcante	Este proceso se detalla a través de la construcción de un lenguaje de descripción arquitectónica que solo concibe particularidades del negocio y permite dinamismo en la medida en que deja el espacio para agregar o remover componentes.
Uso de una arquitectura restringida de software por composición	Modelo arquitectónico de componentes que considera condiciones y restricciones específicas de un sistema a partir de la definición formal de la arquitectura	2016	Chouki Tibermacine	A pesar de que su enfoque es el paradigma basado en componentes, la descomposición de los elementos del sistema está basado en cada una de las limitaciones y restricciones expresadas en los componentes de software, y no considera elementos propios del dominio o la interacción con las demás capas del sistema.

Tabla 3-1: Cuadro comparativo con los trabajos relacionados y su respectivo análisis.

A pesar de que cada una de las propuestas descritas anteriormente presenta ciertas limitaciones o restricciones inherentes a su construcción, y no considera elementos propios de la capa de presentación que puedan ser tenidos en cuenta para el modelo de descomposición, pues terminan siendo referencias relevantes para el desarrollo del proyecto en la medida en que, utilizan lenguajes de descripción de arquitectura que comprenden propiedades determinantes para la elaboración de posibles modelos arquitectónicos, emplean diferentes algoritmos que permiten la identificación de los puntos en común entre los componentes, y relacionan aspectos esenciales de los sistemas como las descripciones funcionales del negocio con la finalidad de vincularlos al proceso de desacoplamiento de los componentes de software.

4 Implementación de referencia

Es bien sabido que el mundo del software ha hecho un gran aporte a la industria financiera puesto que ha simplificado de forma radical las políticas que este sector aplica en el desempeño de su operación. En sus inicios, los sistemas contables, financieros y bancarios, soportaban desafíos administrativos que impedían el crecimiento y hacían que el negocio se viera acumulado de trabajo por la falta de alternativas de automatización [37]. Afortunadamente, como al resto de los avances tecnológicos que ha sufrido la sociedad desde inicios del siglo XXI, estos sistemas le dieron cabida al software, y fue así como ciertos procesos que otrora se veían irrealizables, ahora son una transacción adicional en las plataformas web o un registro más de oficina [15].

Esta adhesión tecnológica brinda muchísimas ventajas puesto que es un mecanismo que garantiza la posibilidad de incorporar todos los procesos financieros de una empresa, contribuyendo de manera exponencial no solo a la gestión automática de los procesos, sino que también genera aportes para la toma de decisiones por parte de la compañía [10]. Un beneficio adicional que tuvo la implementación del software en esta área fue el control riguroso por parte de diferentes usuarios sobre los estados financieros y en general, sobre la contabilidad de la compañía. Esta visión y manipulación particular de la información aparte de ser un instrumento que funciona en tiempo real de acuerdo con las transacciones en línea, también es una herramienta tecnológica que expone la relación numérica de la economía empresarial.

Con la entrada del software, el ámbito financiero se vio fortalecido tanto en capacidad para soportar una gran cantidad de procesos, como en la velocidad de procesamiento. Esto, porque el hecho de implementar sistemas de software en ambientes monetarios no solo aumenta la posibilidad de abarcar más procedimientos, sino que minimiza el margen de error para métodos que son dispendiosos, toman mucho tiempo o que antes solo era posible completarlos mediante la capacidad humana. Haciendo una comparación en números, la cantidad de transacciones que obtuvo la industria financiera desde la adquisición de software especializado aumentó considerablemente respecto a los procesos que se realizaban de manera mecánica [12].

Por esta razón, es que uno de los sectores que más consume software dentro del mundo actual son los sectores financieros, bancarios, económicos y contables [10], pues más allá de ofrecer una amplia gama de beneficios, es una migración que se está imponiendo en todas las industrias pues supone un avance de la tecnología y al mismo tiempo, un progreso de la sociedad [20]. En ese sentido, estas razones hacen parte de los motivos que ayudaron a desarrollar la imple-

mentación de referencia que se presentará más adelante, además de ser un reto técnico que no ha tenido el mismo análisis que otros sistemas de software.

4.1 Descripción estructural

La implementación de referencia es entonces un sistema de software con toda una estructura definida de requisitos que tiene como objetivo representar un sistema particular de préstamos financieros. El sistema cuenta con diferentes funcionalidades que están agrupadas dentro de cuatro componentes de software para el procesamiento, tratamiento y almacenamiento de los datos en la capa de lógica de negocios, cuatro componentes de software encargados del almacenamiento de la información en la capa de persistencia de datos y un único componente de software de tipo Front-End en la capa de presentación. Esta implementación se guía bajo el patrón arquitectónico de microservicios para desarrollar toda la lógica del problema, y utiliza un único componente de Front-End para la gestión de la interfaz de usuario.

4.2 Vistas Arquitectónicas

4.2.1 Vista de Capas

La aplicación desarrollada puede ser enmarcada en un diagrama de capas donde se exponen los diferentes segmentos que tiene la implementación. En esta separación que presenta la figura **4-1** se evidencia la aparición de tres capas lógicas como lo son la capa de presentación, la capa de lógica de negocios y la capa de acceso a datos. La primer capa describe la estructura de la interfaz gráfica que contiene tres subsegmentos de distribución: El segmento de vistas, que son todos los elementos que renderizan los navegadores para poder presentar la implementación al usuario, el segmento de componentes, que tiene la cualidad de agrupar las piezas lógicas de la interfaz como temas de enrutamiento o mapeo de información, y por último, el segmento de *store*, que se encarga de solicitar las peticiones a las capas inferiores a través de diferentes módulos.

La segunda capa reseña el comportamiento de los microservicios implementados que se estructuran en tres secciones: Rutas, para la gestión de las URLs de las peticiones, Controladores, para procesar y manipular los datos en la medida que se requiera, y Servicios, que se encarga de realizar las consultas de información a la capa inferior. En efecto, la última capa de la implementación es la encargada de proveer los datos solicitados por la capa superior cuando esta lo solicite.

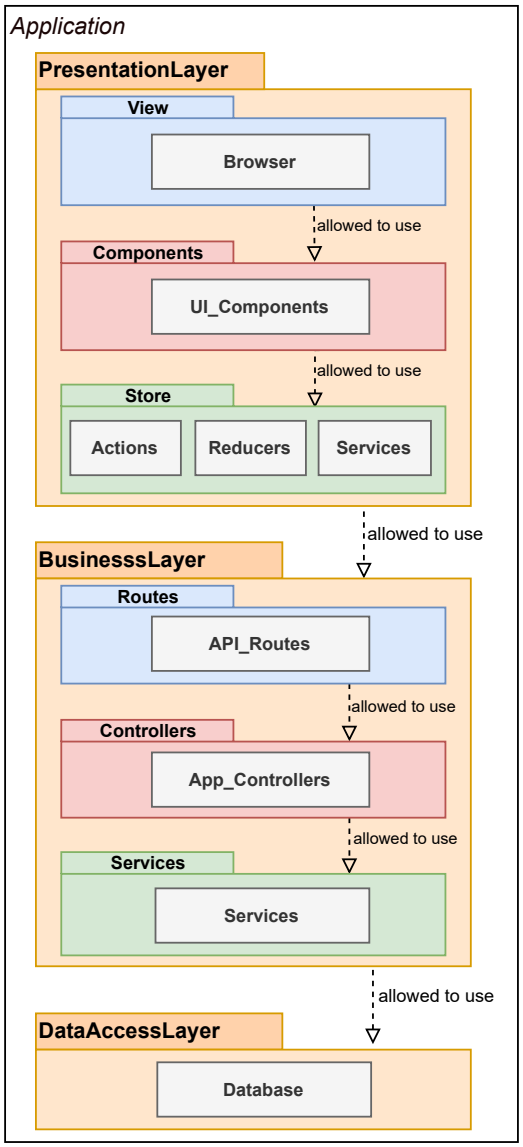

Figura 4-1: Vista de capas de la implementación de referencia.

4.2.2 Vista de Componentes y Conectores

La implementación de referencia cuenta con una vista de componentes y conectores donde se expone de manera clara cada uno de los elementos que están conformando el conjunto y las relaciones que se presentan entre cada uno de ellos. En esta vista se logra apreciar la forma en la que están distribuidos los componentes así como el patrón arquitectónico referenciado. En la parte superior del diagrama, se presenta el único componente de Front-End que es construido bajo la librería *React.JS*, y que consume los servicios que expone el componente API_Gateway.

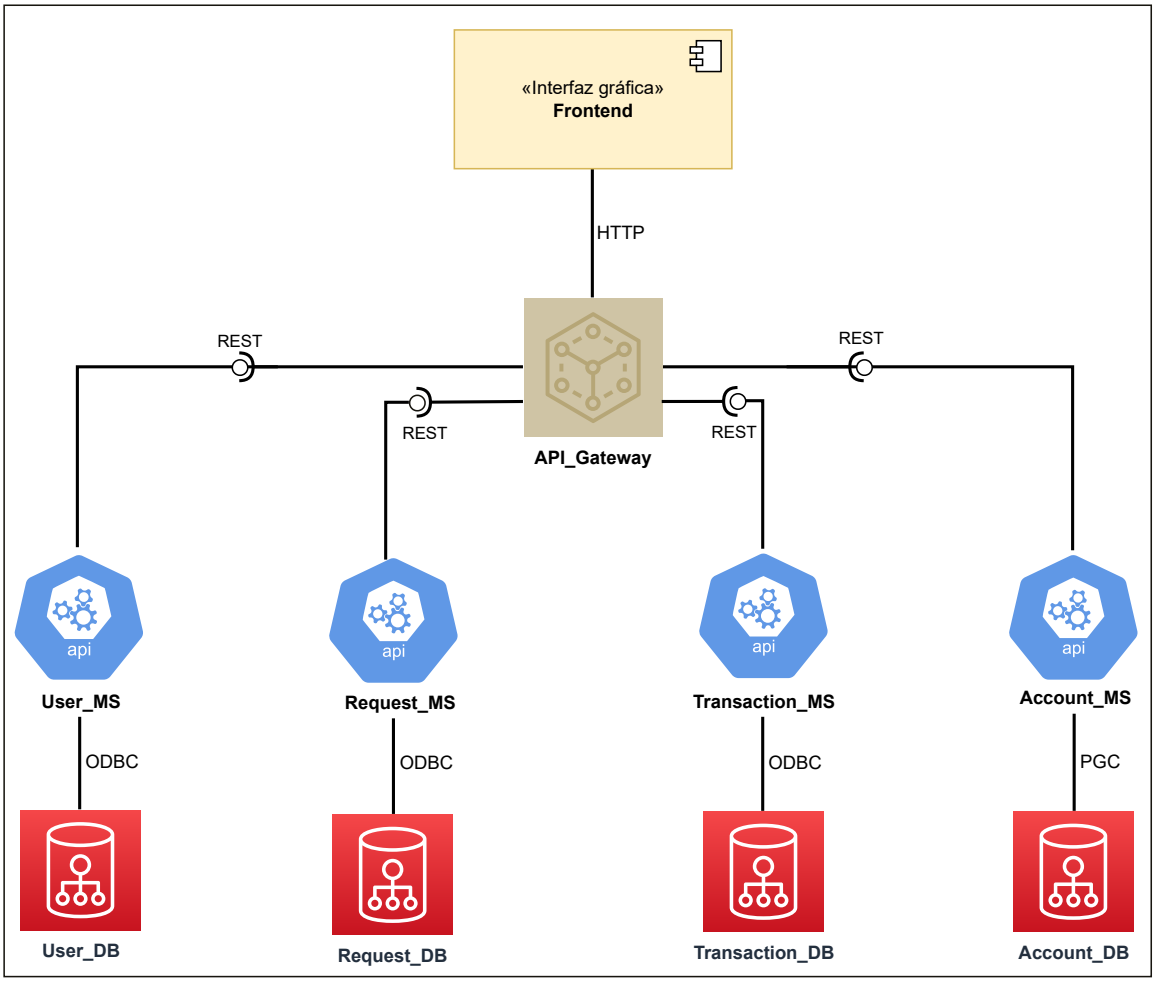

Figura 4-2: Vista de componentes y conectores.

4.2.3 Vista del Modelo de Datos

El sistema de software que se desarrolló posee bases de datos que son independientes del resto de implementaciones, y tiene una estructura relacional para cada una de sus bases de datos. Este modelo está simplificado para que todos los flujos funcionales que se definieron en las diferentes etapas de diseño cumpla por completo con las especificaciones hechas. Los modelos están separados tanto lógica como físicamente, y el resumen que se muestra a continuación solo expone algunos de los campos que se utilizan en la implementación de los motores de bases de datos.

Este sistema cuenta con un total de 18 tablas, que están repartidas dentro de los diferentes componentes de almacenamiento que se nombraron en secciones anteriores. Una visión general del modelo nos permite visualizar que en las cuatro relaciones existe al menos una relación de llave foránea para vincular los registros a partir de la cuenta o a partir del identificador único del

usuario.

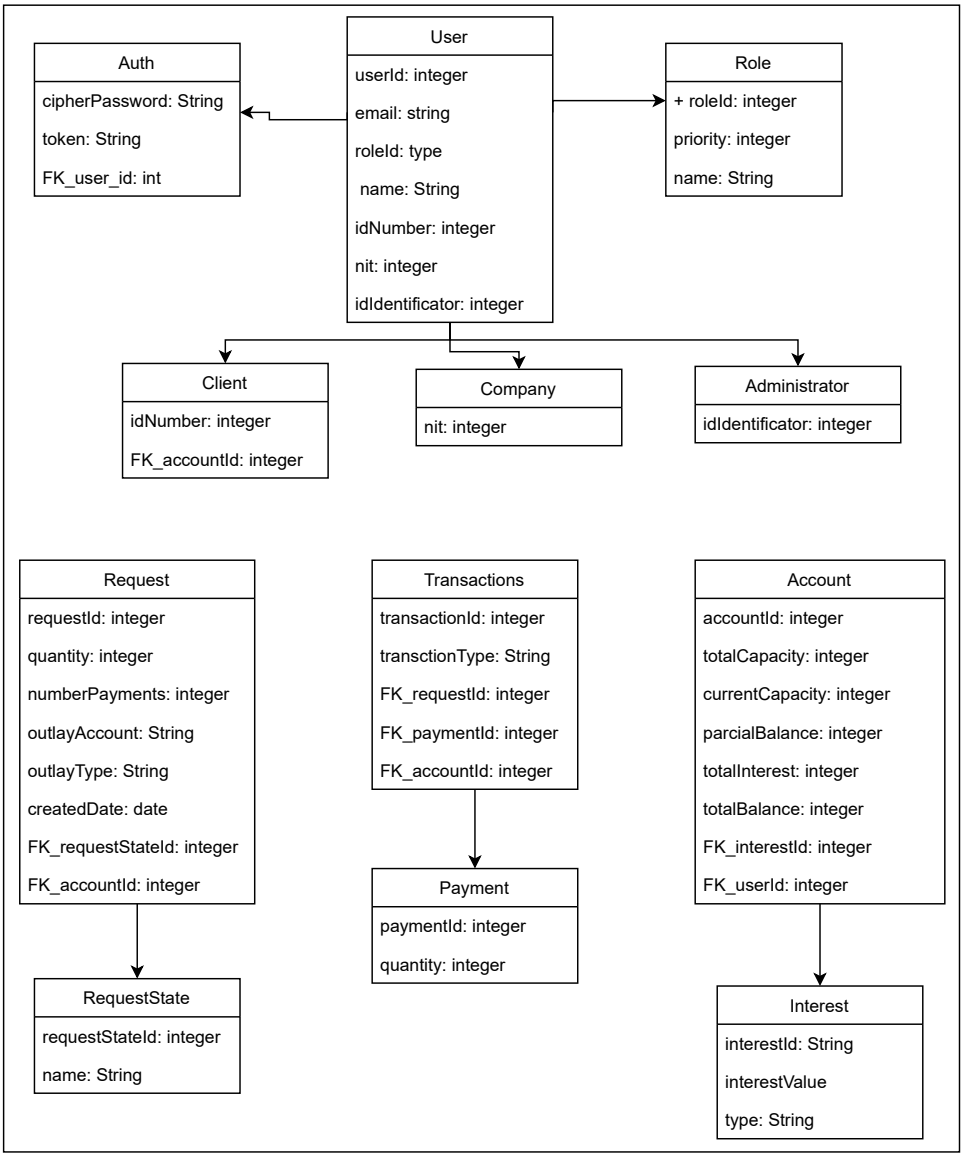

Figura 4-3: Vista parcial del modelo de datos.

4.2.4 Vista de Descomposición Funcional

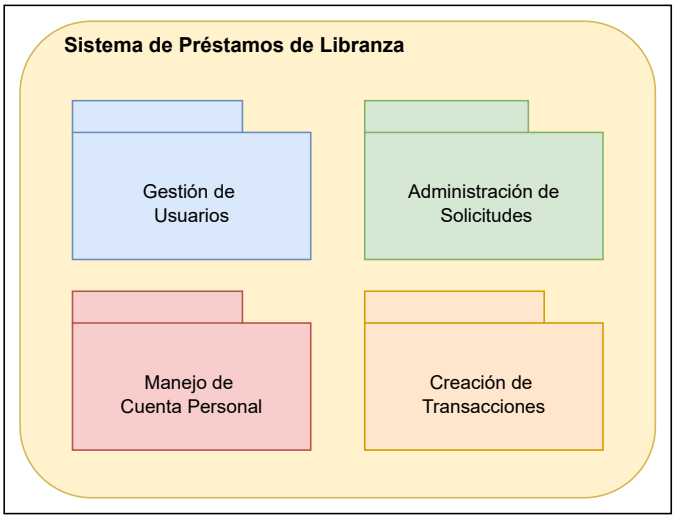

Figura 4-4: Vista de descomposición funcional.

Algunas de las descripciones de los componentes de software utilizados en la construcción del sistema de software se presentan a continuación:

a. **Registro y gestión de usuarios**: Para la gestión de los usuarios, se construyó un componente de software que tiene el propósito de crear, autenticar, actualizar, modificar y eliminar los usuarios del sistema. Este componente actúa como un microservicio dentro de la arquitectura, y está desarrollado bajo el lenguaje de *Javascript* y la librería de *Express.js*.

b. **Administración de solicitudes**: Para la gestión de las solicitudes que crean los usuarios en el sistema, se tiene un módulo particular que tiene como finalidad tratar cada una de estas solicitudes como un registro particular, y las cuales tienen todo un flujo de aprobación que pasa por diferentes usuarios. Este componente actúa como un microservicio dentro de la arquitectura del sistema, está desarrollado bajo el lenguaje de *Python* y la librería de *Flask*.

c. **Gestión de cuentas**: Otro módulo importante dentro de esta plataforma es la administración de cada una de las cuentas de los usuarios del sistema. Allí se maneja información sensible y detallada de cada usuario, donde se tiene el detalle de los balances financieros, los topes máximos a prestar y su cupo disponible actual. Este componente actúa como un microservicio dentro de la arquitectura del sistema, está desarrollado bajo el lenguaje de *Ruby* y una estructura de modelo-controlador para responder a las solicitudes.

d. **Creación de transacciones**: Finalmente, uno de los componentes que se ocupa del registro de las transacciones dentro de la aplicación es el microservicio que se construyó en

Javascript, y que tiene como objetivo recibir cuando todo un flujo de solicitud es completado para hacer la consignación dentro de los reportes del sistema para llevar una trazabilidad de los procesos que se han completado a cabalidad y sin ningún inconveniente.

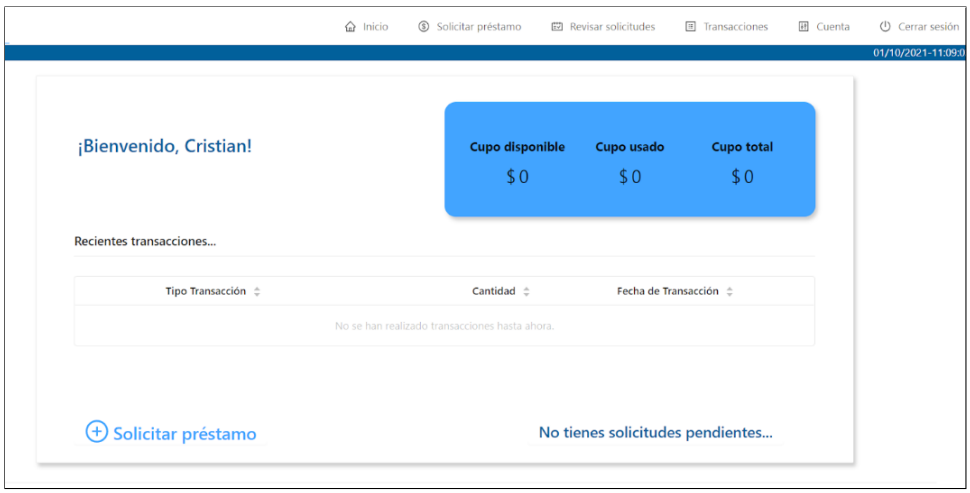

Figura 4-5: Ejemplo de una de las páginas de la implementación de referencia.

5 Modelo de descomposición de arquitectura

Una vez se han comprendido los detalles de la implementación de referencia, es posible contar con la base de funcionalidades que debe cumplir el sistema en cualquier escenario de descomposición. En ese sentido, realizar un diagnóstico de cada uno de los elementos de la implementación permite trazar la hoja de ruta que debe seguir el modelo de descomposición a proponer. Esto, con el objetivo de cumplir con los requisitos establecidos en la fase de diseño, y también, con las decisiones arquitectónicas descritas en la aplicación de referencia.

Con esta intención, se plantea un modelo de desacoplamiento de arquitectura que es capaz de soportar las mismas funciones de la implementación de referencia, y que al mismo tiempo satisface las propiedades operativas del sistema. Adicionalmente, el modelo considera el criterio de reglas que se establecieron en la fase de análisis (propiedades del patrón arquitectónico de microservicios, bases de datos independientes, implementación de un API Gateway, etc.) y realiza su aplicación en la misma medida que la capa de lógica de negocios del desarrollo original.

5.1 Estrategia de descomposición

El modelo de descomposición que se plantea se basa en los principios de la arquitectura de microservicios para realizar un desacoplamiento del componente de Front-End teniendo en cuenta que esa separación modular debe proporcionar la misma respuesta que su componente nativo [7]. Este proceso de descomposición del componente comprende la siguiente estrategia:

1. **Distinción de los dominios y requisitos funcionales de la aplicación:** Para este paso de la estrategia se identificaron cuatro grandes dominios junto con las funciones que cada dominio desempeña.
    a) **Usuarios:** Registro, Autenticación, Gestión, Aprobación, Rechazo y Soporte.
    b) **Cuentas:** Detalle, Gestión, Generación de paz y salvo, Validación Cupo.
    c) **Transacciones:** Generación, Revisión y Listado.
    d) **Solicitudes:** Creación, Validación, Generación de códigos de verificación y Aprobación o Desaprobación.

2. **Identificación de módulos:** Cumpliendo la propiedad de división por el dominio [29], se realiza la identificación de los principales módulos o "componentes" dentro de la aplicación en los cuales se podría desacoplar el sistema. Para esto se toma como referencia los dominios encontrados en el paso número 1 de la estrategia, y se realiza el análisis modular sobre cada dominio. Allí se encuentra que tanto el dominio de solicitudes como el dominio de transacciones están completamente ligados en el componente original de Front-End por lo que finalmente los módulos resultantes son los siguientes:
   a) Cuentas
   b) Solicitudes
   c) Usuarios

3. **Relación de dominios:** Este paso consiste en la valoración de la mejor técnica para relacionar e integrar los dominios. La evaluación se dio a partir de la investigación de puntos claves en el proceso de fabricación del software tales como la complejidad de desarrollo, la sincronización de componentes, la heterogeneidad tecnológica o la facilidad en el despliegue. En la siguiente tabla se presentan las ventajas y desventajas de esta elección, haciendo la selección de los Web Components como tecnología para relacionar los dominios.

Técnica	Complejidad en implementación	Coordinación y Sincronización	Heterogeneidad tecnológica	Despliegues independientes	Desarrollo en equipo	Paso y comunicación de mensajes
Hipervínculos	Media (Enrutado automático - Gestión manual de rutas)	Problemas inusuales con el caché - Demora redirección de rutas	La gestión de las rutas puede implicar una sola tecnología	✓	Coordinación de rutas y parámetros	Mínimo - Independencia máxima
IFrames	Media (Gestión del HTML - Locaciones especiales)	Sencilla - Cada iFrame funciona de manera independiente	✓	✓	Coordinación de tamaños entre equipos	Mínimo - Función postMessage limitada
Web Components - Custom Elements	Alta (Compilación de scripts - Construcción individual y desde 0)	Depende de la tecnología, puede haber demoras en la carga de los componentes	✓	A veces existe la necesidad de desplegar y testear el componente contenedor	✓	✓

Figura 5-1: Análisis de las técnicas utilizadas para integrar los diferentes dominios.

4. **Selección del tipo de enfoque utilizado para estructurar los componentes de Front-End:** En este punto, se identificó que el mejor enfoque para implementar la interfaz gráfica es la construcción de vistas dirigidas por el dominio. Esta elección se fundamentó en la idea de alinear la distinción de los dominios realizada en los pasos anteriores, junto con el diseño dirigido por el dominio, de tal manera que existiera una correspondencia entre los dominios identificados y la forma de estructurar los componentes de Front-End

emergentes. El proceso de selección está basado en el análisis de cada uno de los tipos de enfoques que existen en el proceso de codificación:

a) *Diseño dirigido por estructura atómica*: Este diseño se basa en utilizar el patrón de reusabilidad para agrupar las carpetas y archivos necesarios sobre una base de componentes pequeños y atómicos que son reutilizados en diferentes componentes a lo largo del proyecto [24]. Este enfoque no fue elegido puesto que se genera redundancia de código en varios módulos, además de producir en cierta medida acoplamiento.

b) *Diseño dirigido por usuarios*: Este tipo de enfoque se fundamenta en la consideración de los usuarios como punto de partida para la estructuración de las vistas, allí se tienen en cuenta cada una de las necesidades y posibilidades del usuario para hacer de esta exigencia el modelo a seguir durante todo el desarrollo [61]. Este diseño no fue seleccionado porque hay usuarios que comparten funcionalidades y servicios, entonces realizar una construcción independiente por cada usuario resulta totalmente innecesario y generaría mucho código duplicado por la gestión de cada rol dentro del sistema.

c) *Diseño dirigido por tipos de archivos*: La idea detrás de esta estructura es relacionar los archivos de un mismo tipo en carpetas o agrupaciones independientes con el fin de tener los archivos separados según su extensión y funcionamiento. En ese sentido, las hojas de estilo pueden estar totalmente separadas de los componentes visuales, así como los componentes utilizados para el enrutamiento [61]. La elección de esta propuesta termina siendo poco viable para el proyecto puesto que los archivos en realidad van a ser muy pocos, y tener los componentes separados de las hojas de estilo puede generar desorden y reducir el nivel de cohesión de los elementos relacionados.

d) *Diseño dirigido por el dominio*: Este diseño tiene como principio definir los dominios de manera clara y precisa antes de iniciar el proceso de escritura del código. Allí se relacionan los archivos que tienen algún vínculo por la funcionalidad que representan o por el tipo de archivo que son [7]. De esta manera, no hay forma que se genere redundancia o duplicación del código, y todos los elementos van a procurar la alta cohesión debido a que están agrupados bajo un mismo fin.

5. **Definición de la tecnología para el contenedor o *wrapper* de la aplicación:** La elección de la tecnología se basó en un análisis de las ventajas y retos técnicos que podría aportar cualquier tecnología a la implementación del modelo. Durante ese proceso se realizó la revisión de diversos aspectos de algunas tecnologías que ya existen en el mercado, y posterior a esto, se completó un proceso de discernimiento con base en cada uno de esos factores. Por esta razón, se escogió la herramienta de *Single-SPA*, que provee una gran cantidad de servicios como el enrutamiento entre componentes, la libertad de frameworks, la aplicación del patrón de carga diferida y la declaración de un contenedor guía para el resto de módulos (*API Shell*).

Tecnología	Complejidad en implementación	Coordinación y Sincronización	Heterogeneidad tecnológica	Despliegues independientes	Desarrollo en equipo	Paso y comunicación de mensajes
Single SPA	Media - La documentación puede no llegar a ser suficiente	Carga de scripts de los microfrontends contenidos	Independencia Tecnológica (React - Angular - Vue – Svelte)	Independiente para cada componente emergente	Existe la posibilidad de trabajar de manera independiente dejando claros los protocolos de comunicación y los segmentos de las vistas a utilizar	Comunicación a través de props en los componentes o mediante el manejo de eventos
BIT	Media, aunque la documentación y los foros de análisis son amplios	Carga a través de scripts	Independencia Tecnológica (React - Angular - Vue – Vanilla Javascript - TypeScript)	Componentes que pueden ser desplegados de forma independiente	Los equipos pueden incorporar cambios de manera aislada	Props – Gestión de Eventos
Module Federation	Muy alta, la gestión de los paquetes es técnicamente manual	Carga de scripts de los componentes contenidos	Cualquier tecnología puede ser utilizada	Los despliegues son independientes, pero hay que construir el archivo de despliegue	Gestión individual de cada equipo	Existe un host para la distribución de nodos y mensajes
Luigi	Alta, por la gestión del Luigi Core	Carga a través de scripts	Independencia Tecnológica (React - Angular - Vue – OpenUI5)	No hay información relacionada	Es independiente, pero está atado al manejo del elemento core	Mensajes personalizados – Props entre componentes – Nodos de navegación

Figura 5-2: Análisis de las tecnologías para el contenedor de la aplicación.

6. **Aplicación de la descomposición:** De acuerdo con las elecciones tomadas en los numerales anteriores referentes a la tecnología, la técnica de integración, y los módulos identificados, se realiza la implementación con estos insumos conservando las consideraciones funcionales y arquitectónicas del sistema.

5.2 Modelo resultante

Luego de la aplicación de la estrategia sugerida, se obtiene el esquema presentado en la figura **5-3** que reúne cada uno de los dominios del componente de Front-End inicial, y que también cumple con cada uno de los pasos que se mencionan en la estrategia de descomposición. La implementación del modelo se realiza mediante la librería encontrada en la investigación, **Single-SPA**, que permite no solo ser la tecnología propia del contenedor de la aplicación, sino que también brinda los mecanismos necesarios para construir los *custom-elements* que soportan las diferentes funcionalidades de la interfaz visual.

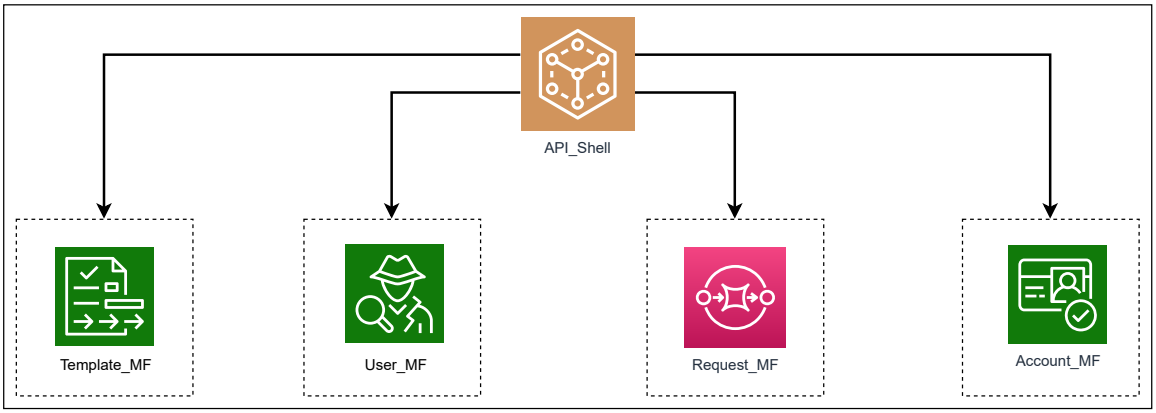

Figura 5-3: Representación del modelo de descomposición para el componente de Front-End.

A partir del modelo de descomposición anterior, se identifican 5 módulos que agrupan la finalidad del componente original:

a. **API_Shell:** Es el componente encargado de orquestar cada uno de los módulos resultantes. Actúa como elemento conector dentro de la aplicación y es la unidad de servicio que define la forma en la que se cargan cada uno de los módulos de acuerdo con la vista requerida por el usuario.

b. **Template_MF:** Este módulo se encarga de renderizar la estructura inicial que se muestra una vez el cliente solicita la interfaz. Aquí se exponen las funcionalidades del log-in, registro de usuarios y la barra de navegación compartida.

c. **User_MF:** Este componente tiene como objetivo mostrar todo el módulo de gestión de usuarios. Dentro de este elemento se exponen las pestañas de inicio, gestión de empresas y gestión de clientes.

d. **Request_MF:** Este módulo se encarga de presentar la interfaz relacionada con las solicitudes de los clientes, tanto para los usuarios que crean la solicitud como para las personas que realizan su aprobación. Adicionalmente, contiene las vistas diseñadas para el dominio de transacciones puesto que tienen una relación estrecha con las solicitudes, y no es pertinente llevar a la práctica una separación con elementos directamente vinculados.

e. **Account_MF:** Esta composición tiene como propósito representar las vistas vinculadas al dominio de Cuentas, donde se describe el detalle de los usuarios y se realiza la generación del paz y salvo de deudas.

5.3 Integración del modelo resultante

Posterior a realizar el proceso de desacoplamiento del componente, se lleva a cabo la integración del modelo de descomposición con el resto de piezas del sistema. Por consiguiente, resulta

oportuno destacar el planteamiento de tres escenarios que recopilan diferentes características arquitectónicas, lo cual garantiza que el modelo final cuente con particularidades que quizás en otro caso no son concebidas.

5.3.1 Formulación de escenarios arquitectónicos

La formulación de diferentes escenarios que permiten integrar el modelo de descomposición nace como respuesta a la gran variedad de alternativas que las arquitecturas distribuidas soportan [70]. Así pues, esta consideración permite tener una amplia gama de alternativas estructurales, y asegura un análisis más tangible de los resultados debido a que se contemplan todos los entornos en los que este problema se desarrolla. En estos esquemas la gestión del enrutamiento en la interfaz gráfica y el proceso de renderización de las vistas siempre es administrado por el componente *API Shell*. Para todos los escenarios, la comunicación entre un microservicio y cualquier elemento se da a partir del uso del estilo arquitectónico REST, la conexión entre un microfrontend y otra pieza del sistema se realiza a través del protocolo HTTP y finalmente, el enlace entre los microservicios y las bases de datos se da mediante la interfaz PGC para los almacenamientos que usan PostgreSQL y ODBC para tecnologías diferentes. En seguida se exponen los tres escenarios:

5.3.1.1 Relación Microservicio-MicroFrontend

En este escenario los microfrontends que resultaron del proceso de descomposición se conectan directamente con el microservicio de su propio dominio en una relación uno a uno, exceptuando el *Template_MF*, que tiene una connotación diferente. En este diseño se destaca la ausencia del *API_Gateway* ya que el consumo de los servicios es manejado de primera mano por el microfrontend. Para este supuesto, el microfrontend de solicitudes tiene una carga mayor de procesamiento de información porque debe gestionar las peticiones de dos microservicios.

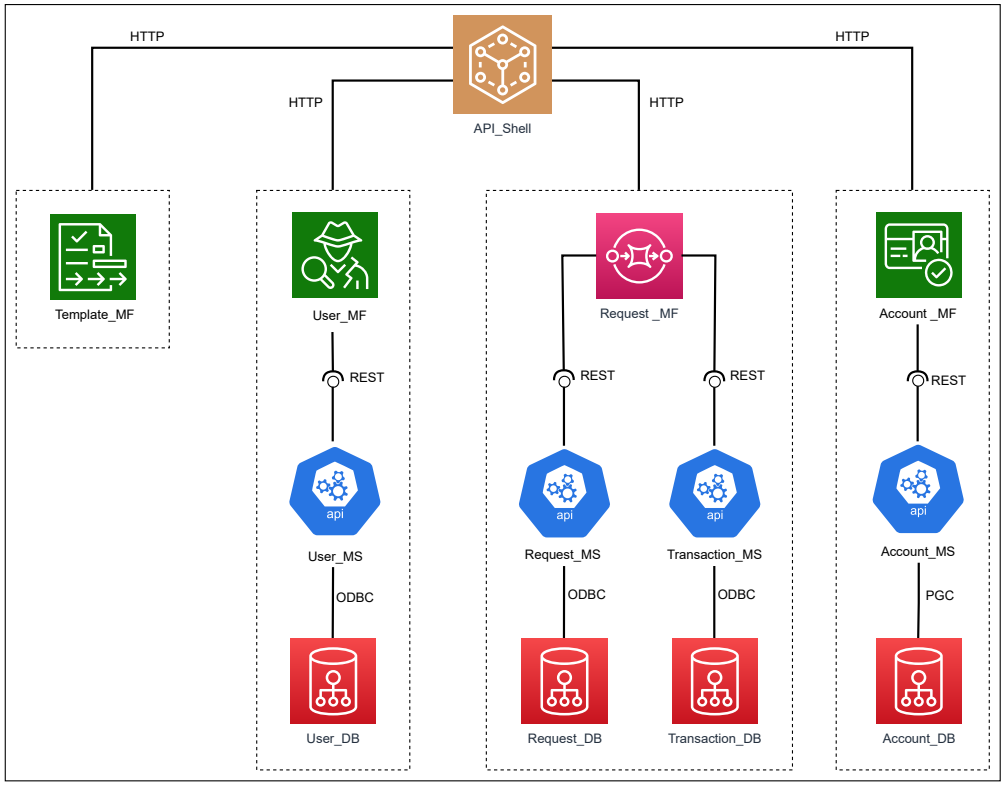

Figura 5-4: Escenario arquitectónico que representa la relación MicroFrontend-Microservicio.

5.3.1.2 Relación API Gateway-API Shell

En este esquema se logra ver una distinción clara entre la capa de presentación y la capa de lógica de negocios. Además, se puede destacar la presencia de dos servicios que gestionan la interacción entre los componentes como lo son el *API Shell* y el *API Gateway*. Aquí, todas las comunicaciones pasan de manera obligatoria por estos canales porque son el único punto de acceso entre una capa y la otra, lo que puede repercutir en puntos adicionales de fallo por tener puertas de enlace únicas.

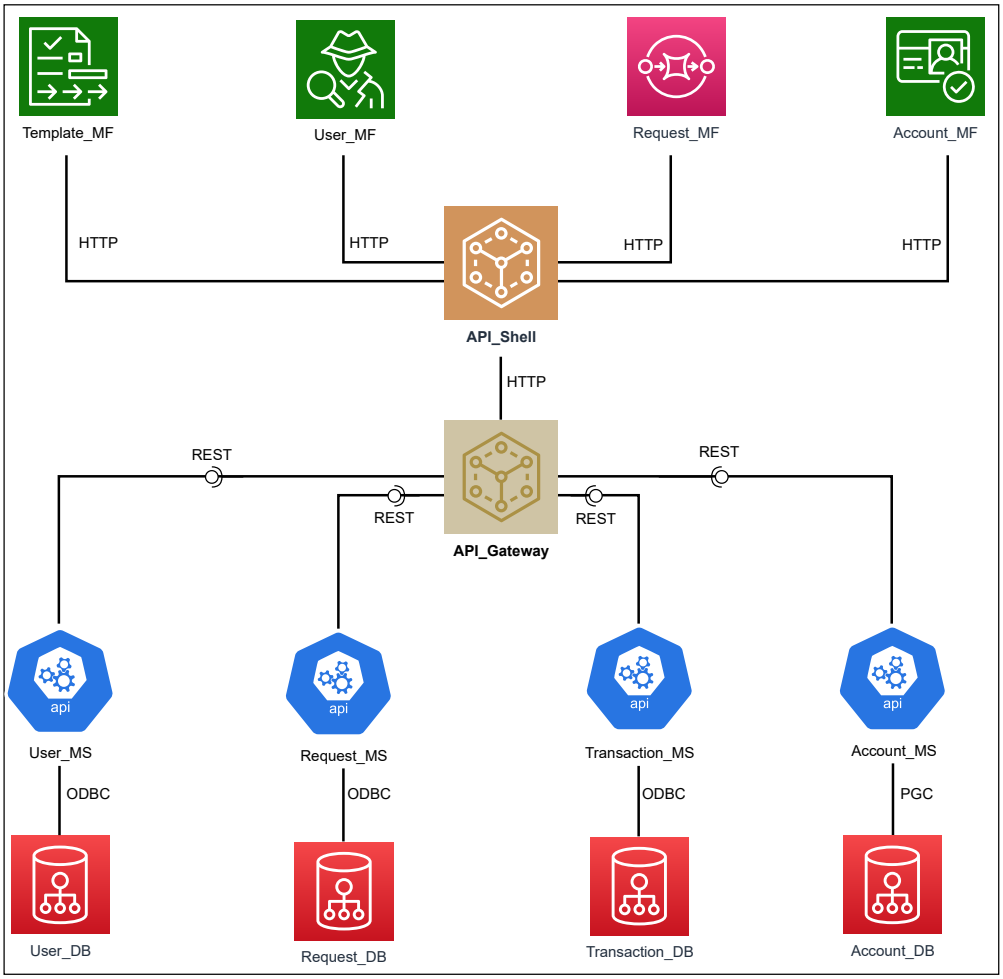

Figura 5-5: Escenario arquitectónico que representa la relación API Gateway-API Shell.

5.3.1.3 Relación Microservicio-API Shell

El tercer escenario tiene la particularidad de que no utiliza uno de los componentes de la implementación original como lo es el *API Gateway*, puesto que parte de la lógica que desarrolla este componente es llevada al *API Shell*. La carga de trabajo del *API Shell* es bastante porque administra el contenido tanto de la capa de presentación como de la capa de lógica de negocios. En este marco cada uno de los microservicios tienen conexión directa con el *API Shell*, quien a su vez es el encargado de consumir cada una de las funciones que estos elementos proveen. Tal como los literales anteriores, es el contenedor de la aplicación el que se hace cargo de la gestión de las vistas según el requerimiento del usuario.

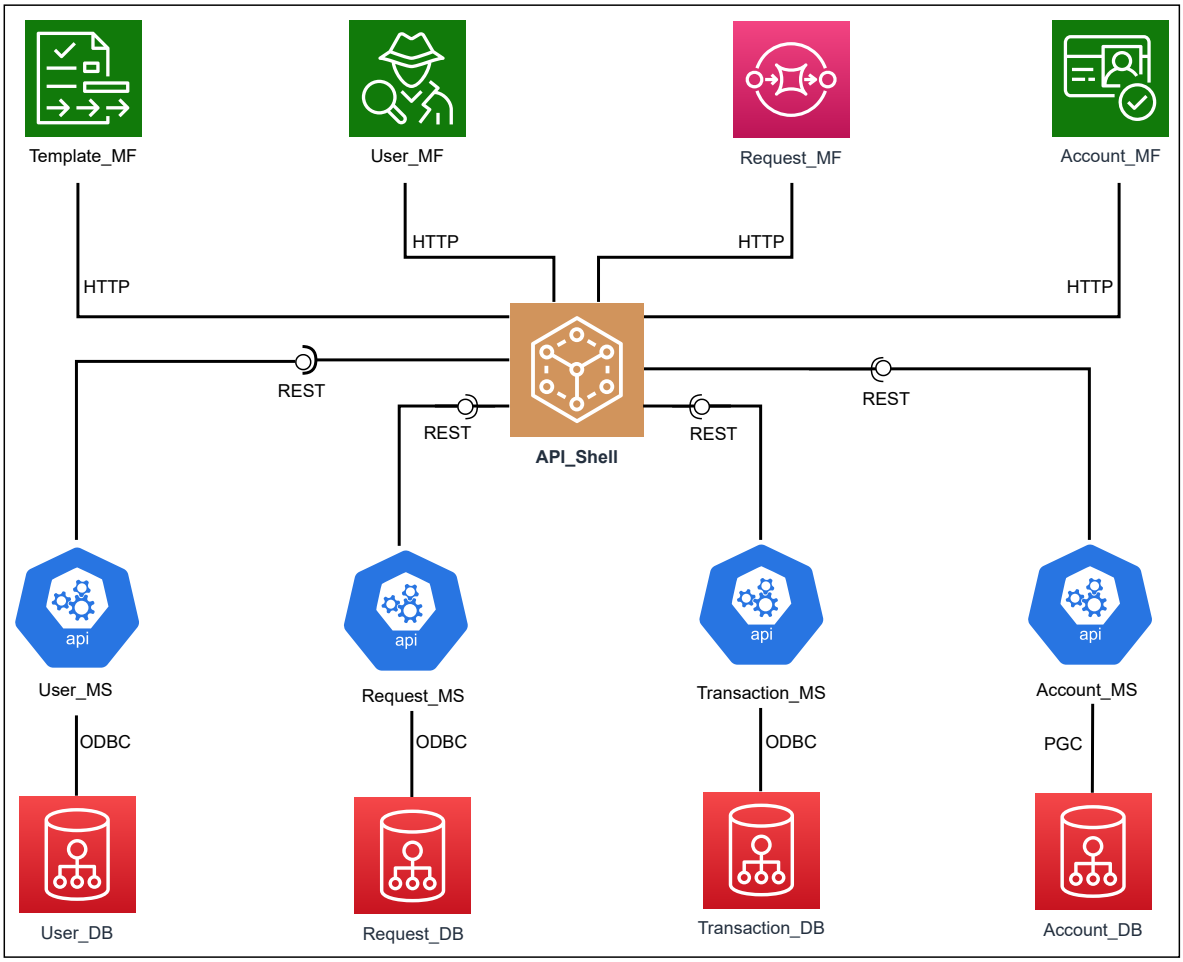

Figura 5-6: Escenario arquitectónico que representa la relación Microservicio-API Shell.

5.3.2 Definición del escenario final

Después de que han sido contemplados los escenarios anteriores y luego de realizar un análisis exhaustivo de los diferentes comportamientos que puede tener la aplicación, se define un cuarto escenario que reúne todos los elementos que puede contener el sistema junto con todas sus posibles conexiones. Un resumen gráfico de la identificación de conexiones que se lograron obtener de los modelos anteriores entre los componentes que actúan como consumidor de un servicio y los componentes proveedores de ese servicio es presentado en la siguiente imagen:

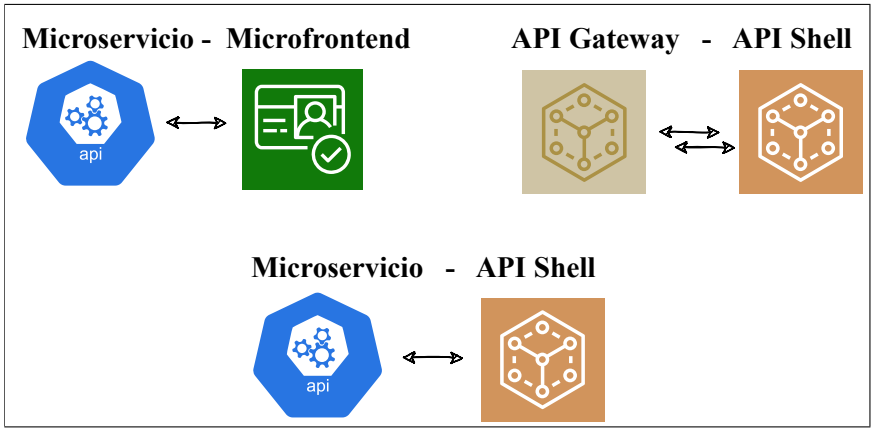

Figura 5-7: Posibles integraciones con base en los escenarios arquitectónicos planteados.

Adicionalmente, la idea detrás de esta evaluación es establecer un modelo que recopile las potenciales interacciones entre los diferentes módulos sin que se afecte alguna operación del sistema cumpliendo con todos los requisitos funcionales especificados en la implementación arquitectónica original. Algunas de las relaciones encontradas entre las piezas del sistema son:

1. Componente de microfrontend y un microservicio
2. Componente de microfrontend y el *API Gateway*
3. Componente de microfrontend y el *API Shell*
4. Componente *API Shell* y el componente *API Gateway*
5. Componente *API Gateway* y un microservicio
6. Componente de base de datos y un microservicio

En consecuencia, surge el modelo que se presenta en la figura **5-8** que considera cada una de las particularidades de la implementación de referencia y que termina siendo el diseño que se utiliza de manera definitiva para realizar el proceso de desarrollo e integración. Un resultado que vale la pena destacar de este proceso de análisis es la existencia obligatoria del *API Shell* en la capa de presentación del sistema de software. Este compromiso funcional se presenta debido a que debe existir un componente que integre de manera uniforme cada uno de los componentes de Front-End emergentes, y gestione temas como el enrutamiento o la sincronización de las vistas cuando estas se tienen que renderizar.

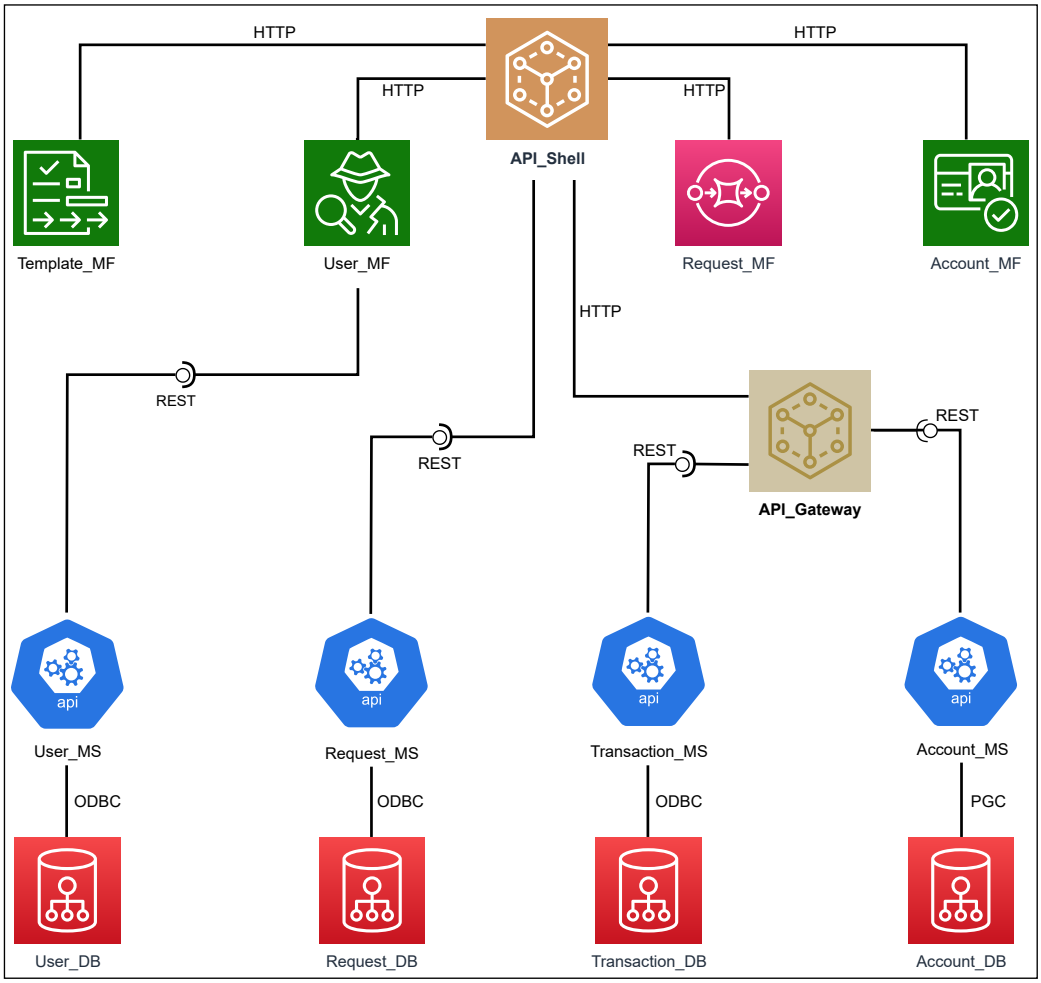

Figura 5-8: Representación de la integración del modelo de descomposición y la implementación de referencia.

6 Evaluación del modelo

Con el objetivo de realizar un análisis de la estrategia de descomposición propuesta, y llevar a cabo un proceso de evaluación del modelo de desacoplamiento resultante, se implementa una métrica que permite estimar la cohesión de los componentes del sistema resultante a través de un estudio cuantitativo del flujo de datos entre componentes. Esta métrica hace parte de un gran conjunto de apreciaciones en el mundo del software que calculan la relación entre un elemento y otro a partir de un concepto clave como lo es la cohesión [8][38]. Por lo habitual, estas mediciones tienden a estar más relacionadas con propiedades exclusivas de la programación orientada a objetos como las clases o los métodos [34], pero su perspectiva de evaluación se puede simular en otros contextos donde exista un intercambio de información entre diferentes componentes de software.

De forma análoga, el propósito de efectuar este tipo de estudio sobre el esquema resultante es dar una perspectiva tangible de las decisiones de diseño implementadas en el sistema, ya que esta revisión brinda parámetros de comparación que pueden servir para posteriores análisis [35] y en definitiva, pueden proporcionar una base referencial a tener en cuenta cuando se trata de descomposición de componentes Front-End de tipo web. En ese sentido, la evaluación del modelo comprende cuatro pasos principales: La comprensión de una estrategia de evaluación, la definición de un flujo de información útil para el proceso, la especificación de las consideraciones a tener en cuenta para aplicar la estrategia, y el cálculo de los valores a partir de la relación de los elementos del sistema.

6.1 Métrica implementada

La inclusión de métricas para la cohesión dentro de diferentes arquitecturas de software concede la posibilidad de diagnosticar propiedades estructurales del sistema en la medida en que se analiza la interacción entre una pieza y la otra [60]. Por esta razón, la métrica utilizada en esta sección se basa directamente en la estrategia de estimación que desarrolló Mahdi Saadati en su proyecto orientado a la medición de cohesión y acoplamiento en sistemas de software que aplican el paradigma de la programación orientada a objetos (POO) [60].

En esta aproximación, además de considerar dos principios fundamentales que procuran la calidad del software desde el ámbito de la ingeniería de software, también toma en consideración

elementos relacionados con el desarrollo de software basado en la programación orientada a objetos. En ese sentido, la ejecución de la métrica sugerida comprende elementos importantes del software como lo son los "*datos de entrada*", y el intercambio de estos datos sobre diversos elementos del software tal como los *métodos públicos* de una clase. Ese tipo de intercambio de datos entre elementos de software no solo brinda la posibilidad de considerar piezas de software vinculadas con el paradigma orientado a objetos, sino que también expande el panorama para analizar otros tipos de componentes en los sistemas.

Por esta razón, se puede generar una analogía entre el paso de información por un componente de software y el atributo de una clase, ya que ambos elementos son datos de entrada con una estructura particular y distintos propósitos, pero al mismo tiempo comparten un conector, un canal o un llamado a un método por el cual se transportan. Por otro lado, los atributos al ser propiedades intrínsecas de las clases, y al ser gestionados a través de funciones, puntualmente por los métodos públicos, permiten diagramar una relación entre los flujos de información que se transmiten entre los diferentes elementos del sistema, que en este caso puntual son los componentes de software.

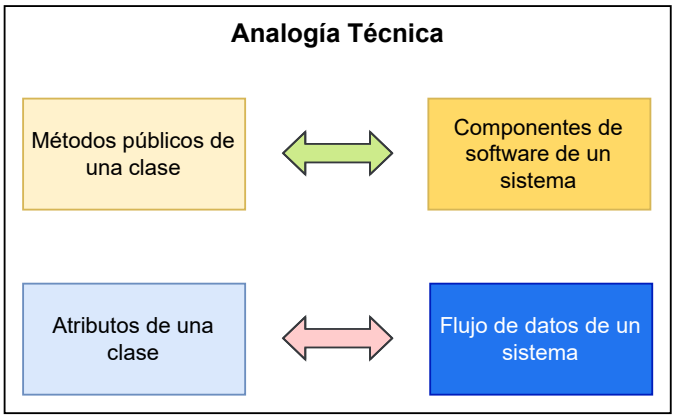

Figura 6-1: Analogía entre el diseño detallado y el diseño arquitectónico.

En definitiva, se realiza una transición entre el diseño detallado que está relacionado con las clases y el diseño arquitectónico que se vincula con los componentes y conectores para que así el criterio de evaluación puede migrarse al modelo de descomposición resultante, pues también hay una contribución similar entre los componentes de software que conforman el sistema y el paso de información que estos servicios proveen.

6.2 Selección del flujo funcional de información

Ahora que la métrica de evaluación está especificada y hay un trasfondo teórico que la respalda, es fundamental contar con un flujo de información que cubra gran parte de la operación funcional

del sistema vinculando la mayor cantidad de componentes posible. Por este motivo, se plantean tres flujos funcionales de la aplicación que abarcan la mayoría de procesos que se desarrollaron en la implementación de referencia. En esta formulación, cada flujo considera al menos siete de los elementos del modelo resultante de descomposición. Los flujos de información encontrados son:

1. **Creación de una solicitud por parte de un cliente nuevo:** Este flujo describe el proceso que debe seguir un cliente una vez ingresa a la plataforma y no tienen ningún registro anterior. Una vez son enviados los datos necesarios para llevar a cabo su creación, el usuario con el rol de administrador es quien realiza su aprobación de acuerdo con la empresa a la que pertenece y el monto predefinido para esa empresa. Luego de esto, el usuario está habilitado para efectuar la generación de la solicitud diligenciando los campos necesarios e ingresando los códigos de validación. Posterior a la validación del proceso, el sistema despliega un mensaje de aviso confirmando la creación.

2. **Rechazo de la solicitud por parte de un administrador:** Esta funcionalidad detalla los pasos necesarios para rechazar una solicitud previamente creada por un cliente desde un usuario con el rol de administrador. Inicialmente, el administrador debe ingresar por la vista de Log-in para suministrar los datos de autenticación, luego debe dirigirse a la pestaña de solicitudes donde se despliega la lista de solicitudes por aprobar. Seguidamente debe desplegar la solicitud del usuario particular que se quiere calificar y escoger un motivo de negación para esa función. El flujo termina cuando el sistema genera una actualización en el registro de cuentas y arroja un mensaje validando el rechazo.

3. **Generación de paz y salvo por el pago de una solicitud con sesión activa:** Esta operación tiene el propósito de generar un certificado de libertad acerca las obligaciones financieras que han sido saldadas por parte del usuario dentro la plataforma. Para ejecutar este método el usuario debe haber recibido un mensaje de confirmación donde se ratifica el desembolso por parte del sistema en su perfil y seguidamente dirigirse a la pestaña de cuentas en la barra de navegación. Cuando el usuario se encuentra en este módulo, la aplicación solicita la información correspondiente al detalle de la cuenta y pide la validación de los datos por parte del usuario. Si los valores expuestos son corroborados por el cliente, el usuario puede generar el reporte descargando el archivo en el botón de generar paz y salvo.

Después de haber analizado cada uno de los flujos de información, se toma como referencia el flujo que más componentes de software utiliza para completar una funcionalidad. Es así como el flujo de creación de una solicitud de préstamos por parte de clientes nuevos es escogido como la operación que recopila la mayor cantidad de interacciones en el sistema.

6.3 Diagrama de estados del flujo seleccionado

Este diagrama representa el flujo escogido anteriormente mediante un diagrama de estados donde se enumeran cada una de las actividades del sistema que permiten desempeñar el proceso. La descripción detallada de las actividades es reflejada en la tabla contigua al diagrama. Vale destacar que son expuestos mediante círculos de inicio los roles de usuario y de administrador, que intervienen tanto para el registro como la aprobación del usuario.

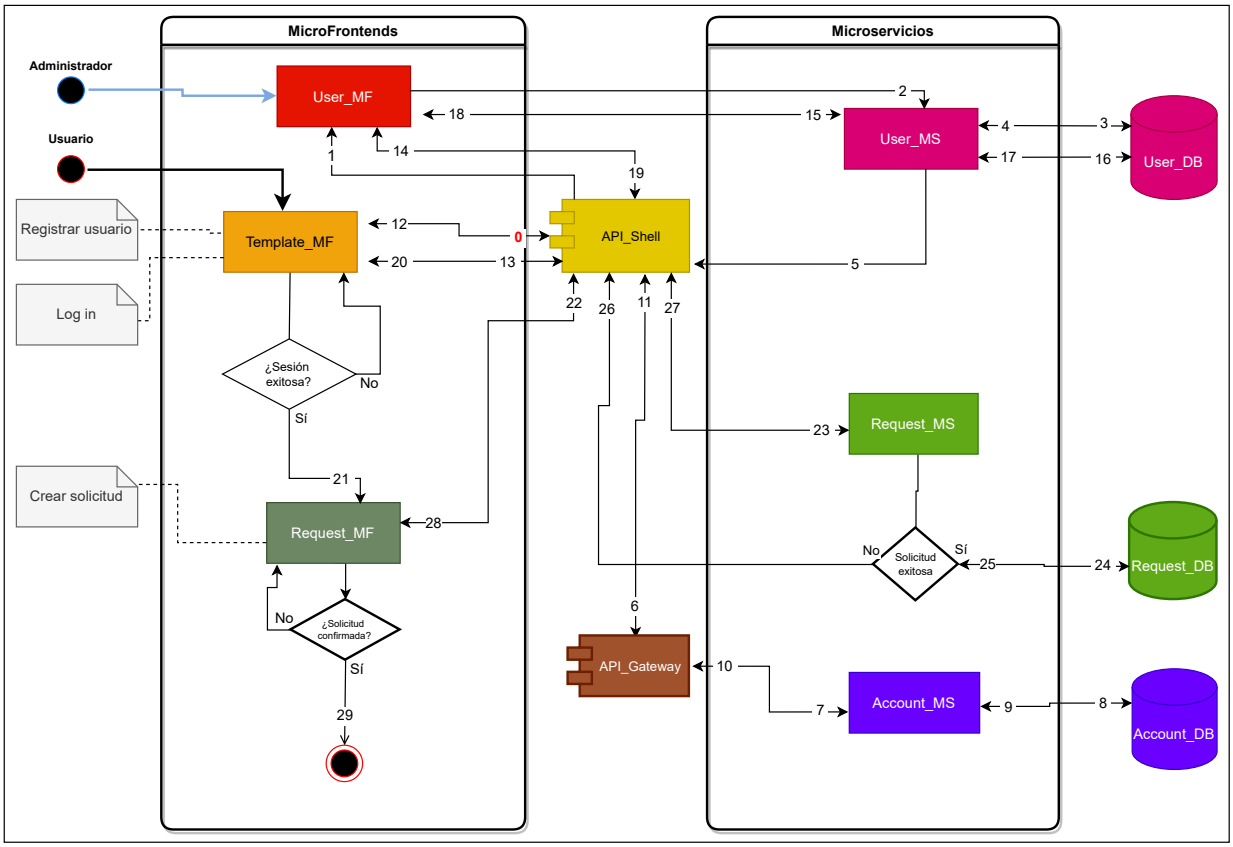

Figura 6-2: Diagrama de estado para la creación de solicitudes.

Flujos
0. Envío de registro de nuevo usuario
1. Enrutamiento de registro de nuevo usuario mediante API Shell
2. Redirección del servicio de nuevo registro
3. Guardar usuario nuevo en la base de datos
4. Respuesta exitosa de la base de datos UserDatabase
5. Envío de solicitud para la creación de cuenta
6. Comunicación con el API Gateway para la creación de cuenta
7. Solicitud del servicio de creación de cuenta
8. Registro de la cuenta en la base de datos
9. Respuesta exitosa de la base de datos UserDatabase
10. Respuesta exitosa del servicio de Cuenta
11. Envío de respuesta exitosa al API Shell
12. Envío de respuesta sobre el registro de usuario
13. Inicio de sesión de nuevo usuario
14. Enrutamiento de inicio de sesión mediante API Shell
15. Redirección del servicio de inicio de sesión
16. Consulta del usuario en la base de datos User Database
17. Respuesta exitosa de la base de datos User Database
18. Envío de token e información al usuario logeado.
19. Envío de respuesta al API Shell.
20. Envío de respuesta sobre inicio de sesión.
21. Redirección al módulo de solicitudes
22. Envío de información para la creación de la solitud
23. Redirección al servicio de creación de solicitudes
24. Registro en la base de datos Request Database
25. Envío de respuesta por parte de la base de datos de Request Database
26. Envío de respuesta negativa por la creación de solicitud
27. Envío de respuesta exitosa por la creación de solicitud
28. Envío de respuesta sobre la creación de la solicitud
29. Finalización del proceso

Figura 6-3: Tabla de actividades del diagrama de estado.

6.4 Consideraciones previas

En este punto es necesario tener en cuenta que hay dos elementos clave que se deben contrastar. Por un lado, los métodos públicos de las clases, que son identificados en el estudio como funciones, ya que son los encargados de retornar un resultado luego de ser suministrados unos datos de entrada, y que, para el caso del modelo de descomposición, representan a los componentes en vista de que tramitan las peticiones entrantes. Los componentes que se utilizan en la estrategia fueron los resultantes de la integración entre el modelo de descomposición y la implementación de referencia.

Por otro lado, los atributos pertenecientes a las clases que tienen la cualidad de representar las características de los objetos y que pueden ser manipulados por agentes externos de la clase, se pueden comparar en esta revisión con todos aquellos datos o contenido que transita de un componente a otro. De acuerdo con lo anterior, es relevante mencionar que la cantidad de datos que se manejan en la implementación es significativa por lo cual se consolidan ciertos grupos de datos con el mismo dominio para ejecutar el planteamiento. Estos conjuntos son:

1. *AuthenticationData:* Hace referencia a los datos de autenticación de la aplicación, desde el proceso de ingreso hasta su respuesta.

2. *NewRegisterData:* Este conjunto contempla todos los datos de registro cuando el usuario desea inscribirse en la aplicación.

3. *RequestBodyData:* Este grupo hace mención a todo el conjunto de datos que interviene en el proceso de construcción de una solicitud.

Por último, es crucial para el proceso de evaluación establecer un rango de valores que permita un análisis más estricto de los resultados por lo que se toma como medida la misma escala utilizada en la métrica de 0 a 1.

6.5 Aplicación de la estrategia

Los pasos que se describen a continuación reflejan de manera ilustrativa la estrategia implementada para la evaluación.

1. **Identificación de las relaciones entre los componentes y los datos:** La idea de este punto es diagramar las relaciones entre los componentes y los conjuntos de datos. La ubicación en la parte central de la gráfica corresponde a los grupos de datos y cada uno de los componentes del modelo resultante están situados en los extremos superiores e inferiores.

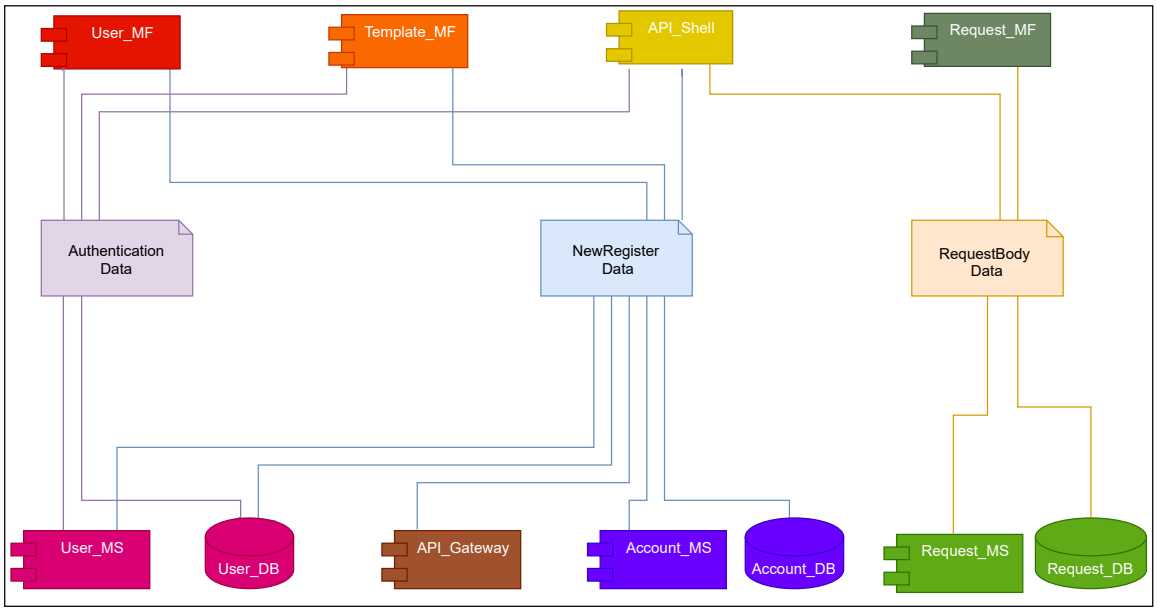

Figura 6-4: Identificación de relaciones entre componentes y datos.

2. **Definición de los diferentes grupos en base a los componentes:** En este paso se identifican cada uno de los grupos que se forman alrededor de un conjunto específico de datos, es decir, todos los componentes de software que están vinculados con un flujo de información en particular. Los grupos encontrados son:

   G1 = {User_MF, Template_MF, API_Shell, User_MS, User_DB}

   G2 = {User_MF, Template_MF, API_Shell, User_MS, User_DB, API_Gateway , Account_MS, Account_DB}

   G3 = {API_Shell, Request_MF, Request_MS, Request_DB}

3. **Cálculo decimal de la relación entre los componentes y la cantidad de grupos encontrados:** Cuando los grupos de vínculo son definidos, se procede a realizar la división entre el número de veces que un componente está incluido dentro de un conjunto y el número total de grupos identificados, que para este caso es 3.

   i. **R(User_MF)** = $\frac{2}{3} = 0.66$

   ii. **R(Template_MF)** = $\frac{2}{3} = 0.66$

   iii. **R(API_Shell)** = $\frac{3}{3} = 1$

   iv. **R(User_MS)** = $\frac{2}{3} = 0.66$

   v. **R(User_DB)** = $\frac{2}{3} = 0.66$

   vi. **R(API_Gateway)** = $\frac{1}{3} = 0.33$

   vii. **R(Account_MS)** = $\frac{1}{3} = 0.33$

   viii. **R(Account_DB)** = $\frac{1}{3} = 0.33$

   ix. **R(Request_MF)** = $\frac{1}{3} = 0.33$

   x. **R(Request_DB)** = $\frac{1}{3} = 0.33$

   xi. **R(Request_MS)** = $\frac{1}{3} = 0.33$

4. **Estimación parcial de la cohesión de la clase con los valores de relación encontrados:** Con los cálculos encontrados en el paso anterior, se calcula la cohesión parcial del flujo de información haciendo una sumatoria de las relaciones en general y dividiendo ese resultado en el número total de grupos. Este resultado no tiene en cuenta los pesos relativos de los componentes sobre los grupos encontrados ni sobre el flujo en general.

   Cohesión Parcial = $\frac{Sumatoria\ Relaciones}{Total\ de\ componentes\ identificados}$

   Cohesión Parcial = $\frac{0.66 + 0.66 + 1 + 0.66 + 0.66 + 0.33 + 0.33 + 0.33 + 0.33 + 0.33 + 0.33}{11}$

   Cohesión Parcial = 0.51

5. **Identificación de subconjuntos según la construcción de grupos en el primer numeral:** En este punto de la métrica, se logra identificar 1 subconjunto de los 3 grupos encontrados inicialmente. Este subconjunto e el *grupo 1 (G1)*, y termina siendo subconjunto porque tiene todos los elementos que el conjunto original.

Conjunto (G2) = {User_MF, Template_MF, API_Shell, User_MS, User_DB, API_Gateway, Account_MS, Account_DB}

i. **Subconjunto (G1)** = {User_MF, Template_MF, API_Shell, User_MS, User_DB}

6. **Construcción del árbol de relaciones en base a los conjuntos y subconjuntos:** Después de encontrar los subconjuntos de relaciones entre los componentes de software, se lleva a cabo la construcción de un árbol de subconjuntos que permite evaluar interacciones más profundas de la arquitectura. Para este diagrama, el conjunto principal es ubicado en la raíz del árbol, y los subconjuntos asociados se ubican en los siguientes niveles del árbol. Adicionalmente, el vínculo jerárquico entre los elementos del organigrama se establece a partir de los pesos que cada conjunto tiene. El cálculo de los pesos se rige por la fórmula que se presenta a continuación:

$$\textbf{Peso del grupo G} = \frac{Número\ de\ componentes\ en\ el\ grupo}{Número\ total\ de\ componentes\ en\ el\ flujo}$$

En ese sentido, la evaluación de los pesos de cada uno de los grupos descritos en el primer paso de esta métrica resulta de la siguiente forma:

i. **Peso (G1)** = $\frac{5}{11} = 0.45$

ii. **Peso (G2)** = $\frac{8}{11} = 0.72$

iii. **Peso (G3)** = $\frac{4}{11} = 0.36$

Según los valores anteriores, es posible entonces construir el árbol con cada una de las especificaciones mencionadas previamente. Este proceso se realiza con el propósito de identificar la contribución que tiene los componentes sobre el flujo que está evaluando.

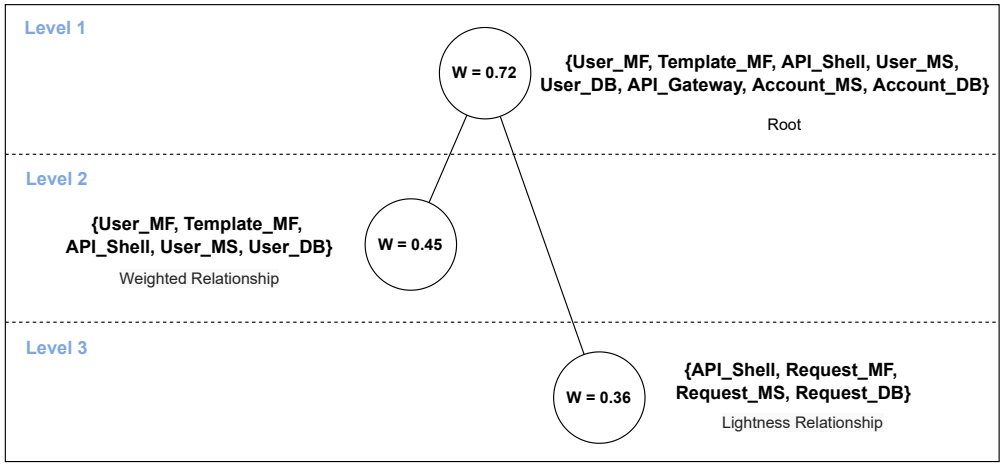

Figura 6-5: Árbol de subconjuntos con los pesos correspondientes.

7. **Cálculo del peso de cada componente sobre el flujo:** Tras realizar el proceso de estimación de los pesos relativos de cada conjunto, y su contribución funcional al flujo de información, es posible definir los pesos específicos de cada componente. Este proceso se efectúa sumando las contribuciones relativas de los conjuntos a los que pertenece el componente. Por lo cual, las aproximaciones del peso son las correspondientes:

   i. **S (User_MF)** = 0.72 + 0.45 = 1.17
   ii. **S (Template_MF)** = 0.72 + 0.45 = 1.17
   iii. **S (API_Shell)** = 0.72 + 0.45 + 0.36
   S (API_Shell) = **1.53**
   iv. **S (User_MS)** = 0.72 + 0.45 = 1.17
   v. **S (User_DB)** = 0.72 + 0.45 = 1.17
   vi. **S (API_Gateway)** = 0.72
   vii. **S (Account_MS)** = 0.72
   viii. **S (Account_DB)** = 0.72
   ix. **S (Request_MF)** = 0.72
   x. **S (Request_DB)** = 0.72
   xi. **S (Request_MS)** = 0.72

8. **Ajuste del peso relativo del componente según el valor máximo de contribución:** Los valores encontrados en la fase anterior pueden ser comparados con el valor máximo de contribución al flujo, es por esta razón que se toma el valor mayor para definir una medida relativa para el árbol de subconjuntos tal como se muestra a continuación donde el común denominador es el 1.53, valor encontrado para el *API Shell*.

   i. **Re (User_MF)** = $\frac{1.17}{1.53} = 0.76$
   ii. **Re (Template_MF)** = $\frac{1.17}{1.53} = 0.76$
   iii. **Re (API_Shell)** = $\frac{1.17}{1.53} = 1$
   iv. **Re (User_MS)** = $\frac{1.17}{1.53} = 0.76$
   v. **Re (User_DB)** = $\frac{1.17}{1.53} = 0.76$
   vi. **Re (API_Gateway)** = $\frac{1.17}{1.53} = 0.47$
   vii. **Re (Account_MS)** = $\frac{1.17}{1.53} = 0.47$
   viii. **Re (Account_DB)** = $\frac{1.17}{1.53} = 0.47$
   ix. **Re (Request_MF)** = $\frac{1.17}{1.53} = 0.23$
   x. **Re (Request_DB)** = $\frac{1.17}{1.53} = 0.23$
   xi. **Re (Request_MS)** = $\frac{1.17}{1.53} = 0.23$

9. **Evaluación de la cohesión del flujo:** Finalmente, al identificar la contribución relativa de cada uno de los componentes de software sobre el flujo de información, es posible determinar la cohesión general del árbol de subconjuntos mediante la sumatoria de las relaciones condicionales de cada elemento del análisis en razón del total de componentes estudiados. La fórmula matemática que nos permite hallar este valor viene dada por la siguiente definición:

$$\textbf{Cohesión (Árbol de subconjuntos)} = \frac{Sumatoria\ de\ contribuciones\ relativas}{Número\ de\ componentes\ en\ el\ árbol}$$

Ahora bien, se efectúa la suma de cada una de las contribuciones según los valores encontrados en el paso anterior:

$$\textbf{Cohesión} = \frac{0.76 + 0.76 + 1 + 0.76 + 0.76 + 0.47 + 0.47 + 0.47 + 0.23 + 0.23 + 0.23}{11}$$

$$\textbf{Cohesión} = \mathbf{0.558}$$

Como en el análisis realizado solo existe un árbol de subconjuntos, la cohesión del árbol de subconjuntos termina siendo el resultado designado para el flujo de información que se definió en etapas anteriores. Con lo cual, se puede deducir que los componentes resultantes del proceso de descomposición procuran tener propiedades cohesivas por ser la evaluación un valor más cercano a 1 que a 0, según la escala definida para la comparación.

7 Conclusiones y Trabajos Futuros

7.1 Conclusiones

El modelo resultante del proceso de descomposición termina siendo fructífero no solo porque cumple con las funcionalidades especificadas en la fase de diseño, sino porque también comprende características cohesivas de acuerdo con la evaluación realizada. Del mismo modo, vale la pena destacar que cada una de las interacciones emergentes del método de desacoplamiento garantiza una respuesta oportuna a las exigencias del usuario y le da la posibilidad de poner a prueba otras configuraciones que en cualquier otro momento no serían factibles.

En ese sentido, la estrategia de desacoplamiento descrita resulta ser un marco de referencia adaptable a otros procedimientos en la medida en que toma aspectos propios de los sistemas de software como el dominio del problema, y lo lleva a un plano arquitectónico más fácil de implementar que otro tipo de abstracciones. Sin duda alguna, este tipo de métodos debe contemplar varias particularidades del entorno porque la desagregación de componentes en sí debe tener al menos un rasgo de diferencia para poder llevar a cabo su ejecución.

Adicionalmente, esta estrategia se sugiere como un modelo de guía complementario puesto que, en otro caso, el hecho de concebir un esquema estándar ligado a una tecnología terminaría siendo improductivo pues las librerías y frameworks de desarrollo están en constante cambio. De manera paralela, la proposición de este modelo tiene la finalidad de reducir la complejidad de construcción de algunos componentes ya que la noción de desacoplamiento puede ser llevada a la fase introductoria de diseño y no como una etapa posterior a la implementación.

Un aspecto significativo de esta investigación es el aporte que brinda el patrón de microservicios sobre la estrategia de descomposición, pues resulta muy útil para todo el proceso el hecho de tomar atributos aplicados en la capa de lógica de negocios y simularlos de cierta forma en la capa de presentación. Ejemplo de este ejercicio es la inclusión de un componente que se encargue tanto de la gestión de las peticiones, como de evitar puntos de acceso a los diversos recursos tal cual lo hace el *API Gateway*. Asimismo, entre detalles adicionales, se destaca que algunas piezas como el contenedor de la aplicación (*API Shell*) o herramientas de integración similares se convierten en elementos imprescindibles en la descomposición del componente de Front-End, porque los módulos salientes no tienen la capacidad de gestionar cuestiones como el enrutamiento o la renderización controlada de las vistas.

Por otra parte, el hecho de que cada componente tenga un contexto distinto hace que los mecanismos sugeridos no sean en ningún momento una teoría absoluta y quizás conduzcan a procesos incluso más rigurosos por la subjetividad de los requisitos del problema. Así pues, la descomposición de cualquier componente de Front-End debe tener en cuenta muchos aspectos estructurales debido a que el desacoplamiento de los módulos de un componente, primero, no es una acción que se pueda llevar a la ligera, y segundo, debe tener en consideración las conexiones con agentes externos que puedan alterar el sistema.

7.2 Trabajos Futuros

El crecimiento excesivo de los sistemas de información globales no es solamente un reto actual, pues desde hace algún tiempo ya se ha venido planteando como un desafío a tratar en la industria y posiblemente, se mantenga en los años venideros. Así pues, la manera cómo se gestiona la información en los diferentes componentes parece trasladarse también al futuro debido al hecho de que los sistemas crecen, las tecnologías cambian y surgen problemas inherentes de interacción. Por este motivo, los constructores de software deben estar preparados para la transición y contar con mecanismos que ayuden a la adaptación de los sistemas.

Hoy en día, muchas estructuras de software complejas sufren de ineficiencia, dependencia y acoplamiento en sus muchos de sus elementos por falta de metodologías o procesos estrictos de trabajo, en este caso en particular, parte de las labores a largo plazo consisten en continuar implementando alternativas que garanticen el manejo de la información en los componentes, que reduzcan la cantidad de código procesable por parte de los compiladores, que mejoren la operación del sistema cuando se deseen agregar funcionalidades, y que innoven el modo de interacción que se tiene para el usuario final.

Además, esta búsqueda debe estar enfocada en resolver inconvenientes emergentes del modelo final como problemas en la sincronización de las respuestas para ciertos microfrontends, ya que la circunstancia de tener un elemento gestor en la interfaz gráfica hace que el paso de mensajes entre componentes no sea el más consistente. De igual forma, otro reto a tener en cuenta para esfuerzos posteriores es el rendimiento del sistema pues agregar cierta independencia en la capa de presentación puede llevar a que los tiempos de respuesta del sistema no sean los mejores.

Bibliografía

[1] Abdullah, Hanin M. ; Zeki, Ahmed M.: Frontend and backend web technologies in social networking sites: Facebook as an example. In: *Proceedings - 3rd International Conference on Advanced Computer Science Applications and Technologies, ACSAT 2014*, Institute of Electrical and Electronics Engineers Inc., apr 2014. – ISBN 9781479918454, S. 85–89

[2] Adam, Julien ; Pérache, Marc: A parallel and resilient frontend for high performance validation suites. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* Bd. 10150 LNCS, Springer Verlag, 2017. – ISBN 9783319619811, S. 248–255

[3] Agarwal, Keshav ; Kaushik, Shardul ; Chhaviraj ; Gulati, Akansh ; Sheoran, Kavita: Quality model for component-based E-commerce application. In: *Proceedings of the 10th INDIACom; 2016 3rd International Conference on Computing for Sustainable Global Development, INDIACom 2016*, Institute of Electrical and Electronics Engineers Inc., oct 2016. – ISBN 9789380544199, S. 1075–1079

[4] Al-Debagy, Omar ; Martinek, Peter: A Comparative Review of Microservices and Monolithic Architectures. In: *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, 2018, S. 000149–000154

[5] Alkhalid, Abdulaziz ; Lung, Chung H. ; Ajila, Samuel: Software architecture decomposition using adaptive K-nearest neighbor algorithm. In: *Canadian Conference on Electrical and Computer Engineering*, 2013. – ISBN 9781479900329

[6] Alkhalid, Abdulaziz ; Lung, Chung H. ; Liu, Duo ; Ajila, Samuel: Software architecture decomposition using clustering techniques. In: *Proceedings - International Computer Software and Applications Conference*, IEEE Computer Society, 2013. – ISBN 9780769549866, S. 806–811

[7] Anil, Nish ; Jain, Tarun ; Wenzel, Maira ; Parente, John ; Warren, Genevieve: Identificación de los límites del modelo de dominio para cada microservicio. In: *Arquitectura para aplicaciones .NET en contenedor - Microsoft Docs* (2022), May, Nr. 1. `https://docs.microsoft.com/es-es/dotnet/architecture/microservices/`

[8] Aral, Atakan ; Ovatman, Tolga: Utilization of Method Graphs to Measure Cohesion in Object Oriented Software. In: *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*, 2013, S. 505–510

[9] Beugnard, Antoine ; Jézéquel, Jean M. ; Plouzeau, Noël ; Watkins, Damien: Making components contract aware. In: *Computer* 32 (1999), jul, Nr. 7, S. 38–45. http://dx.doi.org/10.1109/2.774917. – DOI 10.1109/2.774917. – ISSN 00189162

[10] Bi, Qianwen ; Tang, Jingpeng ; Van Fleet, Bradley ; Nelson, Jason ; Beal, Ian ; Ray, Candra ; Ossola, Andrew: Software Architecture for Machine Learning in Personal Financial Planning. In: *2020 Intermountain Engineering, Technology and Computing (IETC)*, 2020, S. 1–4

[11] Bogado, Verónica ; Gonnet, Silvio ; Leone, Horacio: DEVS-based methodological framework for multi-quality attribute evaluation using software architectures. In: *2017 43rd Latin American Computer Conference, CLEI 2017* Bd. 2017-Janua, Institute of Electrical and Electronics Engineers Inc., dec 2017. – ISBN 9781538630570, S. 1–10

[12] Brandner, Michael ; Craes, Michael ; Oellermann, Frank ; Zimmermann, Olaf: Web services-oriented architecture in production in the finance industry. In: *Informatik-Spektrum* 27 (2004), apr, Nr. 2, S. 136–145. http://dx.doi.org/10.1007/s00287-004-0380-2. – DOI 10.1007/s00287-004-0380-2. – ISSN 0170-6012

[13] Cavalcante, Everton ; Batista, Thais ; Oquendo, Flavio: Supporting Dynamic Software Architectures: From Architectural Description to Implementation. In: *Proceedings - 12th Working IEEE/IFIP Conference on Software Architecture, WICSA 2015*, Institute of Electrical and Electronics Engineers Inc., jul 2015. – ISBN 9781479919222, S. 31–40

[14] Cendejas, José Luis: Implementación del modelo integral colaborativo (MDSIC) como fuente de innovación para el desarrollo ágil de software en las empresas de la zona centro - occidente en México. In: *Centro Interdisciplinario de Posgrados - Universidad Popular Autónoma del Estado de Puebla* (2014), 05, S. 268

[15] Chen, Jenny ; Hsu, Alyn: Implementing a banking front end processor in Taiwan using SOA. In: *Proceedings - IEEE International Conference on e-Business Engineering, ICEBE 2006*, 2006. – ISBN 0769526454, S. 522–527

[16] Clements, P ; Others: Documenting Software Architectures: Views and Beyond. SEI Series in Software Engineering. (2002), Nr. 6, S. 740–741. ISBN 0321552687

[17] Cui, Jian F. ; Chae, Heung S.: Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems. In: *Information and Software Technology* 53 (2011), jun, Nr. 6, S. 601–614. http://dx.doi.org/10.1016/j.infsof.2011.01.006. – DOI 10.1016/j.infsof.2011.01.006. – ISSN 09505849

[18] Derakhshanmanesh, Mahdi ; Ebert, Jürgen ; Grieger, Marvin ; Engels, Gregor: Model-integrating development of software systems: a flexible component-based approach. In: *Software and Systems Modeling* 18 (2019), aug, Nr. 4, S. 2557–2586. http://dx.doi.org/10.1007/s10270-018-0682-5. – DOI 10.1007/s10270-018-0682-5. – ISSN 16191374

[19] Di Francesco, Paolo ; Lago, Patricia ; Malavolta, Ivano: Architecting with microservices: A systematic mapping study. In: *Journal of Systems and Software* 150 (2019), apr, S. 77–97. http://dx.doi.org/10.1016/j.jss.2019.01.001. – DOI 10.1016/j.jss.2019.01.001. – ISSN 01641212

[20] Dinoto, Lucio: The LATAM Financial Software Market. In: *IT Professional* 17 (2015), Nr. 2, S. 10–13. http://dx.doi.org/10.1109/MITP.2015.32. – DOI 10.1109/MITP.2015.32

[21] Elshafei, Sherif ; Menshawi, Mahmoud: A case study on a systematic approach for developing a user interface software for programming intensive software tools. In: *Canadian Conference on Electrical and Computer Engineering* Bd. 2015-June, Institute of Electrical and Electronics Engineers Inc., jun 2015. – ISSN 08407789, S. 132–139

[22] Fedosejev, Artemij ; Boduch, Adam: *React 16 Essentials: A fast-paced, hands-on guide to designing and building scalable and maintainable web apps with React 16*. 2017. – ISBN 9781787126046

[23] Franco, Joao M. ; Barbosa, Raul ; Zenha-Rela, Mário: Reliability analysis of software architecture evolution. In: *Proceedings - 6th Latin-American Symposium on Dependable Computing, LADC 2013*, 2013. – ISBN 9780769549620, S. 11–20

[24] Frost, B.: *Atomic Design*. Brad Frost Web, 2016. https://books.google.com.co/books?id=1e92vgAACAAJ. – ISBN 9780998296609

[25] Funk, Mathias ; Hoyer, Philip ; Link, Stefan: Model-driven instrumentation of graphical user interfaces. In: *Proceedings of the 2nd International Conferences on Advances in Computer-Human Interactions, ACHI 2009*, 2009. – ISBN 9780769535296, S. 19–25

[26] Gholamshahi, Shabnam ; Hasheminejad, Seyed Mohammad H.: Software component identification and selection: A research review. In: *Software - Practice and Experience* 49 (2019), jan, Nr. 1, S. 40–69. http://dx.doi.org/10.1002/spe.2656. – DOI 10.1002/spe.2656. – ISSN 1097024X

[27] Godbolt, Micah: *Frontend Architecture for Design Systems: A Modern Blueprint for Scalable and Sustainable Websites*. 2016

[28] Guerraoui, Rachid ; Frølund, Svend: e-Transactions: End-to-end reliability for three-tier architectures. In: *IEEE Transactions on Software Engineering* 28 (2002), apr, Nr. 4, S. 378–395. http://dx.doi.org/10.1109/TSE.2002.995430. – DOI 10.1109/TSE.2002.995430. – ISSN 00985589

[29] Hammad, Maen ; Banat, Rua'a H.: Automatic Class Decomposition using Clustering. In: *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, 2021, S. 78–81

[30] Harman, Mark ; Jones, Bryan F.: Search-based software engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications. In: *Information and Software Technology* 43 (2001), Nr. 14, 833–839. http://dx.doi.org/10.1016/S0950-5849(01)00189-6. – DOI 10.1016/S0950-5849(01)00189-6. – ISBN 0360-0300

[31] Harms, Holger ; Rogowski, Collin ; Lo Iacono, Luigi: Guidelines for adopting frontend architectures and patterns in microservices-based systems. In: *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering* Bd. Part F1301, Association for Computing Machinery, aug 2017. – ISBN 9781450351058, S. 902–907

[32] Harrison, R. ; Nithi, R. V. ; Counsell, S. J.: An Evaluation of the MOOD Set of Object-Oriented Software Metrics. In: *IEEE Transactions on Software Engineering* 30 (1998), Nr. 06, S. 491–496. – ISSN 1939-3520

[33] Helbich, Jan ; Cerny, Tomas: Energy impact of web user interface technology on mobile devices. In: *2016 6th International Conference on IT Convergence and Security, ICITCS 2016*, Institute of Electrical and Electronics Engineers Inc., nov 2016. – ISBN 9781509037643

[34] Husein, Sukainah ; Oxley, Alan: A Coupling and Cohesion Metrics Suite for Object-Oriented Software. In: *2009 International Conference on Computer Technology and Development* Bd. 1, 2009, S. 421–425

[35] Iyapparaja, M. ; Sureshkumar, S.: Coupling and cohesion metrics in Java for adaptive reusability risk reduction. In: *IET Chennai 3rd International on Sustainable Energy and Intelligent Systems (SEISCON 2012)*, 2012, S. 1–6

[36] Jha, P. C. ; Bali, Vikram ; Narula, Sonam ; Kalra, Mala: Optimal component selection based on cohesion & coupling for component based software system under build-or-buy scheme. In: *Journal of Computational Science* 5 (2014), mar, Nr. 2, S. 233–242. http://dx.doi.org/10.1016/j.jocs.2013.07.003. – DOI 10.1016/j.jocs.2013.07.003. – ISSN 18777503

[37] Jinping, Liu: Computer auxiliary accounting research of environment accounting software. In: *The 27th Chinese Control and Decision Conference (2015 CCDC)*, 2015, S. 2141–2146

[38] Kirğil, Elif Nur H. ; Ayyildiz, Tülin E.: Analysis of Lack of Cohesion in Methods (LCOM): A Case Study. In: *2021 2nd International Informatics and Software Engineering Conference (IISEC)*, 2021, S. 1–4

[39] Laga, Nassim ; Bertin, Emmanuel ; Crespi, Noel: Composition at the frontend: The user centric approach. In: *2010 14th Int. Conference on Intelligence in Next Generation Networks: "Weaving Applications Into the Network Fabric", ICIN 2010 - 2nd Int. Workshop on Business Models for Mobile Platforms, BMMP 10*, 2010. – ISBN 9781424474455

[40] Lai, R. ; Kim, Y.S. ; Mahmood, S.: Survey of component-based software development. In: *IET Software* 1 (2007), apr, Nr. 2, 57–66. http://dx.doi.org/10.1049/iet-sen:20060045. – DOI 10.1049/iet-sen:20060045. – ISSN 1751-8806

[41] Lung, Chung H. ; Xu, Xia ; Zaman, Marzia: Software architecture decomposition using attributes. In: *17th International Conference on Software Engineering and Knowledge Engineering, SEKE 2005*, 2005. – ISBN 9781627486590, S. 747–752

[42] Maheshwari, Anshu ; Tripathi, Aprna ; Kushwaha, Dharmender S.: A New Design Based Software Coupling Metric. In: *2014 International Conference on Information Technology*, 2014, S. 351–355

[43] Malek, Sam ; Edwards, George ; Brun, Yuriy ; Tajalli, Hossein ; Garcia, Joshua ; Krka, Ivo ; Medvidovic, Nenad ; Mikic-Rakic, Marija ; Sukhatme, Gaurav S.: An architecture-driven software mobility framework. In: *Journal of Systems and Software* 83 (2010), jun, Nr. 6, S. 972–989. http://dx.doi.org/10.1016/j.jss.2009.11.003. – DOI 10.1016/j.jss.2009.11.003. – ISSN 01641212

[44] Maple, Carsten ; French, Tim ; Conrad, Marc: A visual formalism for graphical user interfaces based on state transition diagrams. In: *Proceedings of the International Conference on Information Visualisation* Bd. 2003-Janua, Institute of Electrical and Electronics Engineers Inc., 2003. – ISBN 0769519881, S. 183–188

[45] McC Smith, J. ; Stotts, D.: Elemental design patterns: A formal semantics for composition of OO software architecture. In: *Proceedings - 27th Annual NASA Goddard / IEEE Software Engineering Workshop, SEW 2002*, Institute of Electrical and Electronics Engineers Inc., 2003. – ISBN 0769518559, S. 183–190

[46] Mitchell, Brian S. ; Mancoridis, Spiros: On the automatic modularization of software systems using the bunch tool. In: *IEEE Transactions on Software Engineering* 32 (2006), mar, Nr. 3, S. 193–208. http://dx.doi.org/10.1109/TSE.2006.31. – DOI 10.1109/TSE.2006.31. – ISSN 00985589

[47] Newman, Christopher ; Agioutantis, Zach ; Schaefer, Nathaniel: Development of a web-platform for mining applications. In: *International Journal of Mining Science and Technology* 28 (2018), jan, Nr. 1, S. 95–99. http://dx.doi.org/10.1016/j.ijmst.2017.11.016. – DOI 10.1016/j.ijmst.2017.11.016. – ISSN 20952686

[48] Newman, Sam: *Building Microservices: Designing Fine-Grained Systems*. O'Reilly Media, 2015. – 280 S. – ISBN 978-1-491-95035-7

[49] Oktafiani, Intan ; Hendradjaya, Bayu: Software Metrics Proposal for Conformity Checking of Class Diagram to SOLID Design Principles. In: *2018 5th International Conference on Data and Software Engineering (ICoDSE)*, 2018, S. 1–6

[50] Pichler, Josef ; Ramler, Rudolf: How to test the intangible properties of graphical user interfaces? In: *Proceedings of the 1st International Conference on Software Testing, Verification and Validation, ICST 2008*, 2008. – ISBN 076953127X, S. 494–497

[51] Prat, A. ; Catot, M. ; Fletcher, P. ; Lorés, J. ; Southwick, R.: Using the FOCUS architecture for developing knowledge-based front ends: a KBFE for forecasting. In: *Knowledge-Based Systems* 6 (1993), Nr. 1, S. 11–16. http://dx.doi.org/10.1016/0950-7051(93)90004-D. – DOI 10.1016/0950–7051(93)90004–D. – ISSN 09507051

[52] Pressman, Roger: *Software Engineering: A Practitioner's Approach*. Bd. 8th. McGraw Hill, 2014. – Pg: 197–276 S. – ISBN 978–607–15–0314–5

[53] Putz, Steve: Interactive information services using World-Wide Web hypertext. In: *Computer Networks and ISDN Systems* 27 (1994), Nr. 2, S. 273–280. http://dx.doi.org/10.1016/0169-7552(94)90141-4. – DOI 10.1016/0169–7552(94)90141–4. – ISSN 01697552

[54] Qi, Zhao ; Li, Ma: The design and implementation of the front-end software for the telemetry and telecontrol system of satellite. In: *Proceedings - 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, GreenCom-iThings-CPSCom 2013*, 2013. – ISBN 9780769550466, S. 1885–1890

[55] Ramírez, Aurora ; Romero, José Raúl ; Ventura, Sebastián: An approach for the evolutionary discovery of software architectures. In: *Information Sciences* 305 (2015), jun, S. 234–255. http://dx.doi.org/10.1016/j.ins.2015.01.017. – DOI 10.1016/j.ins.2015.01.017. – ISSN 00200255

[56] Reyes-Delgado, Paola Y. ; Mora, Manuel ; Duran-Limon, Hector A. ; Rodríguez-Martínez, Laura C. ; O'Connor, Rory V. ; Mendoza-Gonzalez, Ricardo: The strengths and weaknesses of software architecture design in the RUP, MSF, MBASE and RUP-SOA methodologies: A conceptual review. 47 (2016), aug, S. 24–41. http://dx.doi.org/10.1016/j.csi.2016.02.005. – DOI 10.1016/j.csi.2016.02.005. – ISSN 09205489

[57] Richards, Mark ; America, United S. (Hrsg.): *Software Architecture Understanding Common Architecture Patterns Patterns and When to Use Them*. O'Reilly M. United States of America : O'Reilly Media, Inc, 2015. – 97 S. http://dx.doi.org/10.1002/he.18. http://dx.doi.org/10.1002/he.18. – ISBN 9781491924242

[58] Richardson, Chris: *Microservices Patterns*. 'With examples in Java'. Shelter Island, NY 11964. United States of America : Manning Publications Co., 2019. – 492 S. – ISBN 9781617294549

[59] Rozanski, Nick. ; Woods, Eoin.: *Software Systems Architecture: Working with stakeholders using viewpoints and perspectives*. Addison-Wesley, 2012. – 49–5717–49–5717 S. http://dx.doi.org/10.5860/choice.49-5717. http://dx.doi.org/10.5860/choice.49-5717. – ISBN 0321112296

[60] Saadati, Mahdi ; Motameni, Homayoon: Measuring Cohesion and Coupling of Object-Oriented Systems. In: *Journal of Mathematics and Computer Science. Islamic Azad University, Sari, Iran* (2013), Nr. 1, S. 1–9

[61] Saltiveri, Toni G. ; Vidal, Jesús L. ; Delgado, José Juan C.: *Diseño de sistemas interactivos centrados en el usuario*. Editorial UOC, 2011. – 280 pgs S. https://books.google.com.co/books?id=Bk5Uv0Aiis0C. – ISBN 8497883209

[62] SANTOS, Vismar: LA Industria del Software. Estudio a nivel global y América Latina. In: *Observatorio de la Economía Latinoamericana* (2009), 06, S. 1–23. – ISSN 0032 116

[63] Schnoor, Henning ; Hasselbring, Wilhelm: Poster: Toward Measuring Software Coupling via Weighted Dynamic Metrics. In: *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, 2018, S. 342–343

[64] Seele, Wilbert ; Syed, Shaheen ; Brinkkemper, Sjaak: The functional architecture modeling method applied on web browsers. In: *Proceedings - Working IEEE/IFIP Conference on Software Architecture 2014, WICSA 2014*, IEEE Computer Society, 2014. – ISBN 9781479934126, S. 171–174

[65] Sharma, Anubha ; Kumar, Manoj ; Agarwal, Sonali: A Complete Survey on Software Architectural Styles and Patterns. In: *Procedia Computer Science* Bd. 70, Elsevier B.V., 2015. – ISSN 18770509, S. 16–28

[66] Silveira, Paulo ; Mannan, Umme A. ; Almeida, Eduardo S. ; Nagappan, Nachi ; Lo, David ; Singh Kochhar, Pavneet ; Gao, Cuiyun ; Ahmed, Iftekhar: A Deep Dive into the Impact of COVID-19 on Software Development. In: *IEEE Transactions on Software Engineering* (2021), S. 1–1. http://dx.doi.org/10.1109/TSE.2021.3088759. – DOI 10.1109/TSE.2021.3088759. – ISSN 0098–5589

[67] Sun, Xiaoyang ; Ji, Zhenshan ; Wang, Feng ; Wang, Yang: A new user front end for EAST remote participation. In: *IEEE Transactions on Plasma Science* 46 (2018), may, Nr. 5, S. 1127–1130. http://dx.doi.org/10.1109/TPS.2018.2811413. – DOI 10.1109/TPS.2018.2811413. – ISSN 00933813

[68] Svahnberg, Mikael ; Mårtensson, Frans: Six years of evaluating software architectures in student projects. In: *Journal of Systems and Software* 80 (2007), nov, Nr. 11, S. 1893–1901. http://dx.doi.org/10.1016/j.jss.2007.01.050. – DOI 10.1016/j.jss.2007.01.050. – ISSN 01641212

[69] Szyperski, Clemens: Emerging component software technologies - A strategic comparison. 19 (1998), Nr. 1, S. 2–10. http://dx.doi.org/10.1007/s003780050002. – DOI 10.1007/s003780050002. – ISSN 09458115

[70] Takahashi, Akiko ; Kinoshita, Tetsuo: A behavioral characteristics model for a flexible distributed system. In: *International Journal of Pervasive Computing and Communications* 6 (2010), jun, Nr. 2, S. 192–213. http://dx.doi.org/10.1108/17427371011066419. – DOI 10.1108/17427371011066419. – ISSN 17427371

[71] Taylor, Richard N. ; Medvidoviç, Nenad. ; Dashofy, Eric M. (Eric M.: *Software architecture: Foundations, Theory, and Practice*. Wiley, 2010. – 712 S. – ISBN 0470167742

[72] Tibermacine, Chouki ; Sadou, Salah ; Ton That, Minh T. ; Dony, Christophe: Software architecture constraint reuse-by-composition. In: *Future Generation Computer Systems* 61 (2016), aug, S. 37–53. http://dx.doi.org/10.1016/j.future.2016.02.006. – DOI 10.1016/j.future.2016.02.006. – ISSN 0167739X

[73] Tucholka, Andrzej ; Gurbani, Prem: A Highly Decoupled Front-End Framework for High Trafficked Web Applications. In: *2010 Fifth International Conference on Internet and Web Applications and Services*, 2010, S. 32–36

[74] Vakali, Athena ; Korosoglou, Paschalis ; Daoglou, Pavlos: A multi-layer software architecture framework for adaptive real-time analytics. In: *Proceedings - 2016 IEEE International Conference on Big Data, Big Data 2016*, Institute of Electrical and Electronics Engineers Inc., 2016. – ISBN 9781467390040, S. 2425–2430

[75] Vera, Fredy H.: Modelo inteligente de especificación de granularidad de aplicaciones basadas en microservicios. In: *Escuela de Ingeniería de Sistemas y Computación - Universidad del Valle* (2021), Jan, Nr. 1, S. 16–152

[76] Young, Michal ; Taylor, Richard N. ; Troup, Dennis B.: Software Environment Architectures and User Interface Facilities. In: *IEEE Transactions on Software Engineering* 14 (1988), Nr. 6, S. 697–708. http://dx.doi.org/10.1109/32.6151. – DOI 10.1109/32.6151. – ISSN 00985589