Universidad Tecnologica de Bolivar

Restaurant Chain

Software Architecture Document (SAD)

CONTENT OWNER:

- **Luis Daniel Arias Marrugo**
- **Michel Augusto Castellano Severiche**
- **Brandold Vega Pérez**
- **Andrés Felipe García Sosa**

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
1.0		27-09-24
2.0		28-09-24
3.0		29-09-24
4.0		29-09-24
5.0		9-10-24
6.0		10-10-24
7.0		11-10-24
8.0		24-11-24
9.0		24-11-24
10		26-11-24

Table of Contents

List of Figures

List of Tables

1 Documentation Roadmap

The Documentation Roadmap should be the first place a new reader of the SAD begins. But for new and returning readers, it is intended to describe how the SAD is organized so that a reader with specific interests who does not wish to read the SAD cover-to-cover can find desired information quickly and directly.

Sub-sections of Section 1 include the following.

- Section 1.1 (“Document Management and Configuration Control Information”) explains revision history. This tells you if you’re looking at the correct version of the SAD.

- Section 1.2 (“Purpose and Scope of the SAD”) explains the purpose and scope of the SAD, and indicates what information is and is not included. This tells you if the information you’re seeking is likely to be in this document.

- Section 1.3 (“How the SAD Is Organized”) explains the information that is found in each section of the SAD. This tells you what section(s) in this SAD are most likely to contain the information you seek.

- Section 0 (“Este documento se organiza en las siguientes secciones:

- **Section 1:** Información introductoria y de propósito.
- **Section 2:** Antecedentes de la arquitectura y requerimientos principales.
- **Section 3:** Vistas arquitectónicas principales.
- **Section 4:** Relaciones entre vistas.
- **Section 5-6:** Referencias, glosario y acrónimos.

- Stakeholder Representation”) explains the stakeholders for which the SAD has been particularly aimed. This tells you how you might use the SAD to do your job.

- Section 1.5 (“Viewpoint Definitions”) explains the *viewpoints* (as defined by IEEE Standard 1471-2000) used in this SAD. For each viewpoint defined in Section 1.5, there is a corresponding view defined in Section 3 (“Views”). This tells you how the architectural information has been partitioned, and what views are most likely to contain the information you seek.

- Section 1.6 (“How a View is Documented”) explains the standard organization used to document architectural views in this SAD. This tells you what section within a view you should read in order to find the information you seek.

1.1 Document Management and Configuration Control Information

- Revision Number: 9.0
- Revision Release Date: 24-11-24
- Purpose of Revision:
- Scope of Revision:

1.2 Purpose and Scope of the SAD

El propósito de este *Software Architecture Document* (SAD) es proporcionar una descripción detallada de la arquitectura del sistema para la cadena de restaurantes que se está desarrollando como parte del caso de estudio. Este sistema tiene como objetivo unificar y optimizar las operaciones de los restaurantes mediante la integración de funciones clave como la gestión de pedidos, el manejo de inventarios, los programas de fidelización y las aplicaciones móviles en una plataforma basada en microservicios. La arquitectura del sistema se ha diseñado con el objetivo de ofrecer una solución robusta y escalable que permita la centralización de datos en tiempo real, facilitando la experiencia omnicanal para los clientes y mejorando la eficiencia operativa de la empresa.

El SAD documenta las decisiones de diseño arquitectónicas tomadas durante el proceso de desarrollo y proporciona la base para la implementación, mantenimiento y evolución del sistema. Esta documentación sirve como guía para desarrolladores, administradores de sistemas, gerentes de proyecto y otros stakeholders interesados en la visión global del sistema, incluyendo su estructura, comportamiento y la interacción entre los diferentes módulos del sistema.

Este documento abarca la arquitectura del sistema para la cadena de restaurantes, que está constituido por los siguientes componentes clave:

- **Microservicios** que descomponen el sistema monolítico en unidades independientes, facilitando su mantenimiento y escalabilidad.

- **API Gateway**, que centraliza las interacciones con los microservicios y facilita el acceso a los datos de clientes, pedidos y menús.

- **Plataforma de análisis en la nube**, que se utiliza para procesar los datos del sistema y optimizar decisiones estratégicas mediante el uso de **aprendizaje automático (ML)**.

- **Integración omnicanal**, que asegura que los clientes reciban una experiencia coherente a través de diversos puntos de contacto, como aplicaciones móviles, terminales POS y sistemas de fidelización.

- **Sistemas de seguridad**, que garantizan la protección de los datos y transacciones, cumpliendo con normativas de protección de datos, como GDPR.

El alcance del SAD incluye la descripción de los módulos, componentes y las interacciones entre ellos, así como los principios de diseño que guían la implementación y operación del sistema. No se profundiza en los detalles de implementación de cada microservicio o en la configuración específica de hardware, sino que se concentra en la visión global y las decisiones arquitectónicas que permiten alcanzar los objetivos del sistema de manera eficiente, escalable y segura.

Este documento será utilizado como referencia a lo largo del ciclo de vida del proyecto, desde la fase de desarrollo hasta la de mantenimiento y actualización, asegurando que todos los miembros del equipo comprendan la estructura y el comportamiento del sistema.

1.3 How the SAD Is Organized

Este documento se organiza en las siguientes secciones:

- **Section 1:** Información introductoria y de propósito.
- **Section 2:** Antecedentes de la arquitectura y requerimientos principales.
- **Section 3:** Vistas arquitectónicas principales.
- **Section 4:** Relaciones entre vistas.
- **Section 5-6:** Referencias, glosario y acrónimos.

1.4 Stakeholder Representation

Esta sección identifica los roles de los stakeholders involucrados en el desarrollo de la arquitectura del sistema y las preocupaciones específicas que cada uno tiene, las cuales se

abordan en este documento. A continuación, se describen los principales roles y sus inquietudes:

1. **Administradores del Sistema**
    - **Preocupaciones:** Garantizar la seguridad, mantenibilidad y configurabilidad del sistema. Hay que asegurar que las actualizaciones puedan realizarse con interrupciones mínimas y que el monitoreo del sistema sea eficiente.
2. **Gerentes de Restaurante**
    - **Preocupaciones:** Asegurar la disponibilidad del sistema para realizar operaciones críticas, como la gestión de inventarios y generación de reportes. Facilitar una toma de decisiones basada en datos confiables.
3. **Cajeros**
    - **Preocupaciones:** Interactuar con un sistema fácil de usar y que ofrezca tiempos de respuesta rápidos al realizar transacciones.
4. **Clientes**
    - **Preocupaciones:** Obtener una experiencia de usuario coherente y personalizada a través de todos los puntos de contacto. Garantizar la seguridad de sus datos personales y transacciones.
5. **Analistas de Datos**
    - **Preocupaciones:** Acceder a datos precisos y consistentes para generar análisis e informes. Integrar datos con herramientas externas para análisis avanzados.
6. **Desarrolladores**
    - **Preocupaciones:** Tener una arquitectura modular y bien documentada que facilite la integración de nuevos componentes, reduzca la complejidad y sea compatible con tecnologías modernas.
7. **Certificadores de Seguridad**
    - **Preocupaciones:** Asegurar que el sistema cumpla con estándares de protección de datos, como GDPR o CCPA, y que utilice buenas prácticas de seguridad, como encriptación robusta y autenticación multifactorial.
8. **Gerentes de Proyecto**
    - **Preocupaciones:** Cumplir con los cronogramas y presupuestos del proyecto, y garantizar que los equipos puedan trabajar de manera independiente y eficiente, minimizando dependencias.

1.5 Viewpoint Definitions

Este documento adopta un enfoque de múltiples vistas centrado en los stakeholders, según lo recomendado por el estándar ANSI/IEEE 1471-2000. Este enfoque asegura que la arquitectura del sistema se documente de manera integral, abordando las preocupaciones específicas de cada grupo de interesados a través de puntos de vista relevantes.

- **Definición de un Viewpoint**

Un viewpoint es un conjunto de principios y técnicas que guía la creación de una vista arquitectónica para abordar las preocupaciones de un grupo específico de stakeholders. Cada viewpoint especifica:

- Las preocupaciones que aborda.
- Las técnicas de modelado y evaluación utilizadas.
- Los criterios de consistencia y completitud que deben cumplir las vistas conformes.

Una view, por tanto, es una representación de elementos del software, sus propiedades y relaciones, que sigue las definiciones establecidas en un viewpoint. Las vistas seleccionadas en este SAD buscan proporcionar una descripción holística del sistema.

- **Viewpoints Utilizados en este SAD**

1. **Module Viewpoint**
   Este *viewpoint* organiza el sistema en módulos independientes que representan áreas funcionales específicas, como la gestión de pedidos, inventarios y programas de lealtad. Aborda preocupaciones de desarrolladores y gerentes de proyecto relacionadas con la modularidad, mantenibilidad y asignación de responsabilidades.

2. **Component-and-Connector Viewpoint**
   Este *viewpoint* describe los componentes que se ejecutan en tiempo de ejecución y las conexiones entre ellos. Es fundamental para los administradores del sistema, desarrolladores y certificadores de seguridad, ya que proporciona detalles sobre la comunicación interna, protocolos y garantías de seguridad.

3. **Allocation Viewpoint**
   Este *viewpoint* asigna elementos del software a recursos externos, como servidores, contenedores Docker y bases de datos. Aborda preocupaciones de gerentes de

proyecto y administradores del sistema sobre escalabilidad, rendimiento y disponibilidad.

- **Rationale de Selección de Viewpoints**

La selección de estos *viewpoints* se realizó considerando:

- La necesidad de abordar preocupaciones críticas de stakeholders específicos.
- La complejidad del sistema y su naturaleza distribuida.
- La relevancia de garantizar escalabilidad, seguridad y modularidad en el diseño.

Table 1: Stakeholders and Relevant Viewpoints

Stakeholder	**Viewpoint(s) that apply to that class of stakeholder's concerns**
Administradores del Sistema	Component-and-Connector View, Allocation View
Gerentes de Restaurante	Module View, Component-and-Connector View
Cajeros	Component-and-Connector View
Desarrolladores	Module View, Component-and-Connector View

1.5.1 Module Viewpoint

1.5.1.1 Abstract

El *Module Viewpoint* organiza el sistema en módulos independientes que representan áreas funcionales específicas, como la gestión de pedidos, inventarios y programas de lealtad.

Cada módulo encapsula una funcionalidad que puede ser desarrollada, mantenida y escalada de manera independiente, mejorando la modularidad, la mantenibilidad y la asignación clara de responsabilidades. Este enfoque permite una arquitectura más limpia y flexible, facilitando la división de tareas entre los equipos de desarrollo.

1.5.1.2 Stakeholders and Their Concerns Addressed

Este *viewpoint* está diseñado para abordar las preocupaciones de los siguientes stakeholders:

- **Desarrolladores:** Necesitan una estructura modular para implementar funcionalidades específicas de forma independiente, permitiendo actualizaciones y pruebas más fáciles.

- **Gerentes de Proyecto:** Buscan una asignación clara de responsabilidades entre módulos, lo que facilita la planificación y la estimación de recursos para el proyecto.

Preguntas que este *viewpoint* puede responder:

- ¿Cómo se organiza el sistema en unidades funcionales independientes?

- ¿Qué responsabilidades tiene cada módulo y cómo se asignan?

- ¿Qué impacto tendría un cambio en un módulo en el sistema global?

1.5.1.3 Elements, Relations, Properties, and Constraints

- **Elements:** Los elementos son los módulos del sistema, como "Gestión de Pedidos", "Inventarios", y "Programas de Lealtad". Cada módulo representa una funcionalidad autónoma.
- **Relations:** Los módulos están relacionados entre sí mediante interfaces bien definidas. Por ejemplo, el módulo de "Gestión de Pedidos" interactúa con el de "Inventarios" para verificar la disponibilidad de productos.
- **Properties:** Los módulos tienen propiedades como su nombre, las responsabilidades que manejan, los servicios que proporcionan a otros módulos y los datos que gestionan.

- **Constraints:** Los módulos deben ser lo suficientemente autónomos para que puedan ser modificados sin afectar significativamente a otros módulos. Además, los módulos deben cumplir con interfaces estándar para garantizar su interoperabilidad.

1.5.1.4 Language(s) to Model/Represent Conforming Views

Lenguajes y notaciones:

- **UML (Unified Modeling Language):** Se utiliza para representar los módulos mediante diagramas de clases o componentes.
- **Texto:** Descripciones detalladas de las responsabilidades de cada módulo.

1.5.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

- **Técnicas de evaluación:**
  - Evaluación basada en la consistencia de la asignación de funcionalidades, asegurando que no haya solapamientos de responsabilidad entre módulos.
  - Análisis de impacto: verificar cómo los cambios en un módulo afectan al sistema global.
- **Criterios de consistencia y completitud:**
  - Cada módulo debe tener una única responsabilidad.
  - Cada módulo debe ser independiente, sin dependencias circulares entre ellos.
  - La totalidad de las funcionalidades del sistema debe estar cubierto por un módulo especifico.

1.5.1.6 Viewpoint Source

Este *viewpoint* se basa en prácticas de diseño de software modular descritas en la literatura de arquitecturas de software, específicamente en "Documenting Software Architecture: Views and Beyond" de Clements (2002), que define la importancia de los módulos autónomos en un sistema distribuido.

1.5.2 Component-and-Connector Viewpoint

1.5.2.1 Abstract

El *Component-and-Connector Viewpoint* describe los componentes del sistema que ejecutan procesos en tiempo de ejecución y las conexiones entre ellos. Este enfoque proporciona una visión clara de la estructura dinámica del sistema, mostrando cómo los componentes interactúan entre sí para cumplir con los objetivos del sistema. Es fundamental para entender cómo fluye la información a través del sistema y cómo se mantienen las garantías de seguridad, rendimiento y disponibilidad.

1.5.2.2 Stakeholders and Their Concerns Addressed

Este *viewpoint* aborda las preocupaciones de los siguientes stakeholders:

- **Administradores del Sistema:** Necesitan comprender cómo se comunican los componentes y cómo se gestionan las conexiones para garantizar la seguridad, disponibilidad y escalabilidad del sistema.

- **Desarrolladores:** Buscan detalles sobre la estructura interna del sistema, incluyendo las interacciones entre componentes y cómo manejar las conexiones para garantizar un rendimiento óptimo.

- **Certificadores de Seguridad:** Están preocupados por la seguridad en la comunicación entre componentes, incluyendo protocolos de autenticación y cifrado.

Preguntas que este *viewpoint* puede responder:

- ¿Qué componentes existen en el sistema y cómo interactúan entre sí?

- ¿Qué protocolos y mecanismos de comunicación se utilizan entre componentes?

- ¿Cómo se asegura la seguridad y la integridad de las interacciones entre componentes?

1.5.2.3 Elements, Relations, Properties, and Constraints

- **Elements:** Los componentes son los procesos o unidades de trabajo en ejecución (por ejemplo, "Módulo de Gestión de Pedidos", "API Gateway", "Base de Datos").
- **Relations:** Los componentes se interconectan mediante protocolos de comunicación (por ejemplo, **REST API**, **gRPC**) para intercambiar información.
- **Properties:** Los componentes tienen propiedades como el tipo de operación que realizan, su capacidad para manejar solicitudes concurrentes y la seguridad de las conexiones.

• **Constraints:** Los componentes deben ser resilientes y permitir una comunicación eficiente y segura. Las conexiones deben ser rápidas y tolerantes a fallos, y el sistema debe garantizar la coherencia de los datos a través de las conexiones.

1.5.2.4 Language(s) to Model/Represent Conforming Views

Lenguajes y notaciones:

- **UML:** Se pueden usar diagramas de componentes y secuencias para representar las interacciones entre componentes.
- **Texto:** Descripciones detalladas de cómo se configuran y comunican los componentes, incluyendo los protocolos utilizados.

1.5.2.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

Técnicas de evaluación:

- **Análisis de impacto:** Examinar cómo los fallos en un componente afectan al sistema en su conjunto.
- **Evaluación de rendimiento:** Analizar la eficiencia de la comunicación entre los componentes y su capacidad para manejar cargas de trabajo simultáneas.

Criterios de consistencia y completitud:

- Los componentes deben estar correctamente interconectados mediante las interfaces apropiadas.
- Las conexiones deben garantizar la seguridad, confidencialidad e integridad de los datos.

1.5.2.6 Viewpoint Source

Este *viewpoint* se basa en el enfoque de arquitectura de componentes y conectores descrito en "Documenting Software Architecture: Views and Beyond" de Clements (2002), que hace hincapié en la importancia de definir las interacciones de los componentes en sistemas distribuidos.

1.5.3 Allocation Viewpoint

1.5.3.1 Abstract

El *Allocation Viewpoint* asigna los elementos del software a los recursos físicos o virtuales, como servidores, contenedores Docker y bases de datos. Este enfoque es crucial para la planificación de la infraestructura del sistema, asegurando que los recursos estén distribuidos eficientemente para cumplir con los requisitos de rendimiento, escalabilidad y disponibilidad.

1.5.3.2 Stakeholders and Their Concerns Addressed

Este *viewpoint* responde a las preocupaciones de los siguientes stakeholders:

- **Gerentes de Proyecto:** Necesitan asegurarse de que los recursos se asignen de manera eficiente para cumplir con los requisitos de rendimiento y disponibilidad, y dentro de los límites de presupuesto.

- **Administradores del Sistema:** Están interesados en cómo se distribuyen los recursos para garantizar la escalabilidad y la disponibilidad del sistema.

Preguntas que este *viewpoint* puede responder:

- ¿Cómo se distribuyen los componentes en los diferentes servidores y recursos de infraestructura?

- ¿Qué recursos son necesarios para asegurar el rendimiento y la escalabilidad del sistema?

- ¿Cómo se gestionan los recursos para asegurar que el sistema sea resiliente y altamente disponible?

1.5.3.3 Elements, Relations, Properties, and Constraints

- **Elements:** Los elementos son los recursos asignados a los componentes, como servidores, bases de datos, contenedores Docker, y redes.
- **Relations:** La asignación de componentes a recursos es definida mediante relaciones que conectan las capacidades del hardware o la infraestructura con las necesidades de los componentes del software.
- **Properties:** Los recursos tienen propiedades como capacidad de procesamiento, almacenamiento y tolerancia a fallos.

• **Constraints:** Los recursos deben ser dimensionados correctamente para garantizar el rendimiento bajo carga. Además, la infraestructura debe ser flexible para escalar según las necesidades del sistema.

1.5.3.4 Language(s) to Model/Represent Conforming Views

Lenguajes y notaciones:

- **Diagrama de infraestructura (UML o similar):** Representa cómo los componentes del sistema se asignan a recursos físicos o virtuales.
- **Texto:** Descripción detallada de la asignación de recursos y sus capacidades.

1.5.3.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

Técnicas de evaluación:

- Análisis de rendimiento: Medir la capacidad de la infraestructura para manejar cargas de trabajo bajo diferentes escenarios.
- Evaluación de escalabilidad: Verificar cómo el sistema se adapta a aumentos de carga y cómo los recursos pueden ser reconfigurados para satisfacer la demanda.

Criterios de consistencia y completitud:

- Los recursos deben estar correctamente asignados a los componentes, de acuerdo con sus necesidades de rendimiento.
- La infraestructura debe ser resiliente y permitir la recuperación ante fallos.

1.5.3.6 Viewpoint Source

Este *viewpoint* se basa en las mejores prácticas de asignación de recursos y escalabilidad de sistemas distribuidos, como se describe en "Documenting Software Architecture: Views and Beyond" de Clements (2002).

1.6 How a View is Documented

La **Sección 3** de este SAD contiene una vista para cada *viewpoint* listado en la **Sección 1.5**. Cada vista se documenta como un conjunto de *view packets*. Un *view packet* es el conjunto

más pequeño de documentación arquitectónica que se puede entregar a un stakeholder individual.

Cada vista se documenta de la siguiente manera, donde la letra **i** se refiere al número de la vista (1, 2, etc.):

- **Sección 3.i: Nombre de la vista.**
  En esta sección se nombra la vista que está siendo documentada.

- **Sección 3.i.1: Descripción de la vista.**
  Esta sección describe el propósito y los contenidos de la vista. Se debe hacer referencia (y coincidir) con la descripción del *viewpoint* correspondiente en la **Sección 1.5**, el cual define cómo se documenta esta vista.

- **Sección 3.i.2: Visión general de los *view packets*.**
  Aquí se presenta el conjunto de *view packets* para esta vista, junto con la justificación que explica por qué el conjunto elegido es completo y no duplicado. El conjunto de *view packets* puede ser listado de forma textual o representado gráficamente, mostrando cómo estos particionan la arquitectura completa que se describe en la vista.

- **Sección 3.i.3: Antecedentes de la arquitectura.**
  Mientras que los antecedentes de la arquitectura en la **Sección 2** se refieren a los requerimientos generales y decisiones cuya influencia abarca toda la arquitectura, esta sección ofrece los antecedentes específicos de la arquitectura aplicables a esta vista. Incluye requerimientos significativos, enfoques de diseño, patrones, resultados de análisis y cobertura de requerimientos relevantes para esta vista.

- **Sección 3.i.4: Mecanismos de variabilidad.**
  En esta sección se describen los mecanismos de variabilidad arquitectónica (por ejemplo, datos de adaptación, parámetros de tiempo de compilación, replicación variable, etc.) presentados por esta vista. Se incluye una descripción de cómo y cuándo esos mecanismos pueden ser utilizados y cualquier restricción asociada a su uso.

- **Sección 3.i.5: *View packets*.**
  Esta sección presenta todos los *view packets* asociados a esta vista. Cada *view packet* se describe utilizando el siguiente esquema, donde la letra **j** representa el número del *view packet* (1, 2, etc.):

  - **Sección 3.i.5.j: *View packet* #j.**
    Aquí se describe el *view packet* en cuestión.

- **Sección 3.i.5.j.1: Presentación primaria.**
  Esta sección presenta los elementos y las relaciones entre ellos que conforman este *view packet*, utilizando un lenguaje adecuado, notación o representación basada en herramientas.

- **Sección 3.i.5.j.2: Catálogo de elementos.**
  Mientras que la presentación primaria muestra los elementos importantes y sus relaciones, esta sección proporciona información adicional necesaria para completar la imagen arquitectónica. El catálogo de elementos incluye las siguientes subsecciones:

  - **Sección 3.i.5.j.2.1: Elementos.**
    Esta subsección describe cada uno de los elementos mostrados en la presentación primaria, detalla las responsabilidades de cada elemento y especifica los valores de las propiedades relevantes de los elementos, tal como se definen en el *viewpoint* correspondiente.

  - **Sección 3.i.5.j.2.2: Relaciones.**
    Esta subsección describe las relaciones adicionales entre los elementos mostrados en la presentación primaria, o las especializaciones o restricciones sobre las relaciones presentadas.

  - **Sección 3.i.5.j.2.3: Interfaces.**
    Esta subsección especifica las interfaces de software de los elementos que deben ser visibles a otros elementos, explicando cómo interactúan con otros componentes del sistema.

  - **Sección 3.i.5.j.2.4: Comportamiento.**
    En esta subsección se especifica cualquier comportamiento significativo de los elementos o grupos de elementos interactuantes mostrados en la presentación primaria.

  - **Sección 3.i.5.j.2.5: Restricciones.**
    Esta subsección lista las restricciones sobre los elementos o relaciones que no se han descrito en las subsecciones anteriores.

- **Sección 3.i.5.j.3: Diagrama de contexto.**
  Esta sección proporciona un diagrama de contexto que muestra el contexto de la parte del sistema representada por este *view packet*. También se designa el alcance de este *view packet* con un símbolo distinguido, y se muestran las interacciones con entidades externas en la jerga del *viewpoint*.

- **Sección 3.i.5.j.4: Mecanismos de variabilidad.** Esta sección describe las variabilidades disponibles en la porción del sistema mostrada en este *view packet*, incluyendo cómo y cuándo esos mecanismos pueden ser ejercidos, así como cualquier restricción asociada a su uso.

- **Sección 3.i.5.j.5: Antecedentes de la arquitectura.** Esta sección proporciona la justificación de las decisiones de diseño significativas cuyo alcance está limitado a este *view packet*.

- **Sección 3.i.5.j.6: Relación con otros *view packets*.** Esta sección proporciona referencias a otros *view packets* relacionados, incluyendo el padre, los hijos y los hermanos de este *view packet*. Los *view packets* relacionados pueden encontrarse en la misma vista o en vistas diferentes.

Este enfoque asegura que cada *view* sea documentada de manera completa y clara, permitiendo a los stakeholders comprender todos los aspectos clave del sistema y sus componentes.

1.7 Relationship to Other SADs

1.8 Process for Updating this SAD

2 Architecture Background

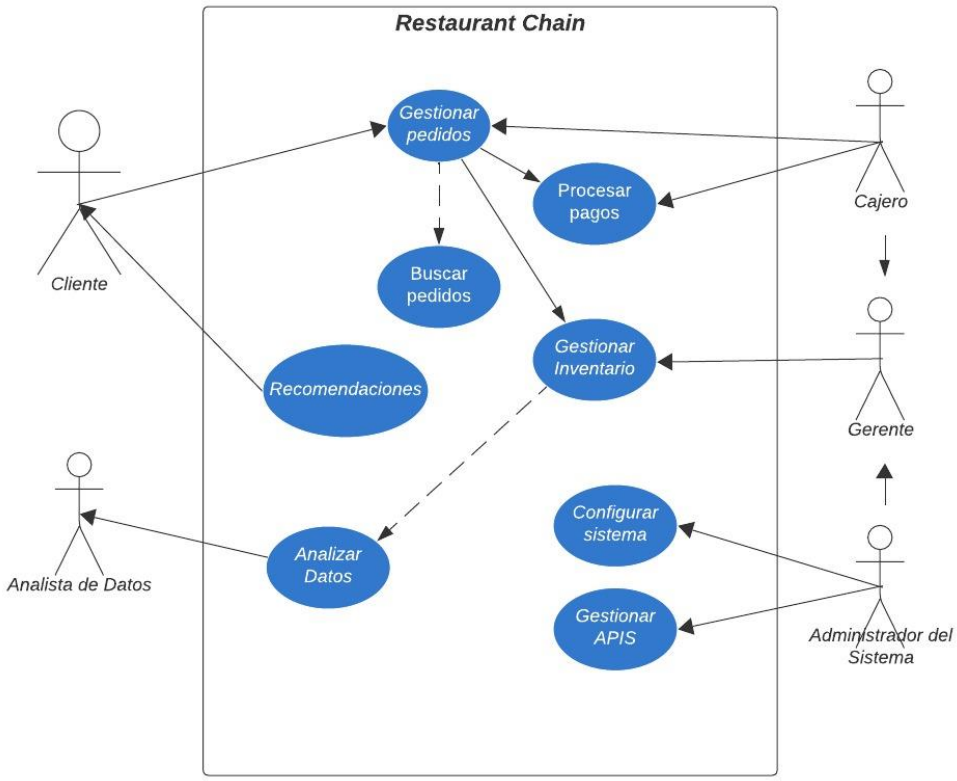
2.1 Problem Background

La cadena de restaurantes opera con un sistema fragmentado que combina sistemas POS heredados, programas de lealtad independientes y aplicaciones móviles separadas. Esto genera una serie de ineficiencias operativas debido a la falta de integración entre estos sistemas, con datos aislados que limitan los conocimientos sobre los clientes. La ausencia de una plataforma unificada impide la escalabilidad efectiva y requiere un esfuerzo significativo para coordinar cambios o agregar nuevas funcionalidades.

Este enfoque fragmentado también limita la capacidad de la cadena para ofrecer experiencias omnicanal coherentes. Por ejemplo, las promociones y los pedidos no están integrados entre los puntos de venta, las aplicaciones móviles y los programas de lealtad, lo que resulta en mensajes inconsistentes y una experiencia insatisfactoria para los clientes.

2.1.1 System Overview

El sistema busca modernizar una cadena de restaurantes reemplazando sistemas monolíticos por microservicios para integrar la gestión de pedidos, inventarios y programas de lealtad. Para ello en el siguiente diagrama de casos de uso se evidencia como interactúan las partes del negocio con cada uno de los microservicios



2.1.2 Goals and Context

El objetivo principal es diseñar e implementar una arquitectura moderna basada en microservicios para abordar las ineficiencias actuales. Esto incluye:

1. Integración Omnicanal: Ofrecer una experiencia consistente para los clientes en todos los puntos de interacción.

2. Eficiencia Operativa: Mejorar la capacidad de agregar nuevas funcionalidades sin afectar otros servicios.

3. Análisis Avanzado: Centralizar los datos en una plataforma de análisis en la nube para habilitar personalización y optimización basada en ML.

4. Escalabilidad: Facilitar la expansión del sistema con cambios incrementales.

El nuevo sistema descompondrá los sistemas monolíticos existentes en microservicios enfocados que se comunican mediante APIs estándar y están gestionados por un API Gateway.

2.1.3 Significant Driving Requirements

Los requisitos clave que impulsan la necesidad de esta solución incluyen:

1. **Interoperabilidad de Sistemas:**

    - La arquitectura debe permitir la comunicación entre sistemas POS, programas de lealtad y aplicaciones móviles mediante APIs estándar.

2. **Personalización y Análisis:**

    - Los datos deben fluir hacia una plataforma de análisis en la nube para permitir personalización en tiempo real y optimización mediante aprendizaje automático (ML).

3. **Consistencia en la Experiencia del Usuario:**

    - Las promociones, precios y datos de inventario deben estar sincronizados entre todos los puntos de interacción (POS, móvil, lealtad).

4. **Escalabilidad:**

- La solución debe soportar la expansión de la cadena con capacidad para agregar nuevas funcionalidades y módulos sin interrupciones significativas.

5. **Seguridad y Gobernanza:**

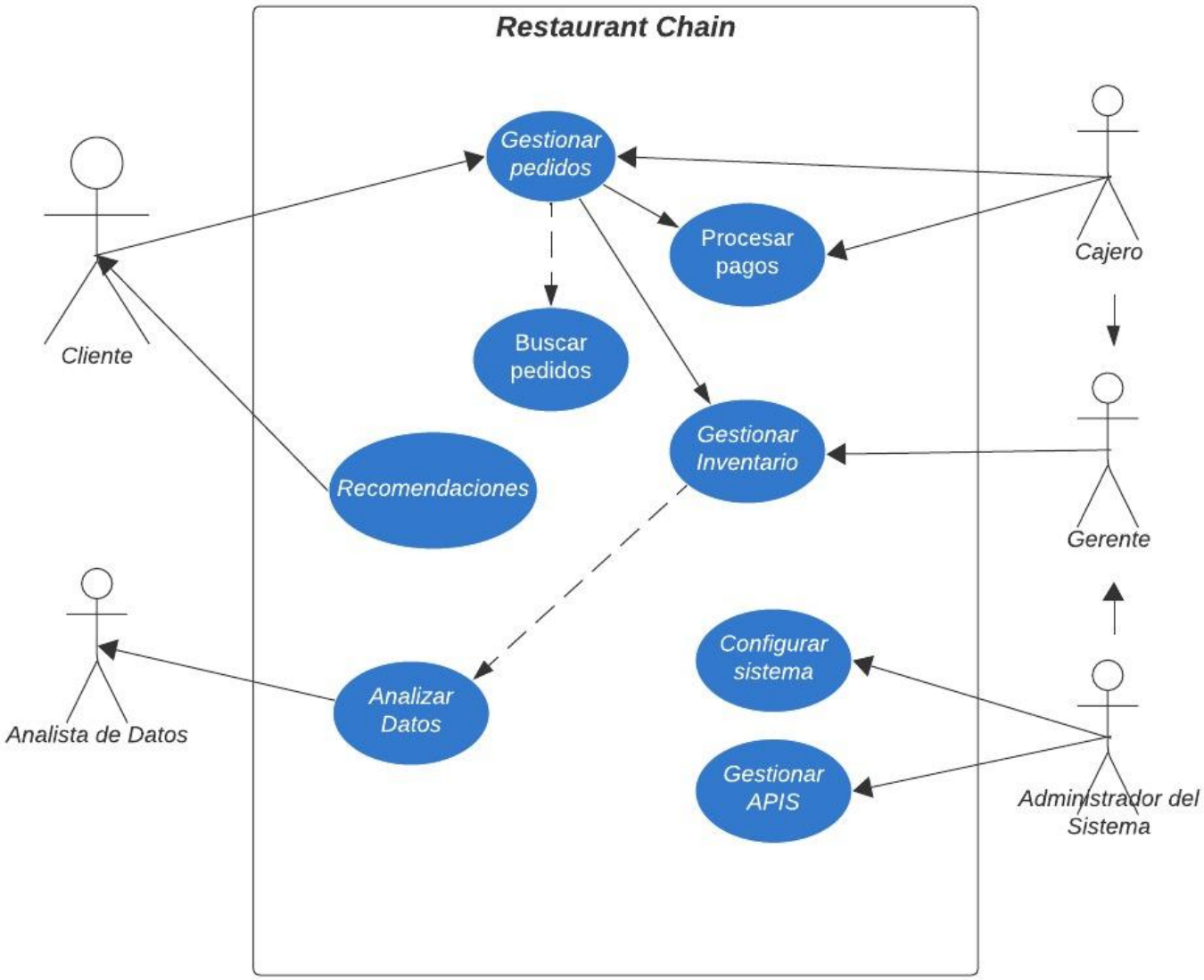
   - Los datos sensibles de clientes y transacciones deben estar protegidos cumpliendo con regulaciones como GDPR y estándares de seguridad modernos.

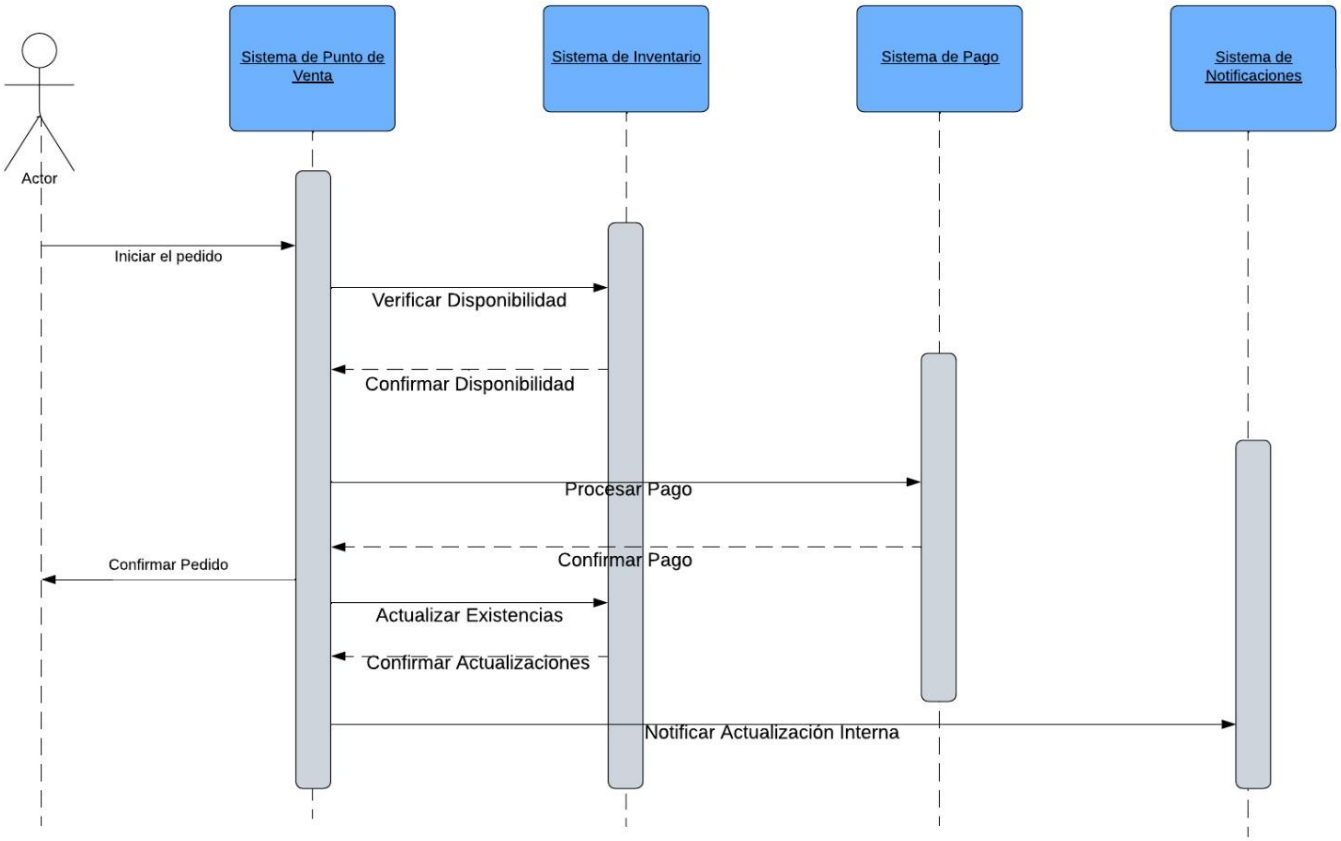
Estos requisitos impulsan la transición hacia una arquitectura basada en microservicios con capacidades robustas de integración y análisis, permitiendo una operación más eficiente y orientada al cliente.

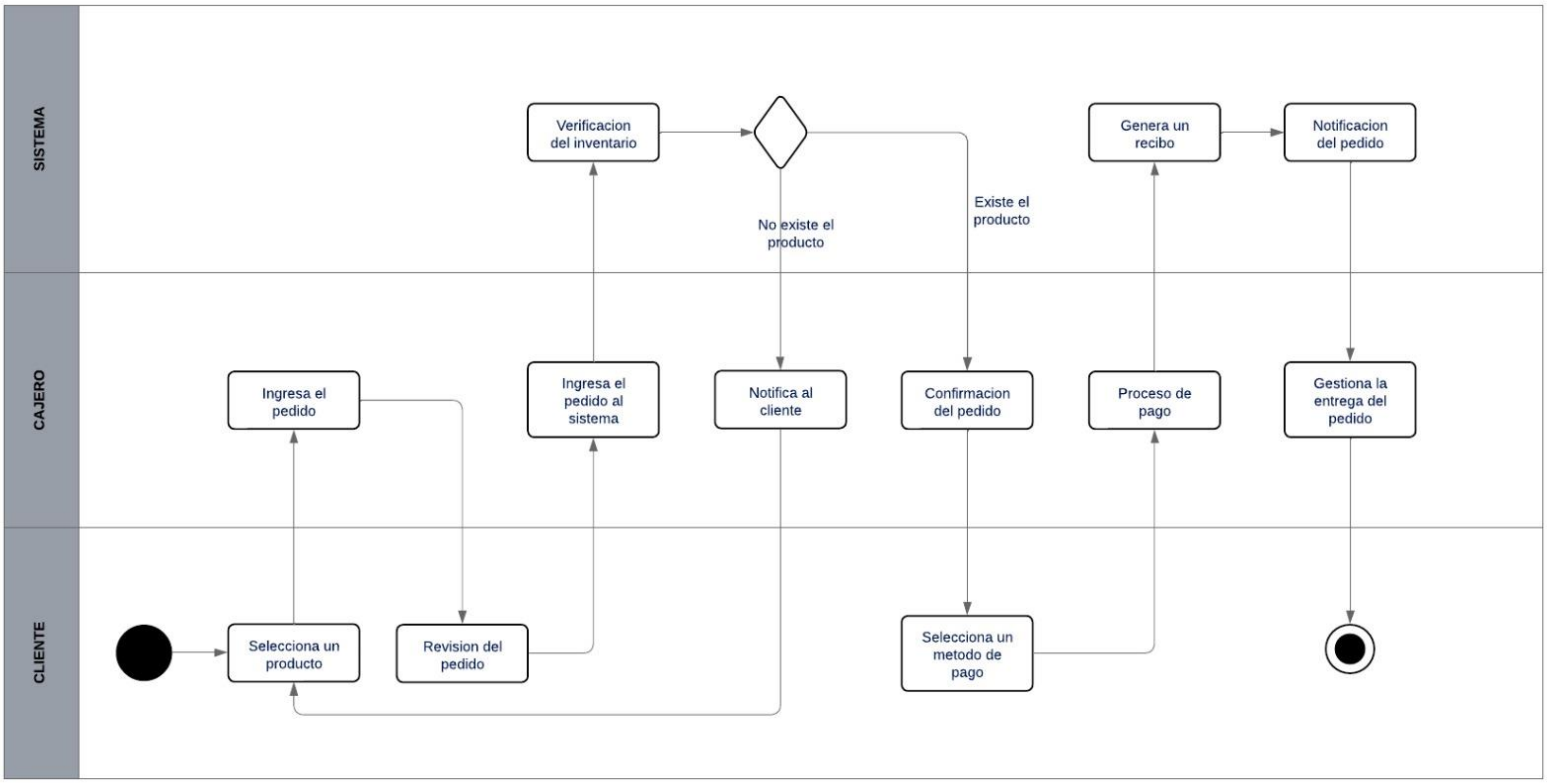
2.2 Solution Background

La solución se encuentra en identificar microservicios que estén orientados a brindar la capacidad para el usuario dentro de una cadena de restaurante. La empresa necesita compartir datos entre canales para permitir interacciones personalizadas, coordinación de inventario y optimización de procesos. La solución implica la creación de microservicios orientados a las capacidades de los restaurantes, como los programas de pedidos y fidelización. Estos expondrían API de datos de clientes, ventas y menús gestionadas por una puerta de enlace. Estos datos se transmiten a una plataforma de análisis en la nube para la optimización del aprendizaje automático. Este enfoque descompone de forma incremental los monolitos con el tiempo en servicios enfocados y desacoplados que brindan acceso unificado a los datos a través de API estandarizadas.

Para ello, se implementó el desarrollo de casos de uso para la resolución de las partes implicadas, a continuación, se evidencia en el diagrama el planteamiento de los casos de uso en el desarrollo del proyecto:




Sistema de Punto de Venta
Sistema de Inventario
Sistema de Pago
Sistema de Notificaciones
Actor
Iniciar el pedido
Verificar Disponibilidad
Confirmar Disponibilidad
Procesar Pago
Confirmar Pago
Confirmar Pedido
Actualizar Existencias
Confirmar Actualizaciones
Notificar Actualización Interna


SISTEMA
CAJERO
CLIENTE
Verificacion del inventario
No existe el producto
Existe el producto
Genera un recibo
Notificacion del pedido
Ingresa el pedido
Ingresa el pedido al sistema
Notifica al cliente
Confirmacion del pedido
Proceso de pago
Gestiona la entrega del pedido
Selecciona un producto
Revision del pedido
Selecciona un metodo de pago

2.2.1 Architectural Approaches

Se utiliza una arquitectura basada en microservicios, gestionada con Docker, y APIs REST para comunicación interna y externa. Al igual que la implementación de una arquitectura limpia dentro de la cual el sistema mantendrá una escalabilidad, mantenibilidad y fácil testeo.

2.2.2 Analysis Results

El análisis ATAM asegura que la arquitectura soporta cambios y crece sin impacto negativo.

2.2.3 Requirements Coverage

La arquitectura de software propuesta para la cadena de restaurantes está diseñada para abordar tanto los requisitos originales como los derivados, garantizando que cada uno sea satisfecho de manera efectiva dentro de la arquitectura. A continuación, se describe cómo cada requisito se aborda en la solución:

Requisitos Originales y su Cobertura en la Arquitectura

1. **Gestión de Pedidos (Original)**
    - **Descripción:** El sistema debe permitir la recepción y gestión de pedidos desde múltiples canales (POS, aplicaciones móviles, programas de lealtad).
    - **Cobertura en la Arquitectura:**
        - Microservicio de pedidos que procesa solicitudes en tiempo real.
        - APIs expuestas por el *API Gateway* para recibir pedidos desde cualquier canal.
        - Conexión directa con el microservicio de inventarios para verificar disponibilidad.
2. **Gestión de Inventarios (Original)**
    - **Descripción:** Actualización en tiempo real de inventarios basada en los pedidos procesados y las promociones aplicadas.
    - **Cobertura en la Arquitectura:**

- Microservicio de inventarios sincronizado con el microservicio de pedidos.
- Transmisión de datos hacia la plataforma de análisis en la nube para prever tendencias y necesidades futuras.

3. **Personalización de Experiencias (Derivado)**
   - **Descripción:** Ofrecer recomendaciones personalizadas basadas en el historial de pedidos y preferencias del cliente.
   - **Cobertura en la Arquitectura:**
     - Microservicio de programas de lealtad que centraliza los datos del cliente.
     - Uso de algoritmos de *machine learning* en la plataforma de análisis para personalizar las ofertas y promociones.
4. **Promociones y Precios Dinámicos (Original)**
   - **Descripción:** Optimización de promociones y ajustes dinámicos de precios basados en la demanda y patrones de comportamiento del cliente.
   - **Cobertura en la Arquitectura:**
     - Microservicio de promociones conectado con el microservicio de inventarios para reflejar cambios en tiempo real.
     - Modelo de ML en la plataforma de análisis que evalúa la efectividad de las promociones y ajusta precios automáticamente.
5. **Interoperabilidad de Sistemas (Original)**
   - **Descripción:** Integrar sistemas POS, aplicaciones móviles y programas de lealtad a través de una arquitectura común.
   - **Cobertura en la Arquitectura:**
     - *API Gateway* que estandariza las comunicaciones y asegura interoperabilidad entre los diferentes canales.

6. **Seguridad de Datos (Derivado)**

    - **Descripción:** Garantizar que los datos sensibles de clientes y transacciones estén protegidos y cumplan con regulaciones como GDPR.

    - **Cobertura en la Arquitectura:**

        - Autenticación multifactor y encriptación de datos implementadas en el *API Gateway*.

        - Políticas de acceso seguro para todos los microservicios.

Dónde se Aborda Cada Requisito en la Arquitectura

- **Requisitos de funcionalidad:** Cubiertos por los microservicios específicos, como pedidos, inventarios, promociones y lealtad.

- **Requisitos de interoperabilidad:** Implementados en el *API Gateway*.

- **Requisitos de personalización y análisis:** Gestionados por la plataforma de análisis en la nube.

- **Requisitos de seguridad:** Incorporados en la configuración del *API Gateway* y las conexiones entre microservicios.

La arquitectura está diseñada para garantizar que estos requisitos sean abordados de manera integral, proporcionando una solución escalable, segura y adaptable.

2.2.4 Summary of Background Changes Reflected in Current Version

La versión actual del Software Architecture Document (SAD) para la cadena de restaurantes refleja los avances realizados desde la versión original que inicialmente fue desarrollada en el documento Software requirements specification Template (SRS), con base en las decisiones arquitectónicas, los cambios en los requisitos y los resultados de análisis que influyeron en la evolución de la arquitectura. A continuación, se presenta un resumen de los cambios más relevantes:

Acciones y Decisiones Tomadas

1. **Adopción de Microservicios:**

- La arquitectura se rediseñó para descomponer los sistemas monolíticos existentes en microservicios independientes, mejorando la modularidad y escalabilidad.

- Los microservicios principales definidos incluyen: gestión de pedidos, inventarios, programas de lealtad y promociones.

2. **Implementación del *API Gateway*:**

   - Se decidió centralizar la gestión de comunicaciones entre microservicios a través de un *API Gateway*.

   - Esto permite la estandarización de las APIs y mejora la seguridad mediante autenticación multifactor y encriptación.

3. **Integración con la Nube:**

   - Los datos generados por los microservicios ahora se transmiten a una plataforma en la nube para análisis avanzados y optimización mediante ML.

   - Esta decisión se tomó para abordar la necesidad de personalización en tiempo real y optimización dinámica de operaciones.

4. **Seguridad y Cumplimiento:**

   - Se reforzaron las políticas de seguridad en las conexiones entre microservicios y en la gestión de datos sensibles para cumplir con normativas como GDPR.

Factores Decisivos y Resultados del Análisis

1. **Estudios de Impacto en la Modularidad:**

   - Un análisis comparativo entre sistemas monolíticos y arquitecturas basadas en microservicios mostró una mejora significativa en la capacidad de implementar cambios sin afectar otras partes del sistema.

   - Esto impulsó la adopción de microservicios como el enfoque principal.

2. **Demanda de Personalización:**

- Los requisitos de los usuarios finales destacaron la importancia de ofrecer promociones personalizadas y experiencias coherentes.

- Esto justificó la integración de análisis avanzados y ML en la nube.

3. **Evaluación de Riesgos de Seguridad:**

    - Las auditorías iniciales identificaron vulnerabilidades en la gestión de datos, lo que llevó a priorizar medidas de seguridad avanzadas como encriptación y autenticación multifactor.

Cambios en los Requisitos que Impulsaron Decisiones

1. **Escalabilidad Mejorada:**

    - Los requisitos evolucionaron para incluir la capacidad de manejar aumentos significativos en la demanda, especialmente durante eventos promocionales. Esto condujo a la implementación de microservicios escalables y balanceo de carga dinámico.

2. **Integración Omnicanal:**

    - Se añadió la necesidad de integrar de manera uniforme las interacciones entre POS, aplicaciones móviles y programas de lealtad. Esto influyó en la introducción del *API Gateway*.

3. **Cumplimiento de Normativas de Privacidad:**

    - Cambios en las regulaciones de privacidad de datos, como GDPR, impulsaron decisiones para implementar políticas más estrictas de manejo de datos y auditorías internas regulares.

Evolución y Cambios en la Arquitectura

1. **Despliegue de Prototipos Iniciales:**

    - Se implementaron microservicios para gestión de pedidos y lealtad como pruebas iniciales. Estas pruebas confirmaron la viabilidad del enfoque modular y guiaron ajustes en la configuración del *API Gateway*.

2. **Optimización de Comunicaciones:**

- Las conexiones entre microservicios se optimizaron utilizando protocolos como gRPC para mejorar la latencia y la eficiencia en tiempo real.

3. **Iteración Basada en Retroalimentación:**

   - La retroalimentación obtenida de los usuarios internos y pruebas iniciales llevó a ajustar el diseño de las APIs para mejorar la experiencia de desarrollo e integración.

2.3 Product Line Reuse Considerations

En el contexto del desarrollo de una línea de productos para la cadena de restaurantes, esta sección detalla cómo el software cubierto por este SAD se planifica o se espera que se reutilice para apoyar la visión de la línea de productos. A medida que la solución evoluciona y se adapta a diferentes necesidades, se considera la reutilización de módulos y componentes dentro de diferentes implementaciones, manteniendo una arquitectura flexible y escalable.

- **Variaciones Planeadas y Mecanismos de Variación**

Variación 1: Gestión de Pedidos Adaptable

- **Descripción:** El módulo de *Gestión de Pedidos* se adaptará para trabajar con diferentes tipos de interfaces de usuario, como aplicaciones móviles, terminales POS y plataformas de pedidos en línea.

- **Incremento(s) de Construcción:**

  - *Incremento 1:* Versión inicial que incluye la gestión de pedidos en el sistema POS tradicional.

  - *Incremento 2:* Expansión para incluir una interfaz de pedidos a través de aplicaciones móviles.

- **Uso de la Variación:**

  - La variación será utilizada en restaurantes con interfaces móviles específicas y será adaptada para su despliegue en plataformas externas, como aplicaciones de terceros.

Variación 2: Programas de Lealtad Personalizables

- **Descripción:** El módulo de *Programas de Fidelización* se diseñará de forma modular para permitir la creación de promociones personalizadas basadas en el comportamiento de compra del cliente. Esta variación permitirá a diferentes restaurantes o cadenas ajustar el tipo de recompensas ofrecidas.

- **Incremento(s) de Construcción:**

    - *Incremento 1:* Implementación básica de un sistema de recompensas.

    - *Incremento 3:* Personalización avanzada de programas de fidelización para integrar datos de compras en tiempo real y preferencias individuales.

- **Uso de la Variación:**

    - La variación será utilizada en establecimientos con necesidades específicas de promociones y recompensas personalizadas para mejorar la experiencia del cliente.

Variación 3: Optimización de Precios Basada en ML (Aprendizaje Automático)

- **Descripción:** El módulo de *Promociones y Precios* será ampliado con capacidades de análisis predictivo y optimización de precios utilizando algoritmos de aprendizaje automático.

- **Incremento(s) de Construcción:**

    - *Incremento 2:* Implementación básica de ajustes de precios.

    - *Incremento 4:* Optimización avanzada de precios basada en datos históricos y en tiempo real.

- **Uso de la Variación:**

    - Esta variación se implementará en restaurantes que operan en múltiples mercados con diferentes comportamientos de consumidores y precios dinámicos.

- **Potencial de Reutilización Adicional**

Aunque las variaciones descritas anteriormente están planificadas para los próximos incrementos de desarrollo, existen otras áreas donde los módulos y sus variaciones podrían ser

reutilizados en el futuro, incluso si no están específicamente planificadas en esta fase. Algunos ejemplos de reutilización adicional incluyen:

1. **Reutilización del Módulo de Gestión de Inventarios**:

    - **Contexto:** Este módulo podría ser reutilizado en sistemas de gestión de almacenes más grandes o para integrar otras cadenas de suministro, manteniendo la coherencia en los niveles de inventario entre varios puntos de venta.

    - **Posibilidad de Reutilización:** Se explorará en una futura expansión del sistema a otras cadenas de restaurantes o tipos de negocio relacionados.

2. **Microservicios de Autenticación y Seguridad**:

    - **Contexto:** El microservicio de autenticación y seguridad para la gestión de cuentas de clientes podría ser reutilizado en otros sistemas empresariales, como plataformas de fidelización de terceros o aplicaciones móviles externas.

    - **Posibilidad de Reutilización:** Aunque no se planifica una reutilización inmediata, se evalúa la posibilidad de integrarlo con servicios externos a medida que la empresa crece.

3 Views

This section contains the views of the software architecture. A view is a representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. Concretely, a view shows a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.

Architectural views can be divided into three groups, depending on the broad nature of the elements they show. These are:

- Module views. Here, the elements are modules, which are units of implementation. Modules represent a code-based way of considering the system. Modules are assigned areas of functional responsibility and are assigned to teams for implementation. There is less emphasis on how the resulting software manifests itself at runtime. Module structures allow us to answer questions such as: What is the primary functional responsibility assigned to each module? What other software elements is a module allowed to use? What other software does it actually use? What modules are related to other modules by generalization or specialization (i.e., inheritance) relationships?

- Component-and-connector views. Here, the elements are runtime components (which are principal units of computation) and connectors (which are the communication vehicles among components). Component and connector structures help answer questions such as: What are the major executing components and how do they interact? What are the major shared data stores? Which parts of the system are replicated? How does data progress through the system? What parts of the system can run in parallel? How can the system's structure change as it executes?

- Allocation views. These views show the relationship between the software elements and elements in one or more external environments in which the software is created and executed. Allocation structures answer questions such as: What processor does each software element execute on? In what files is each element stored during development, testing, and system building? What is the assignment of the software element to development teams?

These three kinds of structures correspond to the three broad kinds of decisions that architectural design involves:

- How is the system to be structured as a set of code units (modules)

- How is the system structured as a set of elements that have run-time behavior (components) and interactions (connectors) ?

- How is the system to relate to non-software structures in its environment (such as CPUs, file systems, networks, development teams, etc.)?

Often, a view shows information from more than one of these categories. However, unless chosen carefully, the information in such a hybrid view can be confusing and not well understood.

Name of view	**Viewtype that defines this view**	**Types of elements and relations shown**		**Is this a module view?**	**Is this a component-and-connector view?**	**Is this an allocation view?**
Vista de Módulos Funcionales	Module View	Módulos funcionales como Gestión de Pedidos, Inventarios, Promociones.	Dependencias entre módulos, como la verificación de disponibilidad.	si	no	no

Vista de Componentes y Conexiones	Component-and-Connector View	Componentes en tiempo de ejecución, como *API Gateway*, *Microservicio de Pedidos*.	Interacciones entre componentes a través de un bus de mensajes interno.	no	si	no
Vista de Asignación de Recursos	Allocation View	Recursos como contenedores Docker, servidores físicos y plataformas en la nube.	Asignación de componentes a recursos físicos o virtuales.	no	no	si

The views presented in this SAD are the following:-

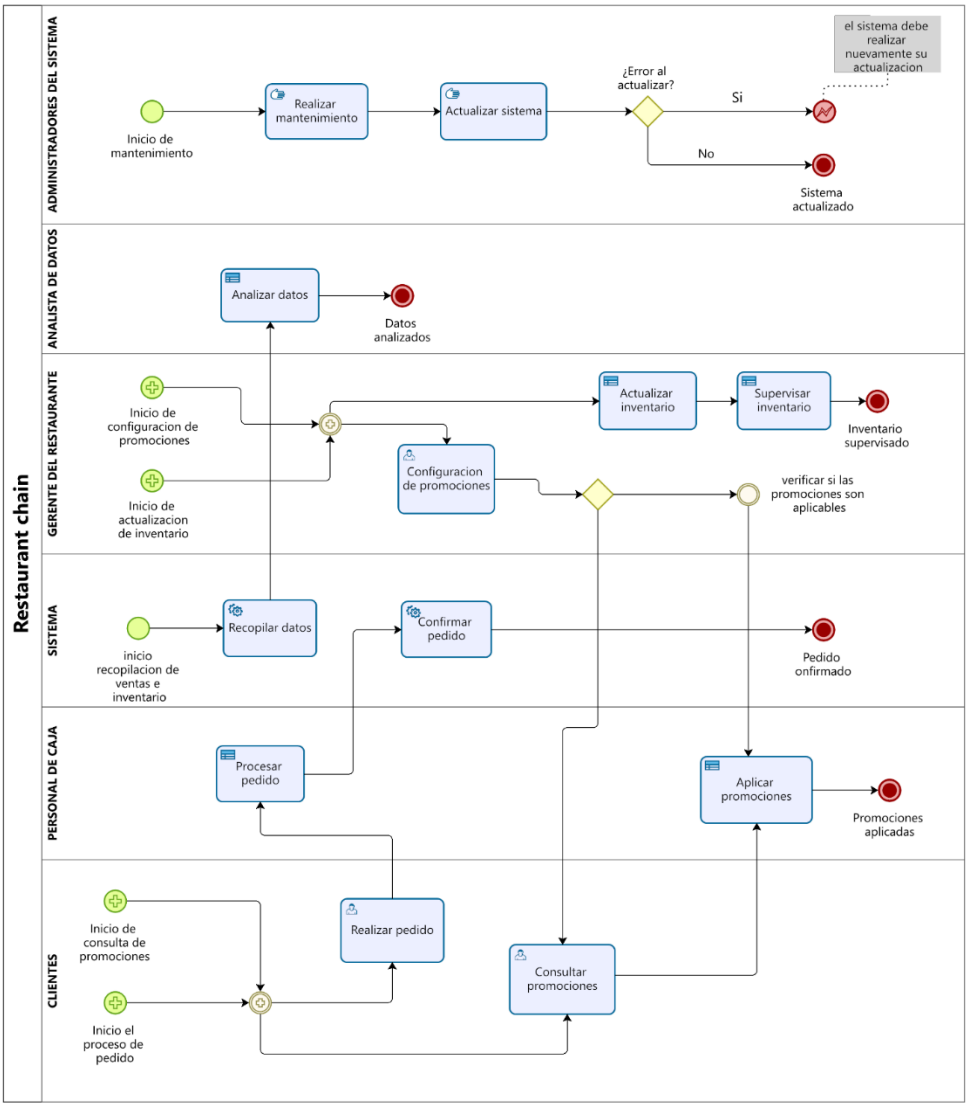
3.1 Component-and-Connector View

3.1.1 View Description

Esta vista describe la interacción entre los microservicios mediante conectores como REST y gRPC.

3.1.2 Business Processes Diagram

Representan flujos clave como la gestión de pedidos, inventarios y pagos, proporcionando un contexto sobre las interacciones principales del sistema.



3.1.3 Primary Presentation

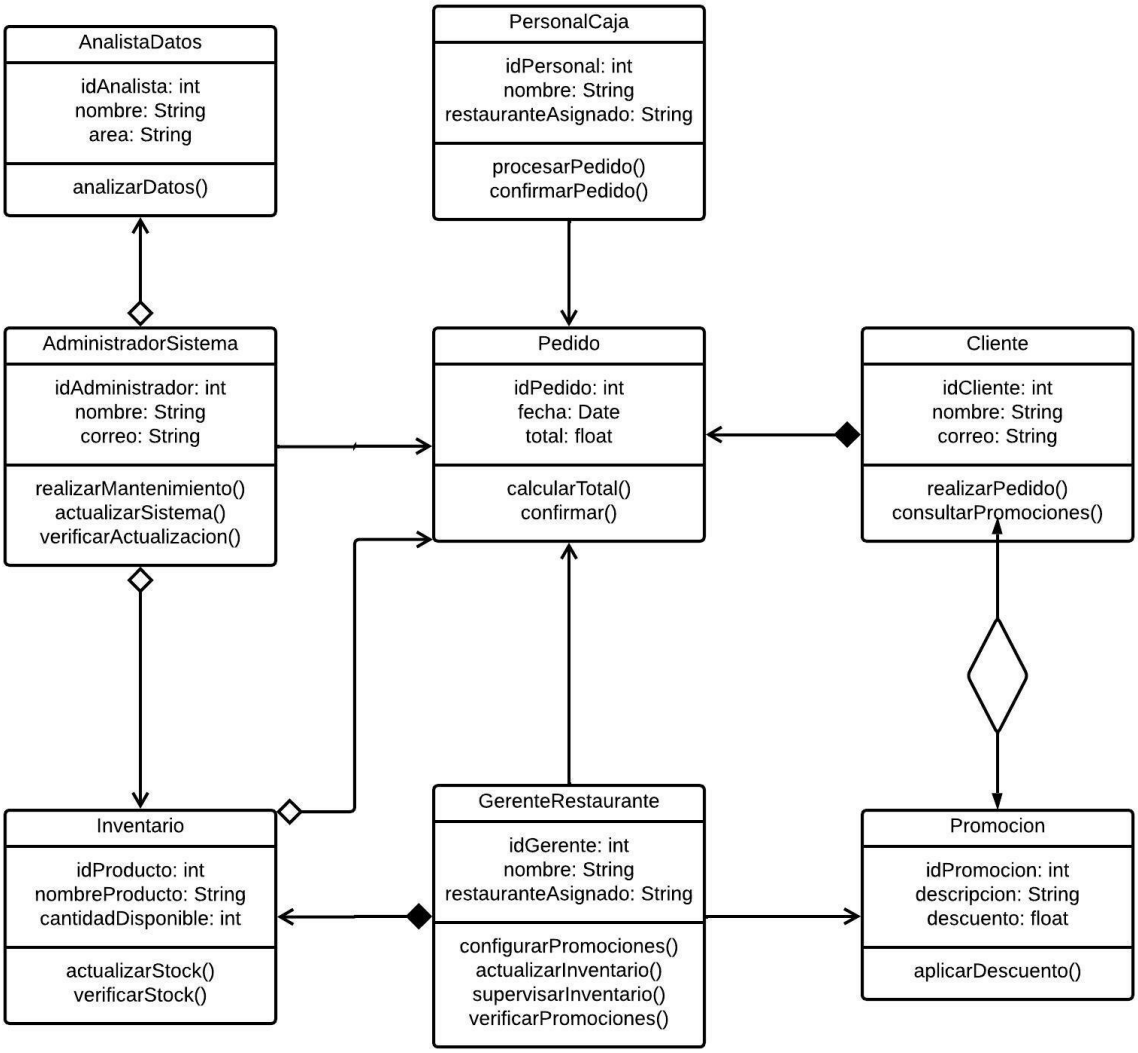
Componentes Principales:

- Base de datos MongoDB.
- Contenedores Docker para cada microservicio.

3.2 Module View

3.2.1 View Description

Esta vista descompone el sistema en módulos como Gestión de Pedidos, Inventarios, y Programas de Fidelización. Describiendo la organización los componentes lógicos. Clientes, pedido, producto. Reflejando atributos y métodos que se pueden utilizar por los microservicios.



3.2.2 View Packet Overview

- **Módulos Principales:**

    1. API Gateway: Punto de acceso central para todas las interacciones.
    2. Módulo de Pedidos: Gestión de pedidos desde múltiples canales.
    3. Módulo de Inventarios: Actualización en tiempo real del inventario.
    4. Plataforma de Análisis: Procesamiento de datos en la nube.

3.2.3 Architecture Background

Cada módulo tiene una API dedicada con endpoints claramente definidos.

4 Relations Among Views

Each of the views specified in Section 3 provides a different perspective and design handle on a system, and each is valid and useful in its own right. Although the views give different system perspectives, they are not independent. Elements of one view will be related to elements of other views, and we need to reason about these relations. For example, a module in a decomposition view may be manifested as one, part of one, or several components in one of the component-and-connector views, reflecting its runtime alter-ego. In general, mappings between views are many to many. Section 4 describes the relations that exist among the views given in Section 3. As required by ANSI/IEEE 1471-2000, it also describes any known inconsistencies among the views.

4.1 General Relations Among Views

El diseño arquitectónico de la cadena de restaurantes se fundamenta en tres puntos de vista principales: el **Viewpoint de Módulo**, el **Viewpoint de Componente y Conector** y el **Viewpoint de Asignación**. Estas vistas están interrelacionadas y ofrecen una perspectiva integral del sistema.

1. **Relación entre el Viewpoint de Módulo y el Viewpoint de Componente y Conector**

    - **Correspondencia:** Los módulos definidos en el Viewpoint de Módulo, que representan capacidades funcionales como la gestión de pedidos, lealtad y promociones, son implementados como componentes específicos en el Viewpoint de Componente y Conector.

    - **Coherencia:** Las relaciones entre módulos (como dependencias o integración) se reflejan directamente en las conexiones de los componentes en el tiempo de ejecución. Por ejemplo, el módulo de *Gestión de Pedidos* se traduce en un componente que interactúa con otros, como el componente de *Base de Datos* o el componente de *API Gateway*.

2. **Relación entre el Viewpoint de Módulo y el Viewpoint de Asignación**

    - **Correspondencia:** Los módulos diseñados son asignados a recursos específicos en el Viewpoint de Asignación, como contenedores Docker, servidores o bases de datos. Esto asegura que las capacidades funcionales de cada módulo tengan los recursos necesarios para su ejecución.

    - **Coherencia:** El diseño modular permite una asignación clara y escalable, asegurando que cada recurso sea utilizado de manera eficiente y que las dependencias entre módulos se mantengan consistentes durante la asignación.

3. **Relación entre el Viewpoint de Componente y Conector y el Viewpoint de Asignación**

- **Correspondencia:** Los componentes definidos en el Viewpoint de Componente y Conector se asignan directamente a los recursos físicos o virtuales en el Viewpoint de Asignación. Por ejemplo, el componente de *Análisis de Datos* puede estar asignado a un clúster en la nube.

- **Coherencia:** Las conexiones entre componentes en tiempo de ejecución se traducen en comunicaciones entre los recursos asignados, como servidores o contenedores, lo que asegura un diseño operativo efectivo.

- **Consistencia entre las Vistas**

La consistencia entre estas vistas se mantiene al garantizar que:

- Las dependencias funcionales y estructurales en el Viewpoint de Módulo se reflejen correctamente en las conexiones del Viewpoint de Componente y Conector.

- Los componentes definidos en el Viewpoint de Componente y Conector sean asignados de manera lógica a los recursos en el Viewpoint de Asignación, respetando las restricciones de rendimiento, seguridad y escalabilidad.

- Las relaciones entre módulos y componentes estén alineadas con las políticas de implementación y diseño general.

- **Inconsistencias Identificadas**

- **Restricciones de Asignación:** En algunos casos, la asignación de recursos podría no reflejar completamente las dependencias entre módulos o componentes debido a limitaciones de infraestructura existente.

- **Actualizaciones en Tiempo de Ejecución:** La comunicación entre componentes puede no adaptarse automáticamente a cambios en la asignación, lo que podría llevar a problemas de latencia o inconsistencias temporales en la sincronización de datos.

4.2 View-to-View Relations

Esta sección describe cómo los elementos definidos en una vista están relacionados con los elementos de otras vistas, asegurando una representación integral y consistente de la arquitectura del sistema para la cadena de restaurantes.

Relaciones entre las Vistas (View-to-View Relations)

- Esta sección describe cómo los elementos definidos en una vista están relacionados con los elementos de otras vistas, asegurando una representación integral y consistente de la arquitectura del sistema para la cadena de restaurantes.

Relación entre el Viewpoint de Módulo y el Viewpoint de Componente y Conector

- **Elementos Relacionados:**
    - Los módulos definidos en el Viewpoint de Módulo se corresponden directamente con componentes en el Viewpoint de Componente y Conector. Por ejemplo:
        - El módulo de *Gestión de Pedidos* se implementa como un componente en tiempo de ejecución que interactúa con otros componentes, como el de *Base de Datos* y el de *API Gateway*.
        - El módulo de *Promociones y Precios* se traduce en un componente que interactúa con la plataforma de análisis en la nube para aplicar reglas optimizadas.
- **Relaciones:**
    - Las relaciones de dependencia entre módulos (por ejemplo, *Gestión de Pedidos* depende de *Inventarios*) se reflejan en las conexiones entre los componentes correspondientes. Estas conexiones pueden incluir protocolos de comunicación como REST APIs o mensajes asíncronos mediante colas de mensajes.

Relación entre el Viewpoint de Módulo y el Viewpoint de Asignación

- **Elementos Relacionados:**
    - Cada módulo del sistema es asignado a recursos físicos o virtuales en el Viewpoint de Asignación. Por ejemplo:
        - El módulo de *Gestión de Pedidos* se asigna a un contenedor Docker que ejecuta el servicio correspondiente en un servidor específico.
        - El módulo de *Fidelización de Clientes* se asigna a un servidor dedicado para garantizar su escalabilidad y disponibilidad.
- **Relaciones:**

- Los recursos asignados en el Viewpoint de Asignación deben cumplir con las necesidades de rendimiento y escalabilidad definidas para cada módulo. Por ejemplo, el módulo de *Análisis de Datos* requerirá un clúster de servidores en la nube con capacidad para manejar grandes volúmenes de datos en tiempo real.

Relación entre el Viewpoint de Componente y Conector y el Viewpoint de Asignación

- **Elementos Relacionados:**
  - Los componentes en el Viewpoint de Componente y Conector se implementan directamente en los recursos especificados en el Viewpoint de Asignación. Por ejemplo:
    - El componente de *API Gateway* se asigna a un servidor balanceador de carga para manejar solicitudes entrantes de manera eficiente.
    - El componente de *Base de Datos* se asigna a un servicio de base de datos en la nube, como Amazon RDS o MongoDB Atlas.
- **Relaciones:**
  - Las conexiones entre componentes en el Viewpoint de Componente y Conector dependen de las configuraciones de red y recursos definidos en el Viewpoint de Asignación. Por ejemplo, la conexión entre el componente de *Gestión de Pedidos* y el de *Inventarios* puede depender de una red interna segura configurada en el recurso de infraestructura.

- **Coherencia entre las Vistas**

  Para mantener la coherencia entre las vistas:

1. Las relaciones entre módulos en el Viewpoint de Módulo deben reflejarse directamente en las conexiones entre componentes en el Viewpoint de Componente y Conector.

2. Los recursos asignados en el Viewpoint de Asignación deben garantizar que los componentes definidos en el Viewpoint de Componente y Conector puedan operar de manera efectiva.

3. Las configuraciones de infraestructura y asignación deben alinearse con las dependencias funcionales definidas en el Viewpoint de Módulo.

5 Referenced Materials

1.	• *SRS for Restaurant Chain*, 25-11-2024.
2.	3.
4.	5.
6.	7.
8.	9.

6 Directory

6.1 Index

    - **Utilizada en:**
        - Sección 3.1.2 (Relación entre el módulo de Gestión de Pedidos y el módulo de Inventarios).
        - Sección 3.2.1 (Vista de Componentes y Conectores: Traducción de dependencias a conexiones).
- **Conexión REST API**
    - **Definida en:**
        - Sección 1.5.2.3 (*Component-and-Connector Viewpoint*: Relación entre componentes).
    - **Utilizada en:**
        - Sección 3.2.1 (Conexión entre el componente de API Gateway y los módulos funcionales).
        - Sección 3.3 (Vista de Asignación: Comunicación entre servidores).
- **Relación de Asignación**
    - **Definida en:**
        - Sección 1.5.3.3 (*Allocation Viewpoint*: Relación entre componentes y recursos).
    - **Utilizada en:**
        - Sección 3.3 (Vista de Asignación: Correspondencia entre componentes y recursos físicos).
- **Propiedades**
- **Nombre del Módulo**
    - **Definida en:**
        - Sección 1.5.1.3 (*Module Viewpoint*: Propiedades de los módulos).
    - **Utilizada en:**
        - Sección 3.1.1 (Identificación de cada módulo funcional).
- **Responsabilidad del Módulo**
    - **Definida en:**
        - Sección 1.5.1.3 (*Module Viewpoint*: Propiedades de los módulos).
    - **Utilizada en:**
        - Sección 3.1.2 (Descripción de las responsabilidades específicas de cada módulo).
- **Tiempo de Respuesta de los Componentes**
    - **Definido en:**
        - Sección 1.5.2.3 (*Component-and-Connector Viewpoint*: Propiedades de los componentes).
    - **Utilizado en:**
        - Sección 3.2.1 (Evaluación del rendimiento de los componentes).
- **Capacidad de los Recursos**
    - **Definida en:**
        - Sección 1.5.3.3 (*Allocation Viewpoint*: Propiedades de los recursos).
    - **Utilizada en:**

- Sección 3.3 (Asignación de componentes a recursos según capacidades requeridas).

6.2 Glossary

Este glosario proporciona una lista de términos y acrónimos especiales utilizados en este SAD, junto con sus definiciones. Si algún término tiene un significado diferente al utilizado en documentos relacionados o en un SAD principal, esta sección explica las razones de dicha discrepancia.

Términos

1. **Microservicio:**
   Unidad de implementación independiente que encapsula una funcionalidad específica del sistema. Los microservicios se comunican entre sí mediante APIs estándar para formar un sistema distribuido.

2. **API Gateway:**
   Componente central que actúa como intermediario entre los clientes y los microservicios. Gestiona solicitudes, autentica usuarios y enrutamiento de mensajes hacia los servicios correspondientes.

3. **Omnicanal:**
   Experiencia del cliente consistente y sin interrupciones a través de múltiples canales de interacción, como aplicaciones móviles, sistemas POS y programas de lealtad.

4. **Base de Datos en la Nube:**
   Sistema de almacenamiento de datos alojado en un servicio de nube, como Amazon RDS o MongoDB Atlas, que permite acceso remoto, escalabilidad y alta disponibilidad.

5. **Plataforma de Análisis:**
   Sistema que procesa grandes volúmenes de datos y utiliza algoritmos de aprendizaje automático (ML) para optimizar decisiones empresariales, como promociones y precios.

6. **Tiempo de Respuesta:**
   Cantidad de tiempo que tarda un sistema o componente en responder a una solicitud del cliente, medido desde que se recibe la solicitud hasta que se envía una respuesta.

7. **Escalabilidad:**
   Capacidad del sistema para manejar un aumento en la carga de trabajo añadiendo

recursos, como servidores o instancias de microservicios, sin comprometer el rendimiento.

1 Acrónimos

1. **API (Application Programming Interface):** Conjunto de definiciones y protocolos que permite que diferentes aplicaciones se comuniquen entre sí.
2. **POS (Point of Sale):** Sistema utilizado para gestionar transacciones de venta en un establecimiento, como terminales de caja registradora.
3. **ML (Machine Learning):** Técnica de análisis de datos que permite a los sistemas aprender y mejorar automáticamente a partir de la experiencia sin ser programados explícitamente.
4. **GDPR (General Data Protection Regulation):** Reglamento de protección de datos de la Unión Europea que regula el tratamiento y la privacidad de los datos personales.
5. **REST (Representational State Transfer):** Estilo arquitectónico para diseñar servicios web que permite la comunicación entre sistemas utilizando protocolos HTTP estándar.
6. **UML (Unified Modeling Language):** Lenguaje de modelado gráfico utilizado para especificar, visualizar y documentar sistemas de software.

1 Discrepancias con un SAD Principal

No se identificaron discrepancias significativas en las definiciones utilizadas en este SAD respecto a documentos relacionados o un SAD principal. Sin embargo, el uso de términos como *Omnicanal* y *Microservicio* ha sido adaptado al contexto de este caso de estudio para reflejar las necesidades específicas de la cadena de restaurantes.

Este glosario tiene como objetivo aclarar términos especializados y facilitar la comprensión del SAD por parte de los diferentes stakeholders. Si se requieren términos adicionales, pueden ser añadidos en futuras versiones del documento.

Term	Definition
software architecture	The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 2003]. "Externally visible" properties refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
view	A representation of a whole system from the perspective of a related set of concerns [IEEE 1471]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.
view packet	The smallest package of architectural documentation that could usefully be given to a stakeholder. The documentation of a view is composed of one or more view packets.
viewpoint	A specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view, and the techniques for its creation and analysis [IEEE 1471]. Identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view.

6.3 Acronym List

API	Application Programming Interface; Application Program Interface; Application Programmer Interface
ATAM	Architecture Tradeoff Analysis Method
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CORBA	Common object request broker architecture
COTS	Commercial-Off-The-Shelf
EPIC	Evolutionary Process for Integrating COTS-Based Systems
IEEE	Institute of Electrical and Electronics Engineers
KPA	Key Process Area
OO	Object Oriented
ORB	Object Request Broker
OS	Operating System
QAW	Quality Attribute Workshop
RUP	Rational Unified Process
SAD	Software Architecture Document
SDE	Software Development Environment
SEE	Software Engineering Environment
SEI	Software Engineering Institute Systems Engineering & Integration Software End Item
SEPG	Software Engineering Process Group
SLOC	Source Lines of Code
SW-CMM	Capability Maturity Model for Software
CMMI-SW	Capability Maturity Model Integrated - includes Software Engineering
UML	Unified Modeling Language

7 Sample Figures & Tables

Table 2: Sample Table

Table Heading	Table Heading	Table Heading	Table Heading
Table Body	Table Body	Table Body	Table Body
Table Body	Table Body	Table Body	Table Body
Table Body	Table Body	Table Body	Table Body
Table Body	Table Body	Table Body	Table Body

Appendix A Appendices

1. **Apéndice A.1: Especificaciones de la API**

   - Contiene detalles técnicos sobre las APIs utilizadas para la integración entre microservicios y otros sistemas externos (por ejemplo, sistemas POS, aplicaciones móviles y plataformas de análisis en la nube).

   - **Referencia en el documento principal:** Se mencionan en las vistas de componentes y conexiones, donde se describen las interacciones entre los microservicios y las plataformas externas.

2. **Apéndice A.2: Diagramas de Arquitectura de Microservicios**

   - Incluye diagramas visuales detallados de la arquitectura de microservicios, mostrando la distribución y la relación entre los distintos componentes, como el *API Gateway*, *Gestión de Pedidos*, *Base de Datos*, etc.

   - **Referencia en el documento principal:** Se hace referencia a estos diagramas en las secciones de vistas de módulos y vistas de componentes y conectores.

3, **Apéndice A.3: Políticas de Seguridad y Autenticación**

   - Documento detallado sobre las políticas de seguridad, incluyendo autenticación multifactor, gestión de sesiones y políticas de acceso a datos.

   - **Referencia en el documento principal:** Se menciona en la sección sobre la vista de componentes y conectores, donde se describe la seguridad de las interacciones entre los microservicios.

4. **Apéndice A.4: Información sobre el Proceso de Escalabilidad**

   - Detalles sobre las estrategias y tecnologías implementadas para garantizar que el sistema pueda escalar de manera eficiente conforme crece la cadena de restaurantes.

   - **Referencia en el documento principal:** Se hace referencia en las vistas de asignación, donde se describe cómo se asignan los recursos para asegurar la escalabilidad.