

## **Arquitectura de Software**



### **Taller #2**

#### **Integrantes**

Nicolás David Rincón Ballesteros

Nicolas Quintana Cuartas

Sebastián Isaza Jiménez

**Profesor:** Andrés Armando Sánchez Martín

**Pontificia Universidad Javeriana**

Bogotá

28 de septiembre, 2024

Que es Pub-Sub .....	4
Definiciones .....	4
Características .....	5
Historia y evolución .....	6
1. RabbitMQ .....	6
2. .Net .....	6
3. Vue.js .....	7
4. SQL-server .....	7
5. Publicador Subscriptor .....	7
Casos de uso y caso de aplicación: .....	8
1. RabbitMQ con el estilo Publicador-Subscriptor (Pub/Sub) .....	8
2. Framework Frontend Vue.js .....	9
3. Backend con .NET .....	9
4. Base de Datos SQL Server .....	9
Ventajas y Desventajas : .....	10
1. RabbitMQ .....	10
2. Vue.js .....	11
3. .NET .....	11
4. SQL Server .....	12
5. Publicador / Subscriptor .....	13
Popularidad Stack .....	14
Matriz de análisis .....	14
Principios SOLID vs Temas .....	14
Single Responsibility : .....	15
Open/Closed : .....	15
Liskov Substitution : .....	16
Interface Segregation : .....	17
Dependency Inversion : .....	17
Atributos de Calidad vs Temas .....	18

Funcionalidad :	18
Fiabilidad :	19
Usabilidad :	19
Eficiencia :	20
Mantenibilidad :	20
Portabilidad :	21
Tácticas vs Temas.....	21
Mercado laboral vs Temas .....	25
Demanda (Media): .....	26
Salario promedio (Alto):.....	26
Nuevas ofertas (Media): .....	27
Competencia (Media):.....	27
Patrones laboral vs Temas.....	28
Trabajo remoto (Medio): .....	28
Metodologías ágiles (Alto): .....	29
Contratos freelance :	29
Roles de DevOps :	30
Diagramas .....	30
Alto Nivel .....	30
Diagrama Contexto .....	33
Diagrama de Contenedores.....	34
Dinámico C4 .....	36
Despliegue C4 .....	37
Diagrama de paquetes UML.....	38

## Que es Pub-Sub

Pub-Sub (Publicación-Subscripción) es un patrón de diseño de mensajería que permite que los diferentes componentes de un sistema se comuniquen de manera asíncrona y desacoplada. Este patrón se basa en la idea de que un publicador emite mensajes o eventos, mientras que uno o más suscriptores los reciben si están interesados en ellos.

Este enfoque es especialmente útil en arquitecturas de microservicios, donde múltiples servicios pueden estar interesados en un mismo evento. Por ejemplo, en un sistema donde se crea una nueva cuenta de usuario, varios servicios como auditoría, notificaciones y marketing pueden estar interesados en procesar ese evento sin que el publicador necesite saber cuántos o cuáles servicios lo están consumiendo.

Características clave del Pub-Sub:

- **Comunicación desacoplada:** El publicador no necesita saber quién o cuántos suscriptores están recibiendo el mensaje, lo que permite añadir o eliminar suscriptores sin afectar al sistema.
- **Entrega a múltiples suscriptores:** Un solo mensaje puede ser distribuido a múltiples consumidores, lo que es ideal para eventos que deben ser procesados por varios servicios.
- **Asincronía:** Los suscriptores no necesitan estar activos o conectados al momento de la publicación del mensaje. Los mensajes pueden ser almacenados en colas hasta que los suscriptores estén disponibles para consumirlos.
- **Uso eficiente de recursos:** Algunos sistemas, como RabbitMQ, optimizan el uso de memoria al almacenar una única copia del mensaje y permitir que múltiples colas lo referencien, en lugar de duplicarlo.

## Definiciones

**RabbitMQ:** Broker de mensajería RabbitMQ es una pieza clave dentro del desarrollo de aplicaciones que facilita la comunicación entre diversos sistemas. Se trata de un broker de mensajería que permite el envío, enrutamiento y almacenamiento de mensajes de manera eficiente. Este componente es capaz de funcionar de manera autónoma, facilitando la interacción entre diferentes servicios o microservicios, y es altamente utilizado para gestionar colas de tareas y eventos en arquitecturas distribuidas.

**Vue.js:** Framework JavaScript para la creación de interfaces de usuario Vue.js es un framework progresivo de JavaScript que se utiliza para construir interfaces de usuario. Se basa en tecnologías web estándar como HTML, CSS y JavaScript, y adopta un enfoque declarativo basado en componentes, lo que permite a los desarrolladores crear interfaces interactivas y reactivas de manera eficiente. Vue.js destaca por su simplicidad, flexibilidad y la facilidad con la que se puede integrar en proyectos nuevos o existentes.

**.NET:** Plataforma de desarrollo de aplicaciones multiplataforma .NET es una plataforma de código abierto que da un entorno para crear aplicaciones de escritorio, web y móviles, capaces de ejecutarse de forma nativa en cualquier sistema operativo. Ofrece un conjunto completo de herramientas, bibliotecas y lenguajes que soportan el desarrollo de software moderno, escalable y de alto rendimiento. Gracias a su enfoque multiplataforma y al soporte para lenguajes como C#, F# y VB.NET, .NET es una opción ideal para proyectos de gran envergadura.

**SQL Server:** Plataforma de integración y administración de datos SQL Server es una plataforma de alto rendimiento que se utiliza para la gestión de bases de datos relacionales. Además de sus capacidades de almacenamiento y consulta de datos, SQL Server incluye características avanzadas de integración, como paquetes de extracción, transformación y carga (ETL), que permiten el procesamiento eficiente de grandes volúmenes de datos. Estas capacidades lo convierten en una herramienta indispensable para soluciones de almacenamiento de datos y análisis empresarial.

## Características

Nuestros temas de elección en donde utilizábamos 5 temas para construir una aplicación de publicador - subscriptor

1. **RabbitMQ:** Esencial en la construcción de aplicaciones distribuidas. RabbitMQ actúa como un bróker de mensajería, facilitando la comunicación entre sistemas mediante el envío de mensajes de forma confiable y asíncrona. Su capacidad para manejar flujos de mensajes distribuidos lo convierte en una pieza clave en entornos escalables y de alta disponibilidad.
2. **Vue.js:** Framework JavaScript que permite desarrollar Single Page Applications (SPA) interactivas y dinámicas. Basado en HTML, CSS y JavaScript, Vue.js ofrece un enfoque declarativo y modular, facilitando la creación de aplicaciones modernas, altamente reactivas y fáciles de mantener, ideal para mejorar la experiencia del usuario al cargar contenido sin necesidad de recargar toda la página.

3. **.NET:** Plataforma versátil y de código abierto diseñada para el desarrollo de aplicaciones web, de escritorio y móviles. .NET permite la creación de software multiplataforma, proporcionando herramientas robustas, bibliotecas extensas y soporte para múltiples lenguajes, ideal para proyectos de alto rendimiento y escalabilidad.
4. **SQL Server:** Potente sistema de gestión de bases de datos relacionales, optimizado para el procesamiento eficiente de transacciones y análisis de grandes volúmenes de datos. Destaca por su integración nativa con herramientas de inteligencia de negocios (BI) como SSIS y SSRS, y su compatibilidad con entornos de Big Data.
5. **Modelo Publicador / Subscriptor:** Patrón de comunicación asíncrono ampliamente utilizado en arquitecturas de microservicios y sistemas distribuidos. Este modelo permite la publicación de eventos a múltiples suscriptores, lo que facilita el manejo de eventos en tiempo real y la escalabilidad del sistema, mejorando la flexibilidad y la eficiencia en la comunicación entre componentes.

## Historia y evolución

### 1. RabbitMQ

Creado en 2007 por la empresa LShift, liderada por Alexis Richardson y Matthias Radestock, como un mensaje bróker que implementara el protocolo AMQP (Advanced Message Queuing Protocol), diseñado para la interoperabilidad entre diferentes sistemas. En 2010, RabbitMQ fue adquirido por VMware como parte de su estrategia de expansión en la infraestructura de nube y software. En 2012, VMware lanzó Pivotal Software, bajo la cual RabbitMQ se continuó desarrollando. Medida que las arquitecturas distribuidas y los microservicios se volvieron más populares, RabbitMQ evolucionó para soportar múltiples protocolos de mensajería como STOMP (Streaming Text Oriented Messaging Protocol) y MQTT (Message Queuing Telemetry Transport). Esta flexibilidad permitió que RabbitMQ se adaptara a una amplia gama de aplicaciones.

### 2. .Net

Microsoft buscaba una manera de competir con Java y otros entornos de desarrollo multiplataforma. Esto llevó al desarrollo de .NET Framework, anunciado en 2000 por Bill Gates como una plataforma integral para construir aplicaciones web, de escritorio y móviles en entornos Windows. En 2002, se lanzó la primera versión de .NET Framework 1.0, que incluía el Common Language Runtime (CLR), un entorno de ejecución que soportaba múltiples lenguajes de programación (C#, VB.NET, etc.). Esta versión permitió la creación de aplicaciones robustas sobre el sistema operativo Windows. Microsoft lanzó .NET Core,

una versión de .NET diseñada para ser multiplataforma (Windows, Linux, y macOS) y de código abierto, lo que marcó un cambio significativo en la estrategia de Microsoft para hacer .NET más accesible y adaptable a las nuevas arquitecturas en la nube y microservicios.

### 3. Vue.js

Fue creado en 2014 por Evan You, un exdesarrollador de Google. Frustrado por la complejidad de otros frameworks de JavaScript como Angular, decidió crear una herramienta más accesible y fácil de aprender. En 2016, Vue 2 fue lanzado, consolidando a Vue como una de las opciones más populares en el desarrollo frontend. Esta versión trajo mejoras en rendimiento, una mejor gestión de los componentes, y una sintaxis más refinada para trabajar con datos reactivos. En 2020, se lanzó Vue 3, que introdujo la Composition API, permitiendo una mayor flexibilidad en la creación y organización de componentes. Vue 3 también se centró en mejorar el rendimiento y la escalabilidad, haciéndolo ideal para aplicaciones más grandes y complejas. Además, Vue.js es una de las opciones favoritas para desarrollar interfaces de usuario por su simplicidad y rendimiento. progresivo, lo que significa que los desarrolladores podían adoptarlo en proyectos existentes de manera incremental.

### 4. SQL-server

Fue lanzado en 1989, fruto de una colaboración entre Microsoft, Sybase, y Ashton-Tate. Inicialmente, SQL Server estaba basado en la tecnología de Sybase, diseñado para correr en sistemas operativos OS/2. Sin embargo, en 1994, Microsoft se separó de Sybase y continuó desarrollando SQL Server de manera independiente. A principios de los 2000, con SQL Server 2000 y posteriormente SQL Server 2005, Microsoft comenzó a incluir herramientas para Business Intelligence (BI), como SQL Server Analysis Services (SSAS), SQL Server Reporting Services (SSRS), y SQL Server Integration Services (SSIS), posicionando a SQL Server como una solución integral para bases de datos y análisis de datos empresariales. En años recientes, SQL Server ha adoptado características modernas como Big Data Clusters, integraciones con Azure para la nube, y mejoras en el manejo de grandes volúmenes de datos con tecnologías como PolyBase.

### 5. Publicador Subscriber

Los primeros versiones de Pub/Sub eran modelos principalmente de comunicación era punto a punto, cada emisor debía saber a quién enviar un mensaje. Aunque era efectivo en pequeños entornos, presentaba problemas de escalabilidad y flexibilidad a medida que las redes crecían y la cantidad de nodos aumentaba. Con el crecimiento de los sistemas

distribuidos, surgió la necesidad de un sistema más flexible y escalable que permitiera el desacoplamiento entre emisores y receptores de mensajes. El modelo de publicador-suscriptor surgió como una respuesta a esta necesidad, permitiendo que los emisores (publicadores) no tuvieran que conocer a los receptores (suscriptores) de los mensajes.

En los años 2010 surge las conexiones en la nube usando el modelo publicador suscriptor aquí surgen Amazon SNS y Google Pub/sub donde servicios en la nube que permiten a los desarrolladores integrar fácilmente la arquitectura publicador-suscriptor en sus aplicaciones. También surgen arquitectura como La arquitectura basada en eventos se convirtió en un enfoque popular para desarrollar sistemas escalables y desacoplados, especialmente en microservicios. En este modelo, los componentes del sistema se comunican a través de eventos en lugar de llamadas directas, lo que reduce la dependencia entre ellos y aumenta la flexibilidad.

## Casos de uso y caso de aplicación:

### 1. RabbitMQ con el estilo Publicador-Subscriber (Pub/Sub)

- **Microservicios:** RabbitMQ se usa para coordinar mensajes entre diferentes microservicios en aplicaciones distribuidas. Este patrón es útil cuando tienes servicios que deben reaccionar a eventos sin depender entre sí.
- **Notificaciones en tiempo real:** Aplicaciones como sistemas de mensajería o notificaciones push en tiempo real pueden beneficiarse de RabbitMQ con Pub/Sub. Un servicio publica un mensaje y varios suscriptores reciben actualizaciones instantáneamente.
- **Integración entre sistemas heterogéneos:** Cuando múltiples sistemas necesitan interactuar, RabbitMQ puede usarse para enviar mensajes a diferentes consumidores sin acoplarlos directamente.

### Casos de Aplicación:

- **Netflix:** Usa RabbitMQ para gestionar colas de trabajos asíncronos y notificaciones, lo que les permite escalar su infraestructura para millones de usuarios.
- **Uber:** Utiliza RabbitMQ para eventos y manejo de transacciones distribuidas en tiempo real, mejorando la coordinación entre diferentes servicios en su plataforma.
- **Aplicaciones IoT:** RabbitMQ es común en aplicaciones de IoT para gestionar mensajes entre dispositivos y servidores.



## 2. Framework Frontend Vue.js

- **SPAs (Single Page Applications):** Vue.js es ideal para aplicaciones de una sola página que necesitan una experiencia fluida, rápida y reactiva, como paneles de control, sistemas de gestión o aplicaciones de redes sociales.
- **Aplicaciones web interactivas:** En sitios donde la interfaz de usuario necesita actualizarse frecuentemente sin recargar toda la página, como plataformas de e-commerce o foros.
- **Sistemas modulares y escalables:** Vue.js se puede usar en grandes proyectos con una arquitectura modular, permitiendo componentes reutilizables y mantenibles.

### Casos de Aplicación:

- Alibaba: Utiliza Vue.js en sus interfaces de usuario, aprovechando su capacidad para construir aplicaciones de alto rendimiento.
- Behance: Utiliza Vue.js en su plataforma para mejorar la experiencia de usuario y manejar interfaces de usuario altamente dinámicas.
- Xiaomi: La empresa lo usa para su tienda en línea, asegurando una interfaz rápida y moderna.

## 3. Backend con .NET

- **Aplicaciones empresariales a gran escala:** .NET es ideal para crear sistemas robustos y escalables, como ERP, CRM o plataformas de gestión empresarial.
- **APIs RESTful:** Utilizado para desarrollar APIs que conectan diferentes servicios y aplicaciones, con soporte robusto para manejo de cargas altas.
- **Sistemas de facturación:** .NET se usa frecuentemente en aplicaciones de gestión financiera, donde la seguridad y la precisión son clave.

### Casos de Aplicación:

- Stack Overflow: El backend de Stack Overflow está construido en .NET, gestionando millones de solicitudes diarias.
- Microsoft: Obviamente, muchas aplicaciones internas y productos de Microsoft están desarrollados en .NET.
- GoDaddy: Utiliza .NET para manejar la infraestructura de su servicio de dominios.

## 4. Base de Datos SQL Server

- **Sistemas OLTP (Procesamiento de transacciones en línea):** SQL Server se usa en aplicaciones transaccionales como plataformas bancarias, donde se necesita confiabilidad, integridad de datos y alto rendimiento.
- **Aplicaciones de análisis de datos:** SQL Server, combinado con SSIS y SSRS, se utiliza en aplicaciones de análisis de datos y BI (Business Intelligence).
- **Aplicaciones de comercio electrónico:** Manejo de inventarios, gestión de usuarios, pedidos y pagos en plataformas de e-commerce.

### Casos de Aplicación:

- Domino's Pizza: Utiliza SQL Server para manejar su sistema de pedidos en línea, procesando transacciones en tiempo real y gestionando grandes volúmenes de datos.
- UPS: Usa SQL Server para su sistema de logística, optimizando el seguimiento de paquetes y gestión de rutas.
- Stack Overflow: Además de usar .NET, también se apoya en SQL Server para almacenar millones de registros de preguntas, respuestas y usuarios.

## Ventajas y Desventajas:

### 1. RabbitMQ

#### Ventajas

- Facilita la comunicación entre servicios sin que dependan unos de otros, lo que permite flexibilidad y escalabilidad en sistemas distribuidos.
- Puede manejar grandes volúmenes de mensajes, siendo una solución robusta para aplicaciones que requieren procesamiento en paralelo.
- Ofrece confirmación de entrega de mensajes y permite reintentar en caso de fallos.
- Compatible con varios patrones de mensajería (cola de mensajes, pub/sub, routing, etc.), lo que lo hace versátil.

#### Desventajas

- **Complejidad:** La implementación de RabbitMQ puede requerir configuración avanzada y manejo de errores, lo que introduce complejidad en la arquitectura.
- **Sobrecarga de mensajes:** Si no se configura adecuadamente, puede haber problemas de acumulación de mensajes en colas, afectando el rendimiento.

- **Requiere monitoreo:** La administración y monitoreo continuo de las colas de mensajes es esencial para asegurar un rendimiento óptimo.

## 2. Vue.js

### Ventajas

- **Curva de aprendizaje rápida:** Comparado con otros frameworks, Vue.js es fácil de aprender, especialmente para desarrolladores que ya conocen JavaScript.
- **Flexibilidad:** Puede integrarse gradualmente en proyectos existentes o ser usado para aplicaciones grandes gracias a su enfoque modular.
- **Reactivo:** La actualización automática del DOM (Document Object Model) cuando cambian los datos mejora la experiencia del usuario con aplicaciones más rápidas y fluidas.
- **Documentación extensa:** Vue.js tiene una de las mejores documentaciones, facilitando la resolución de problemas y la implementación.

### Desventajas

- **Comunidad más pequeña:** Aunque está creciendo, la comunidad de Vue.js es más pequeña que la de React o Angular, lo que puede hacer que algunos paquetes y bibliotecas estén menos desarrollados.
- **Soporte limitado por grandes empresas:** Vue.js no cuenta con el respaldo de grandes corporaciones tecnológicas como Angular (Google) o React (Facebook), lo que puede generar dudas en algunos proyectos corporativos.
- **Integración con SEO:** Las aplicaciones basadas en Vue.js, como las SPAs, pueden tener dificultades con el SEO si no se implementan soluciones como SSR (Server-Side Rendering).

## 3. .NET

### Ventajas

- Ofrece un alto rendimiento, seguridad avanzada y soporte multiplataforma, lo que lo convierte en una opción robusta para desarrolladores de sistemas empresariales.
- **Multiplataforma:** Con .NET Core, ahora es posible desarrollar aplicaciones en Windows, Linux y macOS, lo que brinda más opciones y flexibilidad para los equipos de desarrollo.

- **Rendimiento:** .NET ofrece un rendimiento sólido, siendo adecuado para aplicaciones que manejan grandes cargas de trabajo, como sistemas empresariales y de alta disponibilidad.
- **Gran ecosistema:** Cuenta con un amplio ecosistema de herramientas, bibliotecas y soporte tanto de la comunidad como de Microsoft.
- **Seguridad:** Incluye características avanzadas de seguridad, como autenticación y manejo de identidades, lo que facilita la creación de aplicaciones seguras

### Desventajas

Puede tener una curva de aprendizaje pronunciada y costos de licencia asociados en entornos Windows.

- **Curva de aprendizaje pronunciada:** Aunque es poderoso, puede ser más complejo para desarrolladores sin experiencia previa en .NET o en el ecosistema de Microsoft.
- **Costos de licencia (Windows Server):** Aunque .NET Core es open-source, si se implementa en entornos Windows, puede haber costos asociados a licencias de Windows Server y SQL Server.
- **Peso de las aplicaciones:** Las aplicaciones desarrolladas en .NET pueden ser más pesadas en términos de consumo de recursos comparadas con otras tecnologías ligeras.

## 4. SQL Server

### Ventajas

- Se caracteriza por su fiabilidad, herramientas de BI y sólidas características de seguridad, lo que lo hace adecuado para entornos empresariales críticos.
- **Fiabilidad y rendimiento:** SQL Server es reconocido por su capacidad de manejar grandes volúmenes de datos y ofrecer un rendimiento consistente y fiable en entornos empresariales.
- **Soporte empresarial:** Es ampliamente utilizado en grandes corporaciones y cuenta con el respaldo de Microsoft, lo que garantiza actualizaciones regulares y soporte.
- **Herramientas de BI (Business Intelligence):** Incluye herramientas como SSRS (Reporting Services) y SSIS (Integration Services), que facilitan la integración y análisis de datos.
- **Seguridad:** Ofrece sólidas características de seguridad, como cifrado y control de acceso basado en roles, lo que lo hace adecuado para aplicaciones críticas.

## Desventajas

- Los costos de licencia pueden ser elevados y su adopción puede estar limitada a entornos Windows, lo que reduce su flexibilidad en ciertas organizaciones.
- **Costos de licencia:** SQL Server no es una base de datos gratuita (aunque tiene una versión Express limitada), lo que puede representar un costo significativo para algunas empresas.
- **Comunidad más cerrada:** A diferencia de las bases de datos open-source como MySQL o PostgreSQL, SQL Server tiene una comunidad más cerrada y depende en gran medida del soporte de Microsoft.
- **Requiere Windows (en ciertas versiones):** Algunas versiones de SQL Server están optimizadas para ejecutarse en entornos Windows, lo que puede limitar su adopción en entornos basados en Linux.

## 5. Publicador / Subscriber

### Ventajas:

- **Desacoplamiento:** Los publicadores no necesitan saber quiénes son los suscriptores ni cuántos son. Esto facilita la evolución independiente de los componentes, permitiendo añadir nuevos suscriptores sin modificar a los publicadores.
- **Escalabilidad:** Es altamente escalable, ya que múltiples suscriptores pueden recibir mensajes simultáneamente sin sobrecargar al publicador. Esto es ideal para sistemas distribuidos y de gran escala.
- **Flexibilidad:** Soporta una arquitectura orientada a eventos y permite que los mensajes se publiquen cuando ocurren eventos relevantes, desencadenando acciones en los suscriptores. Esto es útil en sistemas donde los eventos ocurren de manera impredecible.
- **Asincronía:** Los publicadores y suscriptores funcionan de manera asíncrona. Esto significa que los mensajes pueden procesarse en diferentes momentos, lo que mejora el rendimiento al evitar bloqueos y tiempos de espera innecesarios.

### Desventajas:

- **Complejidad:** A pesar de su flexibilidad, la implementación y gestión de un sistema de publicador-suscriptor puede ser compleja, especialmente en lo que respecta a la coordinación de mensajes y la gestión de suscripciones.

- **Garantía de entrega:** Dependiendo del sistema utilizado, puede ser difícil garantizar que todos los suscriptores reciban todos los mensajes. Algunos sistemas ofrecen solo entrega “al mejor esfuerzo”, lo que significa que podrían perderse mensajes en situaciones de alta carga o fallos.
- **Latencia:** En sistemas distribuidos, la latencia puede aumentar debido a la naturaleza asíncrona de la comunicación y el uso de intermediarios (brokers). Esto puede ser un problema en aplicaciones donde la baja latencia es crítica.
- **Orden de los mensajes:** Garantizar el orden de los mensajes en todos los suscriptores puede ser un desafío, especialmente si los mensajes se distribuyen en un sistema distribuido o con múltiples brokers. Esto puede complicar el manejo de datos secuenciales o dependientes.

## Popularidad Stack

En el ámbito del desarrollo de software, se observa una combinación muy popular en artículos y repositorios de GitHub que incluye tecnologías como .NET y C#, ambas de altísima popularidad y demanda entre los desarrolladores. Al mismo tiempo, Vue.js y JavaScript han experimentado un crecimiento constante en popularidad y demanda, consolidándose como herramientas clave para el desarrollo front-end. Por otro lado, RabbitMQ, una herramienta de gestión de colas se ha posicionado como la segunda más popular en su categoría y mantiene una alta demanda en el mercado. Sin embargo, SQL Server Enterprise Edition ha perdido algo de popularidad y demanda en los últimos años, aunque sigue siendo una de las bases de datos más utilizadas en entornos empresariales.

## Matriz de análisis

### *Principios SOLID vs Temas*

La siguiente tabla relaciona los principios SOLID con tecnologías clave como SQL Server EE, Vue.js, RabbitMQ y .NET, además del patrón de diseño Publicador/Subscriptor. Los principios SOLID son fundamentales para el desarrollo de software mantenible y escalable. Cada tecnología es evaluada en términos de su alineación con estos principios: Responsabilidad Única (Single Responsibility), Abierto/Cerrado (Open/Closed), Sustitución de Liskov (Liskov Substitution), Segregación de Interfaces (Interface Segregation) y Inversión de Dependencias (Dependency Inversion). Esta clasificación proporciona una idea de qué tan bien cada tecnología soporta estos principios en la práctica, lo que facilita la toma de decisiones en el diseño y arquitectura del software.

	SQL Server EE	Vue.js	RabbitMQ	.NET	Publicador/ Subcriptor
Single Responsibility	Medio	Bajo	Alto	Alto	Alto
Open/Closed	Bajo	Medio	Medio	Alto	Medio
Liskov Substitution	Bajo	Medio	Bajo	Alto	Medio
Interface Segregation	Bajo	Alto	Alto	Alto	Alto
Dependency Inversion	Medio	Bajo	Alto	Alto	Alto

#### Single Responsibility :

**SQL Server:** está diseñado para gestionar bases de datos, lo que significa que su única responsabilidad es almacenar, recuperar y administrar datos de manera eficiente. Esto permite que las aplicaciones se enfoquen en la lógica de negocio, delegando la gestión de datos al sistema de gestión de bases de datos.

**Vue.js:** Está diseñado para manejar la interfaz de usuario de aplicaciones web, lo que le confiere una clara responsabilidad. Cada componente de Vue puede enfocarse en una función específica de la UI, promoviendo un diseño modular y limpio.

**RabbitMQ:** Está diseñado para el enrutamiento de mensajes y la gestión de colas. Su única responsabilidad es manejar la comunicación entre diferentes componentes de la aplicación de forma eficiente, sin preocuparse por cómo se procesan esos mensajes.

**.NET:** Permite la creación de aplicaciones modulares donde cada componente o servicio puede tener una responsabilidad única. Esto promueve un diseño limpio y fácil de mantener.

**Publicador:** Su única responsabilidad es emitir mensajes. No le importa quién los recibe o cómo se procesan.

**Suscriptor:** Solo se encarga de escuchar mensajes y procesarlos según su lógica. Mantiene su responsabilidad enfocada.

#### Open/Closed :

**SQL Server:** permite una funcionalidad ampliada mediante el uso de procedimientos almacenados, sin embargo, algunas modificaciones en la estructura de la base de datos

pueden requerir cambios en las aplicaciones que la utilizan y es posible que estos cambios no estén completamente abiertos.

**Vue.js:** Este permite la creación de componentes que se pueden extender para modificar un código ya implementado. Permitiendo así que la aplicación pueda evolucionar y se adapte a nuevos requisitos sin afectar la funcionalidad.

**RabbitMQ:** puede ser extendido mediante la adición de nuevos tipos de intercambios y colas sin modificar la configuración existente. Esto permite que las aplicaciones se adapten a nuevos requisitos de mensajería fácilmente.

**.NET:** Puede ser extendida mediante la creación de nuevos módulos y servicios sin necesidad de modificar el código existente, utilizando principios de programación orientada a objetos.

**Publicador:** Puede extenderse para publicar diferentes tipos de mensajes, pero puede ser necesario modificar el código si cambian las colas o los canales de publicación.

**Suscriptor:** Se puede extender para procesar nuevos tipos de mensajes sin alterar el comportamiento existente, pero puede requerir modificaciones dependiendo de los nuevos formatos de datos.

[Liskov Substitution :](#)

**SQL Server:** Se puede sustituir diferentes versiones de este servidor sin problemas significativos. No obstante, las funciones más avanzadas pueden afectar el comportamiento del servidor, por lo cual se recomienda ser cuidadoso con las sustituciones.

**RabbitMQ:** es sustituible por otras tecnologías de mensajería (como Kafka) siempre que las nuevas implementaciones mantengan la misma interfaz y comportamiento esperado, lo que facilita la transición.

**Vue.js:** Pueden ser sustituidos siempre que sigan la misma interfaz y estructura. Pero si algún componente depende de ciertas implementaciones que no funcione de la manera esperada.

**.NET:** soporta la herencia y el polimorfismo, permitiendo que las clases derivadas puedan ser utilizadas como sustitutos de sus clases base sin problemas, lo que fomenta la reutilización del código.

**Publicador:** La sustitución de publicadores puede ser viable, pero depende de la infraestructura (por ejemplo, RabbitMQ, Kafka) y puede variar en comportamiento.



**Suscriptor:** Los suscriptores pueden sustituirse más fácilmente, siempre que respeten la interfaz de suscripción y el contrato de los mensajes que reciben.

#### Interface Segregation:

**SQL Server:** Proporciona una interfaz bien definida para interactuar con las bases de datos. Permitiendo así que las diferentes aplicaciones interactúen con estrictamente lo que necesitan.

**Vue.js:** Permite a los desarrolladores crear componentes que solo implementan las funcionalidades necesarias, evitando el acoplamiento con otras partes del sistema que no son relevantes para su función específica.

**RabbitMQ:** Proporciona interfaces específicas para diferentes operaciones de mensajería, permitiendo a los desarrolladores utilizar solo lo que necesitan sin estar atados a funcionalidades adicionales innecesarias.

**.NET:** Permite crear interfaces específicas que definen solo los métodos necesarios, evitando que las clases que implementan estas interfaces dependan de métodos que no utilizan.

**Publicador:** El publicador solo necesita conocer la interfaz de la cola/mensajería (como RabbitMQ o Kafka), sin estar acoplado a interfaces innecesarias.

**Suscriptor:** El suscriptor implementa únicamente las interfaces necesarias para recibir y procesar mensajes, respetando el principio de no depender de métodos que no usa.

#### Dependency Inversion:

**SQL Server:** Es usable a través de diversas capas de abstracción, lo que permite que se pueda interactuar con él sin la necesidad de depender de una implementación específica. Facilitando así el posible cambio de DB.

**Vue.js:** Fomenta el uso de patrones de diseño que permiten la inversión de dependencias, lo que facilita la integración con otras librerías y la sustitución de componentes sin afectar la lógica de la aplicación.

**RabbitMQ:** permite a las aplicaciones interactuar a través de APIs y bibliotecas de cliente, lo que facilita el cambio de sistemas de mensajería sin impactar la lógica de la aplicación.

**.NET:** Soporta la inversión de dependencias a través de patrones de diseño como la Inyección de Dependencias, lo que permite desacoplar las implementaciones y facilita la prueba y el mantenimiento del código.

**Publicador:** Depende de abstracciones (interfaces de mensajería), lo que le permite publicar mensajes a diferentes sistemas de mensajería sin cambiar su código.

**Suscriptor:** También depende de abstracciones para recibir los mensajes, permitiendo que el sistema de mensajería sea fácilmente sustituible sin afectar el código del suscriptor.

## Atributos de Calidad vs Temas

La siguiente tabla compara diversos atributos de calidad como funcionalidad, fiabilidad, usabilidad, eficiencia, mantenibilidad y portabilidad en tecnologías como SQL Server EE, Vue.js, RabbitMQ y .NET. Estos atributos son esenciales para evaluar el rendimiento general y la capacidad de las tecnologías en diferentes escenarios de desarrollo. Cada tecnología es clasificada según su capacidad para satisfacer estos atributos, proporcionando una visión general sobre su adecuación en términos de calidad de software. Esto ayuda a seleccionar las herramientas más apropiadas para cumplir con los requisitos del proyecto, tanto a nivel técnico como de experiencia del usuario.

	SQL Server EE	Vue.js	RabbitMQ	.NET	Publicador/s uscriptor
Funcionalidad	Alta	Media	Alta	Alta	Alta
Fiabilidad	Alta	Media	Alta	Alta	Alta
Usabilidad	Media	Alta	Media	Alta	Media
Eficiencia	Alta	Alta	Alta	Alta	Alta
Mantenibilidad	Media	Alta	Media	Alta	Alta
Portabilidad	Media	Alta	Alta	Media	Alta

### *Funcionalidad:*

Los sistemas de Publicador/Suscriptor están diseñados para ser muy fiables en la entrega de mensajes. Este patrón arquitectónico asegura que los mensajes se envíen a todos los suscriptores sin pérdida, con características de reintentos y garantías de entrega, lo que es crucial en sistemas distribuidos donde múltiples servicios o aplicaciones deben comunicarse de manera efectiva.

SQL Server Enterprise Edition (EE) es conocido por su alta fiabilidad debido a su diseño robusto para el manejo de grandes volúmenes de datos, recuperación ante fallos y

características avanzadas de seguridad y consistencia. Está construido para entornos empresariales críticos, lo que asegura un rendimiento constante y confiable en el manejo de bases de datos.

Vue.js, aunque es muy eficiente y popular en el desarrollo de interfaces de usuario, puede ser considerado menos fiable en comparación con tecnologías más maduras en áreas críticas como la integración con bases de datos o sistemas empresariales complejos. Como framework frontend, su fiabilidad depende de la infraestructura subyacente con la que se integre.

RabbitMQ es una tecnología de mensajería extremadamente fiable para garantizar la entrega y el manejo de mensajes entre aplicaciones. Al estar diseñado para tolerancia a fallos, encolamiento y reintento automático, RabbitMQ proporciona alta fiabilidad en entornos distribuidos donde la comunicación entre servicios es clave.

El framework .NET es muy fiable debido a su madurez y soporte en el ecosistema de Microsoft, proporcionando herramientas sólidas para el desarrollo de aplicaciones de nivel empresarial. .NET maneja eficientemente el ciclo de vida de aplicaciones, lo que lo convierte en una opción confiable tanto para aplicaciones web como de backend.

#### Fiabilidad:

SQL Server está diseñado para garantizar la integridad de los datos y la recuperación ante fallos, utilizando transacciones ACID para asegurar que las operaciones de la base de datos sean fiables.

La fiabilidad de Vue.js depende de la implementación y manejo adecuado de errores, ya que, aunque es robusto, la calidad de la implementación puede afectar su rendimiento.

RabbitMQ garantiza la entrega de mensajes a través de colas persistentes y mecanismos de confirmación, asegurando que los mensajes no se pierdan incluso en caso de fallos.

.NET ofrece herramientas para la construcción de aplicaciones fiables, con un enfoque en la gestión de errores y patrones de diseño que promueven la estabilidad.

El patrón de Publicador/Suscriptor asegura la entrega de mensajes de manera consistente. En caso de fallos, el sistema de mensajería (como RabbitMQ) puede almacenar mensajes para garantizar su entrega cuando el suscriptor vuelva a estar disponible.

#### Usabilidad:

Aunque SQL Server proporciona herramientas gráficas y un lenguaje SQL potente, la gestión de bases de datos puede ser compleja, especialmente para usuarios sin experiencia.

Vue.js es conocido por su facilidad de uso, lo que permite a los desarrolladores aprender rápidamente y comenzar a construir aplicaciones. Su enfoque en la simplicidad lo hace accesible.

La configuración y gestión de RabbitMQ pueden ser complejas, requiriendo comprensión de colas, canales y configuraciones de mensajería.

.NET tiene una curva de aprendizaje, pero su vasta documentación y herramientas visuales ayudan a mejorar la usabilidad para los desarrolladores.

Aunque Publicador/Suscriptor es eficiente para sistemas distribuidos, su usabilidad no es tan alta como en sistemas más sencillos, ya que implica la gestión de colas, canales y configuraciones de mensajería, lo cual puede ser más complejo de operar.

#### Eficiencia:

SQL Server está optimizado para manejar grandes volúmenes de datos y consultas de manera eficiente, con características como índices y planes de ejecución que maximizan el rendimiento.

Vue.js es eficiente en la actualización de la interfaz de usuario gracias a su sistema de reactividad, permitiendo que las actualizaciones del DOM sean optimizadas.

RabbitMQ es eficiente en la distribución de mensajes, permitiendo que los productores envíen mensajes a múltiples consumidores rápidamente.

.NET es eficiente en la ejecución de aplicaciones, con un compilador optimizado y manejo eficaz de recursos, lo que permite un rendimiento rápido.

En un patrón Pub/Sub, la eficiencia es alta, ya que los mensajes se distribuyen rápidamente a múltiples suscriptores sin necesidad de que el publicador espere a que cada uno procese el mensaje.

#### Mantenibilidad:

La complejidad de las estructuras y relaciones en bases de datos puede dificultar las actualizaciones y cambios en el esquema, especialmente en aplicaciones grandes.

El uso de componentes modulares y la separación de preocupaciones permiten un mantenimiento más fácil, donde los cambios en un componente no afectan a otros.

La complejidad de la arquitectura de mensajería puede complicar el mantenimiento, pero una buena separación de componentes y la gestión adecuada de colas pueden facilitarlo.

La estructura modular y el uso de patrones de diseño facilitan el mantenimiento de aplicaciones .NET, permitiendo cambios sin afectar otras partes del sistema.

Debido a la clara separación de responsabilidades entre publicadores y suscriptores, los cambios en uno de los lados no afectan directamente al otro, lo que facilita el mantenimiento. Los suscriptores pueden ser modificados o agregados sin alterar el publicador.

#### Portabilidad:

Aunque SQL Server se puede implementar en diversas plataformas, las características específicas pueden no ser fácilmente transferibles a otras bases de datos sin modificaciones.

Vue.js puede ser utilizado en múltiples plataformas y entornos, permitiendo que las aplicaciones se desplieguen fácilmente en diferentes sistemas.

RabbitMQ se puede desplegar en diversas plataformas y se integra con múltiples lenguajes y tecnologías, lo que lo hace muy portátil.

Con el lanzamiento de .NET Core y .NET 5/6, .NET se ha vuelto muy portátil, permitiendo la ejecución de aplicaciones en diversas plataformas como Windows, Linux y macOS.

El patrón Publicador/Suscriptor es fácilmente adaptable a diferentes sistemas de mensajería (como RabbitMQ, Kafka) y se puede desplegar en diversas plataformas, aumentando su portabilidad en sistemas distribuidos.

### Tácticas vs Temas

La siguiente tabla presenta una comparación de diversas tecnologías utilizadas en un entorno de desarrollo moderno, incluyendo SQL Server EE, Vue.js, RabbitMQ y .NET, evaluadas en función de varios factores clave de rendimiento y confiabilidad. Estos factores, como el uso de caché, replicación, particionamiento, balanceo de carga, compensación, monitoreo y fallback, son fundamentales para asegurar la escalabilidad, robustez y eficiencia de las aplicaciones. Cada tecnología es clasificada según su capacidad para manejar estos aspectos, proporcionando una visión general sobre qué soluciones son más adecuadas en diferentes escenarios y requerimientos.

	SQL Server EE	Vue.js	RabbitMQ	.NET	Publicador/Suscriptor
Caché	Medio	Bajo	Medio	Alto	Medio
Replicación	Alto	Bajo	Alto	Medio	Alto

Particionamiento	Alto	N/A	N/A	Medio	Alto
Balanceo de carga	Bajo	Medio	Alto	Alto	Alto
Compensación	Medio	Bajo	Alto	Alto	Alto
Monitoreo	Alto	Medio	Alto	Alto	Alto
Fallback	Medio	Bajo	Alto	Alto	Alto

Esta tabla evalúa varios factores de rendimiento y confiabilidad en el contexto de un sistema que incluye SQL Server EE, Vue.js, RabbitMQ y .NET. Los aspectos evaluados son:

1. Caché: la capacidad de utilizar almacenamiento temporal para mejorar el rendimiento.
2. Replicación: la habilidad de copiar y mantener bases de datos o mensajes sincronizados entre diferentes servidores o instancias.
3. Particionamiento: el proceso de dividir datos o mensajes en segmentos para mejorar la administración y el rendimiento.
4. Balanceo de carga: la distribución eficiente de tareas o mensajes a través de diferentes servidores o procesos.
5. Compensación: la capacidad de manejar transacciones fallidas de manera que se reviertan correctamente.
6. Monitoreo: la capacidad de supervisar el estado y el rendimiento del sistema.
7. Fallback: la habilidad de proveer soluciones de respaldo en caso de fallos o interrupciones.

## Caché

SQL Server ofrece capacidades de caché mediante índices y almacenamiento en memoria, lo que mejora el rendimiento, pero requiere una configuración adecuada.

Vue.js no incluye mecanismos nativos de caché para datos. Sin embargo, el rendimiento puede optimizarse con bibliotecas externas o técnicas de almacenamiento en el frontend.

RabbitMQ no tiene un sistema de caché tradicional, pero puede almacenar mensajes temporalmente en memoria antes de que se procesen, actuando como una especie de buffer.

.NET soporta caché en diferentes niveles (memoria, disco, distribuidos), ofreciendo soporte nativo para mejorar la eficiencia de las aplicaciones.

El sistema puede utilizar un caché temporal para evitar la duplicación de mensajes o mejorar la latencia en el proceso de distribución.

## Replicación

SQL Server soporta replicación de bases de datos en varios nodos, asegurando alta disponibilidad y recuperación ante fallos, lo que lo hace ideal para entornos críticos.

Vue.js, siendo una librería de frontend, no maneja replicación directamente. Esta funcionalidad suele recaer en la capa del servidor o de la base de datos.

RabbitMQ admite replicación de colas para garantizar la continuidad y confiabilidad del servicio en entornos distribuidos.

.NET no maneja replicación de forma nativa, pero se puede implementar mediante otras herramientas o frameworks que corran sobre .NET.

Los sistemas Pub/Sub generalmente permiten replicación de mensajes en múltiples nodos para asegurar la entrega a todos los suscriptores en caso de fallos.

## Particionamiento

SQL Server permite el particionamiento de tablas y datos para mejorar el rendimiento y la escalabilidad, especialmente en grandes volúmenes de datos.

Vue.js no tiene soporte para particionamiento, ya que es una librería de frontend enfocada en la interfaz de usuario y no en la gestión de datos.

RabbitMQ no utiliza particionamiento como tal, pero puede distribuir colas y mensajes entre múltiples nodos.

El particionamiento puede ser implementado manualmente en .NET mediante diseño arquitectónico, pero no es una característica nativa.

En un sistema Pub/Sub, es posible particionar los mensajes entre diferentes colas o canales para mejorar la eficiencia y distribución de la carga.

## Balanceo de Carga

SQL Server no tiene balanceo de carga nativo. Generalmente se implementa a través de soluciones externas o en la capa de aplicación.

Aunque Vue.js no maneja balanceo de carga directamente, aplicaciones basadas en Vue pueden beneficiarse de balanceadores de carga a nivel de servidor.

RabbitMQ ofrece balanceo de carga al distribuir mensajes entre múltiples consumidores, asegurando que las cargas se manejen de manera uniforme.

.NET tiene soporte para balanceo de carga en servicios web y aplicaciones distribuidas, especialmente con tecnologías como microservicios y gRPC.

El patrón Pub/Sub maneja automáticamente el balanceo de carga mediante la distribución de mensajes a diferentes suscriptores y nodos.

## Compensación

SQL Server permite compensaciones a través de transacciones distribuidas y procesos de rollback, aunque esto puede requerir configuraciones avanzadas.

Vue.js no tiene mecanismos de compensación, ya que opera a nivel de interfaz. La compensación debe manejarse en la capa del backend o middleware.

RabbitMQ soporta patrones de compensación mediante confirmaciones y manejo de mensajes no procesados, lo que permite volver a enviar mensajes fallidos.

.NET permite la implementación de mecanismos de compensación mediante transacciones distribuidas o mediante lógica de recuperación de fallos.

El patrón Pub/Sub puede implementar compensación mediante reintentos automáticos o almacenamiento de mensajes en caso de fallos.

## Monitoreo

SQL Server ofrece robustas herramientas de monitoreo, como SQL Server Management Studio (SSMS) y monitoreo en tiempo real, que ayudan a garantizar el rendimiento y la estabilidad.

Vue.js tiene herramientas de monitoreo como Vue Devtools, pero no ofrece un monitoreo profundo como el que se encuentra en tecnologías de backend.



RabbitMQ tiene un sistema de monitoreo integrado, permitiendo la visualización de métricas de colas, consumidores y productores.

.NET tiene soporte para monitoreo mediante herramientas como Application Insights, que permiten rastrear el comportamiento y rendimiento de aplicaciones.

En sistemas Pub/Sub, el monitoreo es fundamental y puede realizarse en cada etapa del proceso (publicación, distribución y suscripción) para garantizar el correcto funcionamiento.

## Fallback

SQL Server tiene opciones de fallback mediante mecanismos de recuperación ante desastres, pero requiere configuraciones específicas.

Vue.js no ofrece soluciones de fallback nativas. La implementación de fallback dependerá del backend o middleware asociado.

RabbitMQ permite el uso de mecanismos de fallback mediante la persistencia de mensajes y reenvío en caso de fallos, garantizando alta disponibilidad.

.NET puede implementar fallbacks mediante patrones de diseño como Circuit Breaker, asegurando que las aplicaciones puedan recuperarse de fallos temporales.

Los sistemas Pub/Sub implementan fallback mediante la retención de mensajes en colas y el reenvío automático en caso de fallos en los suscriptores.

## Mercado laboral vs Temas

	SQL Server EE	Vue.js	RabbitMQ	.NET	Publicador/Subscriptor
Demanda	Alto	Alto	Media	Alto	Medio
Salario promedio	Alto	Media	Alto	Alto	Alto
Nuevas ofertas	Alto	Alto	Media	Alto	Medio
Competencia	Media	Alto	Media	Media	Medio

En el contexto de las oportunidades profesionales, las diferentes tecnologías y patrones tienen niveles variables de demanda, salarios promedio, nuevas ofertas y competencia. A continuación, se presenta una tabla comparativa entre SQL Server EE, Vue.js, RabbitMQ, .NET y el patrón publicador/subscriptor

#### Demanda:

SQL Server: La demanda para ingenieros que estén trabajando con SQL sigue siendo alta, especialmente en grandes empresas que necesitan gestionar grandes volúmenes de datos.

Vue.js: Es una de las nuevas principales librerías para front, y está en constante crecimiento gracias a su facilidad de uso y flexibilidad a la hora de programar. Todo esto junto genera una alta demanda que aún no para de crecer.

RabbitMQ: Es muy popular en sistemas distribuidos y microservicios, pero no es tan común en comparación con bases de datos o frameworks de frontend, lo que posiciona su demanda en un nivel medio

.NET: Un framework ampliamente utilizado en el desarrollo de aplicaciones empresariales, tanto en web como en backend, lo que asegura una alta demanda para los desarrolladores de .NET.

La arquitectura Publicador/Suscriptor es más común en sistemas distribuidos y aplicaciones que requieren alta escalabilidad, como microservicios. La demanda de profesionales que manejan este tipo de arquitectura es media, especialmente en empresas con arquitecturas más complejas.

#### Salario promedio:

SQL Server: Los profesionales que manejan SQL Server a nivel empresarial (EE) suelen tener un salario alto debido a la criticidad de la gestión de datos en las grandes empresas que manejan datos sensibles e importantes.

Aunque Vue.js tiene alta demanda, el salario promedio tiende a ser un poco menor en comparación con tecnologías similares. No obstante, sigue siendo competitivo en el sector frontend.

La especialización en sistemas de mensajería como RabbitMQ puede aumentar el salario debido a la complejidad técnica que requiere y su uso en sistemas críticos.

Los desarrolladores de .NET son bien remunerados, especialmente aquellos con experiencia en aplicaciones empresariales y servicios en la nube.

Los profesionales que dominan arquitecturas Publicador/Suscriptor, junto con herramientas como RabbitMQ o Kafka, suelen obtener salarios altos debido a la

especialización y la necesidad de conocimientos profundos en sistemas distribuidos y mensajería.

#### Nuevas ofertas:

Las empresas continúan buscando profesionales que manejen bases de datos SQL a nivel empresarial, y las nuevas ofertas para administradores de bases de datos o desarrolladores SQL son abundantes.

Con el aumento del desarrollo frontend, hay una cantidad significativa de nuevas ofertas para desarrolladores que trabajan con Vue.js, especialmente en startups y empresas que buscan interfaces de usuario modernas.

Si bien RabbitMQ no es la tecnología más común en todos los sectores, nuevas ofertas que requieren su conocimiento están presentes en áreas de sistemas distribuidos y microservicios.

Las ofertas para desarrolladores .NET son constantes, gracias a la adopción masiva del framework en la industria del software, lo que genera una gran cantidad de oportunidades laborales.

Las ofertas laborales específicas para arquitecturas Publicador/Suscriptor no son tan abundantes como para tecnologías más generalistas como .NET o SQL Server, pero hay una cantidad moderada de nuevas posiciones, especialmente en sectores tecnológicos avanzados como FinTech, IoT o comunicaciones en tiempo real.

#### Competencia:

La competencia en este campo es media, ya que los profesionales que se especializan en estas arquitecturas no son tan numerosos, pero los que tienen experiencia sólida pueden destacarse rápidamente en el mercado laboral.

Debido a la creciente popularidad de Vue.js, la competencia es elevada. Muchos desarrolladores frontend están adoptando esta tecnología, lo que incrementa el nivel de competencia.

La competencia en RabbitMQ no es tan alta, ya que es una tecnología más especializada. Quienes dominan su uso tienen buenas oportunidades de destacar en el mercado.

Aunque hay mucha demanda, también hay un número considerable de profesionales con conocimientos en .NET, lo que genera una competencia moderada en este campo.

Aunque hay demanda constante, la competencia es relativamente media porque hay muchos profesionales con conocimientos en bases de datos, por lo que destacar puede ser un reto.

## Patrones laborales vs Temas

	SQL Server EE	Vue.js	RabbitMQ	.NET	Publicador /Subscriber
Trabajo remoto	Medio	Alto	Medio	Alto	Medio
Metodologías ágiles	Alto	Alto	Alto	Alto	Alto
Contratos freelance	Medio	Alto	Medio	Alto	Medio
Roles de DevOps	Bajo	Medio	Alto	Medio	Alto

En el ámbito del desarrollo de software, los distintos patrones laborales y tecnologías presentan diferentes niveles de adopción y relevancia. A continuación, se muestra una tabla comparativa entre algunos temas clave como SQL Server EE, Vue.js, RabbitMQ, .NET y el patrón publicador/suscriptor frente a tendencias laborales como trabajo remoto, metodologías ágiles, contratos freelance y roles de DevOps.

### Trabajo remoto:

Los roles relacionados con SQL Server suelen requerir acceso a infraestructuras empresariales y bases de datos corporativas, lo que puede limitar el trabajo remoto, aunque es posible en organizaciones con infraestructura adecuada.

Vue.js, siendo una tecnología frontend, permite un alto grado de trabajo remoto, ya que el desarrollo de interfaces de usuario puede hacerse desde cualquier ubicación.

Los roles relacionados con RabbitMQ, especialmente en sistemas distribuidos y microservicios, pueden permitir trabajo remoto, pero suelen requerir colaboración constante con otros equipos y configuraciones complejas.

Los desarrolladores .NET, especialmente aquellos que trabajan en proyectos de software empresarial, pueden trabajar remotamente en una gran cantidad de proyectos, especialmente si utilizan servicios en la nube.

Aunque es posible encontrar roles de Publicador/Suscriptor en remoto, muchos de estos puestos requieren una interacción cercana con la infraestructura o equipos de desarrollo, lo que limita las oportunidades remotas en comparación con tecnologías puramente frontend como Vue.js o más tradicionales como .NET.

### Metodologías ágiles:

Los proyectos que involucran bases de datos empresariales a menudo adoptan metodologías ágiles para gestionar el desarrollo iterativo y la mejora continua en la gestión de datos.

Vue.js es una tecnología altamente compatible con metodologías ágiles debido a su enfoque modular y su capacidad para integrarse fácilmente en sprints y ciclos de desarrollo rápidos.

RabbitMQ, siendo parte fundamental en arquitecturas distribuidas y escalables, es clave en entornos ágiles, ya que permite una rápida iteración y adaptación de los sistemas.

Las metodologías ágiles son ampliamente utilizadas en el desarrollo con .NET, especialmente en proyectos empresariales, donde la capacidad de adaptarse rápidamente a los cambios es crucial.

Las arquitecturas Publicador/Suscriptor, utilizadas en sistemas distribuidos y microservicios, son parte integral de equipos que trabajan con metodologías ágiles. Las iteraciones rápidas y la entrega continua facilitan la integración de nuevas funciones de manera eficiente.

### Contratos freelance:

En el caso de SQL Server, puede no ser tan común, ya que muchos roles requieren una mayor integración, lo que limita un poco las oportunidades. Aun así, es posible encontrar contratos para tareas más específicas.

Por otro lado, en el caso del frontend, y más específicamente con Vue.js, existen muchas oportunidades laborales. Esto se debe a que es común contratar para proyectos de mediano o corto plazo.

En el caso de RabbitMQ, la demanda no es tan alta, ya que está orientado a proyectos específicos de alta especialización, donde en ciertos casos se contrata personal adicional para tareas puntuales.

Existen muchas oportunidades de contratos freelance para desarrolladores .NET, especialmente en proyectos de desarrollo web, aplicaciones empresariales y servicios en la nube.

Existen oportunidades freelance, pero la naturaleza compleja y especializada de las arquitecturas Publicador/Suscriptor tiende a ser más requerida en roles permanentes o de consultoría, a diferencia de desarrolladores de frontend o backend.

## Roles de DevOps:

SQL Server, aunque es importante para la infraestructura, no está tan integrado en los roles de DevOps como otras tecnologías. Los administradores de bases de datos tienen un rol más especializado y están menos involucrados en las operaciones continuas.

Por otro lado, aunque Vue.js no está directamente vinculado a DevOps, en proyectos donde se despliegan aplicaciones frontend de forma continua, los desarrolladores pueden colaborar con equipos de DevOps, especialmente en entornos de integración continua.

.NET está cada vez más involucrado en el ámbito de DevOps, especialmente al integrarse con Docker y Kubernetes para el desarrollo de aplicaciones. Sin embargo, no es un rol intrínsecamente vinculado a DevOps, sino que su participación depende de las necesidades específicas de los proyectos en los que se utilicen estas tecnologías.

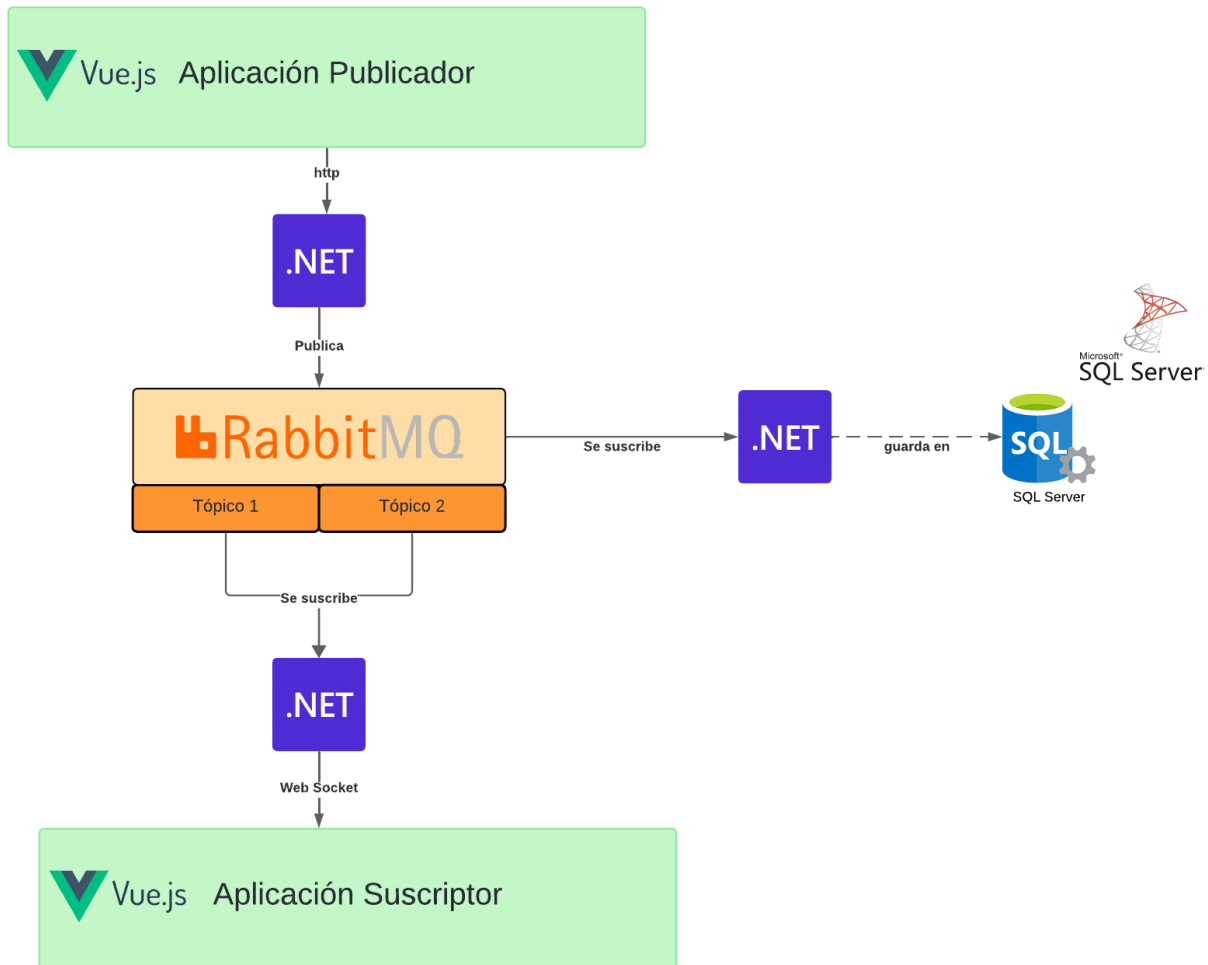
RabbitMQ es una de las herramientas más importantes para muchos equipos de DevOps, especialmente en sistemas distribuidos y/o microservicios, ya que su configuración y manejo están estrechamente ligados a la comunicación eficiente entre servicios. Su capacidad para gestionar colas de mensajes de manera confiable es fundamental en este tipo de arquitecturas.

La arquitectura Publicador/Suscriptor es clave en los entornos DevOps, ya que permite la integración y el despliegue continuo (CI/CD), la automatización y la gestión de sistemas distribuidos, lo que la valora en estos roles.

## Diagramas

### Alto Nivel

El siguiente es el diagrama de alto nivel diseñado para el taller, teniendo en cuenta las tecnologías asignadas.



Explicación de cada uno de los componentes del diagrama:

### **Aplicación Publicador (Vue.js):**

Los usuarios interactúan con la Aplicación Publicador, que está construida en Vue.js. Esta interfaz permite enviar mensajes asociados a tópicos. Cuando un mensaje es enviado, se realiza una petición HTTP al backend, implementado en .NET.

### **Backend .NET Publicador:**

El backend en .NET Publicador actúa como intermediario entre la aplicación de frontend y RabbitMQ. Recibe las solicitudes desde la aplicación publicadora y publica los mensajes en RabbitMQ, asignándolos a un tópico específico, ya sea Tópico 1 o Tópico 2. Este componente se encarga de gestionar toda la lógica de envío de mensajes hacia el sistema de mensajería.

### **RabbitMQ:**

RabbitMQ es el broker de mensajería que gestiona la distribución de los mensajes. Los mensajes publicados desde el backend .NET se distribuyen entre diferentes tópicos, donde los suscriptores se conectan para recibir esos mensajes. RabbitMQ facilita una comunicación desacoplada entre el publicador y el suscriptor, permitiendo que ambos funcionen de manera independiente.

#### **Backend .NET Suscriptor:**

En el lado del suscriptor, el backend en .NET se suscribe a los tópicos relevantes en RabbitMQ y recibe los mensajes. A través de WebSockets, el backend en .NET establece una comunicación en tiempo real con la Aplicación Suscriptor para entregar los mensajes.

#### **Aplicación Suscriptor (Vue.js):**

La Aplicación Suscriptor, también desarrollada en Vue.js, permite a los usuarios ver los mensajes que han sido publicados en los tópicos a los que se han suscrito. Los mensajes llegan en tiempo real gracias a la conexión WebSocket gestionada por el backend.

#### **.NET Persistencia:**

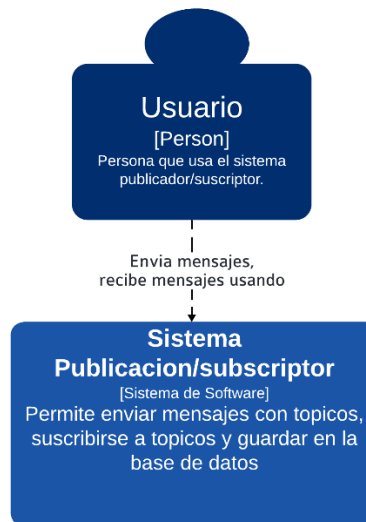
Este componente en .NET se encarga de suscribirse a los tópicos relevantes en RabbitMQ, procesar los mensajes recibidos y almacenarlos en la Base de Datos SQL Server. Garantiza la persistencia de los mensajes, almacenándolos para futuras consultas o auditorías.

#### **Base de Datos SQL Server:**

Además de gestionar los mensajes entre publicadores y suscriptores, el sistema también almacena los mensajes publicados en una Base de Datos SQL Server. El backend .NET se conecta a la base de datos para guardar los mensajes y realizar accesos a los datos cuando sea necesario, garantizando la persistencia de la información.



## Diagrama Contexto

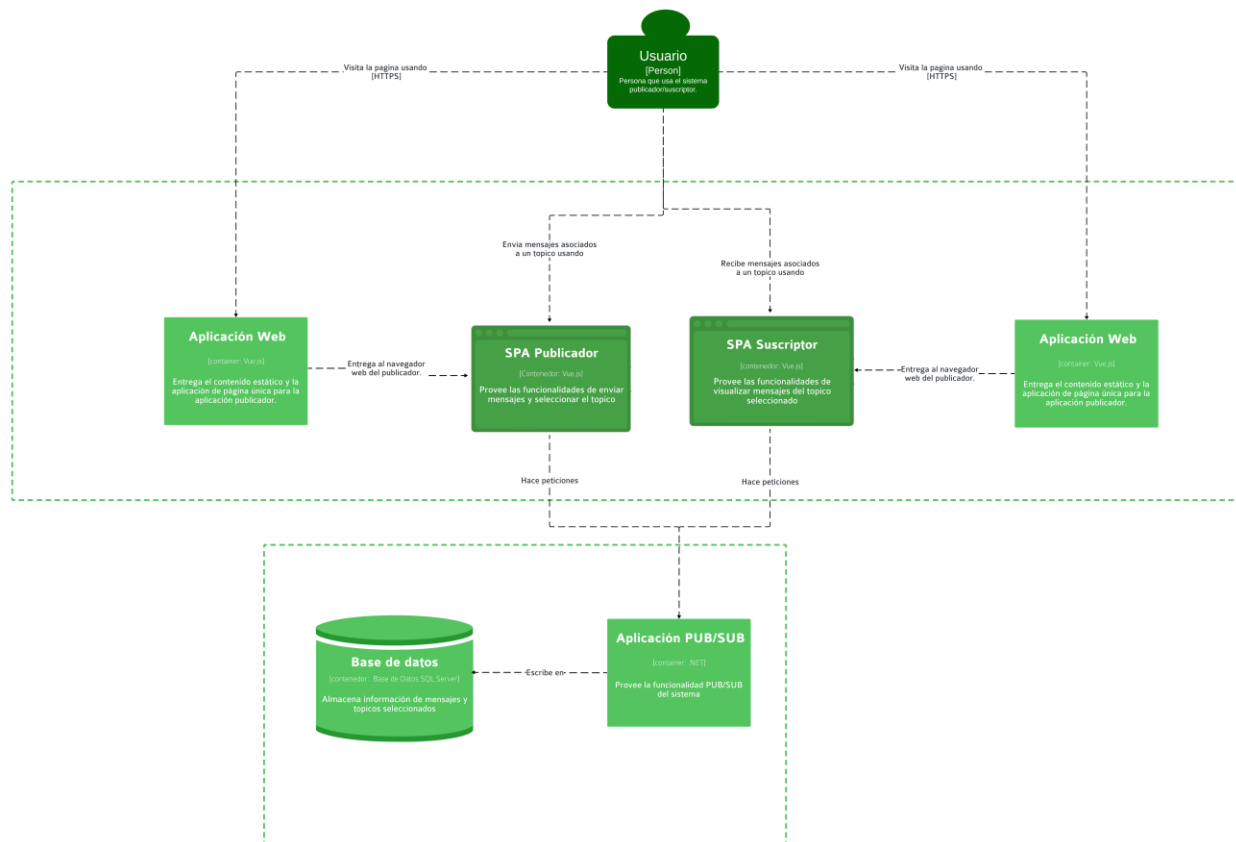


Este diagrama de contexto representa cómo el Usuario interactúa con el Sistema de Publicación/Suscriptor. El Usuario es una persona que, mediante el sistema, tiene la capacidad de enviar y recibir mensajes. Esta interacción es el núcleo de la funcionalidad del sistema, ya que permite que el usuario participe activamente tanto en la publicación de mensajes con tópicos específicos como en la suscripción a dichos tópicos para recibir actualizaciones.

El Sistema de Publicación/Suscriptor gestiona todo el proceso de envío y recepción de mensajes, permitiendo que los usuarios se suscriban a los tópicos de interés. Además, el sistema guarda los mensajes en una base de datos, asegurando su persistencia para futuros accesos o consultas. La relación entre el usuario y el sistema está bien definida, donde el usuario utiliza este servicio de mensajería para comunicarse eficientemente a través de tópicos.

Este diagrama pone énfasis en la facilidad con la que el usuario puede interactuar con el sistema para realizar estas tareas clave de enviar y recibir mensajes.

## Diagrama de Contenedores



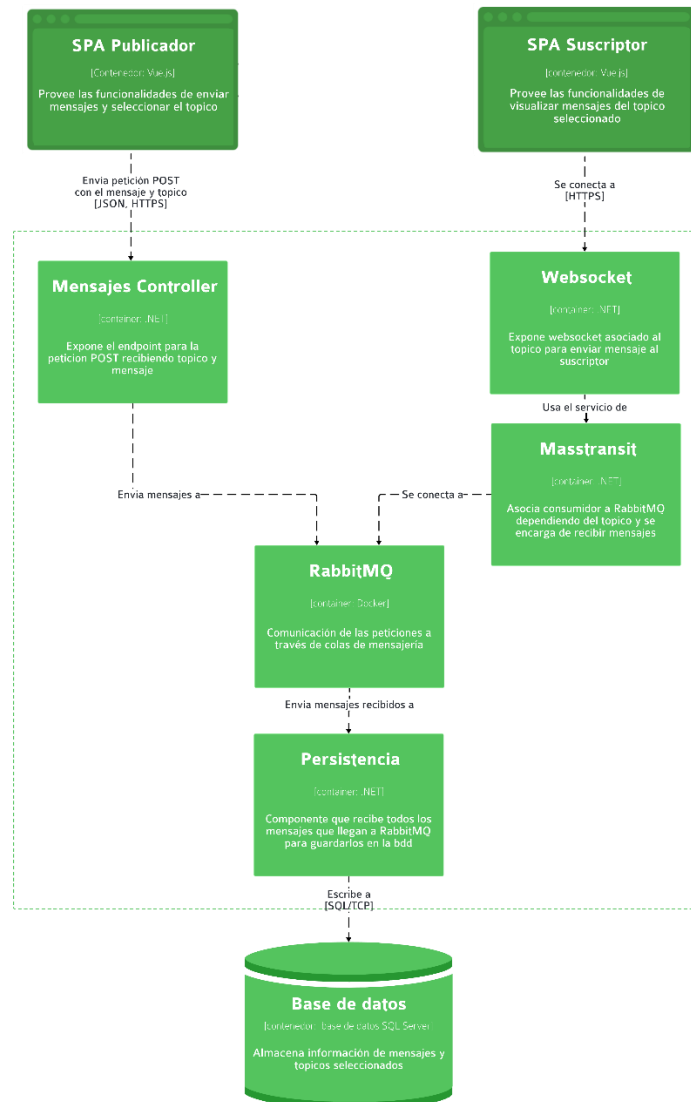
Este diagrama de contenedores representa la interacción entre los Usuarios Publicadores y Usuarios Suscriptores con el sistema Publicador/Suscriptor. Los usuarios acceden a través de una Aplicación Web desarrollada en Vue.js, que les permite enviar mensajes o visualizar los mensajes suscritos.

La SPA Publicador permite a los usuarios enviar mensajes a un tópico, mientras que la SPA Suscriptor les permite recibir mensajes del tópico seleccionado. Ambas aplicaciones interactúan con la Aplicación PUB/SUB desarrollada en .NET, que gestiona la lógica de mensajería del sistema y almacena la información en una Base de Datos SQL Server.

En resumen, este diagrama muestra cómo los componentes trabajan juntos para facilitar la publicación y recepción de mensajes en el sistema, garantizando la correcta gestión y almacenamiento de los datos.

## Diagrama de Componentes

Este diagrama de componentes representa las principales piezas que conforman el sistema de Publicador/Suscriptor y cómo interactúan entre sí para permitir el envío y recepción de mensajes utilizando tópicos. A continuación, se presenta el diagrama

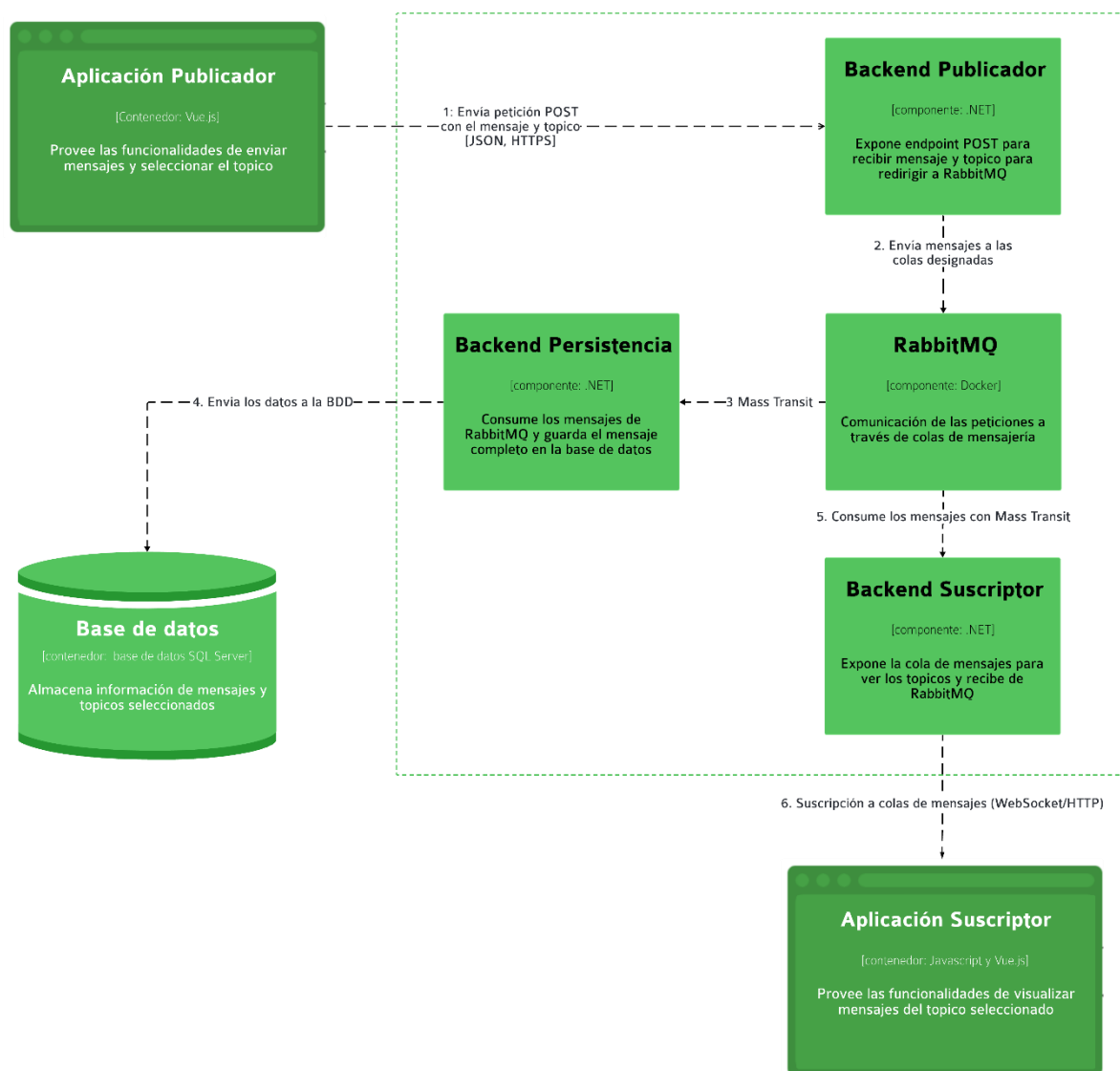


Este diagrama de componentes detalla cómo los diferentes módulos del sistema Publicador/Suscriptor trabajan juntos para manejar el envío y la recepción de mensajes basados en tópicos. Los usuarios interactúan con el sistema a través de las SPA (aplicaciones de página única) desarrolladas en Vue.js: la SPA Publicador permite enviar mensajes mediante peticiones POST hacia el Mensajes Controller en el backend. Este componente, desarrollado en .NET, reenvía los mensajes a RabbitMQ, que actúa como el broker de mensajería.

RabbitMQ, implementado en un contenedor Docker, se encarga de distribuir los mensajes a los suscriptores correspondientes utilizando Masstransit, que asocia los consumidores a los tópicos. Los suscriptores, utilizando la SPA Suscriptor, reciben los mensajes en tiempo real a través de un WebSocket.

Por último, el componente de Persistencia almacena todos los mensajes en una Base de Datos SQL Server, garantizando que los datos se guarden de manera segura para futuras consultas o auditorías. El diagrama resalta cómo los mensajes fluyen desde el publicador hasta el suscriptor y cómo se asegura su persistencia.

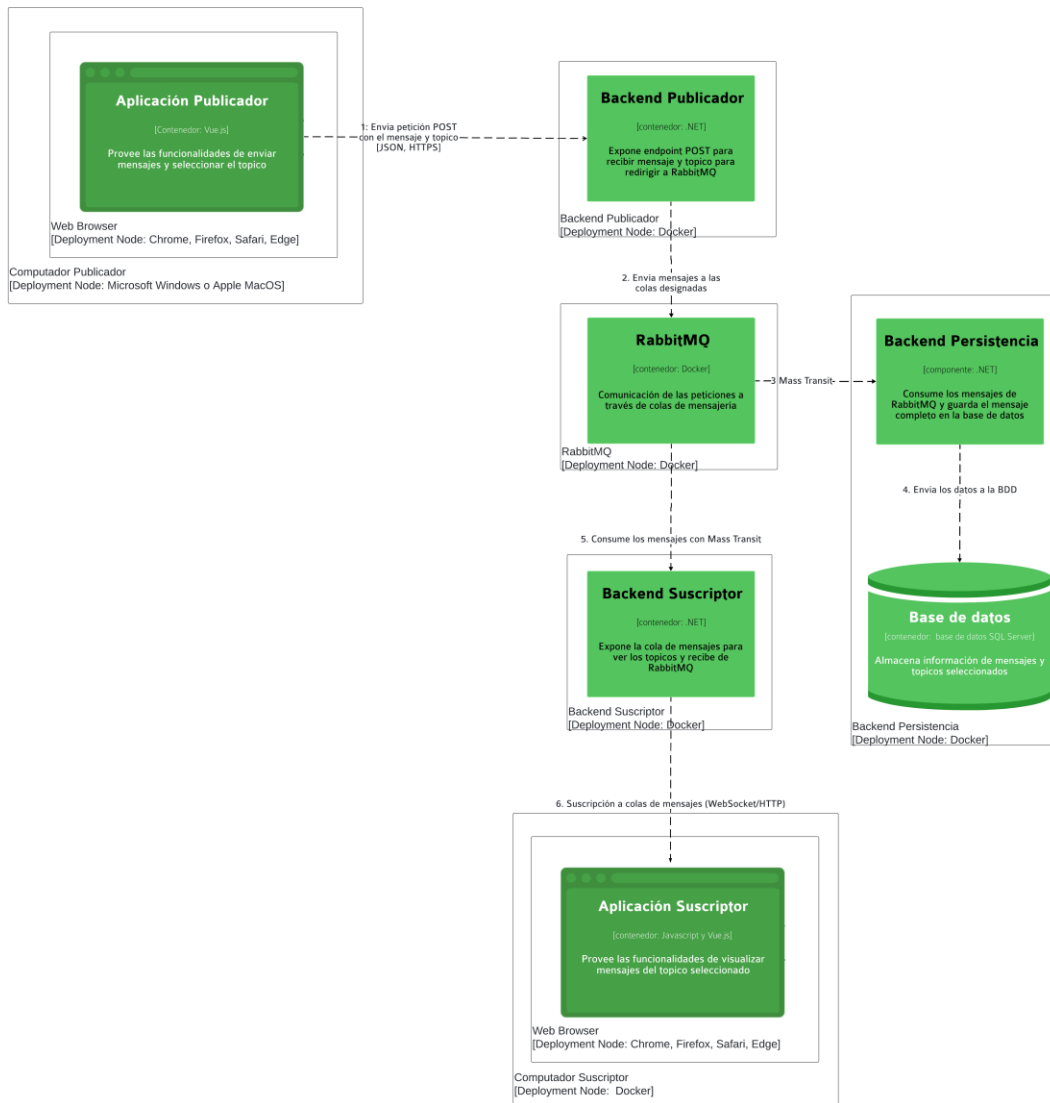
#### Dinámico C4



Este es un diagrama dinámico que muestra cómo los diferentes elementos del modelo estático colaboran en tiempo de ejecución para implementar el patrón de arquitectura Publicador/Suscriptor. En este diagrama, se describe el flujo de mensajes entre una Aplicación Publicador (Vue.js) y una Aplicación Suscriptor (Vue.js), donde el backend .NET publica los mensajes en RabbitMQ a través de peticiones HTTP. Luego, los mensajes se procesan y almacenan en una base de datos SQL Server mediante el Backend Persistencia, mientras que el Backend Suscriptor consume los mensajes desde RabbitMQ y los expone a los suscriptores a través de WebSockets.

## Despliegue C4

Diagrama de Despliegue



Este diagrama de despliegue muestra la distribución de los componentes en la arquitectura Publicador/Suscriptor, detallando cómo interactúan en la infraestructura.

La Aplicación Publicador, accesible desde navegadores web en dispositivos con Windows o macOS, permite a los usuarios enviar mensajes que son procesados por el Backend Publicador en un nodo Windows 11. Este backend, desarrollado en .NET, se encarga de redirigir los mensajes a RabbitMQ, que está desplegado en un contenedor Docker y actúa como el sistema de mensajería.

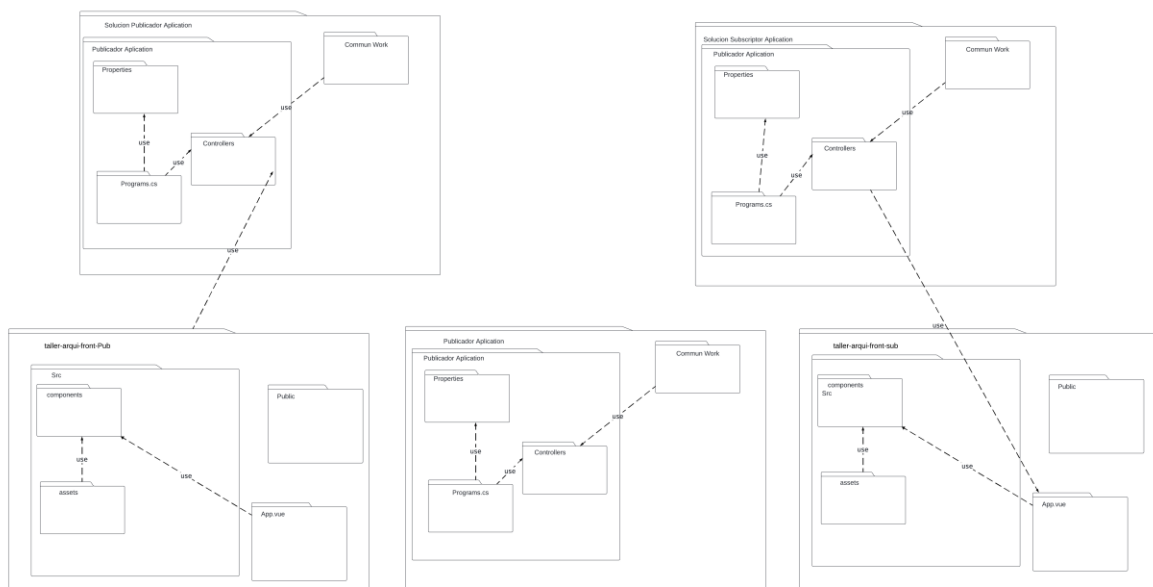
El Backend Persistencia, también en Windows 11, consume los mensajes de RabbitMQ mediante Mass Transit y los guarda en la Base de Datos SQL Server 16, encargada de almacenar la información relevante de los mensajes.

Por otro lado, el Backend Suscriptor, desplegado de forma similar, recibe los mensajes de RabbitMQ y los envía a la Aplicación Suscriptor, que utiliza WebSockets o HTTP para mostrar los mensajes en tiempo real a los usuarios.

Este diagrama muestra como los componentes se implementan y se comunican, asegurando un flujo eficiente de mensajes en la arquitectura Pub/Sub.

### Diagrama de paquetes UML

El siguiente es el diagrama de paquetes UML de cada componente de la aplicación




Este diagrama de paquetes UML ilustra la arquitectura de un sistema basado en el patrón

publicador-suscriptor, donde tanto la Solución Publicador Aplicación como la Solución Suscriptor Aplicación comparten componentes comunes a través del módulo Commun Work. Cada solución está organizada en módulos bien definidos, como **"Controllers"**, **"Programs.cs"**, y **"Properties"**, que reflejan la estructura lógica de la aplicación backend. Además, se visualizan las interfaces frontend de ambas soluciones, construidas con Vue.js, mostrando cómo los componentes y activos interactúan con el archivo principal App.vue. El uso de relaciones use entre los paquetes indica la dependencia entre los módulos, lo que resalta una arquitectura modular y reutilizable, optimizada para la separación de responsabilidades y la reutilización de código en ambos frentes del sistema.

## Referencias

- [1] RabbitMQ tutorial - "Hello World!" | RabbitMQ. (2024). Retrieved September 28, 2024, from Rabbitmq.com website: <https://www.rabbitmq.com/tutorials/tutorial-one-dotnet>
- [2] Vue.js. (2024). Retrieved September 28, 2024, from Vuejs.org website: <https://vuejs.org/>
- [3] SQL Server 2022 | Microsoft. (2022). Retrieved September 28, 2024, from Microsoft.com website: <https://www.microsoft.com/es-co/sql-server/sql-server-2022>
- [4] IBM MQ 9.4. (2024, July 2). Retrieved September 28, 2024, from Ibm.com website: <https://www.ibm.com/docs/es/ibm-mq/9.4?topic=messaging-publishers-subscribers>
- [5] .NET | Build. Test. Deploy. (2024). Retrieved September 28, 2024, from Microsoft website: <https://dotnet.microsoft.com/es-es/>
- [6] ThePower Education. (2023, June 26). Principios SOLID: 5 claves para tu desarrollo de software. Retrieved September 28, 2024, from Thepower.education website: <https://thepower.education/blog/principios-solid>
- [7] Reportes del Mercado Laboral. (2024). Retrieved September 28, 2024, from Banrepcultural.org website: <https://publicaciones.banrepcultural.org/index.php/reporte-mercado-laboral>
- [8] RobBagby. (2024). Patrón de publicador y suscriptor - Azure Architecture Center. Retrieved September 28, 2024, from Microsoft.com website: <https://learn.microsoft.com/es-es/azure/architecture/patterns/publisher-subscriber>

- [9] Ramírez, F. (2023, July 19). Qué es SQL Server, su historia, ventajas y beneficios. Retrieved September 28, 2024, from ITSoftware | Apps | Software | Data Analytics website: <https://itsoftware.com.co/content/que-es-sql-server-su-historia-ventajas-y-beneficios/#:~:text=La%20historia%20de%20SQL%20Server,de%20SQL%20Server%20en%201993>.
- [10] Carmona, J. G. (2022). Breve historia de NET [YouTube Video]. Retrieved from <https://www.youtube.com/watch?v=rhsQJ6X0VTs>
- [11] The. (2021). Introducción a RabbitMQ  - Instalación en Windows y Docker // Librería que DEBES conocer (C#) [YouTube Video]. Retrieved from [https://www.youtube.com/watch?v=j3tDM\\_hi90g&t=411s](https://www.youtube.com/watch?v=j3tDM_hi90g&t=411s)
- [12] Learn With Tapos. (2021). RabbitMQ ASP NET CORE 5 (part-01) [YouTube Video]. Retrieved from <https://www.youtube.com/watch?v=QBi82HRwCcQ&list=PLefG3VDSLxSy2DNezEos2uozgfC5VcroD>
- [13] JumpstartCS. (n.d.). RabbitMQ - Tutorial 9 - PubSub [Video]. YouTube. <https://youtu.be/rgTW-cserPo?si=jl9zfI5m81eYIoi0>
- [14] Reclu IT. (n.d.). <https://recluit.com/que-es-rabbitmq/>
- [15] Case Study: Domino's Pizza UK Keeps Orders Flowing with SQL Server Monitoring - AmCham Bulgaria. (2023, April 27). AmCham Bulgaria. <https://amcham.bg/2023/04/27/case-study-dominos-pizza-uk-keeps-orders-flowing-with-sql-server-monitoring/>