Servidor WebSocket para mantener la conectividad entre Whakau y sistemas de autenticación

Juan Camilo Marín García

Trabajo de Grado para Optar al Título de Ingeniería de Sistemas

Director

Fernando Antonio Rojas Morales

Msc, Computer Science

Tutor

Francisco Javier Gonzales Silva

Líder técnico y desarrollador senior

Universidad Industrial de Santander

Facultad de Ingeniería Fisco-Mecánicas

Escuela de Ingeniería de Sistemas e Informática

Ingeniería de Sistemas

Bucaramanga

2022

Dedicado a

A mis padres y hermana por brindarme su apoyo incondicional.

A mi madre, Esmeralda García por motivarme día a día para perseguir mis sueños, para nunca rendirme y siempre dar lo mejor de mí.

A mi padre, Víctor Marín por siempre confiar en mí y nunca poner en duda el hecho de que este momento llegaría.

A mi hermana, Valentina Marín que, aunque no entendía mucho cuando hablaba sobre los temas que me apasionan de mi profesión, siempre me incitó a seguir aprendiendo y ser el mejor.

A mi abuelita que, aún en la distancia, siempre ha estado pendiente de mi dándome toda su energía.

A los indecisos, mi grupo de la universidad con quienes desde primer semestre pude compartir y disfrutar al máximo de lo que era la vida universitaria.

A JuanPa, Angie, Edward, Marly y Sergio que se ganaron todo mi cariño y respeto durante estos 5 años, espero poder seguir compartiendo en los años venideros viéndolos alcanzar cada una de sus metas.

A Juan Carlos Abaunza, por darme la oportunidad de hacer parte de A&A Soluciones - TIC, más que un jefe, ha sido un mentor de quien he aprendido y espero seguir aprendiendo para algún día llegar a su nivel de excelencia profesional y personal.

Agradecimientos

A mi director de proyecto, profesor Fernando Antonio Rojas Morales quien confió en mí y asumió la dirección de mi proyecto de grado.

A la empresa A&A Soluciones – TIC y su CEO Juan Carlos Abaunza por permitirme realizar la practica empresarial con ellos ayudándome a complementar mi formación profesional.

A Francisco Javier Gonzales Silva, tutor y asignado por A&A Soluciones – TIC, por su disposición al momento de asesorarme con las distintas dudas que surgieron durante el desarrollo del proyecto.

A mi familia por todo el apoyo y cariño brindado durante toda la etapa de pregrado, sus consejos y enseñanzas que fueron vitales para culminar esta experiencia.

A la Universidad Industrial de Santander por abrirme las puertas y permitirme formarme para ser un excelente profesional y persona.

Tabla de Contenido

Pág.

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Lista de Apéndices

pág.

Los apéndices están adjuntos y puede visualizarlos en la base de datos de la biblioteca UIS

Resumen

Título: Servidor WebSocket para mantener la conectividad entre Whakau y sistemas de autenticación[*]

Autor: Juan Camilo Marín García[**]

Palabras Clave: Websocket, Whakau, Biometría

Descripción: Los sistemas biométricos se definen como técnicas automáticas de reconocimiento y autenticación que utilizan el análisis de patrones y/o características físicas y de comportamiento exclusivas de cada persona. El proceso de verificación de la identidad de las personas se realiza comparando los registros de sus datos biométricos con datos que previamente han sido almacenados en la base de datos, el resultado de esta comparación determina la identidad de la persona (Giraldo Giraldo y Gomez Ramirez, 2017).

Whakau es una plataforma que permite conectar distintas soluciones de autenticación biométrica y/o proximidad, fabricadas por empresas tales como Suprema, HID, Lenel, Zk teco, BOSCH, entre otros. Lo hace a través de APIs y/o SDKs de estos fabricantes, conectándose directamente a su software de gestión o al dispositivo de lectura (terminal biométrico o de proximidad). Esto con el fin de aprovechar esta autenticación de las personas para usos distintos al control de accesos y horarios (propósito con el cual fueron fabricados estos sistemas) tales como membresías de gimnasio o clubes, soluciones fintech, conciertos, entre otros.

Para el óptimo funcionamiento y mayor rendimiento en las funciones de Whakau, se busca crear una capa de conectividad siempre activa y abierta (en tiempo real y bidireccional) entre los sistemas conectados a Whakau y la plataforma, con el fin de evitar interrupciones en el uso y consumo de los datos que se transfieren en ambas vías y que dan vida al uso de la plataforma.

---

[*] Trabajo de Grado

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: Fernando Antonio Rojas Morales. Maestría en Ciencia Computacional. Codirector: Francisco Javier Gonzales Silva. Ingeniería de Sistemas

Abstract

Title: WebSocket server to maintain connectivity between Whakau and authentication systems [*]

Author(s): Juan Camilo Marín García[**]

Key Words: Websocket, Whakau, Biometry

Description: Biometric systems are defined as automatic recognition and authentication techniques that use the analysis of patterns and/or exclusive physical and behavioral characteristics of each person. The process of verifying the identity of people is carried out by comparing the records of their biometric data with data that has previously been stored in the database, the result of this comparison determines the identity of the person (Giraldo Giraldo y Gomez Ramirez, 2017).

Whakau is a platform that allows connecting different biometric and/or proximity authentication solutions, manufactured by companies such as Suprema, HID, Lenel, Zk teco, BOSCH, among others. It does so through APIs and/or SDKs from these manufacturers, connecting directly to their management software or to the reading device (biometric or proximity terminal). This in order to take advantage of this authentication of people for uses other than access control and schedules (purpose for which these systems were manufactured) such as gym or club memberships, fintech solutions, concerts, among others.

For optimal operation and better performance of Whakau functions, it aims to create a connectivity layer that is always active and open (in real time and bidirectional) between the systems connected to Whakau and the platform, in order to avoid interruptions in the use and consumption of the data that is transferred in both ways and that gives life to the use of the platform.

---

[*] Degree Work

[**]Faculty of Physical-Mechanical Engineering. School of Systems Engineering and Informatics. Director: Fernando Antonio Rojas Morales. Master of Computer Science. Tutor: Francisco Javier González Silva. Systems Engineer.

Introducción

En este documento se busca presentar el trabajo realizado que permitió el desarrollo de una nueva funcionalidad para Whakau, con la cual y por medio de un servidor WebSocket, le permite mantener una conectividad activa, bidireccional y siempre abierta con los sistemas que tiene conectados.

Whakau es una plataforma que ha sido desarrollada con el fin de conectar distintas soluciones de autenticación biométrica y/o proximidad, siendo mediadora entre otras plataformas existentes, desarrollos a medida y otros usos distintos a los de autenticación de usuario tales como membresías de gimnasio o clubes, soluciones fintech, conciertos, entre otros. En la versión que se tomó al inicio, Whakau ya contaba con la integración de varios fabricantes, pero cada vez que se necesitaba iniciar sesión en el módulo y sección en donde hay que autenticarse para un fabricante, se requería ir allá y hacerlo manualmente para que se activara la comunicación en dos vías, lo que implicaba interrupciones en el flujo de información que debe transitar en ambas vías.

En este orden de ideas, el trabajo que se desarrolló durante este proyecto radica en el diseño y desarrollo de una nueva funcionalidad para Whakau. La nueva funcionalidad consistió en una capa de conectividad siempre activa y abierta (en tiempo real y bidireccional) entre los sistemas conectados a Whakau y la plataforma, con el fin de evitar interrupciones en el uso y consumo de los datos que se transfieren en ambas vías y que dan vida al uso de la plataforma.

Esta funcionalidad se desarrolló siguiendo las practicas DevOps junto a la metodología ágil Scrum y teniendo como base el ciclo de vida del software, de tal manera que en cada iteración se obtuvo una versión mejorada de la funcionalidad hasta conseguir que la comunicación entre

Whakau y las demás plataformas no se apagara nunca. Además, se hizo uso de las áreas de practica del Modelo Integrado de Madurez de Capacidades, CMMI, con el fin de mejorar el rendimiento de los procesos que comprendieron el desarrollo del proyecto.

1. Planteamiento y Justificación del Problema

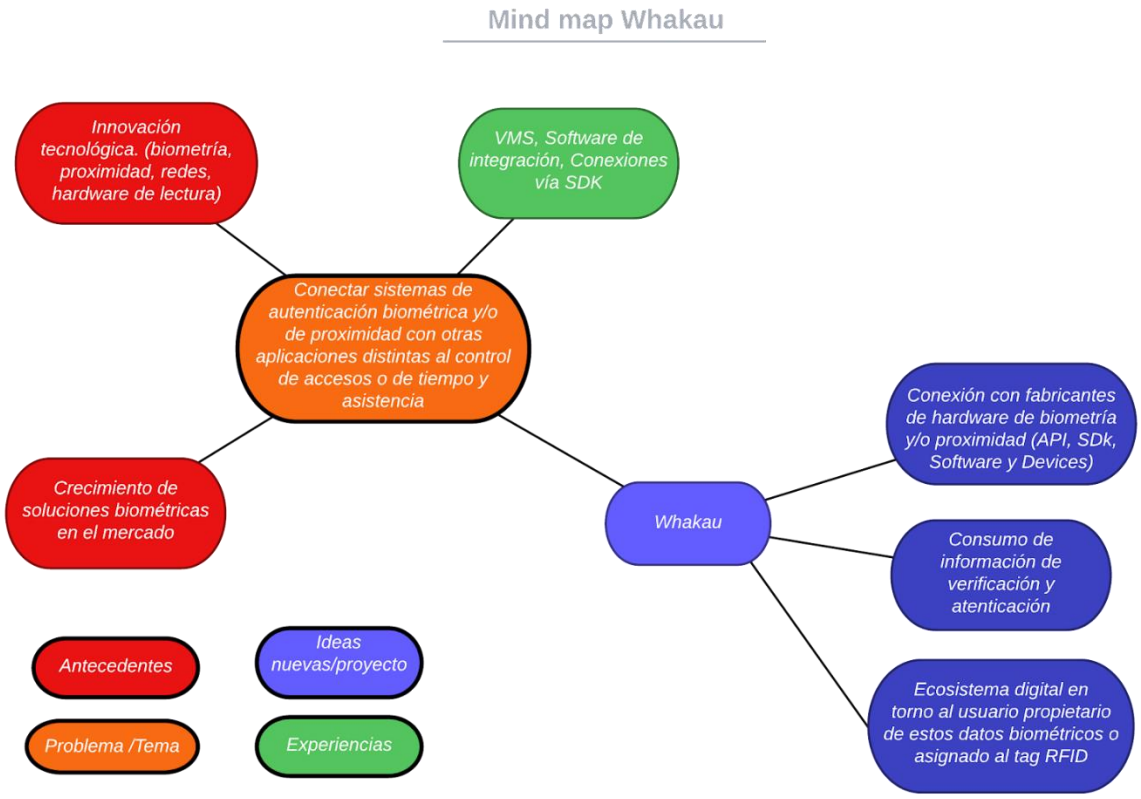
¿Cómo modificar el sistema actual de comunicación que tiene Whakau para la conectividad con las diversas plataformas de autenticación que tiene integradas, de tal manera que esta se mantenga siempre activa y abierta (en tiempo real y bidireccional)?

El proyecto que se desarrolló es una adecuada opción de trabajo de grado ya que para su desarrollo se hizo un trabajo de ingeniería siguiendo el ciclo de vida del desarrollo de software. Al final de esta experiencia se creó un ecosistema de conectividad en tiempo real y asíncrona escalable para la autenticación a través de diferentes plataformas.

Whakau nace de la necesidad recurrente de tener una herramienta que integre las soluciones de software que fabricantes de tecnología de control de accesos tienen. La intención es agrupar diversas tecnologías y poder realizar integraciones entre las mismas evitando que cada que exista un requerimiento con ese fabricante se tenga que desarrollar algo a medida.

Figura 1

Mind Map de Whakau



Por lo anterior, Whakau es un mediador entre plataformas existentes, desarrollos a medida y otros usos distintos a los de autenticar un usuario; tomando un usuario existente y registrado en estos sistemas y siendo capaz de traer la data, administrarla si es posible y conectarla con otros permisos distintos al paso por el acceso, como pueden ser: saldo a favor, servicios incluidos, permisos especiales, reportes, entre otros según la razón de ser de los clientes que contraten la plataforma.

El proyecto que se desarrolló durante la práctica en A&A Soluciones – TIC consistió en una nueva funcionalidad para Whakau. La nueva funcionalidad (Objeto del presente proyecto de grado) permite que la conectividad que tiene Whakau con las demás plataformas de autenticación

que tiene integradas y otras que pueden ser agregadas en un futuro, sea eficiente y bidireccional, que siempre esté abierta y se pueda enviar y recibir información en tiempo real.

Durante todo el proceso que abarcó la construcción de la nueva funcionalidad hasta su despliegue en Whakau, se utilizaron prácticas esenciales para la trazabilidad como lo son la de la cultura DevOps y la metodóloga ágil Scrum, junto con las áreas de practica de CMMI, todo esto con el fin de desarrollar un producto de calidad, eficiente y seguro.

2. Objetivos

2.1 Objetivo General

Diseñar y desarrollar un servidor WebSocket que permita la conectividad en tiempo real y asíncrona entre tecnologías de distintos fabricantes de sistemas de autenticación de usuarios basados en biometría o proximidad utilizando una arquitectura Multitenant.

2.2 Objetivos Específicos

Analizar cómo agrupa Whakau diversas tecnologías y poder realizar integraciones entre las mismas, para establecer la estrategia de conectividad asíncrona.

Diseñar y desarrollar mediante una arquitectura basada en Multitenant un servidor WebSocket que sea mediador entre plataformas existentes, desarrollos a medida y otros usos distintos a los de autenticar un usuario.

Desplegar esta nueva funcionalidad en el producto.

Validar el sistema en toda su funcionalidad, a través de pruebas de aceptación y realizando mediciones de desempeño.

3. Marco de Referencia

A continuación, se encuentran los conceptos más importantes que se deben tener en cuenta para el entendimiento del proyecto que se desarrolló.

3.1 Marco Teórico

3.1.1 Whakau

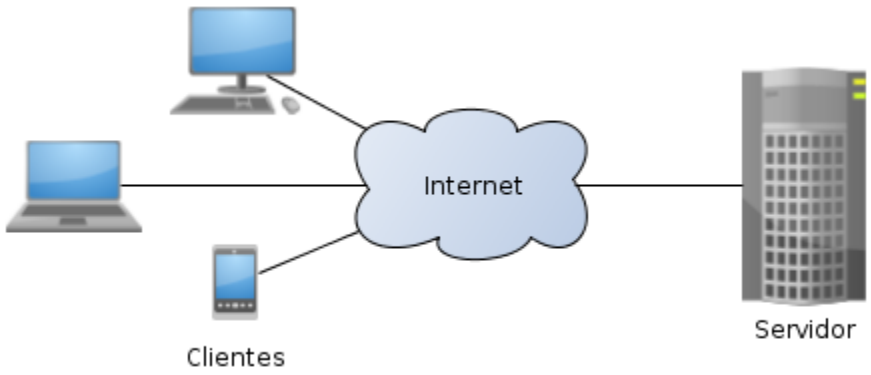
Es una plataforma que permite conectar distintas soluciones de autenticación biométrica y/o proximidad, fabricadas por empresas tales como Suprema, HID, Lenel, Zk teco, BOSCH, entre otros. Lo hace a través de APIs y/o SDKs de estos fabricantes, conectándose directamente a su software de gestión o al dispositivo de lectura (terminal biométrico o de proximidad). Esto con el fin de aprovechar esta autenticación de las personas para usos distintos al control de accesos y horarios (propósito con el cual fueron fabricados estos sistemas) tales como membresías de gimnasio o clubes, soluciones fintech, conciertos, entre otros.

3.1.2 WebSockets

Normalmente la comunicación entre un servidor y un cliente se da mediante por medio de peticiones http. Si se quiere traer un dato, es necesario obtenerlo por medio de una solicitud http.get, con ello se le indica al servidor que el cliente desea que se le envíe información. Al hacerlo el servidor enviará una respuesta, puede ser un error, una advertencia o como tal la información que le ha sido solicitada. ¿Pero qué sucede si la información dentro del servidor cambia?, el servidor no tendrá manera de avisarle al cliente que hay datos nuevos, a menos que el cliente vuelva a hacer una petición solicitud http.get para obtener la información directamente el mismo. Esto puede visualizarse mejor en la siguiente ilustración:

Figura 2

Arquitectura Cliente Servidor



Nota. Representación Arquitectura Cliente-Servidor. Tomada de *Un diagrama cliente-servidor vía Internet* [Diagrama]. Colaboradores de Wikipedia, 2022, https://es.wikipedia.org/wiki/Cliente-servidor

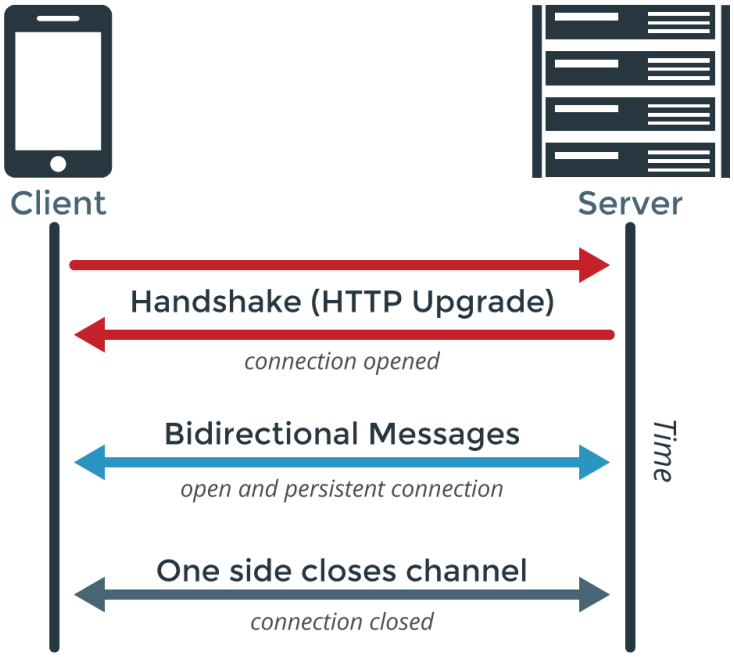
Ahora, ¿qué sucede si un cliente requiere comunicarse con otro siendo la comunicación por medio de un servidor?, si esto sucede cada cliente deberá hacer solicitudes HTTP.GET para saber si el otro cliente le ha enviado algo, lo que produciría una comunicación interrumpida, algo que no puede ocurrir en aplicaciones donde la comunicación debe ser en tiempo real y asíncrona. WebSocket nació con la idea de hacerle frente a este problema.

WebSocket es una tecnología que da la oportunidad desarrollar aplicaciones orientadas a la Web y que necesitan compartir información en dos vías manteniendo una conexión persistente. En (Lombardi, 2015) se indica que usar WebSocket “permite construir aplicaciones web que posean un canal orientado a la comunicación del tipo full-duplex con un comportamiento

asíncrono", lo que permite que cada cliente reciba actualizaciones en tiempo real sin necesidad de hacer solicitudes al servidor.

Figura 3

Arquitectura de WebSocket



Nota. Flujo de comunicación mediante protocolo websocket. Tomada *de ¿Qué es exactamente un WebSocket?* [Figura]. Schmitz, 2020, https://www.dotcom-monitor.com/blog/es/2020/05/25/websocket-application-monitoring/

WebSocket es principalmente útil para servicios que demandan intercambio de información continua, por ejemplo, videojuegos en línea, sistemas de mensajería, aplicaciones basadas en web para llamadas y video llamadas entre otros. También es importante mencionar que la comunicación bidireccional por medio de WebSockets se hace mediante una única conexión.

4.1.3 Multitenat

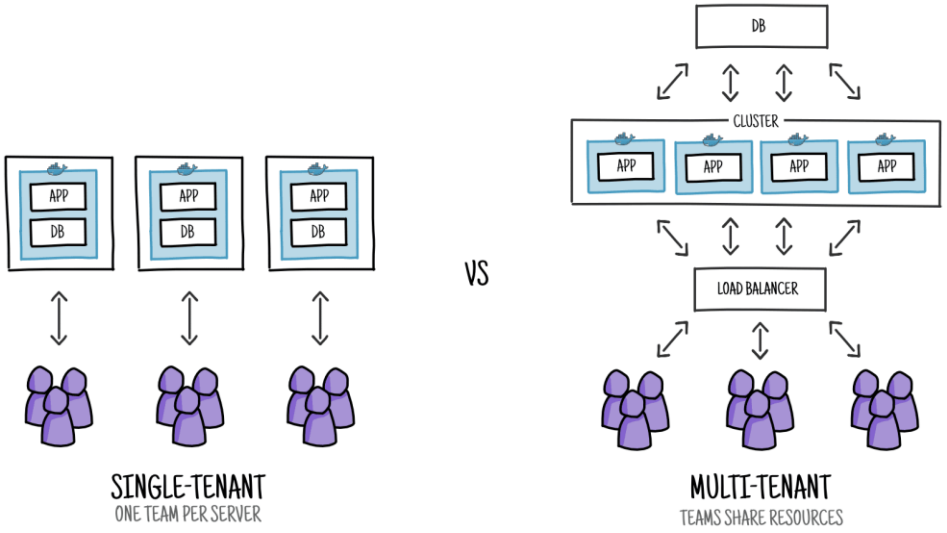
Es una arquitectura de software en la cual se centraliza bajo una única instancia el acceso de varios clientes o grupos, también llamados inquilinos. En otras palabras, una única solución basada en software puede ser utilizada por múltiples usuarios, pero manteniendo el acceso y la visibilidad a la información separada para cada uno de ellos.

Para Alfonso (2018), la estructura de esta arquitectura consiste en los siguiente: “tener una sola base de código que se ejecuta para todos los clientes, con una misma estructura de datos, pero la capa de datos separadas en distintos espacios de trabajo por cada organización”.

Esta arquitectura ofrece tanto ventajas como desventajas, por lo que si se implementa o no depende del problema que se desee atacar.

Figura 4

Arquitectura Estándar vs Multitenat



Nota. Comparativa entre arquitectura estándar vs arquitectura Multitenat. Tomada de *¿Qué es una Arquitectura Multi-tenant?* [Figura]. Pulido, R. U, 2019, https://www.maquinasvirtuales.eu/que-es-una-arquitectura-multi-tenant/

Existen tres enfoques diferentes para aplicar la arquitectura multitenat, su elección depende principalmente del tipo de proyecto y los recursos disponibles para su implementación y su diferencia radica en el nivel de aislamiento:

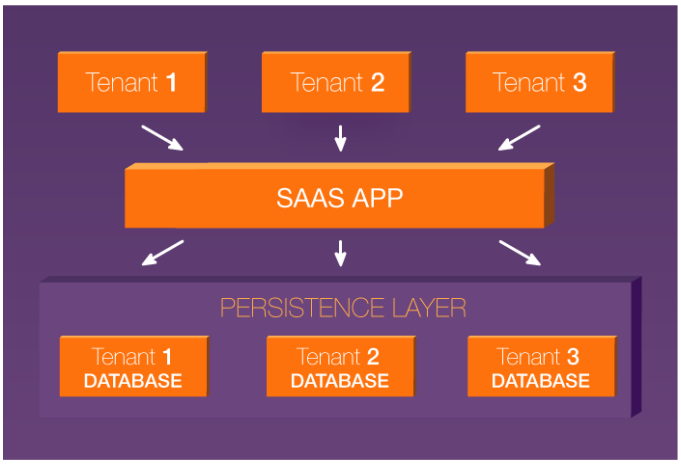
3.1.3.1 Servicio Multi-Inquilino (Base de Datos para cada Cliente)

En este enfoque, cada cliente cuenta con su propia base de datos manejada por un servidor separado lo que garantiza un algo nivel de seguridad.

Como se puede ver en la Ilustración 5, cada uno de los inquilinos se conecta a una única instancia de la aplicación, esta última cuenta con la lógica necesaria para redirigir a cada cliente a su propio servidor y con esto a su propia base de datos. Entre las ventajas que da usar este esquema es que de ser necesario hacer un mantenimiento en el servidor de un cliente, esto no afectará a los demás clientes, adicionalmente, facilita la gestión de backups para cada inquilino conectado. “En caso de cualquier problema de corrupción de información, solo afectaremos a una de las organizaciones (Alfonso, 2018)”.

Figura 5

Una base de datos para cada organización



Nota. Representación Multitenat con diferentes bases de datos por cliente en una capa de persistencia. Tomada de *Una base de datos para cada organización* [Figura]. Alonso, 2018, https://adrianalonsodev.medium.com/enfoques-de-arquitectura-multitenant-para-aplicaciones-saas-cf210d6c2f10

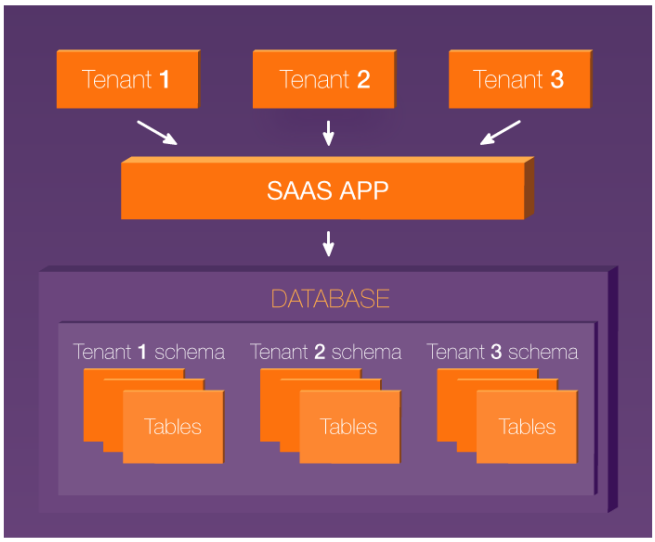
3.1.3.2 Servidor Único Compartido (Esquemas Separados por cada Organización)

El enfoque anterior es una alternativa bastante adecuada puesto que ofrece un nivel de aislamiento bastante alto, pero su implementación es muy costosa debido a que se hace necesario crear un servidor para cada cliente que requiera.

Si se maneja un servidor único compartido, pero se mantiene una base de datos separada para cada cliente, puede garantizarse un aislamiento seguro, además que se su implementación será mucho más económica. “Sin embargo, las cargas de trabajo de un clúster comparten los mismos recursos y red, por lo que su aislamiento es limitado” (Balaji & Manoj, 2021).

Figura 6

Servidor Único Compartido



Nota. Representación Multitenat con misma base de datos pero esquema diferente por cliente. Tomada de *Esquemas separados por cada organización* [Figura]. Alonso, 2018, https://adrianalonsodev.medium.com/enfoques-de-arquitectura-multitenant-para-aplicaciones-saas-cf210d6c2f10

En la Figura 6 se puede ver que cada inquilino se conecta a una única instancia de la aplicación, pero a diferencia del enfoque uno, la capa de persistencia es un único servidor que contiene la base de datos de cada cliente, garantizando que cada cliente pueda acceder solamente a su información.

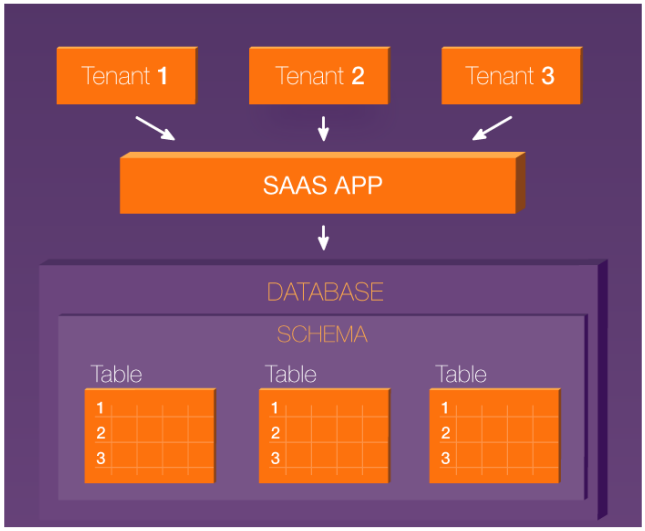
3.1.3.3 Esquemas Compartidos por Organizaciones

Aunque este esquema puede ser más económico monetariamente hablando, su implementación es más costosa. Alfonso (2018) afirma que “los datos de todas las organizaciones estarán almacenados en las mismas tablas, por lo que para recuperar la información se deberá de asignar un identificador que represente quien es el dueño de ese dato”, lo que implica que se deba

tener mucho más cuidado a la hora de implementar la lógica que mantenga la información de cada cliente separada.

Figura 7

Esquemas Compartidos por Organizaciones



Nota. Representación Multitenat con esquema compartido por clientes. Tomada de *Esquema compartido por las organizaciones* [Figura]. Alonso, 2018, https://adrianalonsodev.medium.com/enfoques-de-arquitectura-multitenant-para-aplicaciones-saas-cf210d6c2f10

Como se puede observar en la ilustración 7, el manejo de la persistencia se gestiona por medio de un solo esquema de la base de datos, y este se encuentra en un servidor. La aplicación cuenta con la lógica necesaria para mantener la información de cada inquino aislada de tal forma que solo el “dueño” pueda acceder a esta.

3.1.4 Biometría

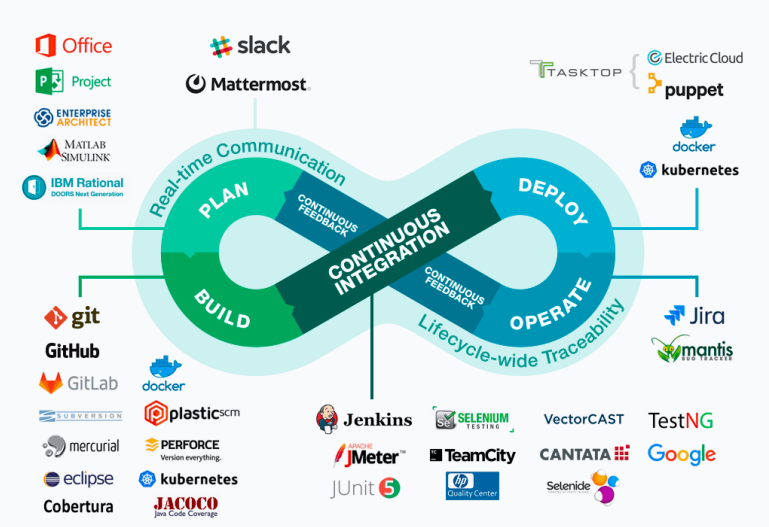
Normalmente cuando se piensa en Biometría se cree que se refiere solamente a reconocimiento por medio de la huella dactilar, pero no es así. Este es un método de reconocimiento de personas, basado en sus características fisiológicas o de comportamiento. Dependiendo del tipo de autenticación que se vaya a aplicar dentro de una organización, es requerido aplicar una técnica diferente, por lo que los parámetros que se también serán diferentes. De estos parámetros se extrae un patrón único para cada persona, que será el que se utilice para posteriores comparaciones. (INCIBE, 2016)

3.1.5 DevOps

DevOps es una cultura de trabajo que permite unir diferentes roles de forma ordenada y colaborativa, con el propósito de producir productos mejores y más confiables en menos tiempo (Microsoft, s.f.). Su aplicación a la hora de desarrollar proyectos se logra a través de la adopción de diferentes prácticas que se desarrollar durante las diferentes fases del ciclo de vida del software.

Figura 8

Esquema DevOps



Nota. Esquema DevOps junto con herramientas para ejecutar sus prácticas. Tomada *de DevOps y Transformación Digital* [Figura]. Abaunza, 2020, https://juancarlosabaunza.com/devops-y-transformacion-digital/

Estas prácticas se llevan a cabo en ocasiones por medio de herramientas que permiten automatizar procesos, pero más allá de utilizar estas herramientas, lo que sugiere un verdadero reto es el cambio de cultura que debe hacerse dentro de una organización para que asuman esta metodología, puesto que implica un cambio bastante grande a como se hacen las cosas normalmente (Amazon, s.f.). A continuación, se explicarán en qué consisten estas prácticas:

3.1.5.1 Integración y Entregas Continuas CI/CD

En esta práctica, cada miembro del equipo integra sus avances hechos durante el desarrollo de un producto a un repositorio de manera periódica, permitiendo encontrar y solucionar errores con mayor rapidez. Una vez se corrigen errores se pasan a la fase de producción y se repite el proceso hasta terminar por completo la aplicación.

3.1.5.2 Control de Versiones

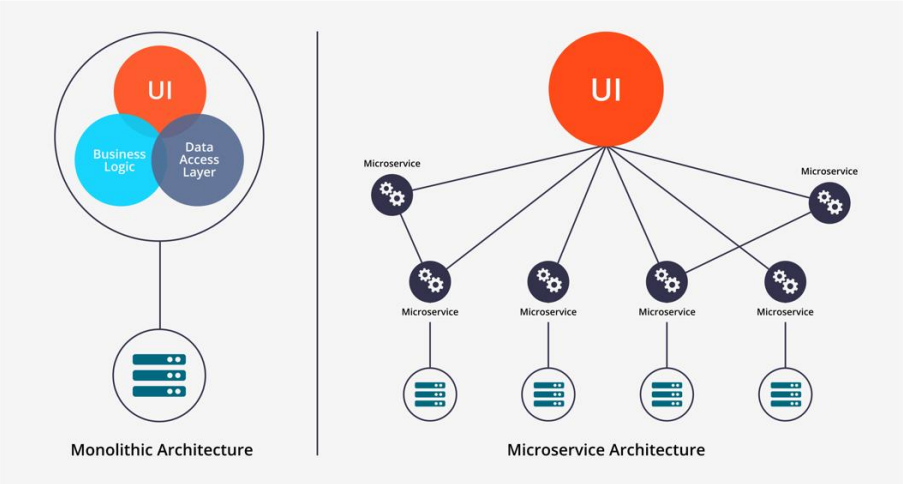
Esta práctica se administra el desarrollo de la aplicación por versiones dando la oportunidad de recuperar el código si es necesario y dando un mejor control y manejo a los avances que se vayan dando durante el ciclo de vida de la aplicación. (Microsoft, s.f.). Adicionalmente se debe manejar un flujo de trabajo que permita la CI/CD, pero sin afectar el trabajo del resto del equipo.

3.1.5.3 Arquitectura Basada en Microservicios

Es una arquitectura que desglosa las funciones principales de un sistema o aplicación en servicios más pequeños, encargados de cumplir una única labor independientemente de los demás servicios. Como se puede ver en la ilustración 7, al usar una arquitectura monolítica las funciones y procesos están estrechamente conectados dentro de un mismo servicio, esto puede ser bastante limitante no solo a la hora de desarrollar la aplicación desde un principio si no cuando sea necesario escalarla para agregar nuevas tareas.

Figura 9

Arquitectura Monolítica VS Arquitectura de Microservicios



Nota. Comparativa entre arquitectura monolítica y una de microservicios. Tomada de *De una arquitectura monolítica a una basada en microservicios* [Figura]. Abaunza, 2020, https://juancarlosabaunza.com/arquitectura-microservicios/

Con la arquitectura de Microservicios, una aplicación está compuesta de servicios independientes que pueden haber sido desarrollado con herramientas diferentes y si uno de estos falla, no afectará el desempeño de los demás. En este sentido, "Los servicios se crean para las capacidades empresariales y cada servicio desempeña una sola función" (Amazon, s.f.), además la comunicación entre ellos se da por medio de interfaces establecidas por medio de APIs. Si en un futuro se requiere que la aplicación cuente con un proceso adicional basta con crear un servicio nuevo que cumpla dicha labor.

3.1.5.4 Infraestructura como Código

En Amazon, s.f. indican que "La infraestructura como código es una práctica mediante la que se aprovisiona y administra infraestructura con técnicas de desarrollo de código y de software, como el control de versiones y la integración continua". Aplicando esta práctica, los equipos tienen una infraestructura sobre cada proceso necesario para crear valor. "Esta solución repetible y confiable para la implementación de entornos permite a los equipos mantener entornos de desarrollo y pruebas que sean idénticos al entorno de producción" (Microsoft, s.f.).

3.1.5.5 Comunicación y Colaboración

Este es uno de los puntos clave para que las demás practicas funcionen correctamente. Utilizar herramientas sofisticadas o mejores prácticas a la hora de hacer software no es suficiente si los miembros de la organización no rompen los límites que se han impuesto por sus roles. Si los diferentes equipos de trabajo establecen normas culturales que se enfoquen en compartir

información y desarrollen el hábito de usar canales de comunicación entre ellos, será más fácil llevar un control y conocimiento del estado actual del producto que se está desarrollando.

3.1.5.6 Desarrollo Ágil

Este es un enfoque de desarrollo de software normalmente incremental, cooperativo y adaptativo para proyectos de gran complejidad y tamaño. Los equipos que trabajan bajo una metodología ágil proporcionan versiones corregidas y actualizadas con más frecuencia a los clientes, recopilan sus comentarios y luego repiten el proceso. El cliente hace parte del equipo de desarrollo, puede estar al tanto de como avanza el desarrollo del producto en cuestión y aportar ideas que permitan mejorar el desarrollo del mismo (Maida & Pacienzia, 2015; Amazon, s.f.).

Como se pudo observar, todas las practicas se complementan entre sí y el pilar fundamental que las sostiene para un correcto flujo de trabajo es el trabajo en equipo.

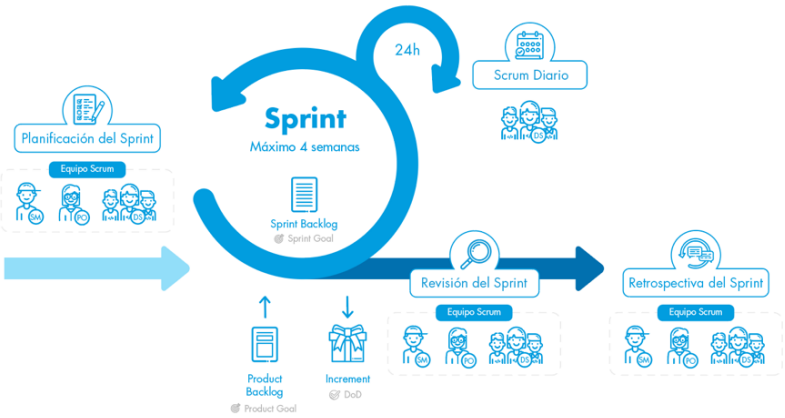
3.1.6 Scrum

Es una metodología ágil para el desarrollo de software que permite ir avanzando en un proyecto de manera incremental. Su idea general consiste en “Iterar rápido, para equivocarse rápido, para corregir rápido”. Schwaber & Sutherland (2020) describe esta metodología como un “marco de trabajo liviano que ayuda a las personas, equipos y organizaciones a generar valor a través de soluciones adaptativas para problemas complejos”.

Scrum permite seguir el ciclo de vida del software, fase por fase, pero de manera iterativa e incremental, de tal forma que cada entrega es una versión actualizada y mejorada de la anterior. Esta característica es la que hace que Scrum sea tan atractivo para los equipos de trabajo, puesto que ofrece la oportunidad de ir desarrollando y corrigiendo al mismo tiempo. Si se aplica correctamente puede aumentar la productividad de toda la operación.

Figura 10

Ciclo de Vida Scrum



Nota. Ciclo de vida de Scrum. Tomada de *Scrum: el pasado y el futuro* [Figura]. Rodríguez, 2021, : https://netmind.net/es/scrum-el-pasado-y-el-futuro/

3.1.6.1 Scrum Team

Para empezar, la unidad fundamental de Scrum es un pequeño grupo de personas, un Scrum Team. “Es una unidad cohesionada de profesionales enfocados en un objetivo a la vez, el Objetivo del Producto” (Schwaber & Sutherland, 2020). El Scrum Team cuenta con diferentes roles, pero cabe resaltar que no existe como tal una jerarquía entre ellos, son más bien definidos como responsabilidades encargadas de crear un incremento valioso durante cada iteración de trabajo.

En Scrum están definidas tres responsabilidades especificas:

- Developers, encargados de desarrollar las nuevas funcionalidades de la aplicación y producir incrementos en la misma, por medio de sus habilidades en el diseño y desarrollo de software.

- Product Owner, “es responsable de maximizar el valor del producto resultante del trabajo del Scrum Team” (Schwaber & Sutherland, 2020). Puede ser el cliente o quien conoce completamente los objetivos y tiene la idea más clara del producto a desarrollar.
- Scrum Máster, es el encargado de liderar al equipo apoyando a los demás miembros del equipo a comprender y aplicar las prácticas de Scrum para que el trabajo sea efectivo.

3.1.6.2 Eventos Scrum

Los eventos en Scrum son oportunidades para mejorar el trabajo que se está realizando; cada evento tiene una finalidad distinta que garantiza mantener la trazabilidad de un proyecto durante su ciclo de vida.

- Sprint, es el incremento que se hace a partir de una versión anterior de la aplicación, puede ser de una semana hasta varios meses. Su objetivo es el de crear y perfeccionar las funcionalidades del producto.
- Sprint Planning, es una reunión para planificar el desarrollo del próximo sprint a realizar.
- Daily Scrum, son reuniones para definir las próximas actividades a realizar e informar el avance en las mismas.
- Sprint Review, son las entregas que se hacen por parte del equipo a los interesados, estos revisan los cambios realizados al producto y colaboran con la ideación de futuras actividades para continuar (Schwaber & Sutherland, 2020; Maida & Pacienzia, 2015).

- Sprint Retrospective, es bastante importante ya que con el se analiza el trabajo realizado por el equipo durante el último sprint para luego buscar estrategias de mejora para sprints futuros.

3.1.6.3 Artefactos en Scrum

Los artefactos son aquellos elementos considerados como valor que se produce al aplicar Scrum. “Cada artefacto contiene un compromiso para garantizar que proporcione información que mejore la transparencia y el enfoque frente al cual se pueda medir el progreso” (Schwaber & Sutherland, 2020).

- Product Backlog, es la lista que enumera lo que se requiere para la creación o mejora del producto.
- Sprint Backlog, es la lista de tareas que se realizarán durante cada sprint, permite estimar el tiempo que se requerirá para cumplir con el objetivo de este.
- Increment, es el resultado de cada sprint, puede verse como la suma de todos los sprint que se han completado.

3.1.7 Cloud Computing

Es un modelo para el uso de recursos informáticos alojados en servidores en la nube tales como aplicaciones, servidores, herramientas de desarrollo, etc; que pueden ser liberados fácilmente por el proveedor del servicio. Este modelo permite conectarse y utilizar a una infraestructura por medio de internet y utilizar sus servicios sin la necesidad de instarlos de manera local (Vennam, 2020; Cierco, 2011). Existen tres modelos bastante utilizados a la hora de optar por cloud computing, a continuación, se explicará en que consiste cada uno:

3.1.7.1 IaaS (Infraestructura como servicio)

Este modelo consiste en la utilización de servicios de computación como el procesamiento y almacenamiento de información dándole la oportunidad al usuario de ejecutar cualquier tipo de software, lo que elimina la necesidad de hacer grandes inversiones innecesarias, evitando el gasto exagerado de recursos para adaptarse al expansionismo de la tecnología (Cierco, 2011; Vennam, 2020).

3.1.7.2 PaaS (Plataforma como servicio)

En este caso, el usuario puede desplegar aplicaciones desarrolladas por el mismo o adquiridas a terceros en la infraestructura del proveedor, teniendo que ajustarse a la estructura establecida por este tanto en el uso de recursos como en el uso de herramientas de desarrollo (Cierco, 2011; Vennam, 2020),"El proveedor aloja el hardware y el software en su propia infraestructura, y ofrece la plataforma al usuario como una solución integrada, una pila de soluciones o un servicio a través de Internet" (Red Hat, 2018).

3.1.7.3 SaaS (Software como servicio)

Es una aplicación desplegada en la nube que puede ser accedida por medio de un navegador web generalmente, "brinda a los usuarios todos los beneficios de los recursos informáticos en las instalaciones sin que esto implique gastos generales, por lo que manejan las aplicaciones, los datos, el sistema operativo, el middleware y los tiempos de ejecución" (Red Hat, 2018). En este modelo el usuario consume el servicio por medio de una API utilizando solamente las funcionalidades liberadas por el proveedor y sin poder modificar o alterar de cualquier forma la infraestructura.

3.1.8 Docker

Esta es una tecnología que nace con la idea de poner fin a la típica respuesta de los desarrolladores cuando el proyecto no sirve en producción “Es que en mi maquina si funciona”. El objetivo principal de Docker es el de encapsular dentro de un contenedor el código fuente de una aplicación y junto a todas las dependencias y cosas que requiera para funcionar correctamente (Charles, 2015).

Docker permite virtualizar aplicaciones dentro de contenedores, la ventaja de esto es que no importa el sistema operativo que tengan los equipos, estos pueden correr las aplicaciones siempre y cuando hagan uso de Docker. “Los contenedores son una abstracción en la capa de aplicación que empaqueta el código y su dependencia” (Eduardo, 2020), son más ligeros que usar la aplicación original y puede ejecutarse desde cualquier sistema operativo.

3.1.9 Kubernetes

Es un sistema de código abierto para el despliegue de aplicaciones, puede verse como una plataforma de contenedores, plataforma de Microservicios o plataforma portable en la nube. Kubernetes es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Facilita la automatización y la configuración declarativa. (Kubernetes, 2022; Ahamed Bhuiyan, Rahman, & Islam Shamim, 2020). “Kubernetes no es una Plataforma como Servicio (PaaS) convencional. Ya que Kubernetes opera a nivel del contenedor y no a nivel del hardware, ofrece algunas características que las PaaS también ofrecen, como deployments, escalado, balanceo de carga, registros y monitoreo”. (Kubernetes, 2022).

3.1.10 Nginx

Es un servidor web que también funciona como proxy inverso, cache de HTTP y balanceador de carga. Nginx está diseñado para brindar un uso de memoria optimo y alta concurrencia. En lugar de crear nuevos procesos para cada solicitud web, Nginx usa un enfoque asincrónico basado en eventos donde las solicitudes se manejan en un solo hilo. Así, un proceso maestro puede controlar múltiples procesos de trabajo. El proceso maestro mantiene los procesos de trabajo, y son estos lo que hacen el procesamiento real (Kinsta Inc., 2022).

3.1.11 WebServices

Es una arquitectura que se basa en la interacción de tres roles, proveedores de servicios, centros de servicios y clientes que solicitan servicios. Las interacciones que se dan entre estos roles involucran el crear, modificar, eliminar y consultar datos. En un caso típico, el proveedor de servicios define la descripción del servicio de WebServices y la publica en el centro de registro de servicios; el solicitante del servicio usa una operación de búsqueda para recuperar la descripción del servicio del centro de registro del servicio y luego usa la descripción del servicio para vincularse con el proveedor del servicio y llamar a la implementación del servicio web o interactuar con él (Yin & Chuangwei, 2011).

3.1.12 API Gateway

Oracle, (s.f.) lo define como “un dispositivo de red virtual en una subred regional. Solo los recursos de la misma subred pueden acceder a gateways de API privada. Los gateways de API pública son de acceso público, incluso desde Internet.”. Un API Gateway es un fragmento de un programa que se encarga de unificar la publicación de APIs para que puedan ser consumidas por otros servicios o por los mismos desarrolladores. (Perez & Evgeniev, s.f.).

3.1.13 CMMI

Es un modelo de buenas prácticas que permiten a las empresas mejorar el rendimiento de los procesos de negocio y fortalecerlos. Su objetivo es permitir a los empresarios tomar decisiones enfocadas alinear las operaciones con los objetivos de negocio, eliminando así el retrabajo y permitiendo automatizar procesos, de manera que, dichas empresas pueden alcanzar sus objetivos mejorando sus capacidades clave y así centrarse en optimizar los resultados de negocio. (ISACA)

3.2 Estado del Arte

Crear una conexión que mantenga la conectividad siempre abierta y bidireccional entre una plataforma y varios servicios es un problema que va más allá de los sistemas de autenticación de usuarios, para comprender su importancia es fundamental conocer un poco de la historia que lo originó.

Anteriormente, la web fue diseñada de tal forma que la consulta de información se hacía estáticamente a archivos HTML. Posteriormente, la tecnología evolucionó y era posible crear sitios Web mucho más interactivos, las consultas ya no se hacían a archivos estáticos, sino que la información se enviaba y recuperaba directamente desde un servidor. Esto permitió que se desarrollarán aplicaciones mucho más versátiles y se solucionarán toda clase de problemas, pero aún no era suficiente. (Aalbers, 2020).

El hecho de que cada vez que se recibiera una respuesta del servidor fuera necesario volver a dibujar la página web completa no era algo muy atractivo para los desarrolladores ni mucho menos para los usuarios. Debido a esta problemática, Microsoft desarrolló una nueva función por medio de JavaScript llamada XMLHttpRequest, la cual fue adoptada por los navegadores de la

época, y hasta hoy en día aún se utiliza (Aalbers, 2020). “XMLHttpRequest permite recibir respuestas de la información de una URL sin tener que recargar la página completa. Una página web puede actualizar sólo una parte de la página sin interrumpir lo que el usuario está haciendo. XMLHttpRequest es ampliamente usado en la programación AJAX” (Mozilla, 2022).

Aunque esta nueva función tuvo bastante éxito y fue adoptada por todos los navegadores, aún mantiene varias limitaciones. La principal limitación es que para que la página web se actualice, es necesario que el cliente envié una petición a este averiguando si hay cambios en la información, lo cual, según las funciones principales de una aplicación, puede ser molesto (Aalbers, 2020).

Por lo anterior, los desarrolladores definieron un nuevo protocolo que permitiera actualizar la información en las aplicaciones sin la necesidad de estar enviando peticiones constantemente al servidor llamador WebSocket. Esta tecnología permite enviar mensajes entre servidor y cliente en cualquier momento. Es posible crear aplicaciones seguras de comunicación en tiempo real sin tener que aprovisionar ni administrar servidores para administrar las conexiones o los intercambios de datos a gran escala (Amazon Web Services, Inc, 2022).

Entre los tipos las aplicaciones más demandantes de esta tecnología están: Aplicaciones de chat (WhatsApp y Messenger), videojuegos online, servicios de streaming, entre otros.

Generalmente, para la implementación de un servidor WebSocket se siguen una serie de pasos, desde la apertura del servidor, hasta el cierre y control de estado de este. Pero entre estos pasos hay dos que son esenciales para el flujo correcto de datos de punto a punto: el control de velocidad y consultas dentro del servidor. Sin embargo, esto no siempre es tenido en cuenta.

Rodas Vasquez & Valencia Carrasquilla, (2018), desarrollaron el prototipo de una plataforma tecnológica para la transmisión de multimedia en tiempo real empleando tecnología websocket, en este dos personas podían mantener una conversación ya fuera por mensajes (texto) o usando video (streaming). Para ello desarrollaron e implementaron un servidor WebSocket encargado de mantener la comunicación entre clientes, de tal forma que esta fuese lo más fluida posible.

En su proyecto el control del límite de velocidad y consultas es esencial puesto que, si en la conversación entre dos clientes uno presenta fallas en su conexión a internet, la conexión puede romperse. Además, suponiendo que se manejen chats grupales, limitar la cantidad de mensajes que cruzarán el servidor debe ser considerado para evitar congestión de mensajes y, por ende, perdida de los estos. Rodas Vasquez & Valencia Carrasquilla, (2018), no definen como darle manejo a este problemática, si se piensa en un ambiente de producción en el mundo real, puede surgir varios problemas que se convertirán en horas adicionales de desarrollo para solucionarlos y quejas de usuarios molestos.

Para el presente proyecto, se han definido una serie de pasos que, si bien incluyen algunos de los mencionados y utilizados por los autores del proyecto anterior, están más desglosados y estructurados. Esto con el fin de garantizar un desempeño optimo del producto que se quiere desarrollar.

Para evitar perdida de información que puede originarse al momento de enviar información de un punto a otro en una posible intermitencia en la conexión, se implementó un buffer que almacena los mensajes que pueden quedar represados hasta el momento en que se restablezca la normalidad y estos puedan ser enviados normalmente sin necesidad de terceros. También se planea

limitar el número de mensajes que circula en el servidor con la idea de que los canales de comunicación puedan funcionar continuamente y sin interrupciones.

4. Desarrollo del Trabajo

Para dar desarrollo a este trabajo, se tomaron como enfoque el conjunto de prácticas DevOps con la idea de que el proceso de construcción del producto a realizar se haga de la manera más optima posible. Adicionalmente, para este proceso se aplicará la metodología Scrum siguiendo una serie de fases establecidas que se indagarán más adelante. Cabe resaltar que estas fases fueron los sprint que se desarrollaron y se aplicaron de manera que se cumpliera completamente con el ciclo de vida del desarrollo de software.

Para el desarrollo Scrum se llevó a cabo Sprints que tuvieron diferentes periodos de tiempo según el tamaño y complejidad de los Sprint Backlog que lo conforman. Para ello, primero se definieron la lista de tareas que conformaron el Product Backlog según las características del producto que se iba a desarrollar, luego se hizo la estimación donde se establecieron variables de interés tales como tiempo de desarrollo por Sprint Backlog, nivel de complejidad y nivel de prioridad.

Para un mejor control de actividades, se usó Zoho Sprints que es una herramienta para la gestión ágil de tareas que componen un determinado proyecto, en esta es posible listar las tareas por realizar, indicar cuando cada una esté en proceso y asignando el tiempo de inicio de finalización de esta junto a los entregables solicitados para su posterior revisión. Una vez la tarea es puesta en revisión, si se cumplió satisfactoriamente se mueve a terminado, permitiendo tener trazabilidad en el flujo de tareas.

Por último, también se aplicaron prácticas que conforman el Modelo Integrado de Madurez de Capacidades, CMMI, con el fin de garantizar un buen rendimiento en el desarrollo de las actividades que comprendieron el desarrollo del proyecto, además de acoplarse a los procesos establecidos en el marco de trabajo de A&A soluciones TIC.

4.1 Fase de Planeación

Esta fase estuvo dividida en dos partes y dio cumplimiento con el objetivo específico 1 (Ver Sección 2.2), aquí se tomaron decisiones que tienen que ver con la arquitectura del producto que se realizó y que herramientas serían las más adecuadas para el desarrollo de este.

4.1.1 Definición del HandShake entre Whakau, Servidor y Fabricantes

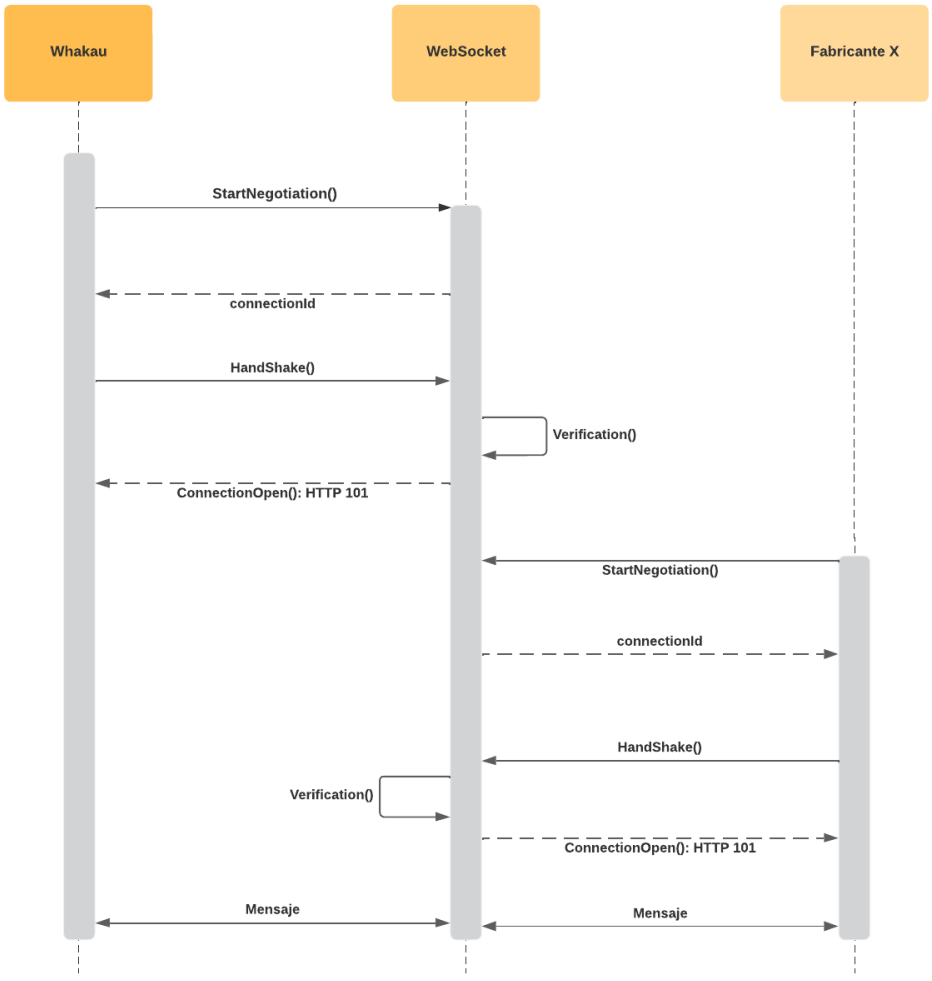
Como actividades iniciales para la fase de planeación, se vio la importancia de establecer como funcionaría la comunicación entre los diferentes clientes y el servidor. Más allá de las tecnologías a utilizar, se requería tener un mayor entendimiento de todo el proceso que permitía teóricamente la comunicación en tiempo real y asíncrona entre aplicaciones. Por lo anterior, se diseñó un diagrama de secuencia (Ver Figura 11), que describe paso a paso como se conecta el frontend de Whakau y los fabricantes al servidor WebSocket.

Es importante comprender que, aunque la comunicación que se busca garantizar es por medio del protocolo WebSocket, el establecimiento de esta se da inicialmente por medio de HTTP. Esto se debe a que antes de que se establezca la conexión, primero se da una negociación entre cliente y servidor. Una vez inicia el cliente, este envía una petición HTTP al servidor para iniciar la negociación en la que envía información importante como el tipo de transportes que permite y un id de conexión.

El servidor analiza la información enviada por el cliente, si todo está en orden hace un proceso de encriptación del id de conexión y un token proveído por el servidor. Por último, envía el nuevo id de conexión al cliente con la configuración necesaria para hacer la conexión.

Figura 11

HandShake entre clientes y servidor



El cliente recibe el nuevo id y se hace el handshake o apretón de manos, es decir, se acepta la apertura del canal de comunicación y la información puede viajar correctamente.

4.1.2 Análisis de la Data Entregada por Fabricantes

Ya que la idea principal de crear un servidor websocket es garantizar un flujo de información, se hizo el análisis de la data que entrega cada fabricante.

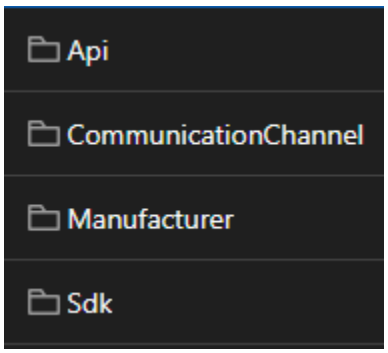
Como se ha dicho anteriormente, Whakau busca integrar diferentes fabricantes de biometría y proximidad, de manera que usando un solo sistema se puedan gestionar los distintos sistemas de seguridad que manejan estos fabricantes. En este orden de ideas, era importante estandarizar la información de manera que se maneje la misma para todos.

4.1.3 Revisión del Proyecto Fabricantes

Se hizo la revisión de los proyectos que componen hasta el momento Whakau, principalmente en aquel que gestiona su backend. Este proyecto está conformado por varios microservicios encargados de manejar diferentes funcionalidades, entre estos se encuentra ManufacturerService, proyecto encargado de gestionar información general de cada fabricante, datos para conexión vía API y SDK y, hasta ese momento, un proyecto que funcionaba para comunicar Whakau con los fabricantes.

Figura 12

Estructura del ManufacturerService



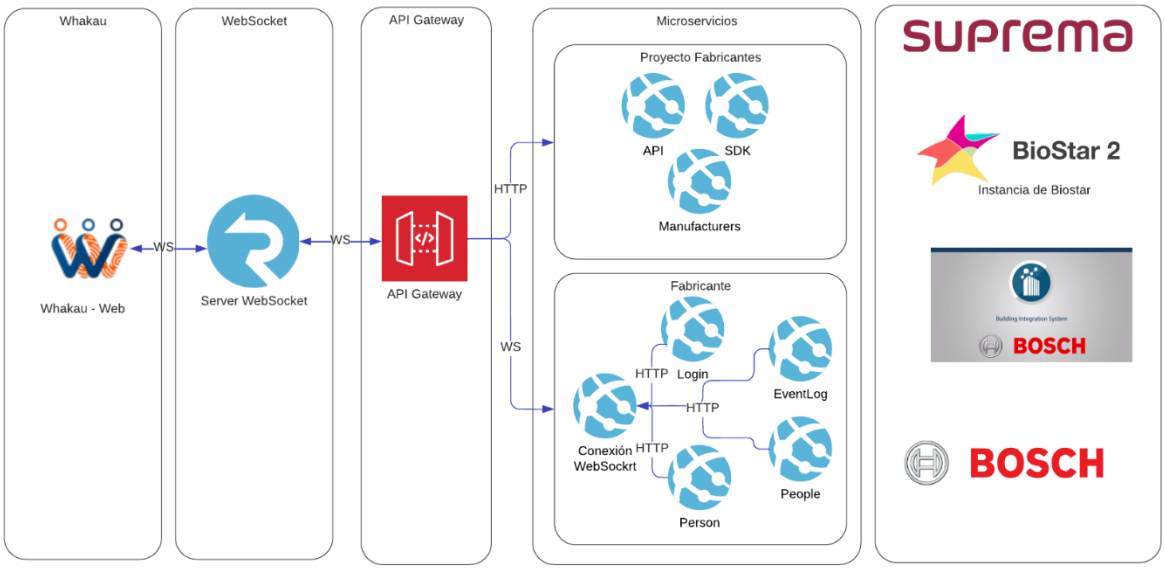
4.1.4 Arquitectura de Integración

Luego de revisar el proyecto de fabricantes, se planteó un modelo inicial para definir la arquitectura que tendría la comunicación de Whakau una vez integrado el servidor WebSockets.

El objetivo principal era garantizar que la comunicación entre Whakau y cada fabricante siempre este abierta. Además, también se buscaba actualizar en tiempo real los eventos detectados por cada sistema de autenticación, por ejemplo, cuando un usuario usa un biométrico, sea que este autorizado o no, el fabricante recibirá una notificación; dicha notificación debía ser enviada también a Whakau para visualizarla desde su propio panel de control.

Figura 13

Arquitectura de Integración WebSocket



En este parte, se vio la necesidad de integrar también un API Gateway que funcionará como mediador entre el servidor WebSocket y los servicios que conforman el backend de Whakau,

además, también se incluyó el cliente que debía comunicarse mediante el protocolo http con los servicios del fabricante y, vía WS con el servidor WebSocket.

4.1.5 Definición de tecnologías.

4.1.5.1 Prueba de Concepto. Al inicio del proceso, se realizó una prueba de concepto con la idea de poder determinar las tecnologías idóneas que se usarían para el desarrollo del proyecto. El ejercicio se enfocó de desarrollar un servidor que implemente las librerías adecuadas, permitiendo la conexión vía WebSocket con los clientes. El servidor a su vez se comunicaría con Firebase para la autentificación de usuarios, retornando el resultado al cliente mediante WebSocket.

Whakau es un proyecto en el que se adaptó la arquitectura de microservicios, debido a la necesidad de integración de fabricantes de biometría y proximidad, se decidió usar como tecnología base ASP.NET Core la cual es un marco de desarrollo, de alto rendimiento y de código abierto para la creación aplicaciones que puedan ser ejecutadas en cualquier plataforma. La razón de esto es que el soporte brindado por los fabricantes para temas de desarrollo con sus dispositivos está en C#.

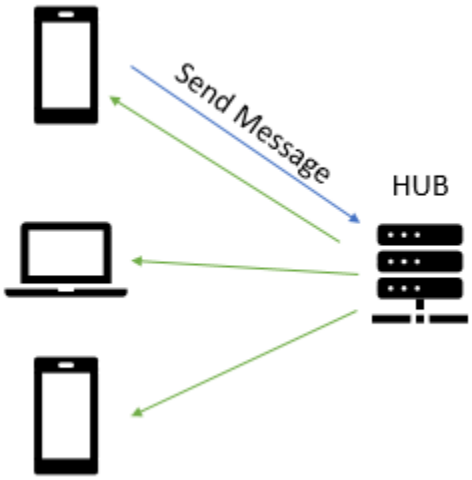
En este orden de ideas, la prueba de conceto que se ejecutó fue usando el marco de desarrollo mencionado anteriormente y verificando el soporte que da este para la comunicación por medio de WebSockets. Había dos alternativas, usar SignalR Core o usar WebSocket, ambas librerías proveídas por Microsoft y que simplifican el proceso de conectividad en tiempo real entre aplicaciones.

Como resultado de la prueba de concepto, se optó por utilizar SignalR Core para el desarrollo del presente proyecto puesto que WebSocket está muy limitada para proyectos de Net Core (Esta optimizada para ASP.NET Framework).

4.1.5.2 SignalR Core. es una biblioteca de código abierto que simplifica la adición de funcionalidad web en tiempo real a las aplicaciones. La funcionalidad web en tiempo real permite que el código del lado del servidor envíe eventos a los clientes conectados a medida que se activan en el servidor (Microsoft, 2022). Con SignalR, los clientes también pueden enviar mensajes al servidor, una capacidad que demuestra la comunicación dúplex.

Figura 14

Representación de un Hub



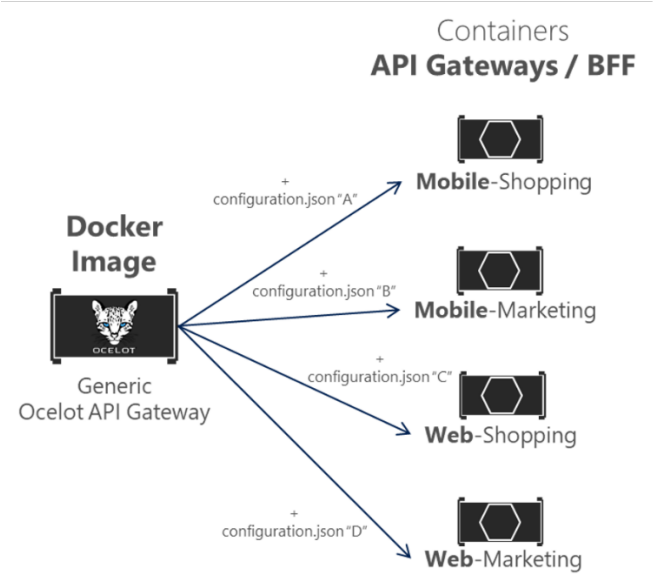
Nota. Comunicación entre cliente y servidor mediante un concentrador. Tomada de *Why SignalR (Core)* [Figura]. Montemagno, 2019 : https://montemagno.com/real-time-communication-for-mobile-with-signalr/

El termino más importante que hay que conocer a la hora de trabajar con SignalR Core es el de concentrador o Hub. Este se utiliza para comunicarse entre clientes y servidores. Un concentrador es una canalización de alto nivel que permite a un cliente y un servidor llamar a métodos entre sí. Con este fin, SignalR maneja el envío a través de los límites de la máquina automáticamente (Uso de concentradores en para ASP.NET CoreSignalR, 2022).

4.1.5.3 Ocelot. Como se determinó al momento de definir la arquitectura de integración, era necesario crear una Api Gateway que funcionará como intermediario para la comunicación entre de servicios de Whakau y el servidor WebSocket. Bajo esta premisa, se buscó una herramienta que permitiera la implementación de este patrón siendo Ocelot la seleccionada.

Figura 15

Funcionamiento de Ocelot



Nota. Representación de API Gateway usando Ocelot. Tomada de *Implementación de puertas de enlace de API con Ocelot* [Figura]. Colaboradores Microsoft Learn, 2022,

https://learn.microsoft.com/es-es/dotnet/architecture/microservices/multi-container-microservice-net-applications/implement-api-gateways-with-ocelot

4.1.5.4 RabbitMQ. Es un qué sistema de mensajería que implementa el Protocolo avanzado de cola de mensajes (AMQP). Este fue de gran importancia para la gestión de información que viajaría por medio de SignalR Core ya que funciona como una cola de mensajes, cada mensaje queda represado en la cola esperando a ser tratado, lo que garantiza que los datos siempre lleguen a su destino, aun si por ejemplo se pierde la conexión.

4.1.6 Configuración del Entorno de Desarrollo

Ya habiendo establecido las tecnologías que se utilizarían en el desarrollo del proyecto, se hizo la configuración del entorno que garantizará el CI/CD y más prácticas de la filosofía DevOps.

4.1.6.1 Creación de Proyectos y Dockerización. Como primera parte, se crearon los proyectos SignalR Whakau, para el desarrollo del servidor WebSocket y Api Gateway para la implementación de la puerta de enlace con Ocelot. Posteriormente, se creó la dockerfile para cada uno de ellos.

Whakau cuenta con un paquete NuGet desarrollado para optimizar y estandarizar algunas funcionalidades que son requeridas en todos los servicios construidos que lo conforman. El nombre del paquete es Whakau Tools y, para poder implementarlo en cada proyecto es importante agregar el origen de datos correspondiente el cual se encuentra registrado en GitLab. Esto hace que sea necesario incluir la carga de este paquete en el docker file de los proyectos que lo requiera, además que añadir el acceso a las credenciales y demás variables de entorno que se requieren para el para la ejecución de funciones que conforman el paquete.

Por último, se crearon los respectivos repositorios remotos en Whakau y se hizo el enlace pertinente con el entorno local para su comunicación.

4.1.6.2 Construcción de Jobs. Los proyectos, aplicaciones o microservicios de Whakau se despliegan implementando contenedores docker, para los cuales primeramente se requiere crear u obtener la imagen final y personalizada del proyecto a implementar, para a partir de la imagen crear el o los contenedores que se deseen.

Para usar GitLab CI/CD es necesario crear un archivo llamado **.gitlab-ci.yml** en la raíz del repositorio, que contiene la configuración de CI/CD. En el archivo se puede definir:

- Los scripts que desea ejecutar.
- Otros archivos de configuración y plantillas que desee incluir.
- Dependencias y cachés.
- Los comandos que desea ejecutar en secuencia y los que desea ejecutar en paralelo.
- La ubicación en la que se va a implementar la aplicación.

Los scripts se agrupan en trabajos (jobs) y estos se ejecutan como parte de una canalización más grande (pipeline). Se puede agrupar varios trabajos independientes en etapas que se ejecuten en un orden definido. La configuración de CI/CD necesita al menos un trabajo que no esté oculto. Cuando agrega un archivo a su repositorio, GitLab lo detecta y una aplicación llamada GitLab Runner ejecuta los scripts definidos en los trabajos configurados.

La nomenclatura propuesta para el nombre de los Jobs es la siguiente:

- Las dos primeras letras del nombre del job indican su rama y su tarea o etapa.
- En nombre del Job se complementa con el nombre del proyecto.

Son tres etapas principales las que deben tenerse en cuenta, cada una de ellas se ejecuta según el entorno que sea requerido.

Figura 16

Ejemplo de un Job

```
DB-SignalR_Whakau:
  image: docker:20.10.16
  stage: Build
  services:
    - docker:20.10.16-dind
  before_script:
    - docker login -u $CI_DEPLOY_USER -p $CI_DEPLOY_PASSWORD $CI_REGISTRY
  variables:
    DOCKER_HOST: tcp://docker:2375
    DOCKER_TLS_CERTDIR: ""
    DOCKER_DRIVER: overlay2
  script:
    - docker build -t registry.gitlab.com/whakau/websockets/dev-signalrwhakau:latest .
    - docker push registry.gitlab.com/whakau/websockets/dev-signalrwhakau
  only:
    - Development
```

4.1.6.2.1 Build. La primera de ellas es la etapa build o construcción, en esta, se construye la imagen en base al dockerfile y se publica en container registry de GitLab.

4.1.6.2.2 Test. Es donde el proyecto es testeado, su código fuente es analizado, probado y se implementan los diversos métodos para validar la calidad de su construcción.

Dentro de Whakau se implementó el uso de SonarQube para evaluar la calidad del código fuente aplicando su funcionamiento en dos entornos, uno local y desarrollo, realizando un análisis estático sobre este, con el objetivo de encontrar diferentes puntos a mejorar.

4.1.6.2.3 Deploy. Como etapa final se tiene la etapa de Deploy, los jobs asociados a esta etapa son los encargados de hacer el despliegue a los diferentes entornos y plataformas.

Para el entorno de Whakau, los proyectos se despliegan/implementan dentro de un clúster de kubernetes en Google Cloud, se toma la imagen del container registry y con ella se crea un deployment en kubernetes, cuyo pod o pods implementaran la imagen durante la construcción del contenedor.

4.2 Fase de Desarrollo

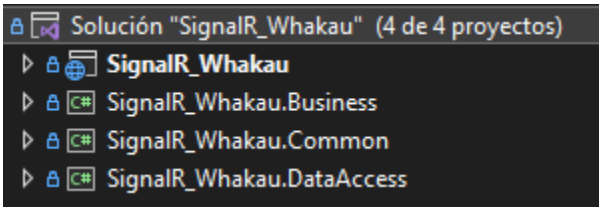
Esta fase sirvió para dar complimiento al objetivo específico 2 (Ver Sección 2.2), donde se hizo el desarrollo del servidor WebSocket de tal manera que este pudiera acoplarse a las arquitecturas basadas en Microservicios y Multitenat con las que se conforma Whakau.

4.2.1 Construcción del Servidor WebSocket

El primer paso para comenzar con el desarrollo del servidor fue la instalación de los paquetes que serían necesarios, en este caso SignalR Core 1.1.0 y RabbitMQ.Client 6.3.0.

Figura 17

Estructura de Proyecto Principal



4.2.1.1 Configuración e Implementación de SignalR Core. Con el paquete NuGet instalado se procedió a realizar la respectiva configuración, inicialmente se creó un Hub fuertemente tipado, el cual permitió mantener un control más riguroso en cuanto a los métodos que lo conformarían, además, de controlar de forma más optima, el tipo de información que se quería o no viajara por el servidor WebSocket.

Para crear un Hub fuertemente tipado, basta con crear una interfaz en la que se listen los diferentes métodos que garantizarán la comunicación entre cliente y servidor, una vez hecho esto, se debe implementar dicha interfaz en la clase Hub que se haya creado (Ver figura 17).

Figura 18

Implementación de Hub fuertemente tipado

```
public class WhakauHub: Hub<IWhakau>
{
```

Posteriormente, en el Startup.cs, el cual es el script que maneja la configuración principal de un proyecto de ASP.NET Core, se hizo el mapeo para que por medio del respectivo endpoint un cliente que usa como transporte de información WebSocket pueda comunicarse con el servidor que se estaba creando (Ver figura 18).

Figura 19

Establecimiento de la canalización con el servidor

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapHub<WhakauHub>("/whakau");
    endpoints.MapControllers();
});
```

Dentro del Hub del Servidor WebSocket se crearon dos tipos de métodos, los primeros que se usan para la invocación por parte del cliente para enviar datos al servidor y los segundos que son para él envió de información de servidor a cliente. En la Figura 10 se puede observar cómo está estructurado, siendo SendEventLog() el método que será invocado por parte del cliente y GetEventLog() el método que estarán escuchando los demás clientes. Esta es la manera estándar

en la que se puede gestionar el envío de información entre clientes por medio de un servidor WebSocket.

Cada uno de los métodos que se crearon dentro del Hub cumple una determinada funcionalidad y trabaja como una canal, de esta manera, se puede mantener un flujo de información constante entre servidor y cliente.

Figura 20

Estructura de un método dentro del Hub.

```
#region SendEventLog
/// <summary>
/// Envia eventos capturados al cliente.
/// </summary>
/// <param name="dto">Objeto tipo Lista EventDto</param>
/// <remarks>Envia mensaje a los demás clientes</remarks>
0 referencias
public async Task SendEventLog(List<EventDto> dto)
{
    _logger.LogInformation($"Eventos enviados");
    await Clients.Others.GetEventLog(dto);
}
#endregion
```

4.2.1.2 Seguridad Una vez se estableció la lógica inicial que permitirá enviar y recibir información por medio del servidor, se definieron funcionalidades que garantizarán tener una trazabilidad en las conexiones al servidor, barreras que permitan que solo los clientes permitidos puedan conectarse a este y contramedidas para evitar ataques (DoS) entre otras cosas.

4.2.1.2.1 Registro y Control. Con el fin de mantener un registro adecuado de las conexiones de nuevos clientes, así como notificaciones provenientes del servidor, se hizo uso de Logger la cual es una herramienta que permite que los logs se puedan visualizar por medio de una consola, además de poder configurar e indicar en qué momento se requiere que se muestren estos registros.

Cada uno de estos logs cuenta con un nivel que se asigna dependiendo de su tipo y la situación que lo generé durante la ejecución del programa. En la tabla 1 se exponen los niveles de logs.

Tabla 1

Enumeración de tipos Log de menor a mayor gravedad

LogLevel	Value	Description
Trace	0	Muestra todas las notificaciones.
Debug	1	Notificaciones de Depuración.
Information	2	Flujo general de la aplicación.
Warning	3	Eventos anormales o Inesperados.
Error	4	Errores y Excepciones.
Critical	5	Fallas que requieren atención inmediata.
None	6	No mostrar registros.

Nota. Enumeración de tipos de Logs junto con su nivel y descripción. Adaptado de *Logging in .NET Core and ASP.NET Core* [Figura]. Anderson, 2022, https://learn.microsoft.com/en-us/aspnet/core/fundamentals/logging/?view=aspnetcore-3.1

Para Whakau y, más específicamente, para el servidor WebSocket, se requiere un proveedor que permita gestionar estos logs de manera externa. Para este caso se implementó **Serilog**, el cual es un framework que permite controlar los logs y almacenarlos en diferentes formas y formatos.

Figura 21

Ejemplo de registro almacenados en SQL Server.

14	14	Request starting HTTP/1.1 OPTIONS http://localhos...	{HostingRequestStartingLog:l}	Information	2022-05-24 15:18:12.717	NULL	<properties>
15	15	CORS policy execution successful.	CORS policy execution successful.	Information	2022-05-24 15:18:12.737	NULL	<properties>
16	16	Request finished in 26.0531ms 204	{HostingRequestFinishedLog:l}	Information	2022-05-24 15:18:12.743	NULL	<properties>
17	17	Request starting HTTP/1.1 POST http://localhost:50...	{HostingRequestStartingLog:l}	Information	2022-05-24 15:18:12.753	NULL	<properties>
18	18	CORS policy execution successful.	CORS policy execution successful.	Information	2022-05-24 15:18:12.753	NULL	<properties>
19	19	Executing endpoint ""/whakau/negotiate""	Executing endpoint '{EndpointName}'	Information	2022-05-24 15:18:12.757	NULL	<properties>
20	20	Executed endpoint ""/whakau/negotiate""	Executed endpoint '{EndpointName}'	Information	2022-05-24 15:18:12.787	NULL	<properties>
21	21	Request finished in 37.3082ms 200 application/json	{HostingRequestFinishedLog:l}	Information	2022-05-24 15:18:12.790	NULL	<properties>
22	22	Request starting HTTP/1.1 GET http://localhost:500...	{HostingRequestStartingLog:l}	Information	2022-05-24 15:18:12.807	NULL	<properties>
23	23	CORS policy execution successful.	CORS policy execution successful.	Information	2022-05-24 15:18:12.810	NULL	<properties>
24	24	Executing endpoint ""/whakau""	Executing endpoint '{EndpointName}'	Information	2022-05-24 15:18:12.813	NULL	<properties>
25	25	Se acaba de conectar un nuevo cliente	Se acaba de conectar un nuevo cliente	Information	2022-05-24 15:18:12.897	NULL	<properties>

En la figura 21 se puede observar los logs de arrojados por el servidor cuando se esta conectando un nuevo cliente a este, muestra desde el inicio de la negociación hasta la ejecución de un log personalizado avisando de la conexión de un nuevo cliente.

4.2.1.2.2 Limitaciones de Velocidad. La limitación de velocidad ayuda a proteger un servidor contra ataques de fuerza bruta. Por ejemplo, un hacker puede usar bots para realizar solicitudes repetidas a un punto de enlace de API. Debido a la cantidad de solicitudes repetidas, el resultado será un servicio no disponible para otros clientes. Esto se denomina ataque de denegación de servicio (DoS).

Existen diferentes maneras de definir reglas para la limitación de peticiones, en este proyecto se usó un paquete NuGet llamado AspNetCoreRateLimit, con el que se estableció una serie de normas para controlar para controlar el número máximo de peticiones que se pueden realizar cada determinado tiempo. Así mismo se definieron algunos endpoint que estarían exentos de este control y se asignó el código de estado que se mostrará cuando se supere el límite establecido, en este caso el 429.

Figura 22

Configuración Básica de AspNetCoreRateLimit.

4.2.1.2.3 Autorización de Conexiones. Para el tema de autorización en el proyecto se usan Json Web Tokens o JWT. Este es un token de acceso estandarizado que permite el intercambio seguro de datos entre dos partes. Es un mecanismo que permite saber si un cliente está o no autorizado para acceder a diferentes funcionalidades de una API (Ionos, 2020).

Whakau cuenta con servicios encargados de manejar diferentes funcionalidades de la aplicación, entre ellas están UserValidation y TokenGenerator. La primera valida el email y contraseña de una persona, si los datos están correctos, devuelve el usuario. El segundo, a partir del usuario genera un token de acceso con el que se abrirá la sesión en Whakau.

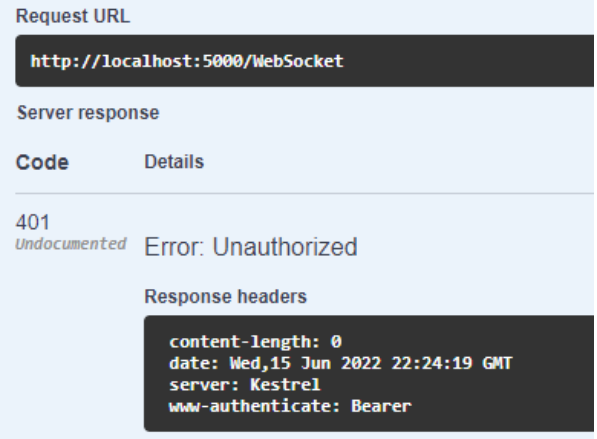
Cada vez que se inicie el proceso de HandShake o negociación entre un cliente y el servidor WebSocket, este deberá proporcionar un Token valido para que así se pueda establecerse la conexión entre ambos y así la información fluya en dos vías. En este orden de ideas, se requiere

que el servidor pueda recibir el Token y validar que este cumple con las características necesarias para dar acceso.

Con esto en mente, se desarrolló un middleware que se encargue se la captura de los Tokens recibidos por parte de clientes que requieran conectarse, seguido a esto validar que este sea autentico, si es autentico, la conexión se establecerá correctamente, de lo contario recibirá un código de estado 401 indicando que no se encuentra autorizado.

Figura 23

Petición rechazada por el servidor al no estar autorizado.



4.2.1.3 Servicios en Segundo Plano. La principal problemática hasta la fecha que se venía presentando en Whakau, era la expiración de los Token de usuario proveídos por cada fabricante. En el caso de Suprema y su sistema de seguridad Biostar2, el token vencía luego de 30 a 60 minutos aproximadamente.

Con esto en mente, se formuló una solución basada en servicios ejecutados en segundo plano que permitiría mantener la conexión entre Whakau y cada fabricante siempre activa,

consiguiendo así que no sea necesario iniciar sesión en cada fabricante cuando se inicie sesión en Whakau.

Cada fabricante maneja una manera de gestionar sus sesiones, ya sea tokens o cookies que deben enviarse en cada petición para poder acceder o enviar información. En Whakau, cada vez que se inicia sesión en un fabricante, este token o cookie es almacenado en Redis asignándole un key que de acceso para un posterior uso.

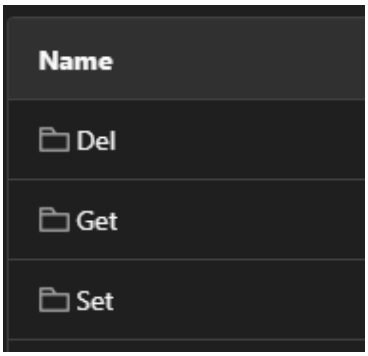
Bajo esta premisa, se crearon tres servicios trabajaran de manera conjunta para que cada determinado tiempo (20 - 30 minutos), consulta los tokens de usuario almacenados en Redis, los agrega a la cola de RabbitMQ y posteriormente se envían uno a uno por medio de SignalR Core a un cliente encargado de hacer la respectiva renovación.

En ASP.NET Core, el desarrollo de servicios en segundo plano se hace mediante la implementación de una interfaz llamada **IHostedService**, esta permite hospedar tareas en segundo plano utilizando dos métodos principales, StartAsync() para iniciar la tarea y StopAsync() para terminarla.

4.2.1.3.1 Modificaciones al Sistema Caché de Whakau. Para el manejo de cache Whakau cuenta un con proyecto encargado de interactuar con una instancia ya desplegada de Redis, el nombre del proyecto es Redis_Whakau. Este proyecto cuenta con tres microservicios como se puede (Ver en la Figura 22), y cada uno de ellos realiza una determinada función.

Figura 24

Estructura de Redis_Whakau.



Como se puede intuir, Set se encarga de insertar elementos clave valor en Redis, Get por su parte permite acceder a estos registros mediante una Key y Del es el encargado de eliminar registros. Los tres hacen uso del paquete NuGet Whakau Tools, ya que en este se establece la lógica para mapear los datos de entrada y salida para Redis.

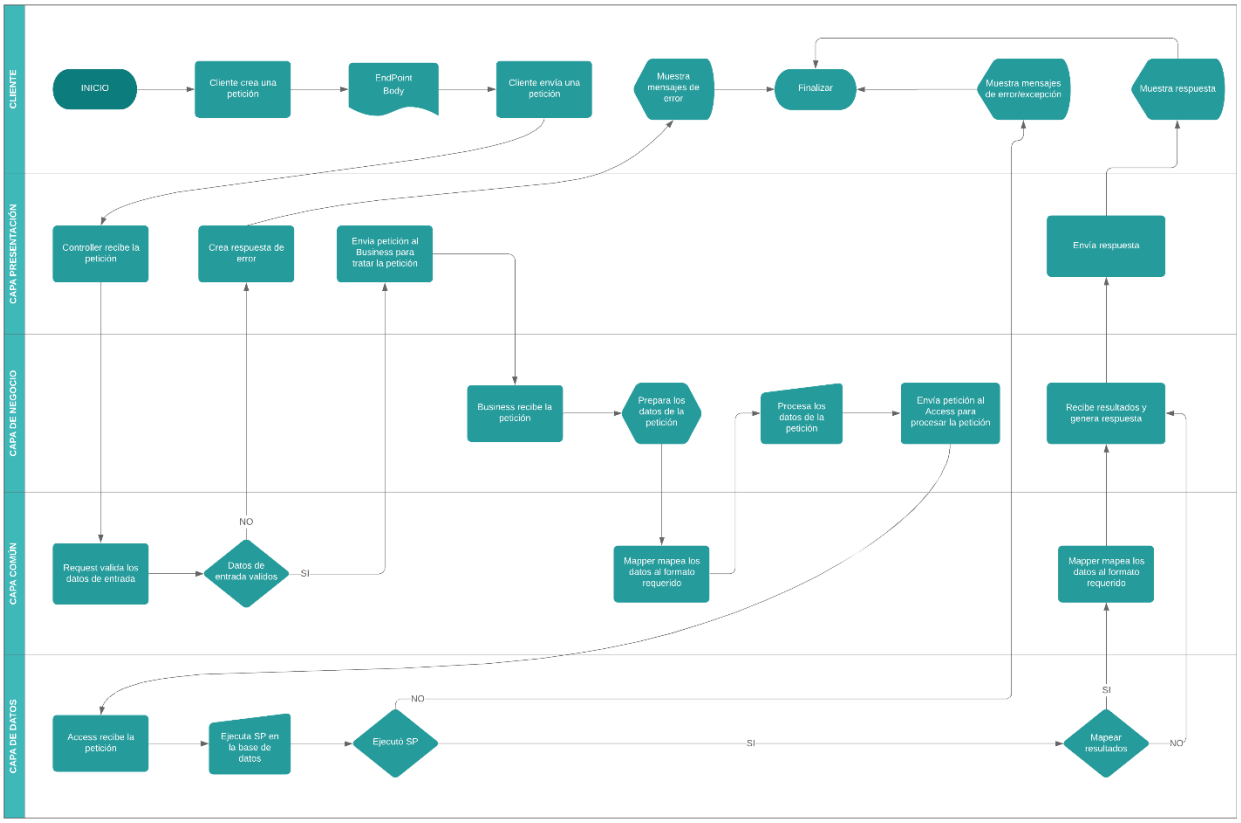
Fue necesario refactorizar tanto estos microservicios como en el paquete NuGet mencionado, puesto que se necesitaba poder almacenar diferentes estructuras que no fueran directamente un String como se venía trabajando hasta entonces, para ello, se modificó de tal forma que los datos almacenados en Redis sean de tipo Byte[], de esta manera cada vez que se guarde algo, será deber del desarrollador convertirlo al tipo de dato correspondiente.

4.2.1.3.2 Servicio Gestor de Tokens. Una vez que se hicieron las modificaciones mencionadas anteriormente, se procedió con el desarrollo de la lógica para consultar la lista de tokens y poder enviarlos al cliente correspondiente.

Lo primero fue establecer la comunicación con el microservicio Get perteneciente al proyecto Redis_Whakau, el objetivo es consultar por medio de una clave secreta y obtener el listado con todos los tokens que están almacenados. Dicha consulta se hará periódicamente con la idea de obtener antiguos y nuevos registros. Si la consulta trae resultados consigo, se ejecuta una función encargada de mapear los datos de Byte[] al formato.

Figura 25

Flujograma de proceso para realizar una petición.



Con los datos ya mapeados, estos se iteran y se van agregando uno por uno a la cola de RabbitMQ.

4.2.1.3.3 Servicios Receptor y Emisor de Mensajes. Existen varias maneras de implementar el funcionamiento de RabbitMQ, para este caso se adoptó la expuesta en la Figura 23 en la que un elemento se agrega a la cola por medio de un productor de mensajes y es retirado puesto a disposición por parte del consumidor.

Figura 26

Flujo aplicado con RabbitMQ.

Nota. Representación del flujo de RabbitMQ. Tomada de *RabbitMQ tutorial - «Hello World!»* [Figura]. RabbitMQ, (s.f.), https://www.rabbitmq.com/tutorials/tutorial-one-dotnet.html

Con esta lógica definida, se crearon dos servicios en segundo plano, Producer y Consumer, ambos escuchando y esperando cambios para interactuar. Cuando se envía un nuevo token para renovar, este es recibido por el Producer quien codifica la información y la agrega a la cola. Una vez acá, el consumidor revisa la cola, saca un elemento, lo decodifica para seguidamente enviarlo vía WebSockets al cliente indicado.

La ventaja de esto es que, si se envían muchos mensajes, estos no se perderán, el consumidor simplemente ira sacando elementos uno por uno, si por algún motivo hay un problema de conexión, estos mensajes se mantendrán en la cola hasta que haya normalidad, una vez se restablezca la conexión, seguirá enviando elementos dese donde se quedó.

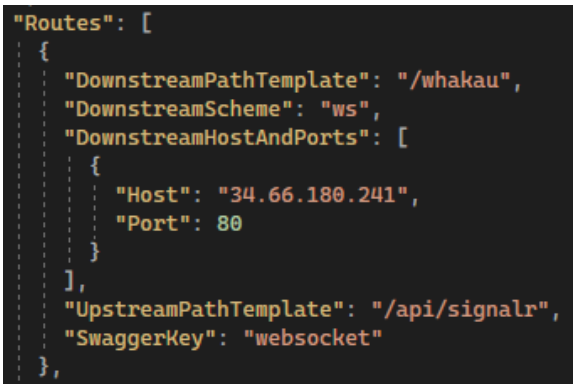
4.2.2 Construcción del Api Gateway

El último paso importante antes de pasar a realizar la integración de la comunicación de los servicios de fabricantes y el servidor WebSocket es el desarrollo del API Gateway.

Como se mencionó anteriormente, la herramienta que se eligió para aplicar este patrón fue Ocelot el cual es un proyecto de código abierto que contiene un grupo de middlewares para la gestión de un API Gateway.

Figura 27

Estructura de enrutamiento de Ocelot.



```
"Routes": [
  {
    "DownstreamPathTemplate": "/whakau",
    "DownstreamScheme": "ws",
    "DownstreamHostAndPorts": [
      {
        "Host": "34.66.180.241",
        "Port": 80
      }
    ],
    "UpstreamPathTemplate": "/api/signalr",
    "SwaggerKey": "websocket"
  },
```

La estructura básica para el enrutamiento y direccionamiento se hace por medio de un script de tipo JSON, este debe ser cargado por medio del Startup.cs para su ejecución. A continuación, esta explicado cada una de las directivas que se deben configurar:

- **UpstreamPathTemplate:** En este objeto se indica la ruta de salida del enlace hecho con otro servicio.
- **UpstreamHttpMethod:** Acá se define el tipo de petición que se puede realizar, si no se configura, Ocelot lo toma por defecto de manera que la ruta servirá para los cuatro verbos principales (POST, GET, PUT, DELETE).
- **DownstreamHostAndPorts:** Se define el host y el puerto del servicio que se enrutará.
- **DownstreamPathTemplate:** Este es el endpoint del servicio que queremos redireccionar por medio del API Gateway.

Por último, está el objeto **DownstreamSheme**, normalmente este no se configura ya que por defecto la comunicación que toma es tipo HTTP, pero teniendo en cuenta que se requiere comunicación mediante WebSocket, es necesario indicarlo. Además, hay que tener cuenta que como se mencionó anteriormente, la negociación entre un cliente y el servidor se da vía HTTP, lo

que hace necesario que se definan dos objetos de enrutamiento, uno para comunicación por WS y, otro para la negociación inicial entre cliente y servidor (Ver Figura 25).

Figura 28

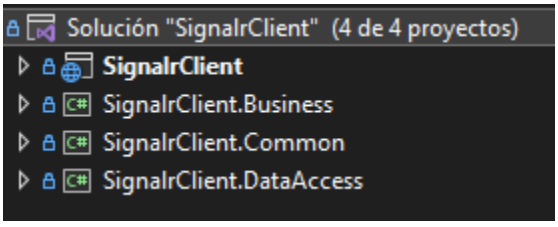
Enrutamiento para la negociación entre cliente y servidor.

4.3 Fase de Integración

Esta fase comprende una parte esencial con la que se dio cumplimiento al objetivo específico 3 (Ver Sección 2.2). Aquí se integraron los fabricantes de biometría y/o proximidad de tal forma que la comunicación entre ellos y Whakau se haga por medio del servidor Web Socket.

Figura 29

Estructura de Microservicios en Whakau.



Una vez se culminó el desarrollo del servidor WebSocket junto con la implementación del API Gateway, el paso a seguir fue desarrollar un nuevo microservicio que funcionara como cliente interactuando con el servidor y a su vez realizando peticiones a los fabricantes. Este microservicio

consta de dos partes, la primera es una tarea ejecutándose en segundo plano que estará escuchando al servidor WebSocket, cada vez que el servidor envíe información, este la capturará, mapeará y hará la petición requerida al software del fabricante. La segunda parte se encargará de esta última parte, la petición que se hará al fabricante, su flujo de proceso es el mismo que el expuesto en la Figura 23.

En el servicio en segundo plano, se estableció la comunicación al servidor WebSocket por medio del endpoint expuesto por el API Gateway y estableciendo un token de acceso para estar autorizado al momento de la negociación, posteriormente, se definieron las funciones que estarían escuchando y recibiendo información por parte del servidor y enviándola también cuando fuese necesario. Una vez que el cliente recibe un dato por parte del servidor, según el canal por el que llego se ejecutará la petición indicada, es en esta parte se pasa de usar el protocolo WS a usar el HTTP. Seguido a esto, si la petición tiene una respuesta, esta es mapeada y enviada nuevamente al servidor WebSocket usando el protocolo requerido.

4.4 Fase de Pruebas

Esta fase dio cumplimiento al objetivo específico 4 (Ver Sección 2.2). Acá se definió un plan de pruebas que permitió evaluar el producto realizado para verificar que cumpla con todas las características establecidas al principio.

A&A Soluciones – TIC, mantiene estandarizados todos los procesos concernientes a las diferentes actividades que dan valor al negocio, entre estos procesos, se encuentra el definido para la planeación y pruebas de los diferentes proyectos que se manejan en la empresa. Por lo anterior, se siguió este estándar con el fin de hacer la respectiva documentación del proceso y así mantener la trazabilidad que permita mejorar en cada iteración las funcionalidades del proyecto.

Las pruebas que se ejecutaron fueron pruebas funcionales, la idea principal era verificar el correcto funcionamiento de las partes que componen el proyecto, no solo en el entorno local, sino también en el despliegue hecho en el clúster de Kubernetes, es decir, que los pods se estuvieran comunicando entre sí por medio de los servicios establecidos internamente.

4.4.1 Validación JWT y Establecimiento de la Conexión con el Servidor. Como primer conjunto pruebas, se verifico que la conexión entre cliente y servidor se estableciera correctamente siempre y cuando el cliente contará con un token valido.

Figura 30

Conexión Satisfactoria entre Cliente y Servidor.

Como se puede observar, primero se hace la verificación del token, una vez este es autorizado, en ese momento se ejecuta el endpoint para comunicación vía WS y se notifica de la conexión de un nuevo cliente. También se pudo comprobar que el servidor si estaba notificando la nueva conexión lo cual es importante para futuras implementaciones que se pueden producir.

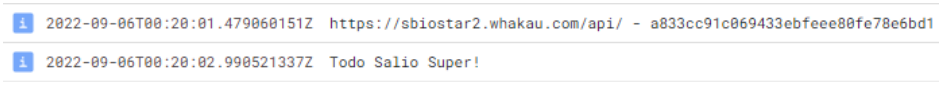
4.4.2 Proceso para la Renovación del Token de Fabricante. Continuando, se requería comprobar que todo el flujo requerido para que se renueven correctamente los tokens de fabricantes si se esté ejecutando correctamente, puesto que ese es el corazón de este proyecto.

Para la ejecución de esta prueba, se debía tener funcional los microservicios encargados de la comunicación con Redis para poder consultar la lista de tokens, la correcta ejecución de la instancia de RabbitMQ junto a las tareas en segundo plano desarrolladas para comunicarse con

esta. Por último, el servicio al cual se haría la petición para evitar que el token caduque también debía estar corriendo correctamente.

Figura 31

Renovación de Token de Fabricante satisfactoria.



Con todo esto, como se puede ver en la Figura 31, el token llego correctamente al cliente y este pudo ejecutar la petición sin problema, si por algún motivo el token ya estuviese vencido o los datos para realizar la petición fueran incorrectos, se recibiría un mensaje indicando que hubo un error, por lo que la prueba se considera como ejecución exitosa.

5. Conclusiones

Después de todo el trabajo realizado durante el desarrollo del trabajo de grado es necesario determinar si se ha logrado cumplir con el objetivo principal de este proyecto: Diseñar y desarrollar un servidor WebSocket que permita la conectividad en tiempo real y asíncrona entre tecnologías de distintos fabricantes de sistemas de autenticación de usuarios basados en biometría o proximidad utilizando una arquitectura Multitenat.

A partir del desarrollo realizado se puede concluir que, si se logró el objetivo principal del proyecto, se desarrolló una capa de conectividad en tiempo real y asíncrona entre los fabricantes de autenticación y Whakau. Para esto, se hizo un trabajo de ingeniería siguiendo el ciclo de vida del software que estuvo repartido en cuatro fases, cada una de estas permitiendo dar cumplimiento a los objetivos específicos planteados al inicio de este documento. Durante todo el proceso, se presentaron diferentes situaciones que pusieron a prueba distintas habilidades fundamentales que debe tener todo ingeniero, como por ejemplo la investigación y análisis, para así encontrar las alternativas adecuadas que permitieron completar con éxito el proyecto.

Es importante concluir que, durante la fase de planeación, se planteó la idea de usar el servicio CommunicationChannel para la comunicación entre el servidor WebSocket y los fabricantes de autenticación, pero luego de revisar su contenido y funcionamiento, se encontró que este funcionaba similar a un API Gateway pero su rendimiento no era optimo; cada vez que se requería hacer una petición a un servicio, se ejecutaban dos peticiones, primero una al servicio CommunicationChannel, y luego este hacía una al fabricante requerido, ocasionando que el tiempo de espera fuera excesivamente largo.

Por último, es significativo recalcar la importancia de una práctica empresarial para complementar la formación universitaria, esta experiencia sirvió para tener un adentramiento más controlado al mundo laboral y profesional, dando la oportunidad de conocer y aplicar metodologías y prácticas de trabajo que se utilizan en la actualidad por la industria de desarrollo de software, junto con el uso de herramientas y tecnologías de vanguardia para la optimización de procesos.

Referencias Bibliográficas

Aalbers, H. (25 de June de 2020). *¿Qué son los Websockets?* IBM: https://developer.ibm.com/es/articles/que-son-los-websockets/

Abaunza, J. C. (20 de Julio de 2020). *De una arquitectura monolítica a una basada en microservicios*. Juan Carlos Abaunza: https://juancarlosabaunza.com/arquitectura-microservicios/

Abaunza, J. C. (23 de Marzo de 2020). *DevOps y Transformación Digital*. Juan Carlos Abaunza: https://juancarlosabaunza.com/devops-y-transformacion-digital/

Ahamed Bhuiyan, F., Rahman, A., & Islam Shamim, M. (2020). XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices. *IEEE*, 59-64.

Alfonso, A. (23 de December de 2018). *adrianalonsodev.* Enfoques de Arquitectura Multitenant para Aplicaciones SaaS: https://adrianalonsodev.medium.com/enfoques-de-arquitectura-multitenant-para-aplicaciones-saas-cf210d6c2f10

Alonso, A. (23 de Diciembre de 2018). *Esquema compartido por las organizaciones*. Enfoques de Arquitectura Multitenant para Aplicaciones SaaS: https://adrianalonsodev.medium.com/enfoques-de-arquitectura-multitenant-para-aplicaciones-saas-cf210d6c2f10

Alonso, A. (23 de Diciembre de 2018). *Esquemas separados por cada organización*. Enfoques de Arquitectura Multitenant para Aplicaciones SaaS: https://adrianalonsodev.medium.com/enfoques-de-arquitectura-multitenant-para-aplicaciones-saas-cf210d6c2f10

Alonso, A. (23 de Diciembre de 2018). *Una base de datos para cada organización*. Enfoques de Arquitectura Multitenant para Aplicaciones SaaS: https://adrianalonsodev.medium.com/enfoques-de-arquitectura-multitenant-para-aplicaciones-saas-cf210d6c2f10

Amazon. (s.f.). *¿Qué es DevOps?* Aws - Amazon: https://aws.amazon.com/es/devops/what-is-devops/

Amazon Web Services, Inc. (2022). *Amazon API Gateway Guía para desarrolladores.* AWS.amazon: https://docs.aws.amazon.com/es_es/apigateway/latest/developerguide/apigateway-dg.pdf#apigateway-websocket-api-overview

Balaji, K., & Manoj, J. (07 de October de 2021). *Introducción a la arquitectura multitenant.* IBM Developer: https://developer.ibm.com/es/articles/introduccion-a-multitenancy/

Charles, A. (2015). Docker. *IEEE Software*, 102-105. https://dev.to/silicosis/que-es-docker-y-para-que-sirve-explicacion-5h2n

Cierco, D. (2011). *CLOUD COMPUTING: RETOS Y OPORTUNIDADES.* Fundacion Ideas.

Colaboradores de Wikipedia. (15 de Septiembre de 2022). *Un diagrama cliente-servidor vía Internet.* Wikipedia: https://es.wikipedia.org/wiki/Cliente-servidor

Eduardo, Z. (18 de December de 2020). *¿Qué es Docker y para que sirve?* Dev: https://dev.to/silicosis/que-es-docker-y-para-que-sirve-explicacion-5h2n

Giraldo Giraldo, A., & Gomez Ramirez, D. (2017). *ESTADO DEL ARTE DE LA SEGURIDAD EN SISTEMAS BIOMETRICOS.* UNIVERSIDAD ABIERTA Y A DISTANCIA -UNAD.

IBM. (s.f.). *Cloud computing: Guía completa*. ibm.com: https://www.ibm.com/es-es/cloud/learn/cloud-computing-gbl

INCIBE. (2016). *Tecnolog as biom tricas aplicadas a la ciberseguridad.* INCIBE_.

Ionos. (10 de Mayo de 2020). *Digital Guide Ionos*. Ionos: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/json-web-token-jwt/

Kinsta Inc. (7 de February de 2022). *¿Qué Es Nginx y Cómo Funciona?* Kinsta: https://kinsta.com/es/base-de-conocimiento/que-es-nginx/

Kubernetes. (11 de February de 2022). *¿Qué es Kubernetes?* kubernetes: https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/

Lombardi, A. (2015). *WebSocket: LIGHTWEIGHT CLIENT-SERVER COMMUNICATIONS.* Sebastopol: O'Reilly Media, Inc.

Maida, E., & Pacienzia, J. (December de 2015). *Metodologías de desarrollo de software*. Biblioteca Digital de la Universidad Católica Argentina: https://repositorio.uca.edu.ar/bitstream/123456789/522/1/metodologias-desarrollo-software.pdf

Microsoft. (21 de Septiembre de 2022). *Microsoft Learn*. Overview of ASP.NET Core SignalR: https://learn.microsoft.com/es-es/aspnet/core/signalr/introduction?view=aspnetcore-6.0

Microsoft. (s.f.). *Azure - Microsoft*. ¿Qué es DevOps?: https://azure.microsoft.com/es-es/overview/what-is-devops/#devops-overview

Microsoft Corporation. (2020). *.NET Microservices: Architecture for Containerized .NET Applications.* Microsoft Developer Division, .NET and Visual Studio product teams.

Microsoft Learn. (22 de Septiembre de 2022). *Implementación de puertas de enlace de API con Ocelot*. Microsoft Learn: https://learn.microsoft.com/es-es/dotnet/architecture/microservices/multi-container-microservice-net-applications/implement-api-gateways-with-ocelot

Montemagno. (24 de Junio de 2019). *Why SignalR (Core)*. Real Time Communication for Mobile with SignalR (Core): https://montemagno.com/real-time-communication-for-mobile-with-signalr/

Mozilla. (19 de February de 2022). *XMLHttpRequest* . developer.Mozilla: https://developer.mozilla.org/es/docs/Web/API/XMLHttpRequest

Oracle. (s.f.). *Documentación de Oracle Cloud Infrastructure*. Oracle: https://docs.oracle.com/es-ww/iaas/Content/APIGateway/Concepts/apigatewayconcepts.htm

Perez, A., & Evgeniev, M. (s.f.). *Tyk y Kong: analizamos estos dos API Gateways*. bbva: https://www.bbva.com/es/tyk-kong-analizamos-estos-dos-api-gateways/

RabbitMQ. (s.f.). *RabbitMQ tutorial - «Hello World!»*. RabbitMQ: https://www.rabbitmq.com/tutorials/tutorial-one-dotnet.html

Red Hat. (14 de March de 2018). *¿Qué es el cloud computing?* redhat.com: https://www.redhat.com/es/topics/cloud

Red Hat. (23 de Abril de 2020). *RedHat.* ¿Qué es la arquitectura multiempresa?: https://www.redhat.com/es/topics/cloud-computing/what-is-multitenancy

Rodas Vasquez, A., & Valencia Carrasquilla, A. (July-December de 2018). Desarrollo e implementación de un prototipo para una plataforma tecnológica para la transmisión de texto y video (streaming) en tiempo real empleando tecnología websocket. *USBMed*, págs.

2-10. Desarrollo e implementación de un prototipo para una plataforma tecnológica para la transmisión de texto y video (streaming) en tiempo real empleando tecnología websocket.

Rodrígez, M. (24 de Diciembre de 2020). *Scrum: el pasado y el futuro*. Netmind: https://netmind.net/es/scrum-el-pasado-y-el-futuro/

Schmitz, M. (25 de Mayo de 2020). *¿Qué es exactamente un WebSocket?* Supervisión de aplicaciones WebSocket: www.dotcom-monitor.com/blog/es/2020/05/25/websocket-application-monitoring/

Schwaber, K., & Sutherland, J. (2020). *La Guía de Scrum.* California: Creative Commons.

Unzue Pulido, R. (21 de Enero de 2019). *¿Qué es una Arquitectura Multi-tenant?* Blog Virtualizacion: https://www.maquinasvirtuales.eu/que-es-una-arquitectura-multi-tenant/

Uso de concentradores en para ASP.NET CoreSignalR. (2022 de Agosto de 2022). Microsoft Docs: https://docs.microsoft.com/es-es/aspnet/core/signalr/hubs?view=aspnetcore-6.0

Vennam, S. (18 de August de 2020). *¿Qué es cloud?* IBM: https://www.ibm.com/co-es/cloud/learn/cloud-computing

Villalba, A., & Carrera, D. (6 de July de 2020). *arxiv*. Multi-tenant Pub/Sub Processing for Real-time Data Streams: https://arxiv.org/pdf/2007.02802.pdf

Yin, X., & Chuangwei, Z. (2011). Design and implementation of single-service multi-function Webservice. *IEEE*, 3912-3915.