Implementación de funcionalidad de Mensajería en una Aplicación Web Progresiva utilizando ReactJS y la API de Twilio

Mario Andrés García Toro

Informe de semestre de industria para optar al título de Ingeniero de Sistemas

Modalidad de Semestre de Industria

Asesora
Deisy Loaiza Berrío, Ingeniera de Sistemas

Universidad de Antioquia
Facultad de Ingeniería
Ingeniería de Sistemas
Medellín, Colombia
2024

Cita	García, 2024
Referencia **Estilo APA 7 (2020)**	(García, 2024). *Implementación de funcionalidad de Mensajería en una Aplicación Web Progresiva utilizando ReactJS y la API de Twilio* [Semestre de Industria]. Universidad de Antioquia, Medellín

Agradecimientos

Agradezco a mi asesora Deisy Loaiza Berrío, Ingeniera de Sistemas y Coordinadora del programa de Prácticas Académicas en el departamento de Ingeniería de Sistemas; por todo su apoyo y paciencia en mi proceso de formación como ingeniero y especialmente en el desarrollo de este trabajo. También deseo expresar mi agradecimiento a la empresa EPAM, quienes me acogieron hace ya algunos años y me han visto crecer como ingeniero, aun sin haberme graduado entonces. Por último, quiero hacer constar mi gratitud con la Universidad de Antioquia por ser mi segundo hogar, el lugar y el espacio en donde he crecido como persona y como profesional durante estos años, el viaje ha sido maravilloso.

Tabla de contenido

Lista de tablas

Lista de gráficos

Siglas, acrónimos y abreviaturas

APA	American Psychological Association
PWA	Progressive Web Application
SMS	Short Message Service
MMS	Multimedia Messaging Service
API	Application Programming Interface
SDK	Software Development Kit
JWT	JSON Web Token
UX	User Experience
UI	User interface
MVP	Minimum Viable Product
Nonprod	Ambiente de no producción
DOM	Document Object Model
CRUD	Create, Read, Update and Delete

Resumen

Luego de un proceso de migración de móvil a web, la empresa cliente contaba con una recién creada Aplicación Web Progresiva (PWA) lista para crecer en funcionalidades y aumentar el alcance a sus usuarios. Esta PWA está construida utilizando ReactJS, Redux Toolkit para el manejo de estados, react-router para la navegación, react-testing-library para las pruebas unitarias y MaterialUI para el diseño de componentes.

El proyecto se centró en la implementación de una característica robusta de mensajería denominada "Messaging", que permitió a los usuarios de la PWA, conocidos como Stylists, comunicarse con sus clientes utilizando la plataforma Twilio y su API para el manejo de mensajes. El objetivo fue integrar esta funcionalidad de manera efectiva, cumpliendo con los requisitos especificados por el cliente, aplicando buenas prácticas de programación y asegurando la calidad del código.

Palabras clave: React, Redux, PWA, Twilio.

Abstract

After a migration process from mobile to web, the client company had a newly created Progressive Web Application (PWA) ready to expand its functionalities and increase its reach to users. This PWA was built using ReactJS, Redux Toolkit for state management, react-router for navigation, react-testing-library for unit testing, and MaterialUI for component design.

The project focused on the implementation of a robust messaging feature called "Messaging," which allowed PWA users, known as Stylists, to communicate with their clients using the Twilio platform and its API for message handling. The goal was to effectively integrate this functionality, meeting the client's specified requirements, applying good programming practices, and ensuring the quality of the code.

*Keywords***:** React, Redux, PWA, Twilio

Introducción

En un mundo cada vez más digitalizado, las empresas buscan constantemente maneras de mejorar sus productos y servicios para ofrecer una experiencia de usuario superior y mantenerse competitivas en el mercado. Recientemente, una empresa cliente ha migrado su aplicación móvil a una Aplicación Web Progresiva (PWA), buscando aprovechar las ventajas de esta tecnología para ofrecer una experiencia más robusta y accesible a sus usuarios. Esta PWA ha sido construida utilizando tecnologías modernas y populares, tales como ReactJS para la interfaz de usuario, Redux Toolkit para el manejo eficiente del estado de la aplicación, react-router para la gestión de la navegación, react-testing-library para asegurar la calidad del código a través de pruebas unitarias, y MaterialUI para el diseño de componentes visuales.

Es imperativo implementar nuevas funcionalidades en la PWA, utilizando una metodología ágil que permita la iteración continua y la adaptación a los cambios en los requisitos del cliente. Para ello, se llevaron a cabo reuniones periódicas con el cliente para discutir y refinar los requisitos, generando un backlog de tareas claras y priorizadas. Cada tarea fue tomada del backlog y desarrollada con enfoque tanto en el frontend como en el backend, siguiendo un proceso riguroso de pruebas unitarias e integradas para asegurar que cada nueva funcionalidad se integre de manera fluida y sin errores.

El desafío que enfrentó la empresa cliente para continuar con el crecimiento de su aplicación se reflejó en la implementación de una nueva y robusta funcionalidad de mensajería denominada "Messaging". Esta funcionalidad permite a los usuarios de la PWA, conocidos como Stylists, comunicarse de manera eficiente con sus clientes utilizando la plataforma Twilio y su API para el manejo de mensajes. La integración de esta funcionalidad fue crucial para mejorar la experiencia del usuario, incrementar la interacción dentro de la plataforma y proporcionar un valor añadido significativo tanto para la empresa como para los usuarios finales.

Este proyecto no solo buscó satisfacer las necesidades actuales del cliente mediante la incorporación de la funcionalidad de mensajería, sino también establecer una base sólida para el crecimiento futuro de la PWA. La implementación de buenas prácticas de programación y un

enfoque riguroso en la calidad del código fueron fundamentales para lograr estos objetivos y entregar un producto que cumpla con los más altos estándares de la industria.

1 Planteamiento del problema

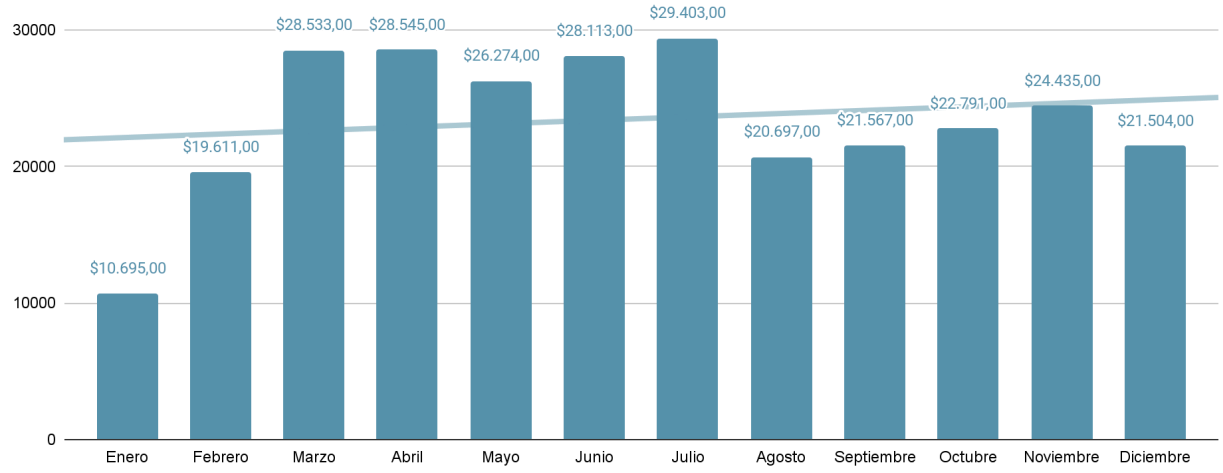
La plataforma actual surgió como una evolución de una aplicación móvil previa, con el objetivo de ampliar el alcance de usuarios y mejorar la capacidad de la herramienta para servir a su público objetivo. Dado que la transición ocurrió recientemente (hace menos de un año) la PWA fue lanzada únicamente con sus funcionalidades esenciales, las que fueron inicialmente migradas antes de que la aplicación móvil original fuera descontinuada. En consecuencia, para cumplir con el objetivo inicial de expansión y mejora continua, se decidió ampliar el equipo de desarrollo con la finalidad de agregar nuevas funcionalidades que incrementaran el valor de la PWA. Sin embargo, más que simplemente incrementar el equipo, se implementó una estrategia de formación de equipos de desarrollo especializados, cada uno centrado en un aspecto clave del negocio, pero colaborando de manera integrada a través de un repositorio compartido para el frontend.

El área del negocio abordada en este proyecto es "Clienteling", cuyo propósito es mejorar la capacidad de los stylists para generar relaciones positivas y duraderas con sus clientes. En este contexto, se busca que el proceso de comunicación entre stylists y clientes ocurra dentro de la misma aplicación, minimizando la dependencia de herramientas externas.

Antes de la implementación de esta nueva funcionalidad, la comunicación entre stylists y clientes se gestionaba a través de Twilio Frontline, una aplicación diseñada para facilitar la creación de relaciones personalizadas mediante el uso de canales como SMS, MMS, WhatsApp, Voz y Chat. Sin embargo, a los costos ya elevados asociados al uso de números telefónicos para los stylists, se le sumaban los cargos generados por el uso de Frontline que, con el pasar del tiempo, mostraban una tendencia a aumentar, tal y cómo se evidencia en el **Gráfico 1.**

Gráfico 1

Costos mensuales de Frontline para el año 2023



La sumatoria de todos los gastos mencionados previamente alcanzaron niveles que motivaron a la empresa a buscar una alternativa más rentable. Esta alternativa consistía en contener, gestionar y mantener todo el flujo de mensajería dentro de su propio entorno, bajo el control de la PWA. De esta manera se podría desacoplar el cobro de Frontline y generar un considerable ahorro en las cuentas totales del proyecto.

Así, surgió una necesidad crítica: investigar la viabilidad de esta nueva solución y, en caso de ser factible, priorizar junto al equipo de desarrollo la implementación de una funcionalidad que pueda no solo reemplazar los servicios ofrecidos por Twilio Frontline, sino también ofrecer mejoras o nuevas funcionalidades.

2 Justificación

El crecimiento de aplicaciones web progresivas (PWA) es una tendencia clave en el desarrollo moderno de software, ya que estas aplicaciones combinan las ventajas de las aplicaciones móviles y web, mejorando la accesibilidad y la experiencia del usuario. En el caso particular de este proyecto, la empresa cliente realizó una transición de una aplicación móvil a una PWA con el fin de aumentar el alcance de sus usuarios y aprovechar las capacidades que estas plataformas ofrecen.

Este proyecto contribuye al campo de la ingeniería de software al abordar un desafío real: la integración de APIs de terceros en una plataforma propietaria, optimizando tanto los recursos económicos como el control sobre los procesos. Además, refuerza la importancia de seguir buenas prácticas de desarrollo, pruebas automatizadas y metodologías ágiles, lo cual garantiza un desarrollo eficiente, escalable y de alta calidad.

La implementación de esta funcionalidad no solo representa un avance tecnológico para la empresa, sino que también contribuye a la ingeniería al ser un ejemplo de cómo las PWA pueden ser una solución efectiva y escalable para empresas que buscan mejorar la interacción con sus clientes a través de un entorno controlado y personalizado.

3 Objetivos

3.1 Objetivo general

Implementar una funcionalidad robusta de mensajería en una Aplicación Web Progresiva (PWA) construída con ReactJS y utilizando la API de Twilio, con el fin de mejorar la comunicación entre los usuarios (Stylists) y sus clientes.

3.2 Objetivos específicos

- Investigar los requisitos técnicos, económicos y administrativos necesarios para añadir Twilio al entorno de la PWA.
- Modificar los repositorios de backend (microservicios) para usar el API de Twilio/Conversations
- Desarrollar componentes de interfaz de usuario que cumplan con los diseños y comportamientos suministrados por los equipos de UX y Producto.
- Generar pruebas unitarias, de integración y de aceptación que verifiquen la correcta implementación y calidad del código.

4 Marco teórico

Las aplicaciones web progresivas (PWA, por sus siglas en inglés) son aplicaciones web construidas y mejoradas con APIs modernas para ofrecer capacidades avanzadas, mientras llegan a cualquier usuario web en cualquier dispositivo con una sola base de código. Combinan el amplio alcance de las aplicaciones web con las ricas capacidades de las aplicaciones específicas de plataforma para mejorar la experiencia del usuario (1).

Las PWA combinan lo mejor de las aplicaciones web y móviles. Pueden funcionar sin conexión, enviar notificaciones push, y aprovechar otras características nativas de los dispositivos, como se describe en el libro "Progressive Web Apps" de Dean Alan Hume (2).

React es una biblioteca de JavaScript para construir interfaces de usuario basada en componentes. Es mantenida por Meta (anteriormente Facebook). Facilita la creación de componentes reutilizables y eficientes, y puede ser usada para crear aplicaciones de una sola página (SPA) y aplicaciones web progresivas (PWA), apps móviles (con React Native) e incluso renderizar en el lado del servidor (usando Node) (3).

El código de React se compone de entidades llamadas componentes. Estos componentes son modulares y reusables, lo que hace sencillo construir capa tras capa, generando complejas interfaces de usuario capaces de cumplir con las necesidades dadas (4).

Una ventaja clave de React es que solo re-renderiza aquellos componentes de la página que hayan sufrido cambios, evitando re-renders innecesarios de elementos del DOM que no han sido modificados (4).

Redux es una librería Javascript para el manejo predecible y mantenible del estado global. Ayuda a escribir aplicaciones que se comporten de manera consistente, que corran en diferentes ambientes (cliente, servidor y nativo) y que sean fáciles de probar (5). Complementando lo anterior, provee una excelente experiencia para desarrolladores, como edición de código en vivo combinada con un depurador que viaja en el tiempo. Redux puede ser usada con react, o con

cualquier otra librería. Es liviana (2kB, incluyendo dependencias), pero tiene un enorme ecosistema de complementos disponibles (5).

Redux Toolkit es la biblioteca oficial de Redux para la configuración y gestión del estado global en aplicaciones React. Envuelve el núcleo de Redux y contiene paquetes y funciones esenciales para la generación de aplicaciones con Redux. Simplifica la creación de reducers y acciones, y mejora el rendimiento y la mantenibilidad del código . Redux Toolkit se integra fácilmente con React, proporcionando una estructura robusta para el manejo del estado de la aplicación (6).

Twilio es una plataforma de comunicación en la nube que permite integrar funciones de mensajería, voz y video en aplicaciones web y móviles. Ofrece APIs que facilitan la integración de capacidades de comunicación sin necesidad de infraestructura adicional. Twilio soporta aplicaciones como verificación de usuarios, recordatorios, alertas, y servicio al cliente.

La API de Conversations permite la creación de experiencias de mensajería interactivas, facilitando la gestión de conversaciones en múltiples canales. La integración de Twilio en una PWA requiere la configuración de servicios backend para manejar las credenciales y la lógica de mensajería, así como componentes frontend para la interacción del usuario.

Las buenas prácticas de programación son métodos y técnicas recomendadas para mejorar la calidad del código, la eficiencia del desarrollo y el mantenimiento de software. Algunas de estas prácticas incluyen:

- Escribir código claro y legible: Nombrar variables y funciones de manera descriptiva.
- Documentación: Comentar el código adecuadamente.
- Modularidad: Dividir el código en módulos o funciones pequeñas y reutilizables.
- Pruebas: Escribir pruebas unitarias y de integración.
- Control de versiones: Usar sistemas como Git para gestionar cambios en el código.
- Revisiones de código: Realizar revisiones y análisis de código entre pares (8).

Estas prácticas ayudan a crear software más robusto, mantenible y escalable.

Las pruebas unitarias son tests que verifican el funcionamiento de unidades individuales de código, como funciones o métodos, de manera aislada para asegurar que cada una realice correctamente su tarea. Por su parte, las pruebas de integración verifican que las diferentes unidades de código funcionen juntas correctamente. Se centran en la interacción entre módulos o componentes para asegurar que colaboren como se espera. Finalmente, las pruebas end-to-end (E2E) son tests que verifican el funcionamiento completo de una aplicación desde el principio hasta el final. Estas pruebas simulan escenarios del mundo real y garantizan que todas las partes del sistema funcionen correctamente juntas, incluyendo la interfaz de usuario, el backend y las bases de datos (8).

La metodología de desarrollo ágil se centra en la colaboración, flexibilidad y entrega rápida de software de alta calidad. Sus principios clave incluyen:

- Individuos e interacciones sobre procesos y herramientas.
- Software funcional sobre documentación extensiva.
- Colaboración con el cliente sobre negociación de contratos.
- Respuesta ante el cambio sobre seguir un plan.

El Manifiesto Ágil destaca la importancia de adaptarse a los cambios y entregar valor continuamente (7).

New Relic es una plataforma de observabilidad basada en la nube que ayuda a los desarrolladores, equipos de operaciones y negocios a monitorear y optimizar el rendimiento de sus aplicaciones y sistemas. Ofrece herramientas para rastrear métricas, eventos, logs y trazas, permitiendo a los equipos detectar y resolver problemas rápidamente. New Relic soporta una amplia variedad de lenguajes de programación y tecnologías, proporcionando una vista unificada del rendimiento de la infraestructura y las aplicaciones (9).

5 Metodología

Este proyecto se llevó a cabo empleando un enfoque cualitativo con principios de metodologías ágiles. Aunque la propuesta inicial seguía un enfoque en cascada, la ejecución real fue iterativa y flexible, lo que permitió adaptarse rápidamente a los cambios en los requerimientos del cliente.

Durante todo el ciclo del proyecto, como parte del standard de trabajo de la empresa cliente, se empleó el marco de trabajo Kanban. Sin embargo, con el fin de tener una comunicación constante y adecuada en el equipo, se optó por incluir algunas reuniones cíclicas, basadas en las ceremonias de Scrum. En total fueron 3 reuniones: Daily, Story review y Retrospective.

Las reuniones de revisión de historias (Story review) y de retrospectiva (Retrospective) se programaron con una periodicidad quincenal. La 'Revisión de Historias' se llevó a cabo en la primera y tercera semana de cada mes, ofreciendo un espacio para discutir el desarrollo y ajustes necesarios de las historias de usuario activas en nuestro tablero Kanban. Paralelamente, las 'Retrospectivas' tuvieron lugar en la segunda y cuarta semana de cada mes, proporcionando un momento crítico para que el equipo evalúe los procesos internos y busque mejoras continuas en nuestra metodología de trabajo. Lo anterior garantizó que cada semana se celebrara una de estas dos reuniones, facilitando una alternancia efectiva que promovió la preparación y la reflexión efectiva entre sesiones.

Además, se adoptó la práctica de realizar una reunión diaria, conocida como 'Daily', inspirada en el formato tradicional de SCRUM. En estas reuniones, los miembros del equipo compartían brevemente sus avances y retos diarios. Sin embargo, se decidió que los jueves serían días sin reuniones, promoviendo así períodos extendidos de concentración y productividad sin interrupciones. Las 'Dailies' se efectuaban, por tanto, de lunes a miércoles y se retomaban el viernes.

El desarrollo del proyecto se estructuró en varias fases clave, cada una diseñada para abordar diferentes aspectos y requerimientos del proceso de implementación. Durante cada fase, se emplearon diversas herramientas tecnológicas y metodológicas específicamente seleccionadas para optimizar los resultados y facilitar la colaboración y el monitoreo del progreso. A continuación, se describe en detalle cómo se dividió el proyecto en estas fases críticas y qué herramientas fueron fundamentales para alcanzar los objetivos planteados.

Reunión de Requisitos y Definición de Especificaciones:

Durante la fase inicial del proyecto, se llevaron a cabo reuniones periódicas con el cliente, así como entrevistas y sesiones de análisis. El objetivo era comprender a profundidad los requisitos detallados de la funcionalidad de mensajería que debía implementarse. Estos requisitos fueron documentados meticulosamente usando herramientas como Jira y Confluence y priorizados para generar un backlog de tareas, que luego fue gestionado mediante el framework Kanban. A lo largo del proyecto, este enfoque permitió una gestión flexible y continua de las tareas, sin depender de ciclos fijos de trabajo (como los sprints de Scrum), pero manteniendo un flujo constante de entregables. El backlog fue ajustado continuamente para priorizar los entregables más importantes. La colaboración constante con el cliente permitió asegurar que todas las especificaciones estuvieran alineadas con las expectativas y necesidades del negocio.

Implementación de la Lógica de Backend:

El desarrollo del backend se llevó a cabo utilizando Node.js y Express, integrando la API de Twilio para gestionar el envío y la recepción de mensajes. Se siguieron estrictamente las mejores prácticas de desarrollo para garantizar que la integración fuera segura y eficiente. Adicionalmente, se crearon endpoints para manejar las operaciones CRUD necesarias en el nuevo flujo de mensajería. Durante la implementación, se llevaron a cabo revisiones continuas de código para asegurar la calidad. Para asegurar la correcta funcionalidad de estos endpoints, se desarrollaron y ejecutaron pruebas unitarias utilizando herramientas como Jest y react-testing-library, logrando verificar la robustez del sistema backend. Además, se utilizaron herramientas adicionales como Postman para la verificación de estos recién creados endpoints. Al

ser gestionado en un flujo Kanban, este trabajo fue iterativo, permitiendo ajustes continuos y revisiones inmediatas a medida que se completaban las tareas.

Desarrollo de la Interfaz de Usuario:

El diseño y desarrollo de los componentes de interfaz de usuario se realizó utilizando ReactJS y MaterialUI, asegurando una experiencia visual intuitiva y consistente con el resto de la PWA. A través de la gestión del estado con Redux Toolkit, se garantizó que el flujo de mensajes funcionara sin interrupciones, actualizando el estado de la aplicación conforme los stylists enviaban y recibían mensajes. Para validar la correcta integración entre el frontend y el backend, se llevaron a cabo pruebas unitarias y de integración, lo que permitió confirmar que todos los componentes funcionaban en conjunto de manera adecuada.

Pruebas de Aceptación de Usuario (UAT):

Una vez desarrollada la funcionalidad de mensajería, se organizaron sesiones de pruebas de aceptación de usuario (UAT) en las que participaron miembros de los equipos de desarrollo, UX y producto. Estas pruebas tenían como fin validar que la implementación cumpliera con todos los requisitos establecidos. Durante estas sesiones se identificaron y corrigieron inconsistencias o errores que no cumplían con las expectativas, devolviendo las tareas a los equipos correspondientes para su corrección y posterior validación. Jira fue utilizado para gestionar los tickets derivados de las pruebas de aceptación y para rastrear las correcciones de errores.

Despliegue a Producción:

Una vez aprobadas las funcionalidades, estas fueron desplegadas en producción utilizando feature flags para controlar qué elementos de la nueva funcionalidad eran visibles para los usuarios finales. Se implementaron dashboards en New Relic para monitorizar el rendimiento y la adaptabilidad de la nueva funcionalidad de mensajería, así como para detectar posibles errores o

inconvenientes en producción. Esto permitió tomar decisiones rápidas y precisas para optimizar el sistema en tiempo real.

Documentación del Proceso y Resultados:

Finalmente, se documentó detalladamente todo el proceso de desarrollo, registrando decisiones clave, problemas encontrados y las soluciones implementadas. Esta documentación fue fundamental para la elaboración de un informe final que resumió los logros del proyecto y los resultados obtenidos, proporcionando una visión completa del impacto que tuvo la implementación de la funcionalidad de mensajería en la PWA.

6 Resultados y Análisis

A medida que avanzaba la implementación del proyecto, se presentó una cuestión crítica relacionada con las opciones de conexión a los servidores de Twilio. Twilio ofrece dos métodos principales para integrar sus servicios: en primer lugar, a través de una API que puede ser implementada desde un backend compatible con múltiples lenguajes de programación, tales como Node.js, Java, y Python. En segundo lugar, proporciona un SDK compatible con JavaScript, iOS y Android, diseñado para su uso en el entorno del cliente. Tras una exhaustiva recopilación y análisis de datos durante la fase de investigación (ver **Tabla 1**), se determinó que una implementación híbrida constituía la alternativa más adecuada para satisfacer las necesidades específicas del caso de uso. Esta decisión se basó en una evaluación detallada de las capacidades y limitaciones de cada enfoque, así como en las demandas del proyecto en términos de eficiencia y funcionalidad.

Tabla 1
Comparación entre Conversations API y Conversations SDK

Característica	Conversations API (Backend)	Conversations SDK (Cliente)
Control y Flexibilidad	Alto control y personalización del flujo de datos.	Flexibilidad en la interacción directa con el cliente.
Experiencia del usuario	Posible latencia debido al procesamiento del servidor.	Respuesta rápida y menor latencia en interacciones.
Carga en el servidor	Alta, maneja toda la lógica y el procesamiento de datos.	Menor. Parte de la carga se traslada al cliente.
Consistencia	Facilita una experiencia uniforme en múltiples plataformas.	Requiere más trabajo para mantener la consistencia.
Seguridad	Posibles puntos fuertes al centralizar el control de la seguridad.	Potencialmente más vulnerable a la manipulación.
Manejo offline	Dependiente de la conectividad para procesar operaciones.	Puede manejar operaciones offline básicas.
Desarrollo y Mantenimiento	Puede concentrar la complejidad en el backend.	Aumenta la complejidad en el cliente, afectando el rendimiento.

La implementación del SDK de Conversations en la PWA se realizó con el objetivo de facilitar un acceso eficiente y en tiempo real a la información relacionada con las interacciones del usuario. Este enfoque permitió alcanzar varias funcionalidades clave, que incluyen:

- Obtener la lista de conversaciones del usuario.
- Obtener el conjunto de mensajes de una conversación.
- Enviar mensajes (texto y multimedia).
- Recibir mensajes en tiempo real.

Esta integración estratégica fue diseñada para mejorar la interactividad y la capacidad de respuesta de la aplicación, alineándose así con los objetivos del proyecto de proporcionar una experiencia de usuario más fluida y efectiva.

Por otra parte, la implementación de la Conversations API se realizó en uno de los repositorios de backend gestionados por el equipo de desarrollo. Esta elección se fundamentó en la necesidad de mantener un control riguroso sobre los datos, así como en la capacidad de personalizar la información recibida y enviada, mediante servicios de tipo consumer, esenciales para el flujo integral de la funcionalidad. Dicho repositorio fue encargado de gestionar las siguientes funcionalidades:

- Inicialmente, un endpoint que proporciona el access token requerido para utilizar el SDK en el frontend de la PWA.
- Endpoint que se conecta al API para crear conversaciones (usado en la PWA)
- Endpoint para eliminar todas las conversaciones de un usuario (usado en scripts)
- Endpoint para actualizar el estado de una conversación.
- Endpoint para obtener todas las conversaciones de un participante, dado un móvil (usado en scripts)

6.1 Backend

El desarrollo de las tareas del backend se inició antes que las del frontend para garantizar un margen de tiempo adecuado y evitar bloqueos derivados de la dependencia entre ambas partes. Inicialmente, se creó un endpoint encargado de obtener y retornar el access token necesario para

la configuración del SDK en el cliente. Durante esta fase, se integró la API de Conversations (twilio-node) en el repositorio, permitiendo la creación de una instancia de Twilio mediante el uso de las variables de entorno preexistentes antes del desarrollo de esta funcionalidad. Posteriormente, usando la instancia de Twilio, se generó un JWT con el identificador único de cada usuario, que actuó como medio de autenticación en las operaciones relacionadas con las conversaciones de Twilio. Esta estrategia aseguró una base sólida para las integraciones futuras y una transición fluida al desarrollo del frontend.

Una vez desplegado este endpoint, y habiendo iniciado en paralelo el diseño del frontend, se prosiguió con la implementación de los otros endpoints planificados para esta fase del proyecto. Estos endpoints estaban destinados tanto para su uso en la PWA como en scripts para tareas específicas.

Uno de los desafíos surgió con el endpoint encargado de crear las conversaciones, principalmente debido a problemas de sincronización con Redux en el cliente. Para abordar estos problemas, fue necesario revisar y ajustar el endpoint para manejar situaciones en las que ya existía una conversación entre los participantes agregados. En lugar de retornar un error en estos casos, se optó por enviar un código de respuesta distinto, incluyendo el identificador de la conversación en el payload. Esta solución mejoró la gestión de excepciones y la comunicación eficiente entre el backend y el frontend.

Finalmente, se procedió con la creación del endpoint responsable de obtener las conversaciones de un participante. Este endpoint se diseñó para ser utilizado en la página de perfil del cliente, facilitando el identificador de la conversación y permitiendo generar un enlace directo a esta. Los demás endpoints planificados para este repositorio se desarrollaron sin contratiempos significativos y se integraron en los scripts que se crearon posteriormente para tareas específicas.

6.2 Frontend

Las tareas relacionadas con el desarrollo del frontend requirieron más tiempo que las de backend, principalmente debido al desafío de aprender a utilizar el SDK de Twilio Conversations y su integración en el estado de la aplicación.

El primer componente diseñado y desarrollado fue el que se encargaría de alojar y mostrar todas las conversaciones del vendedor, ordenadas por fecha de último mensaje y proporcionando una previsualización de este. Junto a este componente, se creó un slice de Redux para gestionar la información de las conversaciones dentro del estado global de la aplicación, permitiendo así compartir datos entre diferentes componentes sin la necesidad de pasar props manualmente a través de cada nivel. Esta estrategia facilitó una gestión más eficiente y coherente del estado en toda la aplicación.

Existen múltiples opciones para manejar el estado de una aplicación, como Redux, React Context y Mobx, cada una con características particulares adaptadas a diferentes casos de uso. Desde sus inicios, la PWA ha utilizado Redux como su gestor de estado. Tal decisión se ha demostrado acertada, especialmente considerando el tamaño y la complejidad de las operaciones involucradas. Las características de Redux, en su mayoría positivas (ver **Tabla 2**), han respaldado eficazmente las necesidades del proyecto, proporcionando una estructura robusta y consistente para el manejo del estado a lo largo de su desarrollo.

Tabla 2
Comparación entre Redux y React Context (10) y (11)

Característica	**Redux**	**React Context**
Estructura	Proporciona una arquitectura más definida y estructura clara para la gestión del estado global.	Es más simple, sin una estructura formal para organizar el estado.
Complejidad	Más complejo de configurar e implementar, especialmente en aplicaciones pequeñas.	Fácil de usar en aplicaciones pequeñas y medianas.

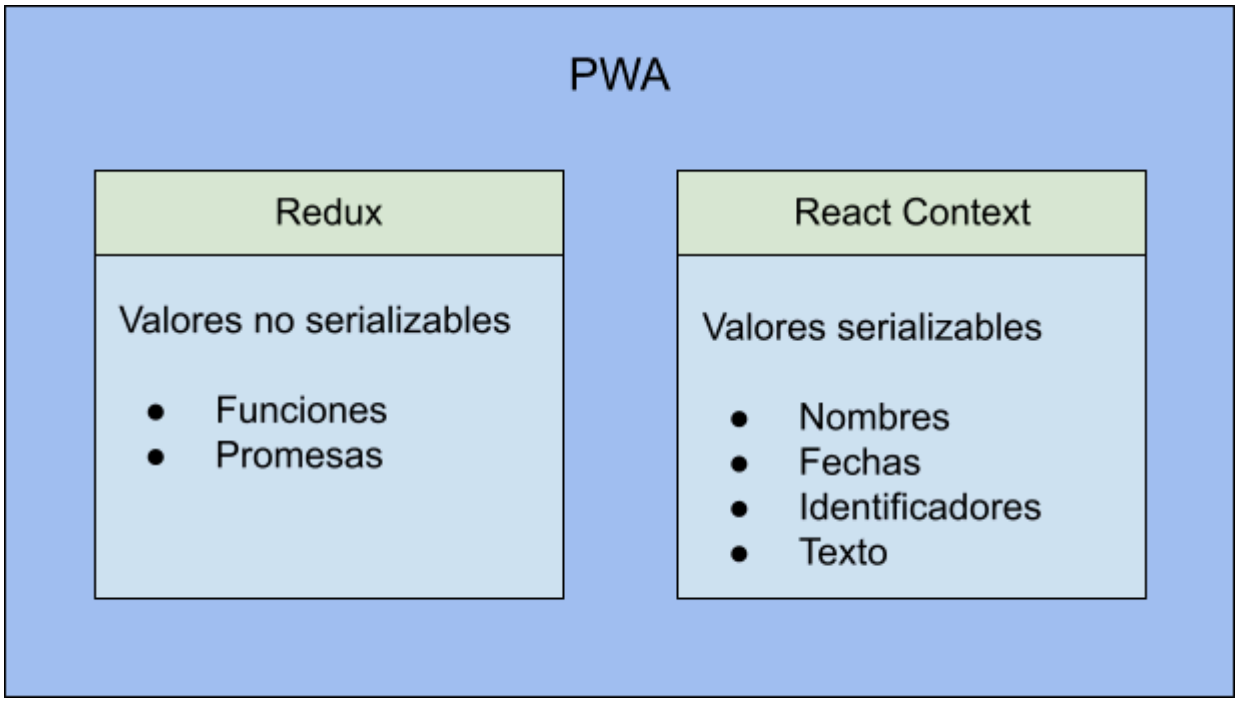
Escalabilidad	Ideal para aplicaciones grandes y complejas que requieren un manejo extenso del estado global.	Maneja bien aplicaciones más pequeñas a medianas, pero puede no ser adecuado para casos de uso muy complejos.
Rendimiento	Optimizado para cambios frecuentes y gran cantidad de datos.	Puede enfrentar problemas de rendimiento con estados muy grandes o actualizaciones profundas.
DevTools	Soporte para depuración y monitoreo con herramientas como Redux DevTools.	No tiene soporte nativo para DevTools. Depuración limitada a logs manuales y React DevTools.
Curva de aprendizaje	Requiere más tiempo y conocimientos previos, especialmente si se usa junto con middlewares y herramientas adicionales.	Curva de aprendizaje más corta. Se basa en la API de Context nativa de React.
Ecosistema y comunidad	Amplio ecosistema y comunidad de soporte, con muchas librerías y herramientas complementarias disponibles.	Menor ecosistema comparado con Redux, ya que se centra solo en la API de React.
Adecuación para manejo de caché o datos	Más adecuado para manejar caché, datos persistentes, y sincronización con fuentes de datos externas como APIs.	Menos eficiente para manejo de datos complejos o sincronización de caché entre múltiples componentes.

No obstante, la implementación de esta funcionalidad generó una necesidad específica que Redux no puede satisfacer: almacenar contenido no serializable (valores que no pueden ser convertidos a un formato, como JSON, que permita ser almacenado o transmitido y luego reconstruido en su forma original). Redux impone la restricción de no incluir valores no serializables, como promesas, funciones, Maps/Sets e instancias de clases, con el fin de asegurar que las capacidades de depuración, como las ofrecidas por Redux DevTools, funcionen correctamente. Dado que el SDK de Twilio devuelve numerosas funciones y promesas que, por motivos de rendimiento y ejecución, no deben ser ejecutadas al iniciar la aplicación—justo cuando se solicita la lista de conversaciones—se presentó un desafío.

Se evaluaron diversas alternativas y, finalmente, se optó por implementar un enfoque híbrido entre Redux y React Context debido a su flexibilidad en este caso específico. React Context se encargó de almacenar las referencias a los objetos no serializables, como funciones y promesas, mientras que Redux continuó gestionando la información destinada a ser renderizada en los componentes visuales, como se ilustra en el **Gráfico 2**. Esta solución permitió conservar la estructura y eficiencia del manejo del estado al mismo tiempo que acomodaba las particularidades del SDK.

Gráfico 2

Responsabilidades de Redux y React Context en la PWA



Con el desarrollo del componente encargado de las conversaciones (CustomerConversations) avanzado, se procedió a implementar otro componente crucial, destinado a mostrar todos los mensajes existentes en una conversación específica entre el vendedor y cada comprador (CustomerMessages). Este nuevo componente también incluye la capacidad de crear y enviar nuevos mensajes, tanto SMS como MMS.

Ambos componentes fueron diseñados con un enfoque modular, distribuyendo las responsabilidades de visualización entre los subcomponentes, pero manteniendo la mayor parte de la lógica dentro del componente principal o en archivos separados para funciones utilitarias. Esta estrategia permitió evitar la concentración excesiva de código en pocos archivos, facilitando así su lectura, comprensión y mantenimiento.

Para optimizar el rendimiento, se implementaron selectores con Redux, de modo que los componentes solo se volvieran a renderizar cuando la propiedad específica seleccionada del estado global cambie, en lugar de hacerlo cada vez que cualquier propiedad del estado mutase. Este enfoque garantizó una eficiente gestión del estado y mejoró el rendimiento de la aplicación.

6.3 Pruebas

Respecto a la PWA, cada subcomponente fue desarrollado junto con su respectiva suite de pruebas unitarias, al igual que los archivos utilitarios. Para los componentes principales, también se añadieron pruebas unitarias, aunque estas resultaron ser más robustas y complejas, debido a la necesidad de preparar extensivamente los casos de prueba, dado el considerable número de funciones, objetos y valores "mockeados" requeridos para cada escenario.

Para mejorar la eficiencia y la consistencia en las pruebas, se implementó un hook con el objetivo de estandarizar adecuadamente los "wrappers" utilizados en los componentes a probar. Estos incluían elementos como Redux, Context, FeatureFlags y MemoryRouter. Esta estandarización aseguró que las pruebas unitarias fueran más robustas y mantenibles, facilitando así el proceso de verificación y validación del funcionamiento de la aplicación.

En cuanto al repositorio de backend, el proceso siguió una dinámica similar. Considerando que cada ruta (endpoint) disponía de un handler (manejador) específico, se desarrolló una suite de pruebas para cada uno, además de otra para el archivo utilitario de servicios. Este enfoque se adoptó porque era el estándar empleado en los demás repositorios de backend desde antes de la implementación de la funcionalidad de mensajería. Al mantener la

consistencia con las prácticas existentes, se aseguró una integración más fluida y coherente de las nuevas funcionalidades, al tiempo que se facilitó el mantenimiento y la evaluación del código.

Aunque la cobertura de código no es un foco principal en las dinámicas del proyecto, la implementación de buenas prácticas en las pruebas unitarias ha resultado en un porcentaje de cobertura bastante elevado, sin ser un aspecto revisado de manera constante por el equipo. Como se observa en la **Tabla 3**, los valores de cobertura son generalmente altos, con pocos casos que descienden al 75%. En el backend, aunque la cobertura no es obligatoria, se le concede suficiente importancia para mantenerla presente. En contraste, en la aplicación web, este aspecto pasa inadvertido, ya que ni siquiera se ha configurado para realizar seguimiento. No obstante, esto no implica que las pruebas unitarias de la PWA sean inadecuadas o ineficaces. Al contrario, se les otorga el suficiente valor para asegurar que no se integren cambios significativos sin la correspondiente cobertura de pruebas, garantizando así la calidad y estabilidad del sistema.

Tabla 3
Cobertura de código para el nuevo controlador en backend

	statements	branches	functions	lines
handler1	96,96	75,86	100	86,87
handler2	100	100	100	100
handler3	85,71	100	100	83,33
handler4	100	84,21	100	100
handler5	92,3	75	100	91,66
serviceHandler	97,56	81,81	100	97,36

Cada solicitud de fusión (merge request) que se generó tuvo que superar un riguroso proceso de revisión por parte del equipo. Este proceso verificó no sólo el cumplimiento de los criterios de aceptación de cada tarea, sino también que el código fuese de calidad y contara con pruebas unitarias relevantes. Dado que el creador de la solicitud no puede aprobar su propio trabajo, y que se requiere la aprobación de al menos dos desarrolladores, se garantizó que el código integrado cumpliera con parámetros de calidad estandarizados por el equipo. Es importante destacar que cada solicitud de fusión es probada por al menos un miembro distinto al

creador, ya sea en un entorno local o desplegando la rama (feature branch) en un entorno no productivo (nonprod).

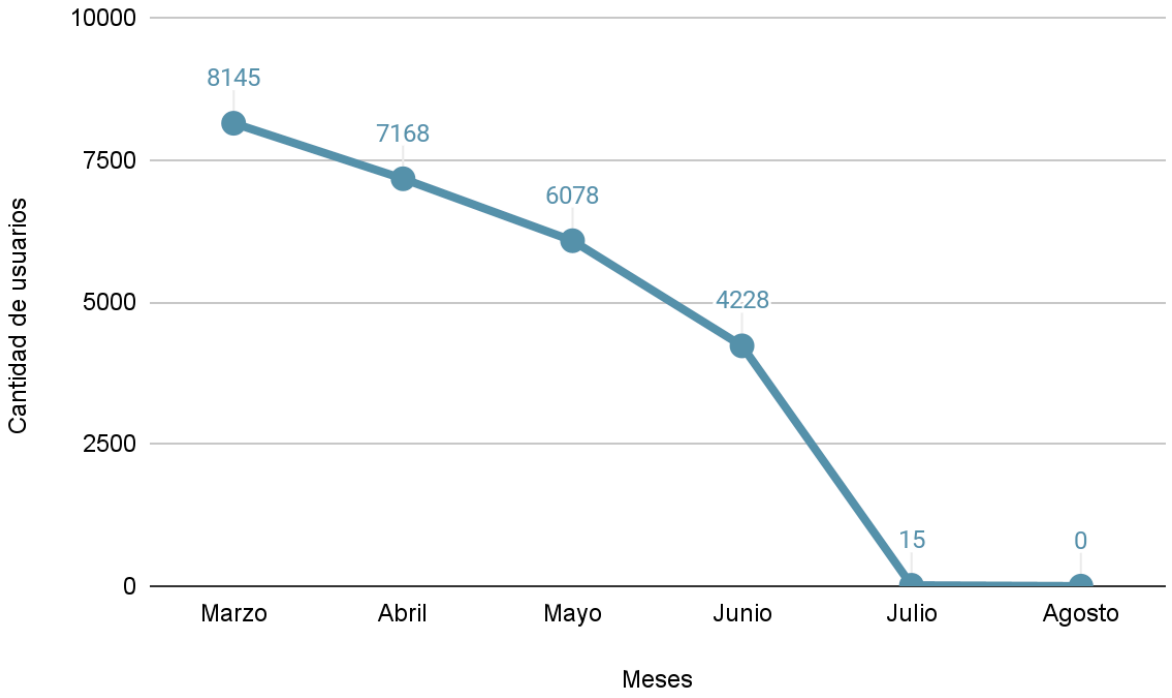
Una vez que el código fue considerado estable tanto en el backend como en el frontend, se organizó una sesión de UAT con la participación de todo el equipo, incluidos desarrolladores, personal de Producto y de UX. El objetivo era verificar que los criterios de aceptación se hubieran implementado correctamente y que la interfaz de usuario cumpliera con los lineamientos establecidos en Figma. De esta sesión surgieron algunas tareas adicionales, centradas principalmente en corregir aspectos de diseño en la UI. Estas tareas no afectaron las fechas de entrega, ya que pudieron resolverse de manera ágil.

6.4 Lanzamiento

Antes del lanzamiento, la empresa cliente había invertido considerables esfuerzos en campañas de capacitación para sus vendedores, preparando la transición de Frontline hacia una solución propia. Esta nueva alternativa no solo cubriría las características requeridas hasta el momento, sino que también se expandiría gradualmente, incorporando nuevas funcionalidades para mejorar las interacciones con los clientes. Como resultado, el número de usuarios activos en Frontline disminuyó progresivamente, hasta llegar casi a cero el día del lanzamiento, tal como se muestra en el **Gráfico 3**. Los 15 usuarios restantes eran casos específicos que no deberían haber existido, pero esto se solucionó posteriormente.

Gráfico 3

Usuarios activos de Frontline en los últimos 6 meses



El lanzamiento del MVP fue un proceso relativamente simple, dado que todo el código ya estaba desplegado en producción, pero oculto bajo un feature flag. Tras la aprobación para la publicación de la funcionalidad, solo se requirió un pequeño ajuste en el manejador de feature flags para que la funcionalidad de mensajería (Messaging) se volviera visible para los usuarios finales.

7 Conclusiones

Considerando que la empresa cliente ya mantenía una relación comercial establecida con Twilio, tanto por el uso de Frontline como por la gestión de números de celular para sus empleados, y que disponía de un equipo de desarrolladores dedicado a la PWA y los microservicios de backend, no se produjeron incrementos económicos significativos asociados a la implementación de esta funcionalidad. A partir de los costos asociados a Frontline, ilustrados en el **Gráfico 1**, y las proyecciones financieras para el año fiscal 2024, se anticipa que el lanzamiento de la funcionalidad de mensajería (Messaging) generará una reducción de gastos superior a los USD 300,000. Esta disminución se atribuye al reemplazo de Frontline como medio de contacto entre vendedores y clientes, lo que demuestra un impacto económico positivo significativo para la organización.

La nueva funcionalidad se integra ahora dentro de un amplio conjunto de características disponibles tanto en la PWA como en otra plataforma web, también propiedad de la empresa, orientada al cliente. A ello se suman los repositorios de backend previamente establecidos, lo cual sugiere que la gestión y el mantenimiento de la funcionalidad de mensajería no deberían representar un desafío significativo para el equipo. Esta situación contrasta favorablemente con los beneficios que dicha funcionalidad aporta, destacando el valor añadido para la organización y sus usuarios.

Gracias al enfoque estratégico del equipo en desarrollar y potenciar la interacción y experiencia de usuario en la aplicación, junto con las campañas de capacitación dirigidas a los vendedores, la nueva funcionalidad fue bien recibida por los usuarios finales. Según las métricas, se reportan hasta 3.7k usuarios únicos por semana, lo que representa un incremento del 34.78% en el porcentaje de adopción en comparación con la herramienta anterior, como se detalla en la **Tabla 4**. Este aumento refleja el éxito de la implementación y la aceptación positiva por parte de los usuarios.

Tabla 4
Resultados visibles pre y post lanzamiento de Messaging

Indicadores	**Pre-lanzamiento**	**Post-lanzamiento**	**Resultado**
Usuarios únicos semanales	2.7k	3.7k	+34.78%
Mensajes enviados diarios	5.0k	8.5k	+70%
Mensajes recibidos diarios	2.3k	3.1k	+34.78%
Tasa de compradores que se dan de baja a los mensajes	1.11%	0.88%	-20.72%

La adición de esta nueva funcionalidad también ha potenciado significativamente las oportunidades de alcance, reflejadas en los mensajes enviados y recibidos. En el contexto de la relación entre stylist y cliente, cada mensaje representa una oportunidad de venta de uno o varios ítems, conocidos colectivamente como "look". Por lo tanto, cuanto más fluidas y continuas sean las interacciones entre el vendedor y sus clientes, mayores serán las posibilidades de venta, beneficiando tanto al vendedor como a la empresa. Tal como se observa en la **Tabla 4**, se registró un incremento aproximado del 35% en la cantidad de mensajes diarios recibidos y un notable aumento del 70% en el total de mensajes enviados. Esto sugiere que la funcionalidad no solo ha sido bien recibida y utilizada de manera efectiva, sino que también ha beneficiado enormemente las relaciones comerciales de sus usuarios, fortaleciendo así la dinámica de negocio.

Es importante destacar que las métricas revelaron una disminución en la tasa de compradores que optan por darse de baja. Este evento ocurre cuando un cliente solicita detener la comunicación con el vendedor hasta que el primero decida reanudarla. En tales casos, el vendedor pierde la oportunidad de interactuar con el cliente, afectando potenciales ventas.

La reducción de este índice en casi un 21% indica una mejora significativa en las interacciones entre vendedores y compradores. Esto sugiere que las comunicaciones se perciben como más valiosas y efectivas, fortaleciendo las relaciones comerciales y maximizando las oportunidades de venta.

En el aspecto técnico, el repositorio de backend se modificó con éxito para integrar la API de Twilio/Conversations. Esto ha permitido un intercambio de datos eficiente y seguro, fundamental para ampliar las capacidades de mensajería. Esta integración no solo mejora la fluidez de la comunicación, sino que prepara el sistema para futuras expansiones y funcionalidades adicionales.

Los componentes de la interfaz de usuario no solo se ajustaron a los diseños y comportamientos establecidos por los equipos de UX y Producto, sino que también mejoraron notablemente la experiencia del usuario, facilitando una interacción fluida y atractiva. Estos aspectos se reflejaron en la destacada adopción registrada, evidenciando el éxito de la implementación y la aceptación positiva por parte de los usuarios.

La implementación de pruebas unitarias, de integración y de aceptación fue crucial para verificar la correcta implementación y calidad del código, permitiendo una gestión oportuna de casos de falla y escenarios límite. Este enfoque garantizó la incorporación de la nueva funcionalidad sin comprometer la estabilidad ni el rendimiento de la PWA. Además, el uso de feature flags facilitó el despliegue continuo y controlado del código en producción, evitando retrasos en las fusiones a la rama principal y asegurando que los usuarios finales no se vieran afectados durante el proceso.

Finalmente, participar en este proyecto demostró ser una experiencia excepcionalmente enriquecedora, tanto a nivel profesional como personal. Implementar una funcionalidad tan crítica permitió profundizar en el manejo de tecnologías avanzadas, como integrar APIs y utilizar eficientemente frameworks de desarrollo. Además, observar de primera mano el impacto positivo de una colaboración eficaz entre equipos de distintas disciplinas resalta la importancia del enfoque interdisciplinario en el desarrollo de software moderno. Ver el resultado reflejado en el aumento sustancial de la adopción y la mejora en la relación entre usuarios es testimonio del valor del esfuerzo conjunto. Este proyecto reforzó la convicción en la capacidad de la innovación tecnológica para transformar dinámicas empresariales y fortalecer conexiones humanas.

Referencias

Abramov, D., Clark, A. (2015). Redux - *A Predictable State Container for JS Apps.* https://redux.js.org/ (5)

Abramov, D. (2018). You Might Not Need Redux. https://bit.ly/3U5PWHK (10)

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... & Thomas, D. (2001). *Manifesto for Agile Software Development.* https://agilemanifesto.org/ (7)

Facebook. (s.f.). *React. GitHub.* https://github.com/facebook/react (3)

Facebook Open Source. *React – A JavaScript library for building user interfaces.* https://reactjs.org/ (3)

Hume, D. A. (2017). *Progressive Web Apps. Apress.* (2)

LePage, P., & Richard, S. (2020, January 6). *What are Progressive Web Apps?* https://web.dev/articles/what-are-pwas (1)

Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.* (8)

New Relic. (s.f.). *About. Tomado de* https://newrelic.com/ (9)

React. (s.f.). *Passing data deeply with context.* https://react.dev/learn/passing-data-deeply-with-context (11)

Redux Toolkit. (s.f.). *Introduction. Redux.* https://redux-toolkit.js.org/introduction/getting-started (6)

Wieruch, Robin (2020). *The Road to React. Leanpub.* ISBN 978-1720043997. (4)