



Ciencia Latina Revista Científica Multidisciplinar, Ciudad de México, México.
ISSN 2707-2207 / ISSN 2707-2215 (en línea), septiembre-octubre 2026,
Volumen 10, Número 5.

https://doi.org/10.37811/cl_rcm.v10i5

MONITOREO MÓVIL EN TIEMPO REAL DE PARÁMETROS FISICOQUÍMICOS DEL AGUA EN ESTANQUES DE ALEVINES DE TILAPIA

**REAL-TIME MOBILE MONITORING OF PHYSICOCHEMICAL
WATER PARAMETERS IN TILAPIA FRY PONDS**

Ezequiel Gómez Domínguez

Instituto Tecnológico de Mérida, México

Jorge Cein Villanueva Guzman

Instituto Tecnológico de Villahermosa, México

Julio Cesar Romellon Cerino

Instituto Tecnológico de Villahermosa, México

Víctor Manuel Arias Peregrino

Instituto Tecnológico de Villahermosa, México

Andy de Jesús González Alcázar

Instituto Tecnológico de Villahermosa, México

Monitoreo Móvil en Tiempo Real de Parámetros Físicoquímicos del Agua en Estanques de Alevines de Tilapia

Ezequiel Gómez Domínguez¹

ezequiel.gd@merida.tecnm.mx

<https://orcid.org/0009-0008-3996-951X>

Tecnológico Nacional de México

Instituto Tecnológico de Mérida

Mérida, Yucatán – México

Jorge Cein Villanueva Guzman²

jorge.vg@villahermosa.tecnm.mx

<https://orcid.org/0000-0003-1307-0801>

Tecnológico Nacional de México

Instituto Tecnológico de Villahermosa

Villahermosa, Tabasco – México

Julio Cesar Romellon Cerino

julio.rc@villahermosa.tecnm.mx

<https://orcid.org/0000-0003-2388-3128>

Tecnológico Nacional de México

Instituto Tecnológico de Villahermosa

Villahermosa, Tabasco – México

Víctor Manuel Arias Peregrino

victor.ap@villahermosa.tecnm.mx

<https://orcid.org/0000-0001-5925-4097>

Tecnológico Nacional de México

Instituto Tecnológico de Villahermosa

Villahermosa, Tabasco – México

Andy de Jesús González Alcázar

L21300636@villahermosa.tecnm.mx

<https://orcid.org/0009-0003-3024-150X>

Tecnológico Nacional de México

Instituto Tecnológico de Villahermosa

Villahermosa, Tabasco – México

RESUMEN

La medición manual de los parámetros físicoquímicos del agua en estanques de alevines de tilapia genera registros inconsistentes y retrasa la respuesta ante valores críticos. Este trabajo desarrolló una aplicación móvil con datos en tiempo real almacenados en la nube para monitorear y gestionar esos parámetros. En cuanto a la metodología, el desarrollo siguió siete etapas: análisis de requerimientos, diseño en Figma, programación con Flutter y Dart, integración con Cloud Firestore mediante el SDK de Firebase, integración de un sistema IoT externo, módulo de alertas y pruebas. Un servicio en Node.js, alojado en Render, envió notificaciones con Firebase Cloud Messaging. Como resultado se obtuvo una aplicación con cinco secciones y un panel de cuatro variables validadas con sensor: temperatura, pH, oxígeno disuelto y sólidos disueltos. La turbidez se implementó en el código, pero no se probó con un sensor físico. El servicio de alertas notificó con prioridad alta ante valores fuera de rango, incluida una estimación de amonio. Las pruebas simuladas confirmaron las alertas y la actualización automática del panel. Se concluyó que integrar Flutter, Firebase y un sistema IoT es viable para consultar la calidad del agua a distancia; falta validarlo en operación prolongada y probar el sensor de turbidez.

Palabras clave: acuicultura; aplicaciones móviles; calidad del agua; firebase; internet de las cosas

¹ Autor principal.

² Correspondencia: jorge.vg@villahermosa.tecnm.mx

Real-Time Mobile Monitoring of Physicochemical Water Parameters in Tilapia Fry Ponds

ABSTRACT

Manual measurement of physicochemical water parameters in tilapia fry ponds generates inconsistent records and delays the response to critical values. This study developed a mobile application with real-time data stored in the cloud to monitor and manage these parameters. Regarding the methodology, the development process followed seven stages: requirements analysis, design in Figma, programming using Flutter and Dart, integration with Cloud Firestore through the Firebase SDK, integration of an external IoT system, implementation of an alert module, and testing. A Node.js service, hosted on Render, sent notifications through Firebase Cloud Messaging. As a result, a mobile application with five sections and a dashboard displaying four sensor-validated variables was developed: temperature, pH, dissolved oxygen, and total dissolved solids (TDS). Turbidity was implemented in the code but was not tested using a physical sensor. The alert service issued high-priority notifications when values fell outside the established ranges, including an estimated ammonium value. Simulated tests confirmed the proper operation of the alerts and the automatic updating of the dashboard. It was concluded that integrating Flutter, Firebase, and an IoT system is a viable approach for remotely monitoring water quality; however, further validation under prolonged operating conditions and physical testing of the turbidity sensor are still required.

Keywords: aquaculture; firebase; internet of things; mobile applications; water quality monitoring

Artículo recibido 20 agosto 2026

Aceptado para publicación: 20 setiembre 2026



INTRODUCCIÓN

La acuicultura es una actividad productiva de relevancia económica y social. En México, la tilapia del Nilo (*Oreochromis niloticus*) es uno de los peces más cultivados por su rápido crecimiento, su resistencia a enfermedades, su facilidad de reproducción y su tolerancia a distintas condiciones ambientales, lo que permite cultivarla en estanques, jaulas y sistemas de recirculación (Sustentable, 2018). Sin embargo, el desarrollo y la supervivencia de los alevines dependen de mantener una buena calidad del agua, porque los parámetros fisicoquímicos influyen directamente en su metabolismo, su crecimiento y su salud (Sustentable, 2018).

Entre estos parámetros destacan la temperatura, el oxígeno disuelto, el pH, el amonio y los nitritos. La temperatura regula el metabolismo, la reproducción y el crecimiento. El intervalo óptimo para la especie es de 28 °C a 32 °C; por debajo de 20 °C se inhibe la reproducción, alrededor de 15 °C los peces dejan de alimentarse y entre 10 °C y 11 °C la temperatura puede ser letal. Para los alevines se recomienda mantener el agua por encima de 25 °C y evitar variaciones bruscas mayores a 5 °C (INTAGRI S.C., 2020). El oxígeno disuelto debe mantenerse entre 5 y 9 mg/L; valores menores a 3 mg/L provocan estrés severo e hipoxia y aumentan la susceptibilidad a enfermedades (El productor, 2017). El pH óptimo se ubica entre 6.5 y 8.5, y los alevines son sensibles a fluctuaciones bruscas, por lo que se recomienda evitar variaciones mayores a 0.5 unidades en un mismo día. Además, un pH elevado favorece la forma más tóxica del amonio (NH_3) (Despesca, 2025). Se recomienda mantener el N-NH_3 por debajo de 0.1 mg/L, porque concentraciones mayores afectan la capacidad de la sangre para transportar oxígeno (SAGARPA, 2008). Los nitritos deben mantenerse por debajo de 0.1 a 0.5 mg/L, ya que se combinan con la hemoglobina, forman metahemoglobina y limitan el transporte de oxígeno; los alevines son especialmente vulnerables a esta toxicidad (Fedeacua, 2018).

Además de mantener estos parámetros en rangos adecuados, se recomienda medir la temperatura, el oxígeno disuelto y el pH al menos dos veces al día durante el alevinaje, de preferencia al amanecer y al atardecer, y evaluar el amonio y los nitritos cada dos días, o diariamente en sistemas intensivos. Llevar una bitácora de las mediciones permite identificar tendencias y tomar decisiones oportunas, como ajustar la alimentación, incrementar la aireación o realizar recambios de agua (SAGARPA, 2008).



En la práctica, estos parámetros suelen medirse de forma manual con equipos de medición. Este procedimiento depende de la intervención humana, puede generar inconsistencias y errores en el registro, y entrega información que no siempre está actualizada, lo que retrasa la toma de decisiones relacionadas con el bienestar de los alevines. Además, cuando no existe una plataforma centralizada para almacenar y visualizar los datos, los acuicultores no consultan en tiempo real el estado de los estanques, y la ausencia de un sistema de alertas impide reaccionar con rapidez ante valores fuera de los rangos recomendados (Villanueva et al., 2025). Como consecuencia, la toma de decisiones es reactiva y no proactiva, lo que puede generar pérdidas en la producción.

La integración de sistemas IoT con aplicaciones móviles para el monitoreo de la calidad del agua en acuicultura ha sido documentada en estudios recientes. Sung et al. (2023) desarrollaron un sistema IoT para estanques de bagre que envía las lecturas de temperatura, pH y turbidez a Firebase y las muestra en una aplicación Android, con un error promedio de medición de 4.49%. Kanwal et al. (2024) propusieron un sistema de toma de decisiones basado en IoT para el monitoreo en tiempo real de estanques multiespecie, que incorpora analítica predictiva para anticipar condiciones críticas del agua. Una aplicación móvil es un software diseñado para operar en dispositivos como teléfonos inteligentes y tabletas y ejecutar funciones específicas. A diferencia de un sitio web, aprovecha características del dispositivo como la cámara, el sistema de geolocalización (GPS) y la interfaz táctil (lab9 Digital Agency, 2024). Según su forma de desarrollo y su nivel de integración con el dispositivo, se clasifican en nativas, web, híbridas y web progresivas. Las aplicaciones nativas se desarrollan para un sistema operativo específico mediante su kit de desarrollo de software (SDK); por ello, para ofrecer una aplicación en sistemas como Android, iOS o Windows Phone se requiere una versión independiente para cada uno (Oliver, 2022). Las aplicaciones web son versiones optimizadas de sitios web para dispositivos móviles: se ejecutan en el navegador mediante una URL, sin importar el sistema operativo, y se basan en HTML, CSS y JavaScript (ISIL, 2023). Las aplicaciones híbridas combinan características de las nativas y la web: se desarrollan con tecnologías web, pero se encapsulan en un contenedor nativo que permite instalarlas, distribuirlas en tiendas de aplicaciones y acceder a componentes como la cámara o la geolocalización (Agency, 2017).



Las aplicaciones web progresivas son una evolución de las webs tradicionales y se comportan de forma similar a las nativas, adaptándose progresivamente a las capacidades del dispositivo (CEUPE, 2019).

Un framework es una estructura de soporte para el desarrollo de software que ofrece herramientas, bibliotecas y convenciones para estandarizar el proceso y evitar que el desarrollador inicie cada proyecto desde cero (Lenovo, 2025b). En el desarrollo móvil, los frameworks reducen los tiempos de desarrollo, disminuyen los errores y favorecen el trabajo colaborativo. Entre las opciones más utilizadas se encuentran React Native, basado en JavaScript y en componentes de React, que permite reutilizar código entre Android e iOS (Redacción Tokio, 2022); Ionic, creado por Drifty Co., que emplea HTML, CSS y JavaScript y se puede integrar con Angular (Caballero, 2022); Xamarin, framework de código abierto de Microsoft que usa el lenguaje C# y Visual Studio para crear aplicaciones multiplataforma (Cordón, 2021); y NativeScript, también de código abierto, que permite usar Angular, Vue.js, TypeScript y JavaScript, con acceso directo a las API nativas de Android e iOS (ESIC University, 2025). Kotlin, por su parte, es un lenguaje de tipado estático que se ejecuta sobre la máquina virtual de Java, tiene una sintaxis concisa y detecta errores en tiempo de compilación (Bernardino, 2024). En este trabajo se utilizó Flutter, un framework desarrollado por Google que emplea el lenguaje Dart, permite una alta reutilización de código y genera aplicaciones compiladas de forma nativa para dispositivos móviles, web y escritorio (Flutter, s. f.).

Un SDK es un conjunto de herramientas y recursos que los fabricantes de plataformas, sistemas operativos o lenguajes de programación proporcionan para facilitar la creación de aplicaciones en entornos específicos. Suele incluir compiladores, depuradores, APIs, bibliotecas, documentación y ejemplos de código (Red Hat, 2023). En Firebase, las bibliotecas de servidor permiten configurar entornos con privilegios en lenguajes como C#, Go, Java, Node.js, PHP, Python y Ruby, y dan acceso completo a la base de datos sin estar sujetas a las reglas de seguridad convencionales, porque se gestionan mediante sistemas de control de acceso como IAM. El SDK de Firebase Admin integra en una sola herramienta el acceso a servicios como autenticación y mensajería (Firebase, s. f.).

Firebase es una plataforma de desarrollo de Google para aplicaciones web, Android e iOS. Es una solución de tipo Backend as a Service (BaaS) que integra servicios en la nube que, en muchos casos, sustituyen la necesidad de desarrollar y mantener un backend propio.



Al basarse en una arquitectura sin servidor (*serverless*), elimina la administración de servidores, la gestión de la escalabilidad y la aplicación de actualizaciones y parches de seguridad (Equipo editorial de IONOS, 2025). Entre sus servicios destacan las bases de datos en tiempo real, la autenticación de usuarios y la mensajería y las notificaciones (Merino, 2024). Cloud Firestore es una base de datos NoSQL que Google considera la opción recomendada; ofrece consultas complejas, escalabilidad automática, reglas de seguridad robustas y sincronización sin conexión optimizada (Google, s. f.-a). Firebase Authentication permite registrar usuarios con correo electrónico y contraseña y autenticarlos mediante cuentas de plataformas externas como Facebook, Google y Twitter (Google, s. f.-b). Firebase Cloud Messaging (FCM) permite diseñar, personalizar, segmentar y enviar notificaciones push (Google, s. f.-c). Aunque Firebase evita la necesidad de un backend propio para el almacenamiento y la sincronización de datos, el proyecto incorporó un servicio adicional en Node.js, alojado en Render, dedicado exclusivamente al procesamiento de alertas; este componente complementa, sin sustituir, los servicios administrados de Firebase.

Un *event listener* (escucha de eventos) es un mecanismo que detecta eventos y ejecuta de forma automática una función asociada cuando ocurren; por ejemplo, para actualizar la interfaz, procesar información o activar otros procesos (Lenovo, 2025a). En Cloud Firestore, el método *onSnapshot()* permite escuchar en tiempo real los cambios de un documento: genera una instantánea inicial con su estado actual y la actualiza cada vez que cambia su contenido. Los cambios locales se notifican de inmediato mediante la compensación de latencia, y la propiedad *metadata.hasPendingWrites* indica si existen cambios locales aún no sincronizados con el servidor (Firebase, 2026).

Con base en lo anterior, el objetivo de este trabajo fue desarrollar una aplicación móvil con conexión a datos en tiempo real almacenados en la nube para el monitoreo y la gestión de parámetros fisicoquímicos de calidad del agua en estanques de crianza de alevines de tilapia de una granja acuícola. Los objetivos específicos fueron: (1) desarrollar interfaces intuitivas y amigables; (2) desarrollar un panel (*dashboard*) para monitorear los parámetros fisicoquímicos; (3) integrar la aplicación con la base de datos en la nube mediante el SDK de Firebase; y (4) desarrollar un módulo que emita alertas en un tiempo oportuno.



El trabajo se limita a la solución digital y a su integración con el dispositivo IoT y los servicios en la nube; el dispositivo IoT que captura las mediciones se consideró un sistema externo. No se plantearon hipótesis.

Se espera que una aplicación móvil que visualice los datos recolectados por un dispositivo IoT permita una gestión más eficiente del agua en los estanques, optimice las condiciones de crecimiento de los peces y reduzca los riesgos asociados a cambios bruscos en la calidad del agua. También facilita la consulta remota de los datos y la recepción de alertas en tiempo real, y representa un avance en la aplicación de tecnologías inteligentes en la acuicultura, alineado con las tendencias de digitalización y sostenibilidad del sector.

METODOLOGÍA

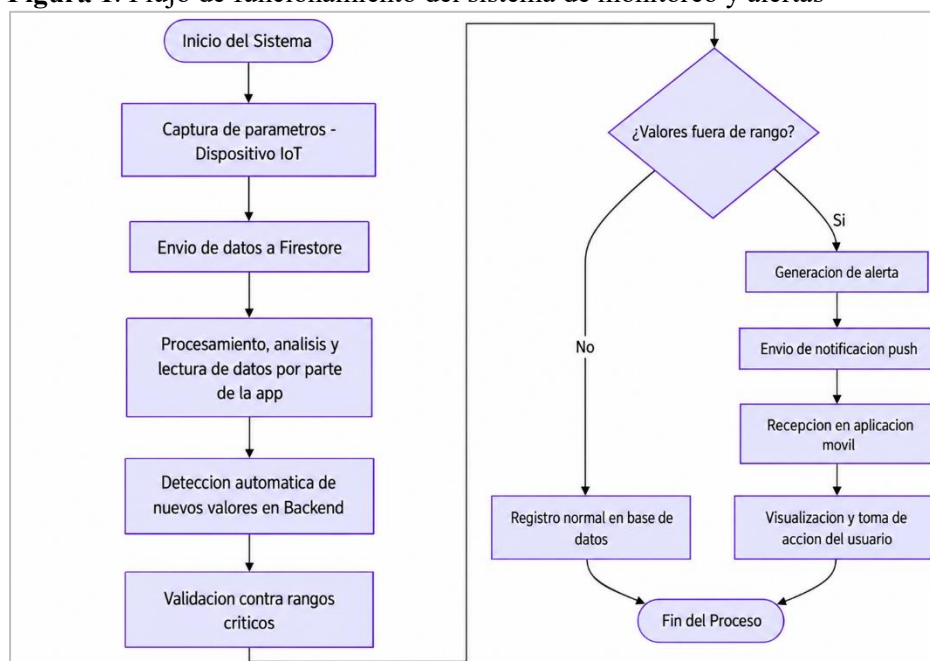
Este trabajo corresponde a un desarrollo tecnológico realizado en coordinación entre el Instituto Tecnológico de Villahermosa y una granja acuícola de la región. El proyecto se ubicó en el área de desarrollo de software y tecnologías de la información y requirió conocimientos de desarrollo móvil, diseño de interfaces de usuario (UI/UX), manejo de bases de datos en la nube y procesamiento de datos en tiempo real. También implicó colaboración con las áreas de desarrollo móvil, sistemas IoT y diseño de experiencia de usuario. El proceso siguió siete etapas, que se resumen en la Tabla 1. Por su naturaleza, el estudio no definió una población ni una muestra y no incluyó análisis estadístico.

Arquitectura del sistema

La arquitectura se definió en la etapa de análisis y establece la comunicación entre tres componentes: el dispositivo IoT, la plataforma Firebase y la aplicación móvil. El dispositivo IoT captura las mediciones de los parámetros fisicoquímicos y las almacena en Cloud Firestore. Este componente formó parte del proyecto general, pero se trató como un sistema externo y su desarrollo queda fuera del alcance de este artículo. Cloud Firestore almacena y sincroniza la información, y la aplicación móvil consulta, procesa y visualiza los datos en tiempo real. Un servicio en Node.js, alojado en la plataforma Render, detecta las nuevas lecturas, las compara con los rangos definidos y envía notificaciones a los dispositivos móviles cuando encuentra valores fuera de rango. La Figura 1 muestra el flujo de funcionamiento del sistema.



Figura 1. Flujo de funcionamiento del sistema de monitoreo y alertas



Fuente: Diagrama elaborado por los autores del proyecto durante la etapa de análisis.

Etapas de desarrollo

El desarrollo se organizó en siete etapas secuenciales. La Tabla 1 resume las actividades y las herramientas de cada una, y los párrafos siguientes describen los aspectos principales.

Tabla 1. Etapas de desarrollo, actividades principales y herramientas

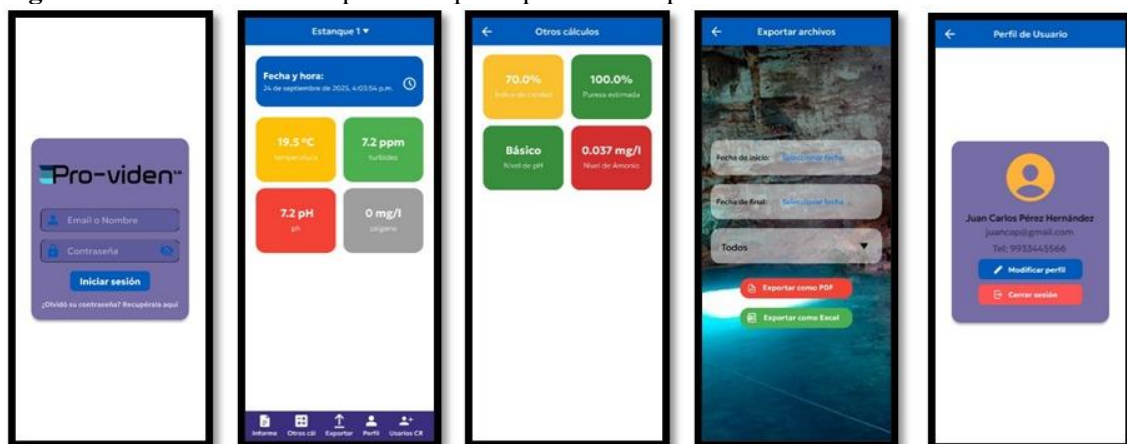
Etapas	Actividades principales	Herramientas
1. Análisis de requerimientos	Investigación documental; definición de requerimientos funcionales, módulos, flujos de información y arquitectura; selección de tecnologías	Diagramas de arquitectura y esquemas de funcionamiento
2. Diseño de interfaces y experiencia de usuario	Bocetos, wireframes, mockups y prototipos; definición de navegación, colores, iconografía y componentes	Figma
3. Desarrollo de la aplicación móvil	Pantallas, módulos independientes (autenticación, monitoreo, exportación, perfil, usuarios y cálculos), servicios y widgets personalizados	Flutter, Dart, Visual Studio Code, emuladores Android
4. Integración con Firebase	Configuración del proyecto, diseño de colecciones y documentos, lectura y escritura en tiempo real con listeners, reglas de seguridad y autenticación	Firebase Console, Firebase SDK, Cloud Firestore
5. Integración de datos del sistema IoT	Análisis de la estructura de datos, clases de servicio, validación de registros, listeners de actualización, pruebas de consistencia y sincronización	Firebase SDK, Cloud Firestore
6. Módulo de alertas	Definición de límites, alertas visuales en la aplicación, servicio de escucha y envío de notificaciones	Flutter, Firebase, Node.js, Render, FCM
7. Pruebas y validación	Pruebas funcionales, de integración y de validación; pruebas con valores simulados fuera de rango; corrección de errores	—

Fuente: Elaboración propia. FCM: Firebase Cloud Messaging.

Etapla 1. Análisis de requerimientos. Se realizó una investigación documental sobre la calidad del agua en sistemas acuícolas para identificar los parámetros más importantes en el desarrollo de los alevines, entre ellos la temperatura, el pH, el oxígeno disuelto y los sólidos disueltos. Después se analizaron las necesidades funcionales: monitoreo en tiempo real, visualización remota de los datos, generación de alertas automáticas y almacenamiento continuo en la nube. A partir de este análisis se definieron los módulos principales (monitoreo en tiempo real, alertas, visualización histórica y administración de usuarios), los flujos de información entre el sistema IoT, Firebase y la aplicación, y la arquitectura general. Se elaboraron diagramas de arquitectura y esquemas de funcionamiento como base de las etapas siguientes. Se seleccionó Flutter como framework principal por su capacidad multiplataforma y su compatibilidad con Firebase.

Etapla 2. Diseño de interfaces y experiencia de usuario. Se elaboraron bocetos y prototipos de navegación para definir la distribución de los elementos visuales. Primero se diseñaron *wireframes* para establecer la estructura básica de las pantallas y después *mockups* con mayor nivel de detalle en Figma. Se definieron la estructura de navegación, los colores, la iconografía y el tamaño de los componentes, y se diseñaron elementos visuales como tarjetas informativas, indicadores de estado y módulos de monitoreo. El diseño se iteró varias veces para mejorar la distribución de los elementos y consideró principios de usabilidad y accesibilidad. Los prototipos funcionales permitieron validar la organización de las pantallas antes de programar.

Figura 2. Wireframes de las pantallas principales de la aplicación



Fuente: Capturas de los prototipos del proyecto, elaborados en Figma.

Etapla 3. Desarrollo de la aplicación móvil. La aplicación se programó con Flutter y Dart, con Visual Studio Code como entorno de programación y emuladores Android para las pruebas preliminares de funcionamiento y de diseño responsivo. Las pantallas se construyeron con *widgets* personalizados de Flutter. Para mantener una estructura organizada y escalable, el proyecto se dividió en módulos independientes de autenticación, monitoreo, exportación de datos, perfil de usuario, gestión de usuarios y cálculos complementarios. Los servicios de comunicación con Firebase y de gestión de datos se separaron de los componentes visuales, lo que favoreció la reutilización de código y las tareas de depuración y mantenimiento. También se desarrollaron los indicadores visuales de tiempo real y los módulos de consulta de información histórica.

Etapla 4. Integración con Firebase y servicios en la nube. Se configuró el proyecto en Firebase Console, se registró la aplicación y se incorporaron los archivos de configuración y las dependencias necesarias para usar Cloud Firestore mediante el SDK oficial de Firebase. La base de datos se organizó en colecciones y documentos según la estructura de datos del sistema IoT, con el fin de permitir consultas eficientes y una sincronización constante entre la nube y la aplicación. Se implementaron métodos de lectura y escritura en tiempo real con listeners de Firebase, que detectan cambios en la base de datos y actualizan las interfaces. También se configuraron reglas de seguridad, autenticación y validación de acceso para proteger la información.

Etapla 5. Integración de datos del sistema IoT. El sistema IoT, externo a la aplicación, capturaba las mediciones y las almacenaba en Firebase. Primero se analizó la estructura de las colecciones y los documentos de Cloud Firestore para ubicar cada parámetro monitoreado. Después se desarrollaron clases de servicio con el SDK de Firebase que leen los datos y los transforman en objetos utilizables por la aplicación, junto con validaciones que verifican la existencia de la información antes de mostrarla y evitan errores por registros incompletos o pérdidas temporales de conexión. Se implementaron listeners que actualizan los valores del panel cada vez que el sistema IoT envía nuevas mediciones. Se hicieron pruebas de consistencia, para verificar que los valores obtenidos de Firebase coincidieran con los registros del sistema IoT, y pruebas de sincronización.

Etapla 6. Desarrollo del módulo de alertas. Se definieron los límites permitidos para variables como la temperatura, el pH y el oxígeno disuelto con base en información técnica sobre la crianza de tilapia, y



se incorporaron a la lógica de la aplicación como condiciones de evaluación que clasifican el estado de cada parámetro. La aplicación compara de forma continua los valores obtenidos de Firebase con esos límites y, cuando una medición los excede, muestra alertas visuales, indicadores de estado y mensajes descriptivos. Además, se desarrolló un servicio de escucha en Node.js para procesar las lecturas en tiempo real y enviar notificaciones a los dispositivos móviles.

Etapa 7. Pruebas, corrección de errores y validación. Se aplicaron pruebas de tres tipos. Las pruebas funcionales verificaron el comportamiento individual de cada módulo, las de integración evaluaron la comunicación entre Flutter, Firebase y el sistema IoT, y las de validación comprobaron que los requerimientos definidos en el análisis se hubieran implementado. Se verificó que las mediciones de los sensores llegaran a la nube, que el panel se actualizara automáticamente ante cambios en la base de datos y que el módulo de alertas generara notificaciones con valores simulados fuera de los rangos permitidos. Los errores detectados de tiempos de respuesta, sincronización de datos y actualización de interfaces se documentaron y se corrigieron en ciclos de prueba y ajuste. Por último, se realizaron pruebas de funcionamiento continuo para validar la estabilidad del sistema completo.

Limitaciones metodológicas

El estudio tiene cinco limitaciones metodológicas. Primero, no se reporta una evaluación formal de usabilidad de las interfaces. Segundo, las pruebas del módulo de alertas incluyeron valores simulados fuera de rango. Tercero, no se reportan métricas cuantitativas de latencia, disponibilidad o exactitud de las lecturas. Cuarto, el dispositivo IoT se consideró un sistema externo, por lo que su selección de sensores, su calibración y su frecuencia de muestreo no se evaluaron en este trabajo. Quinto, no se contó con un sensor físico de turbidez, por lo que este parámetro se implementó en el código y en el modelo de datos, pero no se validó con mediciones reales durante las pruebas.

RESULTADOS Y DISCUSIÓN

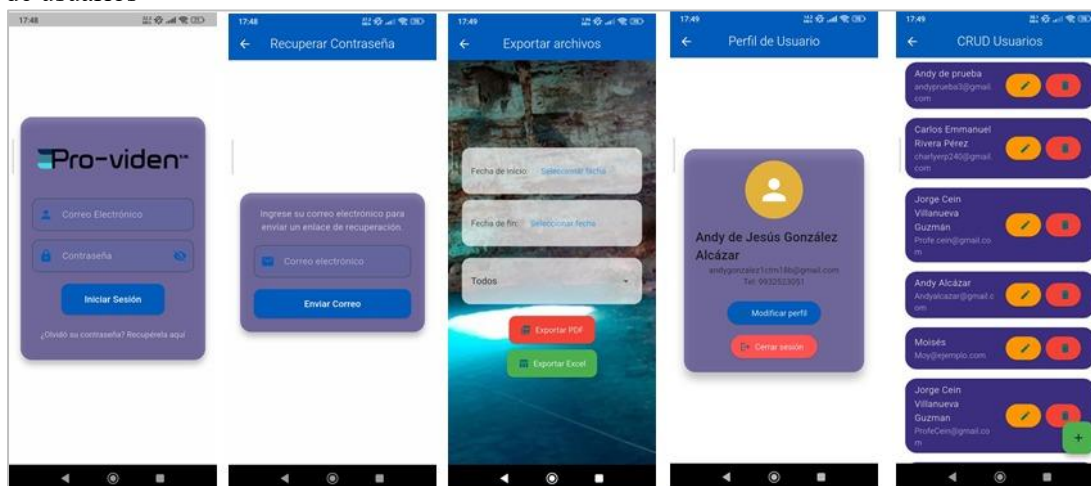
Interfaces intuitivas y amigables

Se desarrolló una aplicación móvil multiplataforma con Flutter cuya barra de navegación inferior da acceso a cinco secciones: informe (monitoreo), cálculos, exportación, perfil y usuarios. El acceso se realiza con correo electrónico y contraseña, y la aplicación incluye recuperación de contraseña mediante un enlace enviado por correo electrónico.



La sección de exportación permite generar reportes en PDF y en Excel a partir de un rango de fechas, y la de usuarios permite administrar las cuentas con las operaciones de alta, edición y eliminación. En el prototipo, la sección de cálculos incluía un índice de calidad, una pureza estimada, el nivel de pH y el nivel de amonio. El código se organizó en módulos independientes de autenticación, monitoreo, exportación, perfil, usuarios y cálculos, con servicios separados de los componentes visuales. Las interfaces usan tarjetas informativas, iconografía descriptiva e indicadores de estado por color para facilitar la interpretación de los datos. La Figura 3 muestra las pantallas de inicio de sesión, recuperación de contraseña, exportación, perfil de usuario y administración de usuarios (CRUD).

Figura 3. Pantallas de inicio de sesión, recuperación de contraseña, exportación, perfil y administración de usuarios



Fuente: Capturas de la aplicación desarrollada en el proyecto.

Panel para monitorear los parámetros

El panel principal muestra en tiempo real los valores de temperatura, pH, oxígeno disuelto y sólidos disueltos del estanque seleccionado, junto con la fecha y la hora de la lectura. El sistema también contempla la turbidez en el modelo de datos y en la lógica de alertas, pero este parámetro no se validó con un sensor físico durante las pruebas, por lo que se excluye de los resultados reportados como variables monitoreadas. Cada variable se presenta en una tarjeta cuyo color indica su estado. Los valores se actualizan automáticamente cuando el sistema IoT registra nuevas mediciones, sin necesidad de recargar la aplicación. La Figura 4 muestra el panel principal de la aplicación.

Figura 4. Panel principal de monitoreo en tiempo real.

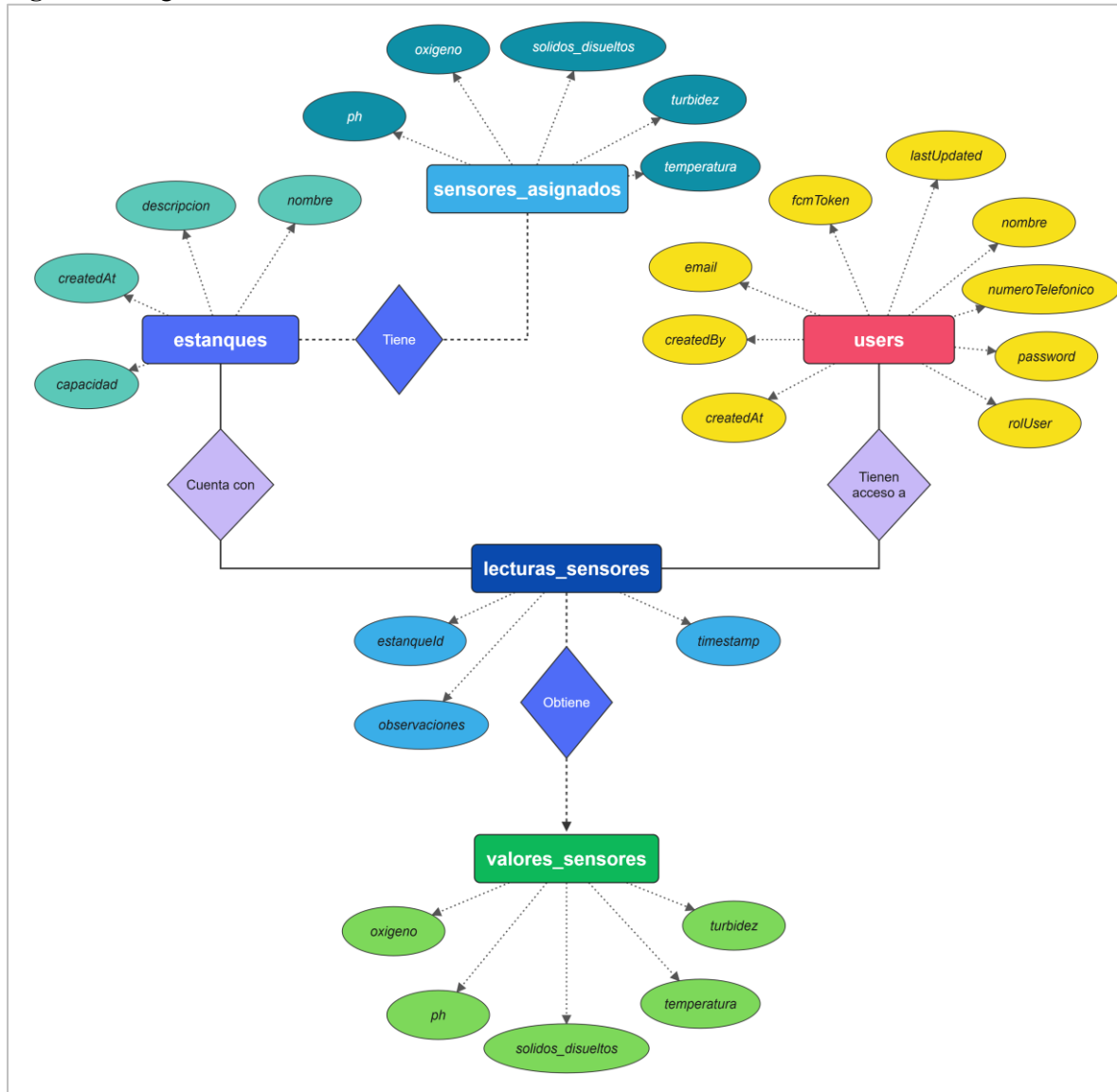


Fuente: Captura de la aplicación desarrollada en el proyecto.

Integración con la base de datos en la nube

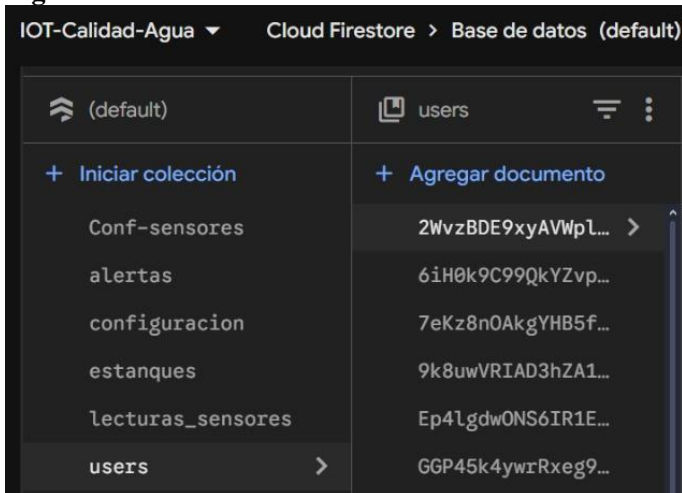
La aplicación se integró con Cloud Firestore mediante el SDK de Firebase. La consola de Firestore del proyecto muestra seis colecciones: Conf-sensores, alertas, configuracion, estanques, lecturas_sensores y users (Figura 6). La Figura 5 muestra el diagrama entidad-relación elaborado para el proyecto, y la Tabla 2 resume las entidades y sus campos a partir de dicho diagrama. La lectura en tiempo real se implementó con listeners de Firebase, que detectan cambios en la base de datos y actualizan la información que ve el usuario. Se implementaron clases de servicio que leen los datos de Firebase y los convierten en objetos de la aplicación, así como validaciones que comprueban la existencia de la información antes de mostrarla. También se configuraron reglas de seguridad, autenticación y validación de acceso. Las pruebas de consistencia y de sincronización verificaron que los valores mostrados coincidieran con los registros generados por el sistema IoT, y la aplicación mantuvo una comunicación continua con la nube para mostrar información actualizada de forma remota.

Figura 5. Diagrama entidad-relación de la base de datos.



Fuente: Diagrama elaborado por los autores del proyecto.

Figura 6. Colecciones creadas en la consola de Cloud Firestore.



Fuente: Captura de la consola de Firebase del proyecto.

Tabla 2. Entidades del modelo de datos y sus campos.

Entidad	Campos	Relación
estanques	nombre, descripcion, capacidad, createdAt	Cuenta con lecturas_sensores; tiene sensores_asignados
sensores_asignados	oxígeno, ph, solidos_disueltos, turbidez, temperatura	Asociada a un estanque
lecturas_sensores	estanqueId, timestamp, observaciones	Obtiene valores_sensores; los usuarios tienen acceso a ella
valores_sensores	oxígeno, ph, solidos_disueltos, turbidez, temperatura	Valores registrados en cada lectura
users	nombre, email, numeroTelefonico, rolUser, fcmToken, createdBy, createdAt, lastUpdated	Tienen acceso a lecturas_sensores

Fuente: Elaboración propia a partir del diagrama entidad-relación del proyecto.

Módulo de alertas en tiempo oportuno

El módulo de alertas tiene dos componentes. El primero son los indicadores visuales de la aplicación, que comparan de forma continua los valores obtenidos de Firebase con los límites configurados y muestran indicadores de estado y mensajes descriptivos cuando una medición los excede. El segundo es un servicio de escucha en Node.js, alojado en Render, que procesa en tiempo real las lecturas del sistema IoT y envía notificaciones a los dispositivos móviles. El servicio se compone de tres módulos principales y un módulo auxiliar (Tabla 3).

Tabla 3. Módulos del servicio de alertas en Node.js.

Módulo	Función principal	Recurso empleado
sensorListener	Escucha la colección lecturas_sensores y detecta la incorporación de nuevas lecturas	Cloud Firestore, onSnapshot()
amonioService	Estima la concentración de amonio a partir de pH, temperatura, oxígeno disuelto, sólidos disueltos y turbidez	Algoritmo propio
notificationService	Evalúa los parámetros con los umbrales, arma el mensaje de alerta y lo envía a los usuarios registrados	Firebase Cloud Messaging
keepAlive	Mantiene activo el servicio en Render con una petición HTTP GET cada 5 minutos	setInterval(), Axios

Fuente: elaboración propia.

El módulo *sensorListener* aplica el método *onSnapshot()* a la colección *lecturas_sensores*, ordenada por *timestamp* de forma descendente y limitada a un documento. Cuando se agrega una lectura nueva, obtiene los valores de oxígeno disuelto, pH, sólidos disueltos, temperatura y turbidez y el identificador del estanque, y los entrega a los módulos de amonio y de alertas. La Figura 7 muestra el fragmento de código correspondiente.



Figura 7. Fragmento de código del módulo sensorListener.

```
function initSensorListener(db) {
  db.collection("lecturas_sensores")
    .orderBy("timestamp", "desc")
    .limit(1)
    .onSnapshot(async (snapshot) => {
      if (!iniciado) {
        iniciado = true;
        console.log("Listener iniciado");
        return;
      }

      snapshot.docChanges().forEach(async (change) => {
        if (change.type !== "added") return;

        const data = change.doc.data();
        if (!data) return;

        const { valores_sensores, estanqueId } = data;

        const oxigeno = valores_sensores.oxigeno ?? 0;
        const ph = valores_sensores.ph ?? 0;
        const solidos = valores_sensores.solidos_disueltos ?? 0;
        const temperatura = valores_sensores.temperatura ?? 0;
        const turbidez = valores_sensores.turbidez ?? 0;
      });
    });
}
```

Fuente: Captura del código fuente del proyecto.

El módulo *amonioService* estima la concentración de amonio porque el sistema no cuenta con un sensor específico para esta variable. El cálculo parte de un valor base de 0.01 y suma incrementos cuando el pH, la temperatura, los sólidos disueltos o la turbidez superan ciertos valores, o cuando el oxígeno disuelto es bajo. El resultado se limita al intervalo de 0 a 1 (Tabla 4) y se usa como indicador complementario del monitoreo. La Figura 8 muestra el fragmento de código correspondiente.

Figura 8. Fragmento de código del módulo amonioService.

```
function calcularAmonioEstimado(ph, temperatura, oxigeno, solidos, turbidez) {
  let amonio = 0.01;

  if (ph > 7.5) amonio += 0.005 * (ph - 7.5);
  if (temperatura > 25) amonio += 0.003 * (temperatura - 25);
  if (oxigeno < 5) amonio += 0.005 * (5 - oxigeno);
  if (solidos > 300) amonio += 0.002 * ((solidos - 300) / 100);
  if (turbidez > 10) amonio += 0.001 * (turbidez - 10);

  return Math.min(Math.max(amonio, 0), 1);
}
```

Fuente: Captura del código fuente del proyecto.

Tabla 4. Reglas de la estimación de amonio.

Condición	Incremento sobre la estimación
Valor inicial	0.01
pH > 7.5	$0.005 \times (\text{pH} - 7.5)$
Temperatura > 25 °C	$0.003 \times (T - 25)$
Oxígeno disuelto < 5 mg/L	$0.005 \times (5 - \text{OD})$
Sólidos disueltos > 300	$0.002 \times ((\text{SD} - 300) / 100)$
Turbidez > 10	$0.001 \times (\text{Tu} - 10)$
Resultado final	Se limita al intervalo de 0 a 1

Fuente: Elaboración propia a partir del código del módulo amonioService. T: temperatura; OD: oxígeno disuelto; SD: sólidos disueltos; Tu: turbidez.

El módulo `notificationService` compara cada valor con las condiciones de la Tabla 5 y, por cada condición que se cumple, agrega un mensaje a una lista de alertas. Si la lista tiene al menos un mensaje, el módulo obtiene los tokens de notificación de los usuarios registrados en la colección `users`, arma un mensaje con el identificador del estanque y los parámetros que generaron la alerta, y lo envía con Firebase Cloud Messaging, con prioridad alta, a todos los dispositivos registrados. Un ejemplo de mensaje recibido en la aplicación fue «Alerta en Estanque1: pH fuera de rango: 8.17 | Sólidos altos: 597». La Figura 9 muestra el fragmento de código correspondiente, y la Figura 10 muestra un ejemplo de la notificación recibida en la aplicación móvil.

Figura 9. Fragmento de código del módulo `notificationService`.

```
let alertas = [];

if (oxigeno < 5 || oxigeno > 8)
  alertas.push(`Oxígeno fuera de rango: ${oxigeno}`);

if (ph < 6.5 || ph > 7.5)
  alertas.push(`pH fuera de rango: ${ph}`);

if (temperatura < 20 || temperatura > 25)
  alertas.push(`Temperatura fuera de rango: ${temperatura}`);

if (solidos > 400)
  alertas.push(`Sólidos altos: ${solidos}`);

if (turbidez > 400)
  alertas.push(`Turbidez alta: ${turbidez}`);

if (amonio > 0.02)
  alertas.push(`Amonio elevado: ${amonio.toFixed(3)}`);

if (alertas.length === 0) {
  console.log("Valores normales");
  return;
}

const usersSnap = await db.collection("users").get();
const tokens = usersSnap.docs.flatMap(
  (doc) => doc.data().fcmTokens || []
);

if (tokens.length === 0) {
  console.log("No hay tokens");
  return;
}

const payload = {
  notification: {
    title: `⚠ Alerta en ${estanqueId}`,
    body: alertas.join(" | "),
  },
  data: {
    estanqueId,
    oxigeno: oxigeno.toString(),
    ph: ph.toString(),
    temperatura: temperatura.toString(),
    solidos: solidos.toString(),
    turbidez: turbidez.toString(),
    amonio: amonio.toFixed(3),
  },
  android: { priority: "high" },
};

await enviarNotificacion(tokens, payload);

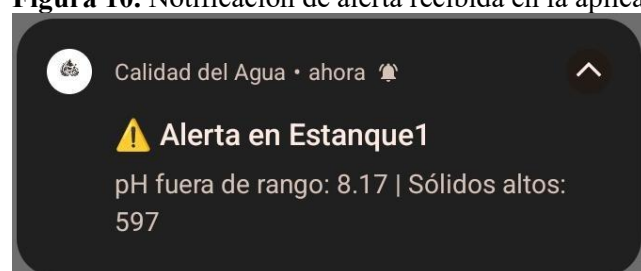
console.log("📢 Notificaciones enviadas");
});

async function enviarNotificacion(tokens, payload) {
  const messaging = getMessaging();

  return messaging.sendEachForMulticast({
    tokens,
    notification: payload.notification,
    data: payload.data,
    android: payload.android,
  });
}
```

Fuente: Captura del código fuente del proyecto.

Figura 10. Notificación de alerta recibida en la aplicación móvil.



Fuente: Captura de la aplicación desarrollada en el proyecto.

Tabla 5. Intervalos de referencia de la literatura y condiciones de alerta implementadas.

Parámetro (unidad)	Intervalo de referencia en la literatura	Condición de alerta implementada
Temperatura (°C)	28 a 32 (óptimo); mantener por encima de 25 en alevines (INTAGRI S.C., 2020)	< 20 o > 25
Oxígeno disuelto (mg/L)	5 a 9 (El productor, 2017)	< 5 o > 8
pH	6.5 a 8.5 (Despesca, 2025)	< 6.5 o > 7.5
Amonio, N-NH ₃ (mg/L)	< 0.1 (SAGARPA, 2008)	> 0.02 (valor estimado)
Sólidos disueltos (ppm)	No incluido en la literatura revisada	> 400
Turbidez	No incluido en la literatura revisada	> 400

Fuente: Elaboración propia.

El módulo *keepAlive* no forma parte del flujo de procesamiento de datos. Su función es evitar que la plataforma Render suspenda el servicio por inactividad. Para ello, ejecuta con *setInterval()* una petición HTTP GET con Axios a la URL del servicio cada 5 minutos; la URL se obtiene de una variable de entorno y, si ocurre un error, este se registra en consola sin afectar la ejecución del sistema. La Figura 11 muestra el fragmento de código correspondiente.

Figura 11. Fragmento de código del módulo *keepAlive*.

```
function startKeepAlive() {
  setInterval(async () => {
    try {
      const url = process.env.RENDER_EXTERNAL_URL;
      if (url) {
        await axios.get(url);
        console.log("Ping enviado");
      }
    } catch (error) {
      console.log("Ping error:", error.message);
    }
  }, 300000);
}
```

Fuente: Captura del código fuente del proyecto.

Pruebas y problemas atendidos

Las pruebas funcionales, de integración y de validación verificaron la llegada de las mediciones a la nube, la actualización automática del panel y la generación de notificaciones con valores simulados fuera de rango. Los errores de tiempos de respuesta, sincronización de datos y actualización de interfaces se corrigieron ajustando los listeners, optimizando las consultas a Firestore y mejorando el rendimiento de la aplicación. Los problemas identificados en el análisis inicial se atendieron de la siguiente manera. La consulta inmediata de los parámetros, que antes requería revisiones presenciales y registros manuales, se resolvió con una aplicación que muestra los datos de Firebase desde cualquier

lugar con acceso a internet. La falta de una plataforma centralizada se atendió con un panel que integra los parámetros principales en una sola interfaz. La sincronización entre el sistema IoT y la aplicación se resolvió con Cloud Firestore y la actualización en tiempo real. La ausencia de advertencias oportunas se atendió con el módulo de alertas.

DISCUSIÓN

Los resultados muestran que Flutter y Firebase permiten construir una aplicación multiplataforma que consulta y sincroniza datos en tiempo real. Flutter se eligió por su capacidad multiplataforma y su compatibilidad con Firebase, características que coinciden con la alta reutilización de código y la compilación nativa para varias plataformas que se atribuyen a este framework (Flutter, s. f.). Firebase, como servicio de tipo BaaS, evitó desarrollar un backend propio para almacenar y sincronizar los datos (Equipo editorial de IONOS, 2025). El envío automático de notificaciones ante nuevas lecturas se resolvió con un servicio adicional en Node.js, que incluye un mecanismo *keep-alive* para evitar su suspensión por inactividad. Por lo tanto, la arquitectura final combina servicios administrados con un componente propio. El uso de listeners con *onSnapshot()* permitió que tanto la aplicación como el servicio de alertas reaccionaran a las nuevas lecturas sin necesidad de recargar la información (Firebase, 2026).

Desde el punto de vista del manejo del cultivo, la práctica recomendada para el alevinaje incluye medir la temperatura, el oxígeno disuelto y el pH al menos dos veces al día y llevar una bitácora para identificar tendencias (SAGARPA, 2008). El sistema desarrollado ofrece consulta remota de las lecturas, un registro en la nube y exportación de reportes, lo que apoya esa práctica. Sin embargo, la frecuencia de muestreo y la calibración de los sensores dependen del sistema IoT externo y no se evaluaron en este trabajo.

Los umbrales implementados en el módulo de alertas resultaron más conservadores que los intervalos óptimos reportados en la literatura consultada (Tabla 5). Por ejemplo, el sistema genera una alerta de pH por encima de 7.5, mientras que la literatura señala un intervalo óptimo de hasta 8.5 (Despesca, 2025); de forma similar, la alerta de temperatura se activa por encima de 25 °C, valor que la literatura identifica como el límite inferior recomendado para los alevines, no el superior (INTAGRI S.C., 2020).



Esta diferencia sugiere que los umbrales se definieron con un criterio de alerta temprana antes de alcanzar el límite crítico, más que como una réplica directa de los rangos óptimos; sin embargo, esta decisión de diseño no está documentada en los materiales del proyecto y debe justificarse y calibrarse formalmente antes de usarse en producción. Para los sólidos disueltos y la turbidez, la comparación no fue posible porque la literatura revisada no reporta un intervalo de referencia para estas variables en alevines de tilapia.

La estimación de amonio es un indicador empírico. Sus coeficientes y su umbral de alerta no cuentan con una validación reportada frente a mediciones directas, y la proporción de amonio no ionizado depende del pH (Despesca, 2025); por ello, la estimación no sustituye una medición directa. Además, el estudio se limitó a una aplicación desarrollada para una granja acuícola, sin evaluación formal de usabilidad y sin métricas cuantitativas de latencia o disponibilidad, y dependió de plataformas de terceros (Firebase y Render) y de un sistema IoT externo.

Los resultados obtenidos coinciden con estudios previos que han utilizado Firebase como plataforma de sincronización en tiempo real para el monitoreo de estanques (Sung et al., 2023) y confirman la factibilidad señalada por Hridoy et al. (2025), quienes identificaron el desarrollo de paneles móviles con alertas en tiempo real como una línea de trabajo futura para el monitoreo de tilapia. A diferencia de Kanwal et al. (2024), cuyo sistema incorpora analítica predictiva para anticipar condiciones críticas del agua, el módulo de alertas del presente trabajo se basa en reglas de umbral fijas, sin componentes predictivos. En cuanto a los rangos de referencia, Hridoy et al. (2025) reportaron para tilapia un intervalo deseable de oxígeno disuelto de 6 a 9 mg/L y de pH de 6.5 a 8.5, cercanos a los valores de la literatura consultada en este trabajo (Tabla 5), lo que refuerza la observación de que los umbrales implementados en el sistema resultaron más conservadores que los rangos reportados en estudios similares.

Como trabajo futuro se propone ampliar la aplicación con herramientas de análisis histórico para identificar tendencias de los parámetros, agregar canales de notificación externos como el correo electrónico, incorporar nuevos parámetros y compatibilidad con otros dispositivos IoT, y continuar las pruebas en condiciones reales de operación durante periodos prolongados.



También se recomienda mantener actualizadas las tecnologías empleadas, en especial Flutter y Firebase, para conservar la seguridad, la estabilidad y la compatibilidad del sistema.

CONCLUSIONES

Se desarrolló una aplicación móvil con Flutter y Firebase que permite monitorear en tiempo real los parámetros fisicoquímicos del agua en estanques de alevines de tilapia y que cumplió los cuatro objetivos específicos: interfaces con cinco secciones, un panel con cinco variables en tiempo real, integración con Cloud Firestore mediante el SDK de Firebase y un módulo de alertas con notificaciones de prioridad alta.

La aplicación atiende los problemas identificados: la dependencia de mediciones manuales, la falta de acceso en tiempo real a la información y la ausencia de alertas oportunas. Los resultados indican que integrar tecnologías móviles, servicios en la nube y sistemas IoT es una alternativa viable para mejorar el acceso a la información y apoyar la toma de decisiones en la acuicultura.

Para su uso en producción queda pendiente validar el sistema en condiciones reales de operación durante periodos prolongados, documentar y calibrar los umbrales de alerta, y validar la estimación de amonio con mediciones directas.

REFERENCIAS BIBLIOGRAFICAS

- Agency, I. C. (2017, 27 de noviembre). Tipos de aplicaciones móviles. <https://ideascreativas.com.ec/blog/tipos-de-aplicaciones-moviles/>
- Bernardino, F. (2024, 24 de julio). Frameworks multiplataforma para el desarrollo de aplicaciones móviles. <https://beecrowd.com/es/blog-posts/frameworks-multiplataforma-2/>
- Caballero, Y. P. (2022, 1 de noviembre). Los mejores entornos para crear aplicaciones Android [sin Java]. <https://yeeply.com/blog/desarrollo-de-apps/entornos-programacion-desarrollar-apps-android/>
- CEUPE. (2019, 15 de enero). Aplicaciones móviles: tipos, ventajas e inconvenientes. <https://www.ceupe.com/blog/aplicaciones-moviles-tipos-ventajas-e-inconvenientes.html>
- Cordón, M. J. (2021, 18 de mayo). Frameworks multiplataforma para desarrollo móvil. <https://www.hiberus.com/crecemos-contigo/framework-multiplataforma-para-desarrollo-movil/>



Despesca. (2025, 24 de diciembre). 10 parámetros de calidad del agua en la tilapicultura: Guía técnica.

<https://despesca.com.br/es/manejo-es/10-parametros-calidad-agua-tilapia/>

El productor. (2017, 11 de agosto). Calidad del agua para la tilapia en estanques rústicos.

<https://elproductor.com/2017/08/calidad-del-agua-para-la-tilapia-en-estanques-rusticos/>

Equipo editorial de IONOS. (2025, 13 de agosto). ¿Qué es Firebase?

<https://www.ionos.mx/digitalguide/servidores/know-how/firebase/>

ESIC University. (2025, 3 de septiembre). ¿Qué es NativeScript? Características.

<https://www.esic.edu/rethink/tecnologia/nativescript-que-es-y-caracteristicas-c>

Fedeacua. (2018). Cartilla de siembra de alevinos. Minagricultura.

Firebase. (s. f.). SDK y bibliotecas cliente.

<https://firebase.google.com/docs/firestore/client/libraries?hl=es-419>

Firebase. (2026, 24 de marzo). Obtén actualizaciones en tiempo real con Cloud Firestore.

<https://firebase.google.com/docs/firestore/query-data/listen?hl=es-419>

Flutter. (s. f.). Flutter: Crea aplicaciones para cualquier pantalla. <https://flutter.dev/>

Google. (s. f.-a). Cloud Firestore. <https://firebase.google.com/docs/firestore?hl=es-419>

Google. (s. f.-b). Firebase Authentication. <https://firebase.google.com/docs/auth?hl=es-419>

Google. (s. f.-c). Firebase Cloud Messaging. <https://firebase.google.com/docs/cloud-messaging/?hl=es-419>

Hridoy, M. A. A. M., Bordin, C., Masood, A., & Masood, K. (2025). Predictive modelling of aquaculture water quality using IoT and advanced machine learning algorithms. Results in Chemistry, 16, 102456. <https://doi.org/10.1016/j.rechem.2025.102456>

INTAGRI S.C. (2020, 13 de enero). Requerimientos del cultivo de tilapia: Calidad del agua.

<https://www.intagri.com/articulos/ganaderia/requerimientos-del-cultivo-de-tilapia>

ISIL. (2023, 13 de mayo). Principales tipos de aplicaciones móviles.

<https://isil.pe/blog/tecnologia/tipos-aplicaciones-moviles/>

Kanwal, S., Abdullah, M., Kumar, S., Arshad, S., Shahroz, M., Zhang, D., & Kumar, D. (2024). An optimal Internet of Things-driven intelligent decision-making system for real-time fishpond



- water quality monitoring and species survival. *Sensors*, 24(23), 7842.
<https://doi.org/10.3390/s24237842>
- lab9 Digital Agency. (2024, 31 de julio). Aplicaciones móviles: definición, características y proceso de diseño y desarrollo. <https://lab9.co/aplicaciones-moviles-definicion-caracteristicas-y-proceso-de-diseno-y-desarrollo/>
- Lenovo. (2025a, 26 de junio). ¿Qué es un event listener? <https://www.lenovo.com/mx/es/glosario/event-listener/>
- Lenovo. (2025b, 26 de junio). ¿Qué es un framework? <https://www.lenovo.com/mx/es/glosario/framework/>
- Merino, Á. M. (2024, 18 de septiembre). Firebase: Una plataforma integral para el desarrollo de aplicaciones web y móviles. <https://www.sngular.com/es/insights/313/firebase>
- Oliver. (2022, 4 de enero). Principales tipos de aplicaciones y ejemplos por sectores. <https://appdesign.dev/principales-tipos-de-aplicaciones-y-ejemplos-por-sectores/>
- Redacción Tokio. (2022, 1 de marzo). El framework para el desarrollo de aplicaciones móviles: Lo que debes saber. <https://www.tokioschool.com/noticias/framework-desarrollo-aplicaciones-moviles/>
- Red Hat. (2023, 4 de agosto). SDK: ¿Qué es y para qué sirve? <https://www.redhat.com/es/topics/cloud-native-apps/what-is-SDK>
- SAGARPA. (2008). Manual de buenas prácticas de producción acuícola de tilapia para la inocuidad alimentaria. CIAD.
- Sung, W., Tofik Isa, I. G., & Hsiao, S.-J. (2023). An IoT-based aquaculture monitoring system using Firebase. *Computers, Materials & Continua*, 76(2), 2179–2200.
<https://doi.org/10.32604/cmc.2023.041022>
- Sustentable, I. M. (2018, 21 de marzo). Acuicultura Tilapia. <https://www.gob.mx/imipas/acciones-y-programas/acuicultura-tilapia>
- Villanueva Guzmán, J. C., Arias Peregrino, V. M., Gómez Domínguez, E., Zamora Reséndiz, P., & López López, N. del C. (2025). Gestión optimizada de datos acuícolas con IOT:



Almacenamiento y procesamiento en la nube. *Revista Ipsumtec*, 8(2), 218–228.

<https://doi.org/10.61117/ipsumtec.v8i2.434>

