# Parcial Nº3 Sistema de Monitoreo de Sensores

**Laboratorio: Sistema de Monitoreo de Sensores con Flask y Firebase**

Jerónimo Vallejo Quintero

**Facultad de Ingenierías / Programa de Ingeniería de Software**

**Docente: Hugo Hernán Henao**

**- Corporación Universitaria Empresarial Alexander Von Humboldt -**

***Fecha de entrega:*** *01 de diciembre de 2024*

## Resumen

Este proyecto consistió en desarrollar un sistema para monitorear y visualizar los niveles de agua en tanques utilizando simulaciones y tecnologías web modernas. Se implementó una API en Flask que interactúa con Firebase para almacenar los datos de sensores simulados. Cada sensor calcula el nivel de agua basado en la distancia medida, transformándola en volumen. La interfaz web, diseñada con HTML, Bootstrap y Chart.js, permite visualizar datos en gráficos de barras actualizados en tiempo real y consultar un historial tabulado. Se incluyeron funcionalidades como exportación a CSV, filtrado por fechas y gráficos generados con Matplotlib para análisis adicional.

La documentación abarcó un manual de usuario, instrucciones técnicas en README.md y un archivo requirements.txt para gestionar dependencias. La evaluación del sistema mostró su eficacia, cumpliendo con los objetivos iniciales. La página web, junto con las gráficas y la interacción con Firebase, valida la viabilidad del sistema para aplicaciones futuras en monitoreo automatizado.

*Palabras clave:* Sensores, API, Flask, Firebase.

**1. Introducción**. En la actualidad, la automatización de procesos en sistemas de control es un pilar fundamental para mejorar la eficiencia operativa en diversas industrias. Un área clave de aplicación es la monitorización y gestión de niveles de líquidos en tanque de almacenamiento. Este proyecto aborda el diseño e implementación de un sensor de llenado de tanques basado en proximidad, una solución innovadora que permite medir el nivel de líquido de manera precisa y no intrusiva.

El desarrollo del sistema se centra en la integración de tecnologías modernas de sensores de proximidad, que ofrecen ventajas como alta sensibilidad, resistencia a condiciones adversas y bajo mantenimiento. El sensor recolecta datos en tiempo real sobre el nivel del líquido y los transmite a un sistema central de monitoreo. Estos datos pueden ser procesados y visualizados mediante interfaces gráficas, facilitando su interpretación y análisis.

Con aplicaciones potenciales en sectores como el industrial, agrícola y doméstico, este sistema tiene como objetivo optimizar el control y la gestión de recursos líquidos, reduciendo desperdicios y mejorando la sostenibilidad. La solución se orienta a ser accesible, eficiente y adaptable a diferentes necesidades operativas, estableciendo un precedente en la innovación tecnológica para la gestión de líquidos.

# 2. Objetivos

## 2.1. Objetivo general

Simular el funcionamiento de un sensor que envía datos a una API desarrollada con Flask.

## 2.2. Objetivos secundarios

* Crear una API que registre los datos enviados por el sensor en una base de datos Firebase.
* Diseñar una interfaz web que permita consultar y graficar los datos registrados, filtrándolos por sensor y rango de fechas.

# 3. Metodología

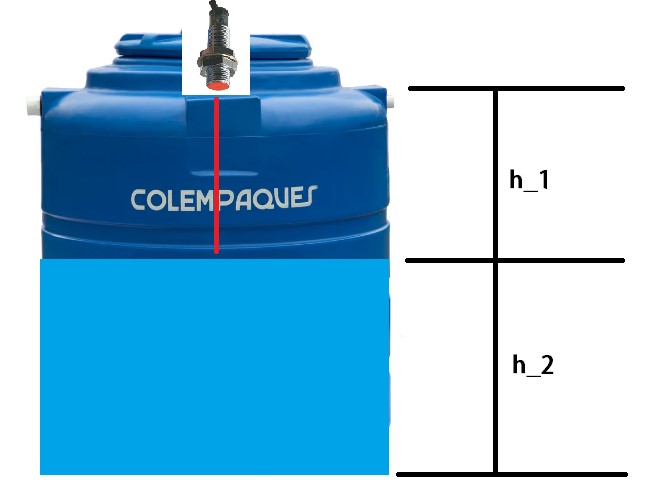
En este proyecto, se realizará la simulación de un sistema de monitoreo para siete tanques de almacenamiento, cada uno con una capacidad de 300 litros, representados en la Figura 4.

El nivel de líquido en cada tanque será determinado mediante la simulación de un sensor de proximidad ubicado en la parte superior. Este sensor calculará la distancia entre la tapa del tanque y la superficie del líquido presente. En este modelo, se asume un entorno ideal en el cual se omiten fenómenos como la reflexión de ondas en la superficie del líquido, garantizando un enfoque simplificado para el análisis.



**Figura 1** Representación de tanques con capacidad de 300 litros.

A partir de la distancia medida por el sensor, se calculará el volumen de espacio vacío en el tanque (*h*1). Este valor será restado del volumen total del tanque para determinar el volumen ocupado por el líquido (*h*2), permitiendo así monitorear el nivel de llenado de manera precisa.



**Figura 2** Enter Caption

Para la implementación del sistema de monitoreo, se empleará la biblioteca Flask en Python, que facilitará la comunicación entre el servidor en tiempo real alojado en Firebase y los datos generados en campo. Flask actuará como intermediario para enviar peticiones HTTP que contienen información estructurada, incluyendo la fecha, el valor del nivel medido y el identificador único de cada tanque. Estas peticiones serán procesadas por el servidor en Firebase, almacenándose en un nodo denominado sensornivel-agua.

En Firebase, se configurarán reglas de acceso específicas para garantizar la seguridad de los datos. Estas reglas limitarán la lectura y escritura únicamente a las identificaciones de tanques autorizadas, mitigando posibles vulnerabilidades y asegurando la integridad del sistema.

La visualización de los datos se realizará mediante un gráfico de barras que representará el nivel de líquido, en litros, de los siete tanques monitoreados. Este gráfico se actualizará automáticamente cada 30 segundos, ofreciendo información en tiempo real al usuario. Adicionalmente, se incluirá un botón para realizar actualizaciones manuales, permitiendo mayor flexibilidad en el monitoreo.

Finalmente, al pie de la interfaz, se dispondrá de un historial universal que recopilará todos los datos ingresados al servidor. Estos datos estarán organizados en una tabla, facilitando su clasificación y consulta según las necesidades del usuario. Esta estructura proporcionará una visión integral del comportamiento del sistema a lo largo del tiempo y será clave para el análisis histórico y la trazabilidad.

# 4. Implementación del Sistema

El proceso de desarrollo del sistema se llevó a cabo siguiendo los pasos descritos a continuación:

1. Se creó una base de datos en tiempo real utilizando la funcionalidad *Real-time Database* de Firebase, configurada para almacenar y procesar los datos generados por los sensores simulados.
2. En la sección de configuración, dentro de *Cuentas de servicio*, se descargó el SDK de Firebase Admin, el cual fue integrado al repositorio del proyecto para garantizar la interacción entre el sistema y el servidor en tiempo real.
3. Se diseñó un archivo .py para el entorno de simulación, que genera datos de manera periódica y simula el comportamiento de los sensores.
4. Se desarrolló un archivo .py principal (main) que recibe los datos generados en el entorno simulado y gestiona las peticiones HTTP al servidor, integrando las funciones del sistema.
5. Se creó un archivo .py adicional encargado de ejecutar simultáneamente el entorno de simulación y el archivo principal, facilitando la sincronización entre ambos procesos.
6. Finalmente, se diseñó un archivo HTML que permite la visualización de los datos recopilados. Este archivo incluye una representación gráfica en forma de barras para mostrar los niveles de líquido en los tanques. La gráfica se actualiza de forma automática o manual según la preferencia del usuario.

Cada etapa del proceso fue documentada cuidadosamente para asegurar la replicabilidad del sistema y su correcta operación en diferentes entornos.

## 4.0.1. Sensores

Usaremos el Sensor Distancia Proximidad Infrarrojo E18-D80NK NPN, debido a lo económico y simples que son.



**Figura 3** Sensor proximidad infrarrojo E18-D80NK

El sistema utiliza una simulación de sensores para monitorear los niveles de agua en tanques de 300 litros. Se han configurado siete sensores virtuales, cada uno identificado de forma única mediante una combinación del nombre del usuario y un número de identificación, asegurando una trazabilidad clara en la base de datos.

Cada sensor mide la distancia desde su ubicación en la parte superior del tanque hasta el nivel del agua. Estos datos se transforman en un volumen equivalente, calculado mediante la fórmula: *volumen* = *π · radio*2 *· nivel*\_*agua ·* 1000

donde *radio* es el radio del tanque, y *nivel*\_*agua* representa la diferencia entre la altura total del tanque y la distancia medida por el sensor.

Los sensores envían los datos en formato JSON a una API desarrollada con Flask. Cada paquete de datos contiene:

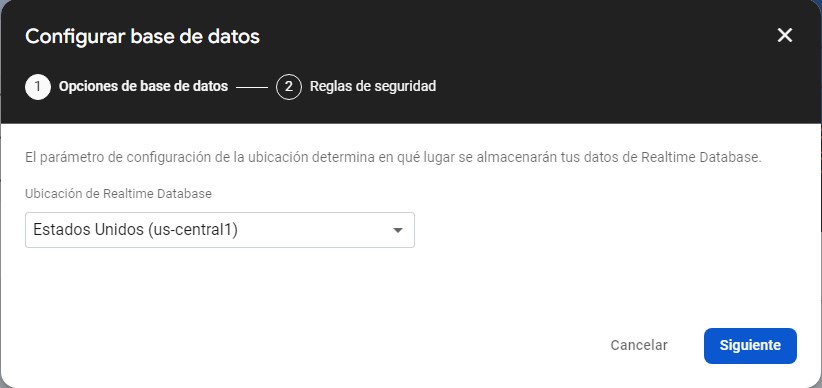
* idsensor: Identificador único del sensor.
* nivel: Nivel del agua en litros.

El envío se realiza a intervalos regulares, definidos en dos segundos, para garantizar una actualización constante y precisa de los niveles de los tanques. En caso de errores durante la transmisión, el sistema registra los fallos para facilitar su diagnóstico. Además, los datos enviados por cada sensor son almacenados en estructuras locales para su posterior visualización en gráficos de barras interactivos.

La implementación en Python permite que cada sensor funcione de manera concurrente mediante hilos de ejecución (*threads*), optimizando el rendimiento y asegurando que los datos de todos los sensores se envíen en tiempo real. Este diseño modular facilita la escalabilidad del sistema y su integración con bases de datos externas, como Firebase.

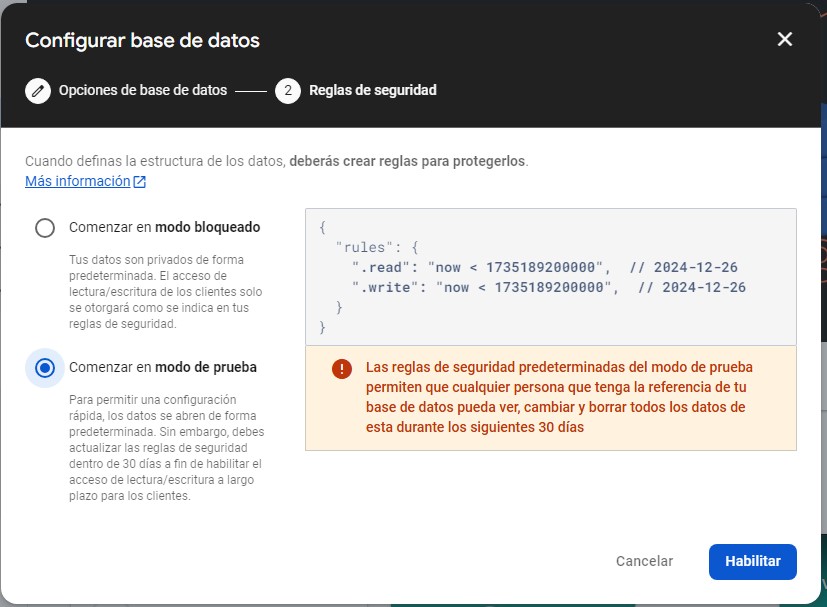
## 4.0.2. Firebase

Para la implementación del sistema, se configuró un proyecto en Firebase, seleccionando como servidor la región de Estados Unidos para optimizar la conectividad y el rendimiento del sistema. En la Figura 4, se presenta la configuración inicial de la base de datos.



**Figura 4** Configuración inicial del proyecto en Firebase.

Durante el proceso, se seleccionó el *modo prueba* para facilitar las pruebas iniciales, como se ilustra en la Figura 5.

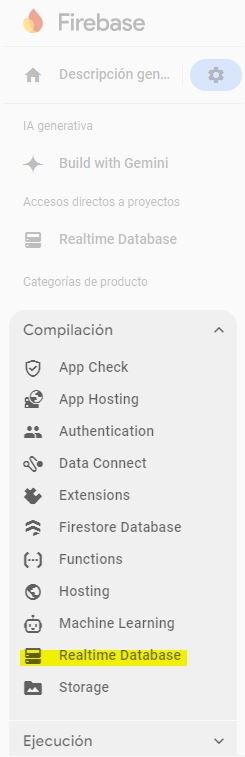


**Figura 5** Selección del modo prueba.

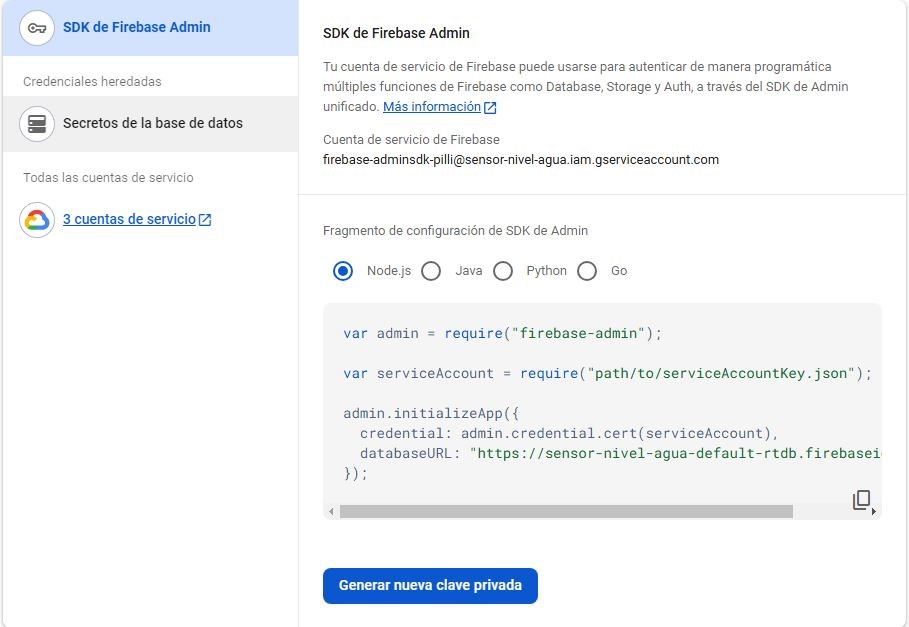
Posteriormente, se creó una base de datos en tiempo real (*Real-time Database*), adecuada para manejar la comunicación continua entre los sensores simulados y el servidor, como se muestra en la Figura 6.

En la sección de configuración, dentro de *Cuentas de servicio*, se descargó el SDK de Firebase Admin. Este archivo se integró en el repositorio del proyecto para garantizar la interacción adecuada entre el sistema local y el servidor en tiempo real. El proceso de generación de la clave privada necesaria se detalla en la Figura 7.

Inicialmente, las reglas de acceso a la base de datos fueron configuradas para permitir cualquier conexión IP. Aunque Firebase advierte sobre los riesgos de esta configuración, al tratarse de un proyecto a pequeña escala, se decidió utilizar estas reglas



**Figura 6** Configuración de la base de datos en tiempo real.

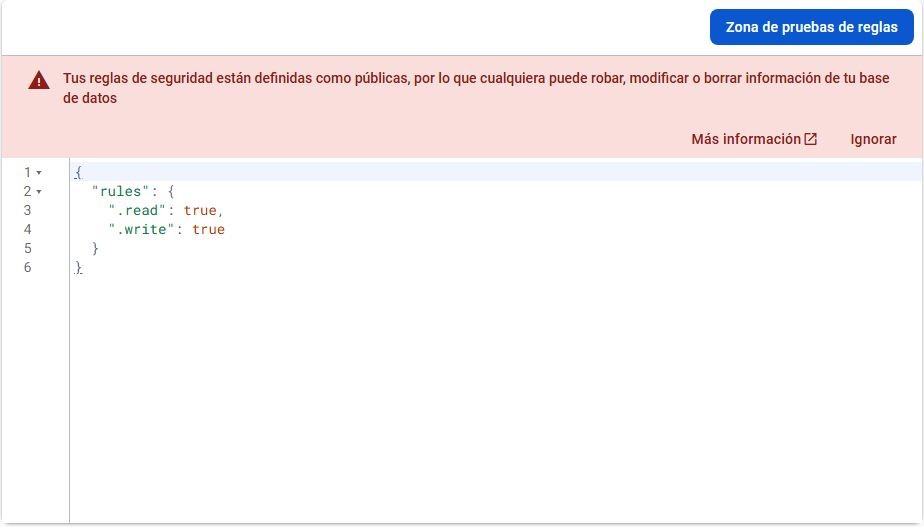


**Figura 7** Generación de clave privada para Firebase Admin SDK.

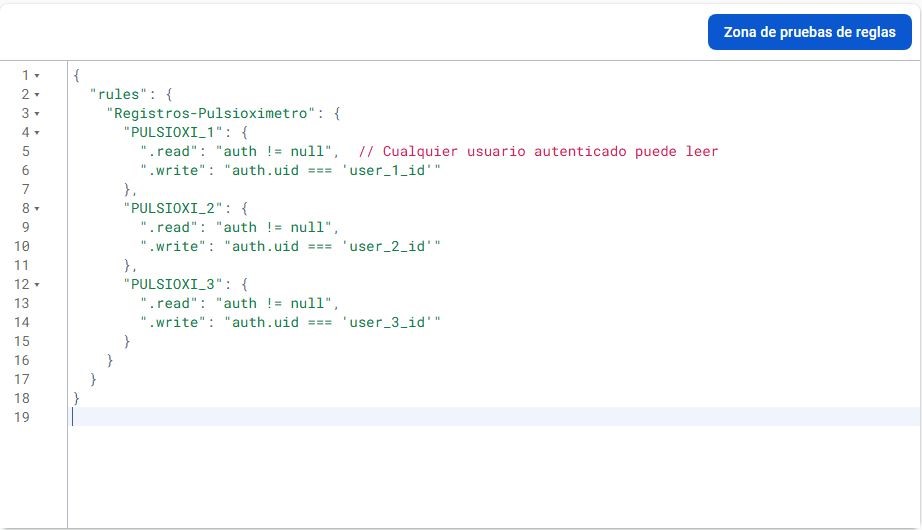
provisionales, como se observa en la Figura 8, para facilitar las pruebas.

Para la fase final, se definieron reglas más estrictas que permiten únicamente la lectura y escritura de datos correspondientes a las identificaciones de los sensores de proximidad. Estas reglas definitivas garantizan la seguridad del sistema, como se muestra en la Figura 9.

Finalmente, se identificó el nodo dentro de la base de datos en tiempo real donde se almacenarán los

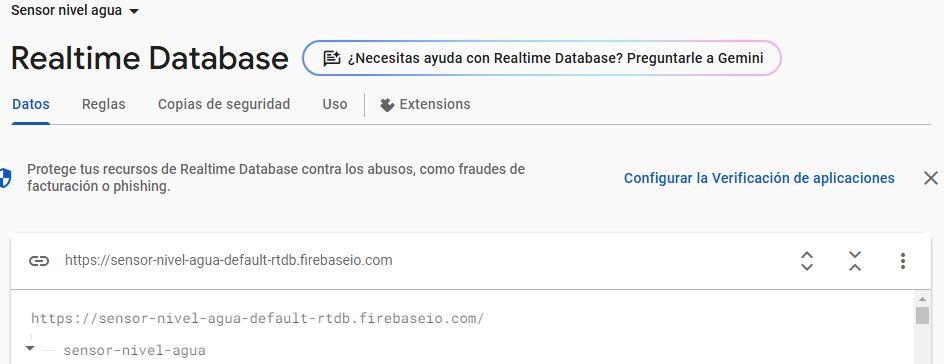


**Figura 8** Reglas provisionales de acceso a la base de datos.



**Figura 9** Reglas definitivas de acceso seguro.

datos enviados por los sensores, como se indica en la Figura 10.



**Figura 10** Identificación del nodo de almacenamiento de datos.

## 4.0.3. API REST

Para implementar la API en Python que conecta los datos de los sensores con Firebase, se siguieron los siguientes pasos:

1. **Inicialización de Flask y Firebase:** Se utilizó Flask como marco de trabajo para la API. Además, se configuró Firebase mediante la integración del archivo .json que contiene la clave generada desde el servicio de cuentas de Firebase Admin.
2. **Validación del formato de la ID del sensor:** Se definió un patrón mediante expresiones regulares (regex) para validar las identificaciones de los sensores (idsensor). Este formato asegura que las ID sigan la estructura definida para los sensores activos (por ejemplo, JERO19791 a JERO19797).
3. **Definición de la función recibir\_datos:** Esta función recibe la información enviada por los sensores simulados en formato JSON. Los datos incluyen:
   * idsensor: Identificador único del sensor.
   * nivel: Nivel del agua en litros.

La función valida los datos, obtiene la fecha y hora actuales, y construye un registro en forma de diccionario. Este registro se almacena en Firebase mediante el comando push, que genera una clave única para cada entrada.

1. **Confirmación de operación:** La función imprime un mensaje en la consola para confirmar si los datos se guardaron exitosamente en Firebase o si ocurrió algún error durante el proceso. Esto facilita la depuración y el monitoreo del funcionamiento de la API.
2. **Visualización de datos almacenados:** Se definió otra ruta (/ver\_datos) que permite recuperar y visualizar los datos almacenados en Firebase. La información se convierte a un formato de lista para su posterior uso o visualización.
3. **Integración con una interfaz web:** Se creó una ruta principal (/) que carga una página HTML. Esta página servirá como interfaz gráfica para visualizar y gestionar los datos enviados por los sensores.

Finalmente, el servidor se ejecuta en modo debug para facilitar las pruebas y ajustes durante el desarrollo. Esta implementación asegura una comunicación robusta y en tiempo real entre los sensores simulados y la base de datos en Firebase.

## 4.0.4. Interfaz Gráfica para Visualización de Datos

Este código HTML, junto con estilos CSS y scripts JavaScript, implementa una interfaz gráfica interactiva para visualizar y gestionar los datos enviados por los sensores. A continuación, se detalla la estructura y funcionalidad de cada sección del código:

1. **Encabezado HTML (<head>)**:
   * Define la codificación de caracteres (UTF-8) y asegura la adaptabilidad a dispositivos móviles mediante la etiqueta viewport.
   * Incluye las bibliotecas externas de

Chart.js para gráficos y Bootstrap para el diseño responsivo y estilización.

* + Agrega estilos personalizados para mejorar la estética del cuerpo de la página.

1. **Cuerpo de la página (<body>)**: Contiene una estructura organizada en una sección principal dentro de un contenedor (<div class=çontainer») que incluye:
   * **Título principal:** Un encabezado centrado (<h1>) para el título "Datos de Sensores".
   * **Botones de acción:**
     + Botón para actualizar manualmente los datos de sensores.
     + Botón para exportar los datos a un archivo CSV.
   * **Filtros por fecha:** Campos de entrada de tipo date para filtrar datos entre dos fechas.
   * **Indicador de carga:** Un spinner que se muestra mientras se cargan los datos desde el servidor.
   * **Gráfica de niveles:** Una representación visual en forma de gráfico de barras para mostrar los niveles de agua de cada sensor.
   * **Tabla de datos:** Una tabla interactiva que muestra los niveles de agua tabulados y permite la ordenación por columnas.
2. **JavaScript para interactividad**:
   * **Funciones principales:**
     + cargarDatos: Recupera los datos de los sensores desde el servidor, actualiza la tabla y la gráfica.
     + actualizarGrafica: Redibuja el gráfico de barras basado en los niveles actuales de los sensores, utilizando colores para indicar diferentes rangos.
     + ordenarTabla: Permite ordenar la tabla de datos por las columnas seleccionadas.
     + filtrarPorFecha: Filtra los datos según un rango de fechas definido.
     + exportarDatos: Genera un archivo CSV con los datos de los sensores para exportación.
     + actualizarDatosManualmente: Refresca manualmente los datos cargados en la interfaz.
   * **Interacción inicial:** Al cargar la página, se ejecuta la función cargarDatos, que recupera la información desde el servidor y la muestra en la tabla y gráfica.

Este código proporciona una experiencia interactiva y visual para el monitoreo de los datos enviados por los sensores, combinando funcionalidad dinámica con un diseño atractivo y responsivo.

## 4.0.5. Documentacion

La documentación del proyecto se elaboró cuidadosamente para garantizar que los usuarios y desarrolladores puedan comprender y utilizar el sistema de manera efectiva. Se generaron los siguientes archivos clave:

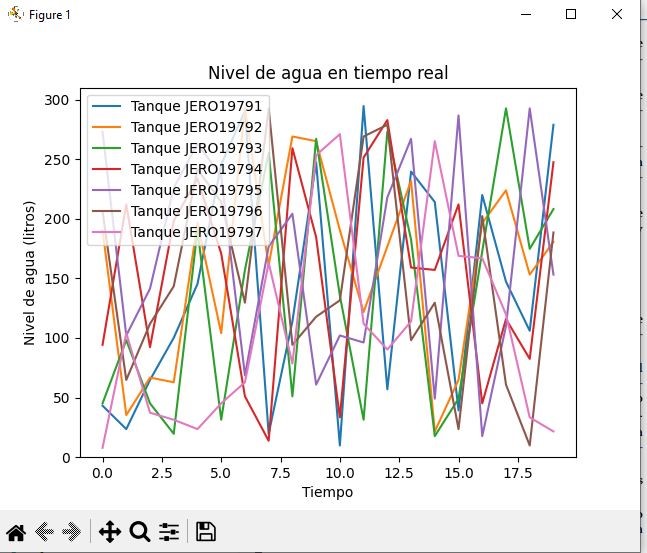
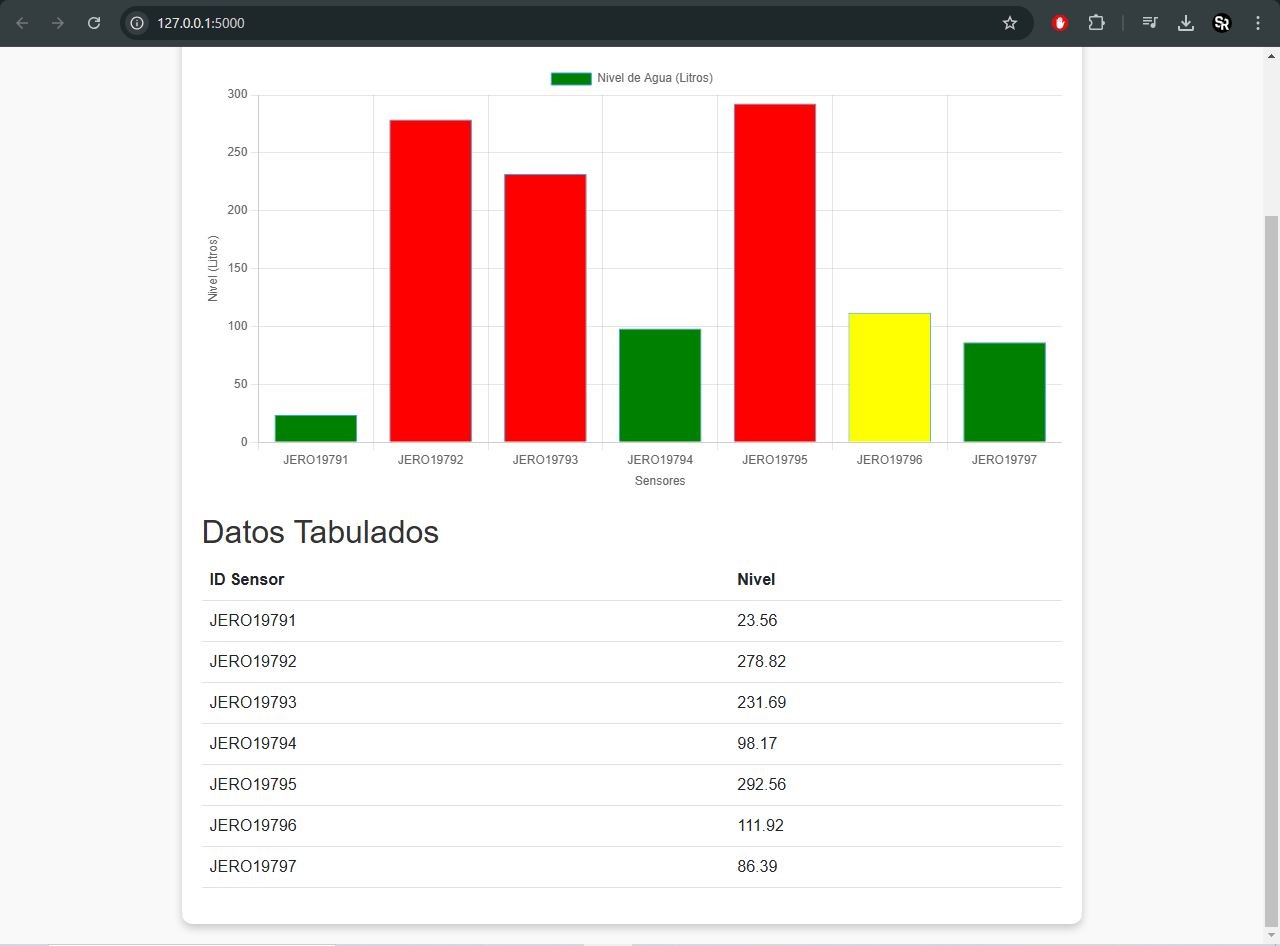
1. **manual\_usuario.md:** Este archivo contiene una guía detallada para el usuario final. Incluye instrucciones paso a paso sobre cómo instalar, configurar y operar el sistema. También describe las funcionalidades principales, como la visualización de datos, exportación a CSV y filtrado por fecha, además de soluciones a problemas comunes.
2. **README.md:** Este archivo actúa como una introducción al proyecto. Incluye una descripción general del propósito del sistema, las tecnologías utilizadas (como Flask, Firebase, y Chart.js) y un resumen de las características implementadas. También proporciona instrucciones rápidas para la instalación, ejecución y pruebas del sistema.
3. **requirements.txt:** Este archivo lista las dependencias necesarias para ejecutar el proyecto, incluyendo las versiones específicas de las bibliotecas requeridas. Permite a los usuarios instalar todas las dependencias de manera eficiente utilizando pip con el comando: pip install -r requirements.txt

Estos archivos aseguran que tanto los usuarios como los desarrolladores cuenten con la información necesaria para instalar, configurar, operar y mantener el sistema, facilitando su adopción y comprensión.

# 5. Resultados y evaluación

El desarrollo del sistema concluyó con una página web funcional que permite la visualización de datos en tiempo real y el historial de registros de los sensores. A continuación, se presentan los resultados obtenidos:

1. **Interfaz Principal:** La página web se diseñó con un enfoque en la usabilidad y claridad. La interfaz principal, como se muestra en la Figura 11, incluye herramientas para la visualización gráfica de los datos, junto con botones para actualizar la información y exportar registros. Además, se incorporaron filtros de fecha para realizar consultas específicas.



**Figura 13**

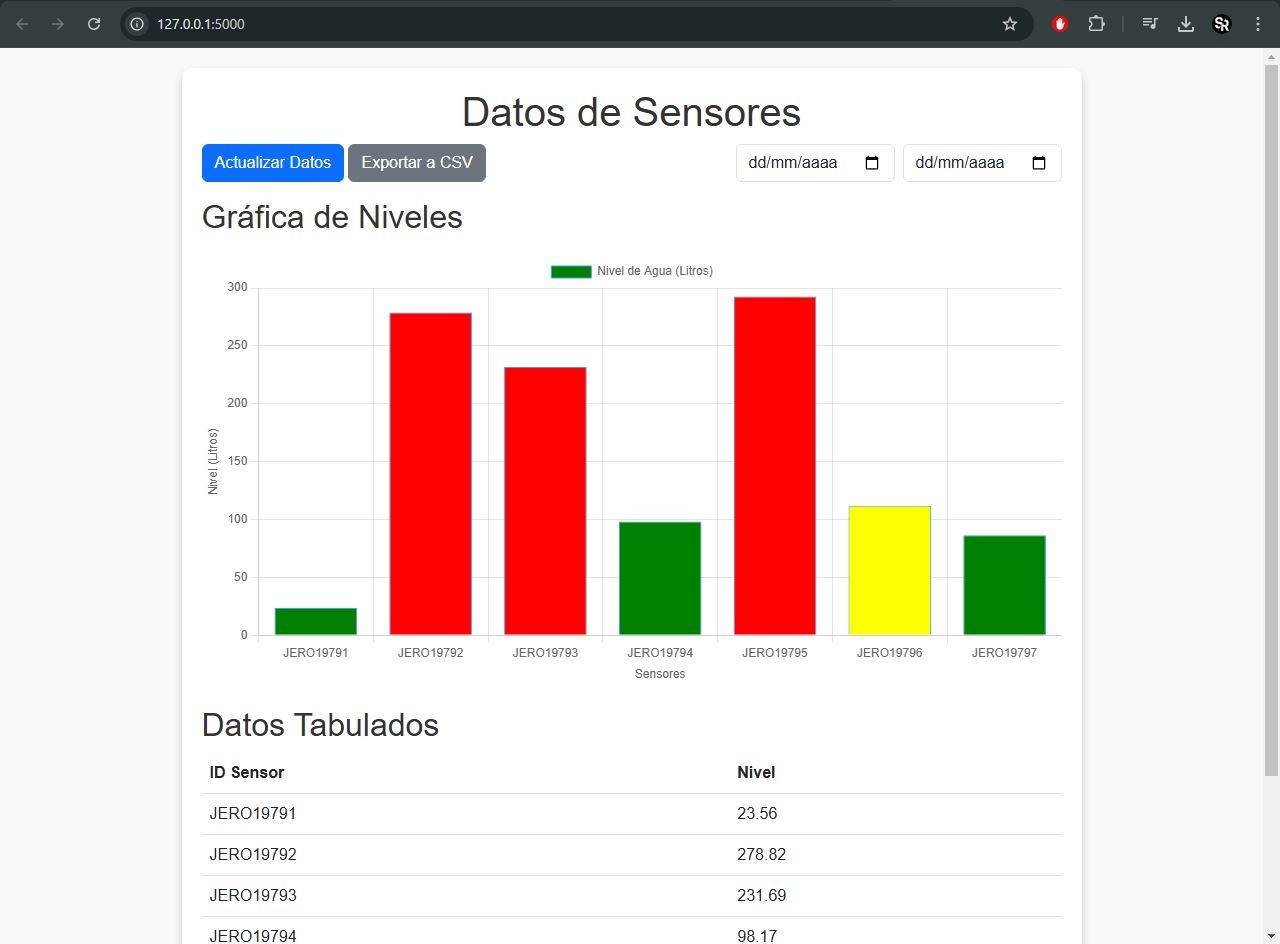
Gráfica en tiempo real - Da-

**Figura 11** Página web - Parte superior.

1. **Tabla de Historial de Datos:** En la parte inferior de la página web, se incluyó una tabla que permite visualizar el historial completo de datos enviados por los sensores. Esta tabla es interactiva, con opciones de ordenación por columnas y filtrado según los criterios seleccionados, como se muestra en la Figura 12.
2. **Gráfica en Tiempo Real:** Además de la interfaz web, se generaron gráficos en tiempo real utilizando la biblioteca matplotlib para analizar los datos simulados por los sensores. Esto permitió verificar la consistencia de los valores generados y su visualización en un entorno gráfico, como se ilustra en la Figura 13.

tos simulados.

Estos resultados demuestran el funcionamiento correcto del sistema, desde la simulación de datos hasta su visualización gráfica y tabular, cumpliendo con los objetivos planteados al inicio del proyecto.



**Figura 12** Página web - Tabla de datos históricos.