UNIVERSIDAD DE TARAPACÁ

FACULTAD DE INGENIERÍA

Departamento de Ingeniería en Computación e Informática

Monitoreo y Control Automatizado de la Alimentación para las Gallinas “Chicken Check”

Autor(es): César Jiménez
Ignacio Garrido
Andrea Navia

Asignatura: Proyecto II
Profesor(es): Diego Aracena Pizarro

ARICA, 03 de Diciembre del 2024

Historial de Cambios

Fecha	Versión	Descripción	Autor(es)
10/09/2024	1.0	Plantear problemática	César Jiménez Ignacio Garrido Andrea Navia
24/09/2024	1.1	Desarrollo del informe	César Jiménez Ignacio Garrido Andrea Navia
30/09/2024	1.2	Finalización de informe/presentación	César Jiménez Ignacio Garrido Andrea Navia
24/10/2024	1.3	Desarrollo C.U.S y requerimientos	César Jiménez Ignacio Garrido Andrea Navia
04/11/2024	1.4	Finalización de informe 2 /presentación 2	César Jiménez Ignacio Garrido Andrea Navia
22/11/2024	1.5	Desarrollo Implementación	César Jiménez Ignacio Garrido Andrea Navia
03/12/2024	1.6	Finalización de informe 3 y producto final	César Jiménez Ignacio Garrido Andrea Navia

Índice de ilustraciones

Índice de tablas

Tabla de contenidos

1. Panorama General

1.1 Introducción

Los gallineros son estructuras diseñadas para albergar gallinas proporcionando un entorno seguro y adecuado para su cría y bienestar.
Una de las funcionalidades que tiene un gallinero es proteger a las aves de depredadores y condiciones climáticas adversas, al tiempo que facilita la recolección de huevos y el manejo general del ganado avícola. Además, los gallineros permiten una gestión eficiente de la alimentación y el agua de parte del propietario.
Estos espacios no solo son esenciales para asegurar la salud y productividad de las aves, sino que también son fundamentales para quienes buscan producir huevos y carne de manera sostenible y eficiente.

1.1.1 Propósito

En la actualidad, la gestión de gallineros es un desafío que requiere atención constante por parte de los propietarios, especialmente en lo que respecta a la alimentación y el suministro de agua para las gallinas. La mayoría de los gallineros tradicionales dependen de la supervisión manual, lo que puede resultar ineficiente y provocar situaciones en las que las gallinas no reciben la comida o el agua necesarias en el momento adecuado. Esto no solo puede afectar la salud de las gallinas, sino también reducir la productividad del gallinero.

1.1.2 Alcance

El proyecto monitoreo automatizado de alimentación y control de alimento de las gallinas pretende mantener el control sobre la constante alimentación de los animales tanto de agua como de alimento, uno de los objetivos principales es que se le abastezca periódicamente de comida en el día según la cantidad de gallinas que hayan en ese momento, se va a contemplar equilibrar la cantidad de comida en el dia por medio de un sensor de peso un servomotor que la va a dispensar, en cuanto al agua, se pretende ir dando constantemente agua con un sensor ultrasónico puede ver el nivel de altura del agua con la cual puede saber si es necesario rellenar el recipiente de agua y tambien esta agua será desechada si se da el caso que con el sensor de Ph se detecte que está insalubre para su consumo y será reabastecida.

Alcance de Proyecto

- Automatizar Dispensador de Alimentación: Estos dispensadores serán automatizados por medio de dispositivos que serán controlados por medio de un raspberry PI que se encargará de proporcionar tanto agua como alimento a la gallina según la calidad del agua, nivel del agua, cantidad de alimento, etc.

- Sensores de detección: usa 2 sensores cuyo propósito es detectar el nivel del agua y la cantidad de comida que contienen los recipientes en ese momento y así informar a los diferentes dispositivos sobre el estado de estos.
- Implementación de notificaciones: una de las principales funcionalidades es mantener un constante monitoreo sobre la alimentación de la gallina y notificar al usuario sobre la cantidad de alimento y agua hay en ese momento y también cuánto será proporcionado.
- Monitoreo remoto: se podrá ver el estado del agua como del alimento por medio de la aplicación móvil conectada al sistema de control del gallinero (Raspberry PI) con una interfaz gráfica intuitiva y amigable.
- Control de Calidad del Agua: busca dar un buen servicio, por lo tanto, debe brindar agua salubre y para esto implementaremos un sensor de Ph el cual va a velar porque el agua se mantenga limpia y que cuando pase un umbral de Ph se drene esta agua para reabastecerse.

Límite de Proyecto

- Esté proyecto no contempla la salud física de las gallinas, solo se encarga de lo principal, que es la alimentación y calibrar este servicio.
- Este proyecto se somete netamente a brindar agua y alimento, y no a otras áreas como pueden ser a la recolección de huevos , desechos de las gallinas o mantener limpio el gallinero.
- Este proyecto no contiene un control de temperatura o en sí el manejo de condiciones de climatización dentro del gallinero.
- Este proyecto no tiene como objetivo abastecer a un gran número de gallinas debido al tamaño de los elementos que la componen.

1.1.3 Objetivos

1.1.3.1 General

El proyecto consiste en un sistema automatizado que controla el suministro de comida y agua para las gallinas de forma remota. Asegura la cantidad adecuada de alimento diario y mantiene el agua limpia mediante un sistema de abastecimiento y desagüe continuo.

1.1.3.2 Especificos

1) Analizar los patrones de alimentación de las gallinas para ajustar la cantidad y frecuencia del suministro.
2) Definir la problemática a resolver sobre las necesidades de las gallinas.
3) Seleccionar dispositivos de automatización (como Raspberry Pi y sensores) para controlar el suministro automático de alimentos y agua.
4) Establecer un presupuesto detallado para cada componente y dispositivo.
5) Documentar y registrar toda la información del proyecto durante el semestre.
6) Investigar las funcionalidades del Raspberry Pi 4 e instalar su sistema operativo.
7) Desarrollar una aplicación móvil para gestionar el dispensador de alimento y agua de forma remota.
8) Establecer la conexión entre la aplicación y el servidor para acceder a los servicios.
9) Implementar el sistema de alimentación con servomotor y sensor de peso para controlar el alimento.
10) Implementar el sistema de agua con válvula solenoide y sensor ultrasónico para controlar el acceso y nivel de agua.
11) Implementar el sistema de drenaje con sensor de pH y servomotor para controlar la calidad del agua y drenar si es necesario.

1.1.4 Suposiciones y Restricciones

1.1.4.1 Suposiciones

- Aplicación para monitorear el nivel del agua y la comida.
- Abastecer de agua periódicamente de forma automática.
- Mantener el agua limpia con sensor de Ph y desagüe.
- Equilibrar la cantidad de comida al día.
- Mantener el control del peso de la comida
- Existe compatibilidad entre los dispositivos/equipos necesarios para elaborar el proyecto.
- Tener los materiales necesarios para elaborar el prototipo a gran escala.
- Tener software disponible de forma gratuita para utilizar nuestros equipos.
- El dispensador está diseñado de manera intuitiva y amigable para el usuario.

1.1.4.2 Restricciones

- Tener los recursos económicos para adquirir los equipos necesarios.
- Tiempo límite destinado para elaborar el proyecto.
- Conocimiento sobre los diferentes equipos.
- Contar con la infraestructura necesaria para desarrollar el proyecto de forma colaborativa.
- Contar con la participación completa del grupo de trabajo.
- Compatibilidad sobre los diferentes equipos necesarios para el proyecto.

1.1.5 Entregables del Proyecto

- Informe 1
- Maqueta 1
- Presentación 1
- Wiki
- Bitácoras
- Carta Gantt

2 Organización del Proyecto

2.1 Personal y entidades internas

- Documentador: Ignacio Garrido. Es el encargado de documentar toda la información en diferentes formatos a lo largo del proyecto y también se encargará de que esta esté actualizada, ya sean estos documentos tanto escritos como digitales.
- Diseñador: Andrea Navia. El diseñador en un proyecto tiene un rol clave en la creación de la apariencia y la funcionalidad visual de un producto, servicio o sistema. Su objetivo principal es garantizar que el diseño sea atractivo, funcional y alineado con los objetivos del proyecto y las necesidades del usuario.
- Analista Programador: Cesar Jimenez. Es el encargado de realizar todos los algoritmos funcionales para los diferentes equipos requeridos en el proyecto, en sí, es la implementación de código para automatizar las tareas y acciones realizadas por los diferentes equipos.
- Jefe de Proyecto: Andrea Navia. Es el encargado de organizar, administrar y liderar el grupo de trabajo. Este rol es el encargado de supervisar y apoyar a cada uno de los integrantes del equipo para lograr sus pequeñas metas diarias y lograr el objetivo general del proyecto.

2.2 Roles y Responsabilidades

- Documentador: Ignacio Garrido
- Diseñador: Andrea Navia
- Analista Programador: César Jiménez
- Jefe de Proyecto: Andrea Navia

2.3 Mecanismos de Comunicación

- Discord
- Whatsapp
- Gmail

3 Planificación de los procesos de gestión

3.1 Planificación inicial del Proyecto

3.1.1 Planificación de estimaciones

Costos de HARDWARE

Productos	Cantidad	Costo
Notebook	3	$ 800.000
Materiales Maqueta	1	$25.000
Sensores	6	$50.000
Raspberry pi	1	$100.000
Micro SD (8GB)	1	$5.000
	Costo Total	$980.000

Tabla 1: Costos de Hardware

Costos Software

Producto	Costo
Visual Studio Code	$0
Discord	$0
Canva	$0
Whatsapp	$0
Phyton	$0
Costo Total	$0

Tabla 2: Costos de Software

3.1.2 Planificación de Recursos Humanos

Cargo	Personas	Valor hora	Horas mensual	Horas extras	Horas totales	Sueldo Mensual	Sueldo total (4 meses)
Programador y analista	3	$10.000	18	15	33	$990.000	$3.960.000
Diseñador gráfico	1	$7.000	18	10	28	$196.000	$784.000
Documentador	1	$6.000	18	10	28	$168.000	$672.000
Jefe de Proyecto	1	$12.000	18	15	33	$396.000	$1.584.000
Total					**Costo**	**$1.750.000**	**$7.000.000**

Tabla 3: Planificación recursos humanos

Costo total del proyecto:

Costo humano($7.000.000) + Costo Productos($980.000) = $7.980.000

3.2 Lista de actividades

3.2.1 Actividades de trabajo

Actividad	Descripción	Responsable
Asignación de roles	Asignación de roles dentro del equipo.	Andrea Navia
Búsqueda de ideas	Búsqueda de ideas de problemática y su solución.	Andrea Navia
Construcción de Maqueta	Creación de maqueta del proyecto	Andrea Navia/ Ignacio Garrido / Cesar Jimenez
Construcción de modelo 3D	Creación de modelo 3D del proyecto	Cesar Jimenez
Redacción de bitácoras	Registro de actividades que se realizan semanalmente.	Ignacio Garrido
Redacción de Carta Gantt	Planificación de actividades a realizar durante el semestre.	Ignacio Garrido
Wiki	Se comparte información del proyecto.	Andrea Navia
Informe I	Escritura del primer informe.	Cesar Jimenez/ Andrea Navia / Ignacio Garrido
Presentación I	Diseño de la primera presentación.	Andrea Navia/ Ignacio Garrido
Estudiar Sensores	Analizar las mejores opciones.	Cesar Jimenez
Estudiar cómo Utilizar Raspberry pi	Aprender a utilizarlo	Cesar Jimenez

Tabla 4: Actividades de trabajo

3.2.2 Asignación de tiempo

Se ha elaborado una Carta Gantt para estimar el tiempo que se emplea en cada actividad del proyecto y organizar el tiempo entre actividades de forma más eficiente.

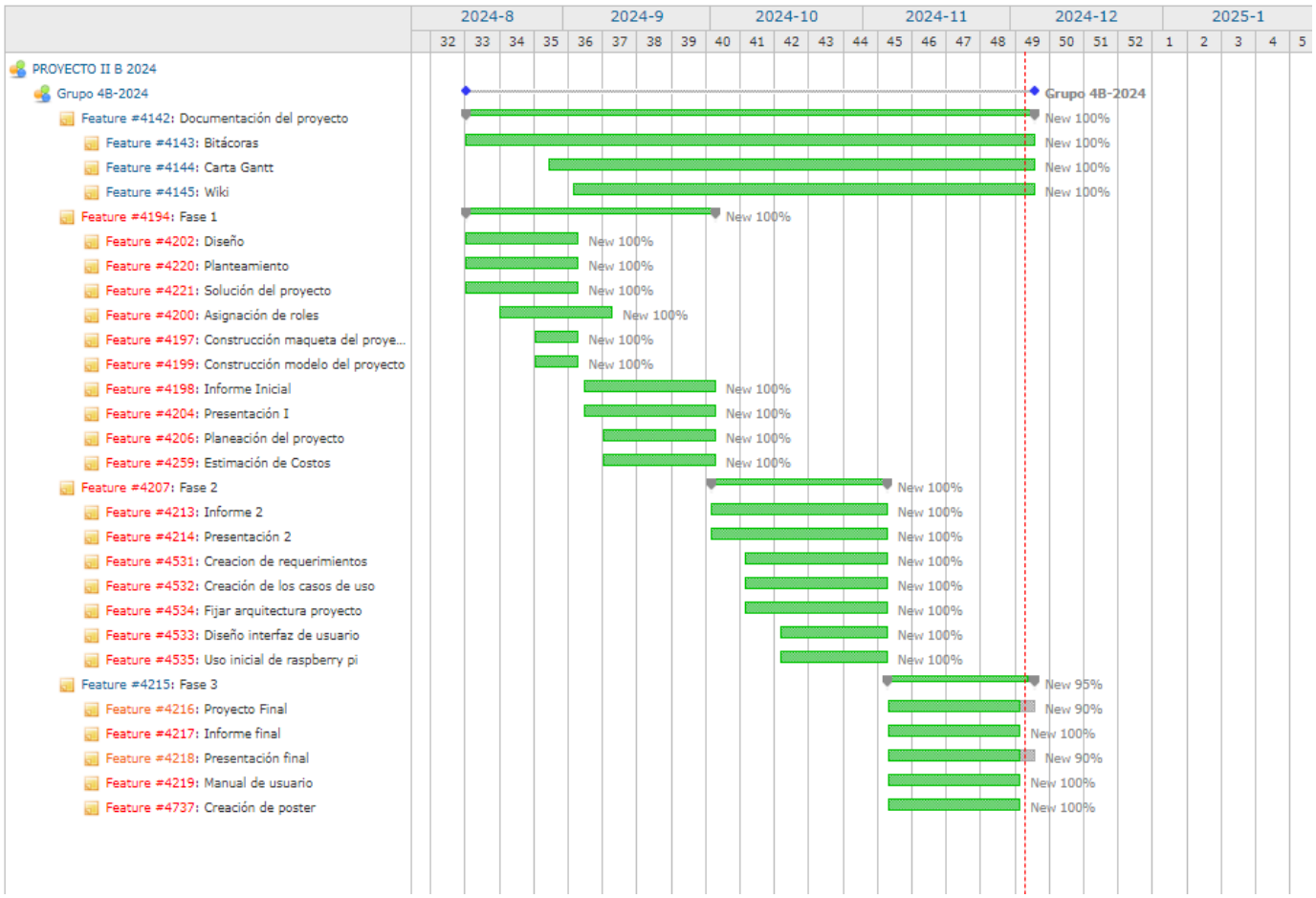

Ilustración 1: Carta Gantt.

3.3 Planificación de la gestión de riesgos

A continuación se presenta la tabla con los riesgos que puede tenerse al transcurso del proyecto, con las siguientes niveles de riesgo:

1. Catastrófico
2. Crítico
3. Marginal
4. Despreciable

Riesgos	Probabilidad de ocurrencia	Nivel de impacto	Acción remedial
Falta de sensores u otros equipos.	80%	3	Buscar la pieza faltante o reemplazarla por una similar.
Ausencia del personal.	10%	2	Reorganizar al equipo para avanzar en la tarea del personal ausente.
Incumplimiento de tareas.	40%	2	Reasignación de roles para encontrar el más óptimo.
Reconstrucción total del proyecto por no cumplir lo requerido.	20%	1	Buscar nuevas ideas que cumplan con lo pedido y llevarlas a cabo.
Problemas de Hardware en Raspberry-Pi3	20%	1	Solicitar reemplazo del hardware.
Precios de los equipos tecnológicos.	10%	3	Cotizar diferentes equipos para adquirir el más económico.
Desgaste y/o mal funcionamiento de motores.	15%	3	Reemplazar la pieza en mal estado.
Exposición de los equipos a condiciones del medio adversas.	10%	3	Proteger los equipos tecnológicos en el ámbito físico para evitar las averías u fallas en su funcionamiento.

Información disponible sobre los equipos tecnológicos.	30%	2	Solicitar ayuda a diferentes profesionales sobre las funcionalidades y riesgos de los equipos.
Falta de decisión del cliente	90%	1	Realizar sugerencias para optimizar recursos y tiempo sobre las mejores opciones para su producto.
Obsolescencia programada en hardware y software	50%	2	Implementar un plan de actualización continua para evitar problemas en el producto.
Cortes repentinos de luz	60%	1	Utilizar alternativas para energizar los componentes como la energía solar.
Falta de experiencia en el equipo	20%	2	Proporcionar capacitación inicial y contar con asesoría técnica en caso de dudas.
Enfermedades o problemas de salud en el equipo	25%	2	Tener un plan de contingencia que incluya redistribución temporal de tareas para mantener el flujo de trabajo en caso de ausencias por salud.

Tabla 5: Planificación de la gestión de riesgos

4. Planificación de los procesos técnicos

4.1 Modelo de proceso

4.1.1 Especificación de Requerimientos

Requerimientos funcionales:

- El sistema debe ser capaz de dispensar alimento y agua de forma periódica.
- El sistema debe ser capaz de dispensar alimento según la cantidad de alimento en la balanza.
- El sistema debe controlar el nivel de agua con un sensor ultrasónico
- El sistema abastecer agua con la válvula solenoide
- El sistema debe drenar el agua si el sensor de pH detecta niveles de agua insalubres.
- Los usuarios deben poder monitorear los niveles de alimento y agua.
- Los usuarios recibirán notificaciones a través de una aplicación móvil.
- El sistema debe notificar los niveles de alimento y agua.
- El sistema debe funcionar correctamente con los dispositivos de hardware utilizados.

Requerimientos no funcionales:

- El sistema debe asegurar un suministro continuo de agua y alimento.
- El sistema puede ser escalable para fines industriales.
- La aplicación móvil debe ser intuitiva y amigable para el usuario.
- El proyecto debe utilizar software gratuito (como Visual Studio Code, Canva, etc.).
- El sistema debe garantizar la integridad de los datos procesados.
- El proyecto debe mantener un balance en costos tanto de hardware como de software relacionado al desarrollo del mismo.

- El proyecto debe estar documentado en los distintos entregables.

4.1.2 Descripción de la Arquitectura

4.1.2.1 de Hardware

Esta arquitectura representa como van a ir enlazados entre sí cada uno de los componentes que se utilizaran para poder desarrollar el proyecto de monitoreo y control de la alimentación para las gallinas.

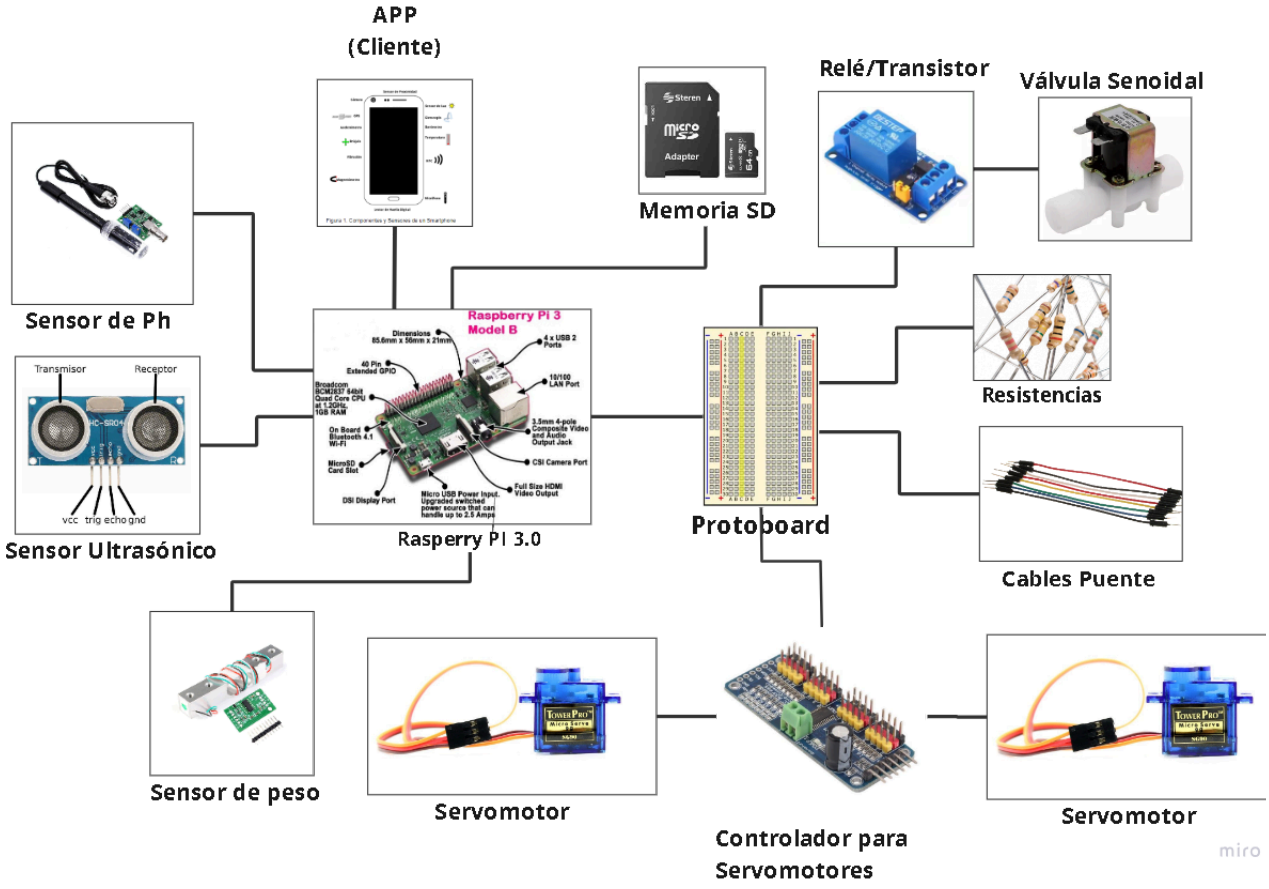

Ilustración 2: Descripción Arquitectura

4.1.2.2 de Conexiones

Esta arquitectura representa como van a ir conectados entre sí cada uno de los componentes que se utilizaran para poder desarrollar el proyecto de monitoreo y control de la alimentación para las gallinas.

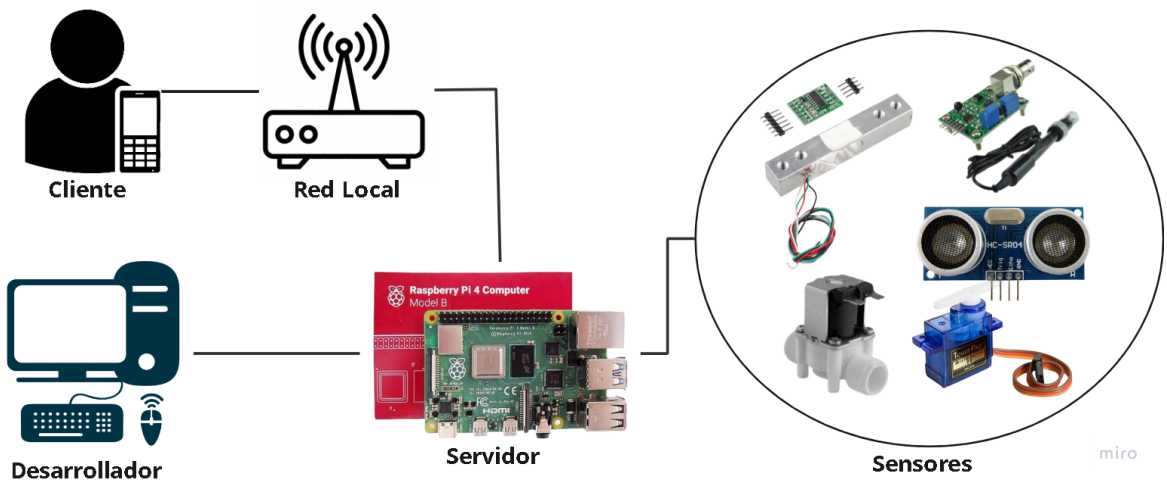

Ilustración 3: Descripción Arquitectura 2

Cliente: es principalmente la aplicación con la cual el cliente va a poder acceder a las principales funciones del proyecto que es administrar y gestionar de forma adecuada cada uno de los sensores que componen el proyecto de un dispensador automatizado de comida y agua.

Red local: es el dispositivo que nos va a proporcionar una conexión en red inalámbrica entre los dispositivos, como: Aplicación móvil y Raspberry Pi 4 Model B. Para establecer una comunicación activa de envío y recepción de datos entre el cliente y el servidor.

Servidor: Está alojado en el dispositivo llamado Raspberry Pi 4 Model B, es el encargado de establecer la intercomunicación entre la aplicación y los sensores con relación a cómo se van a distribuir cada una de las instrucciones indicadas por parte del cliente hacia cada uno de los sensores.

Sensores: son los componentes imprescindibles para la elaboración del proyecto ya que son los que solventan parcialmente la problemática planteada. Deben responder a las instrucciones (datos) que envía el cliente por medio de la aplicación y, además, deben emitir hacia el cliente los diferentes datos recopilados en función de cada sensor.

Desarrollador: es el ente encargado de programar y establecer la conexión

entre los sensores y la aplicación por medio de la implementación de código, y además es quien realiza la administración y mantenimiento de estas conexiones.

4.1.2.3 de Diseño

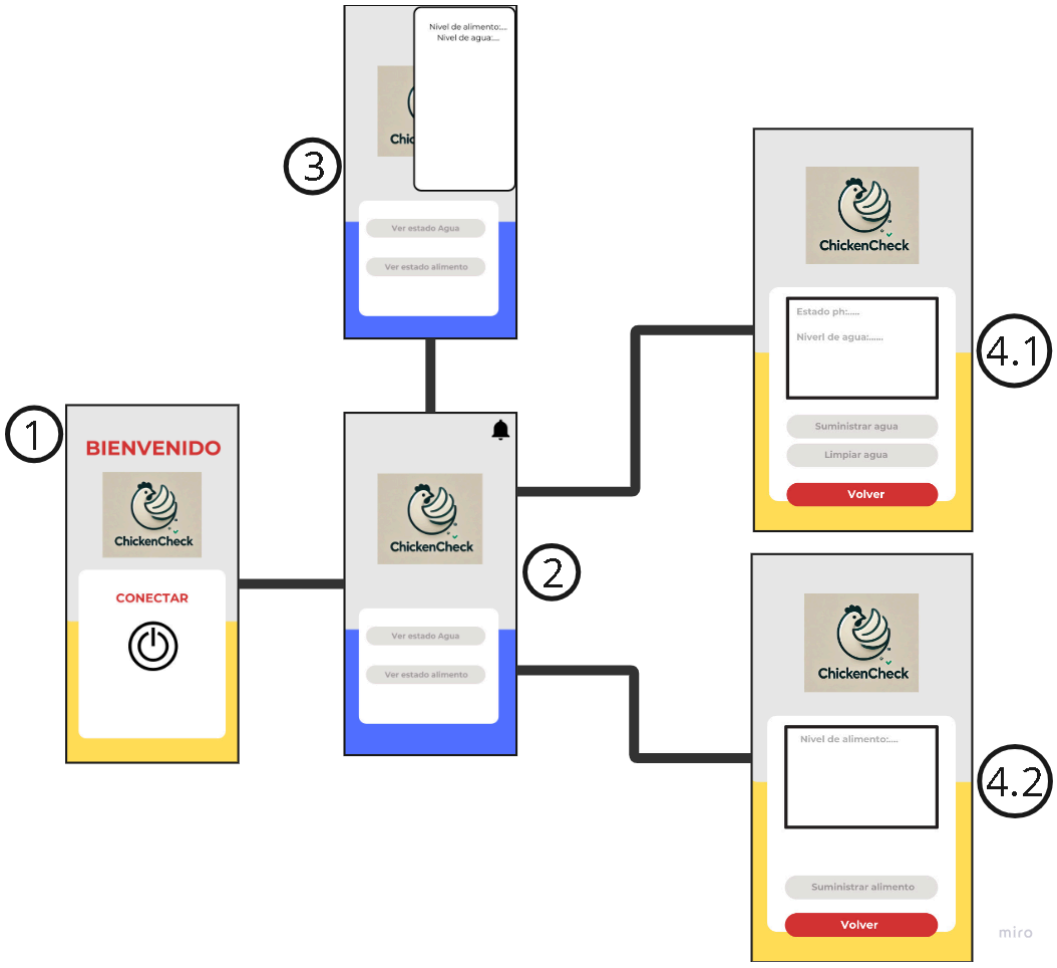

Ilustración 4: Diseño App.

1.- Primera Ventana de la aplicación móvil en la que se destaca un saludo, logo y un botón "conectar"para realizar la conexión del Raspberry PI 4 por medio de su IP e iniciar con la comunicación entre la aplicación y los sensores.
2.- Segunda Ventana es la ventana encargada de derivar a las ventanas operacionales .Aquí encontramos dos botones para ver el estado del agua, alimento, y un botón con un diseño de una campana el cual lleva al usuario a sus notificaciones.
3.- Tercera Ventana es una ventana emergente que se activa al presionar el botón con el diseño de una campana. Esta ventana está destinada a mostrar las notificaciones que emergen según las condiciones de calidad y cantidad del agua y alimento administrado en los recipientes. El botón para activar esta ventana se encuentra ubicado en la parte superior de la ventana 2.
4.1.- Cuarta Ventana operacional de "Ver estado agua" se encuentra un caja de texto donde se visualiza el nivel de agua y el estado de ph actuales, en la parte de abajo se encuentran dos botones rotulados con "Suministrar agua" brindar el agua y "Volver" retornar a la ventana anterior.
4.2.- Cuarta Ventana operacional de "Ver estado alimento" se encuentra un caja de texto donde se visualiza el nivel de alimento actual, en la parte de abajo se encuentran dos botones "Suministrar alimento" para brindar el alimento y "Volver" retornar a la ventana anterior.

4.1.3 Caso de Uso de Contexto

En la fase inicial del diseño del software, se busca definir los elementos principales y las interacciones más importantes del sistema. En este proyecto, se enfoca en crear casos de uso y diagramas sencillos de nivel 0 y 1 para mostrar cómo interactúan los usuarios y el sistema en diferentes situaciones, según los casos de uso previamente definidos.

Estos diagramas nos darán una idea clara de las acciones y respuestas del sistema, ayudando a identificar los puntos clave de interacción y los flujos de información. También usaremos diagramas de clases para mostrar la estructura del sistema, incluyendo sus elementos, sus características y cómo se relacionan entre sí.

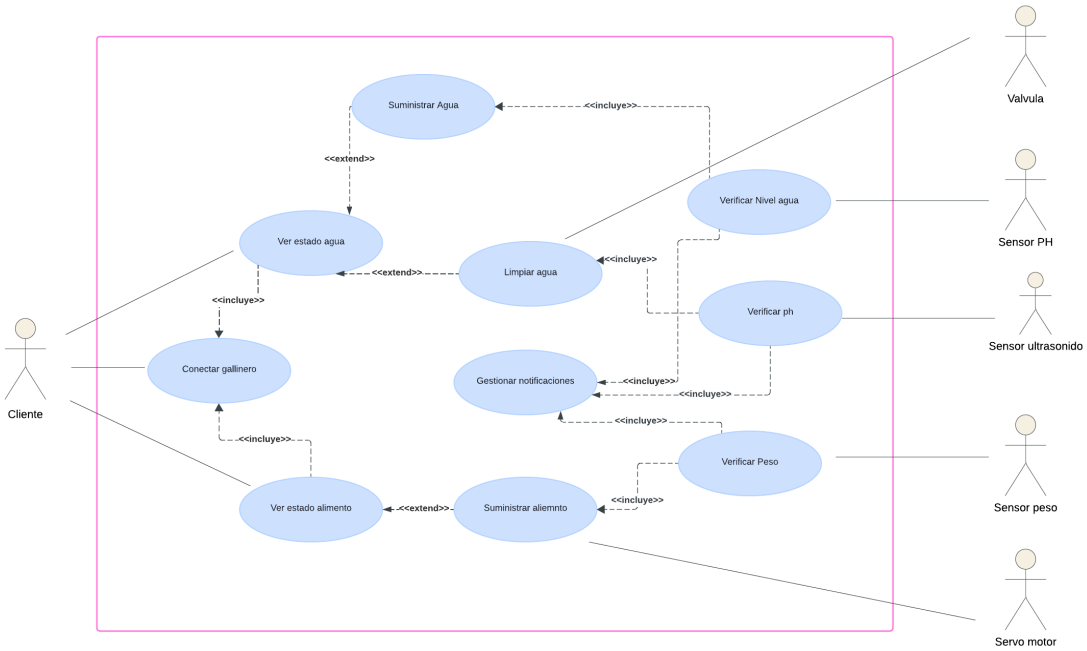

Ilustración 9 : Diagrama Caso de Uso de Contexto

Caso de uso : Conectar Gallinero

Nombre : Conectar gallinero	
Autor/Fecha: Andrea Navia 24-10-2024	
Descripción: Permite al cliente mediante la aplicación conectarse al gallinero correspondiente	
Actor: Cliente	
Precondición: El sistema y la aplicación móvil ambas deben estar conectadas a la misma red wifi	
Flujo Principal: Cliente 2. Selecciona el botón conectar	Flujo Principal: Sistema 1.Muestra el logo, un botón rotulado “conectar” 3. Muestra en la vista conexión un título “conexión exitosa” y cambia a la vista principal
Flujo Alternativo:	Flujo Alternativo: 3.1 Muestra en la vista conexión un título “error de conexión” 3.2 Muestra el logo, un botón rotulado “conectar”
Postcondiciones:	
Valor medible: eficiencia en realizar la coneccion, ahorra tiempo y facilidad para el cliente	

Tabla 6: CUS Conectar gallinero.

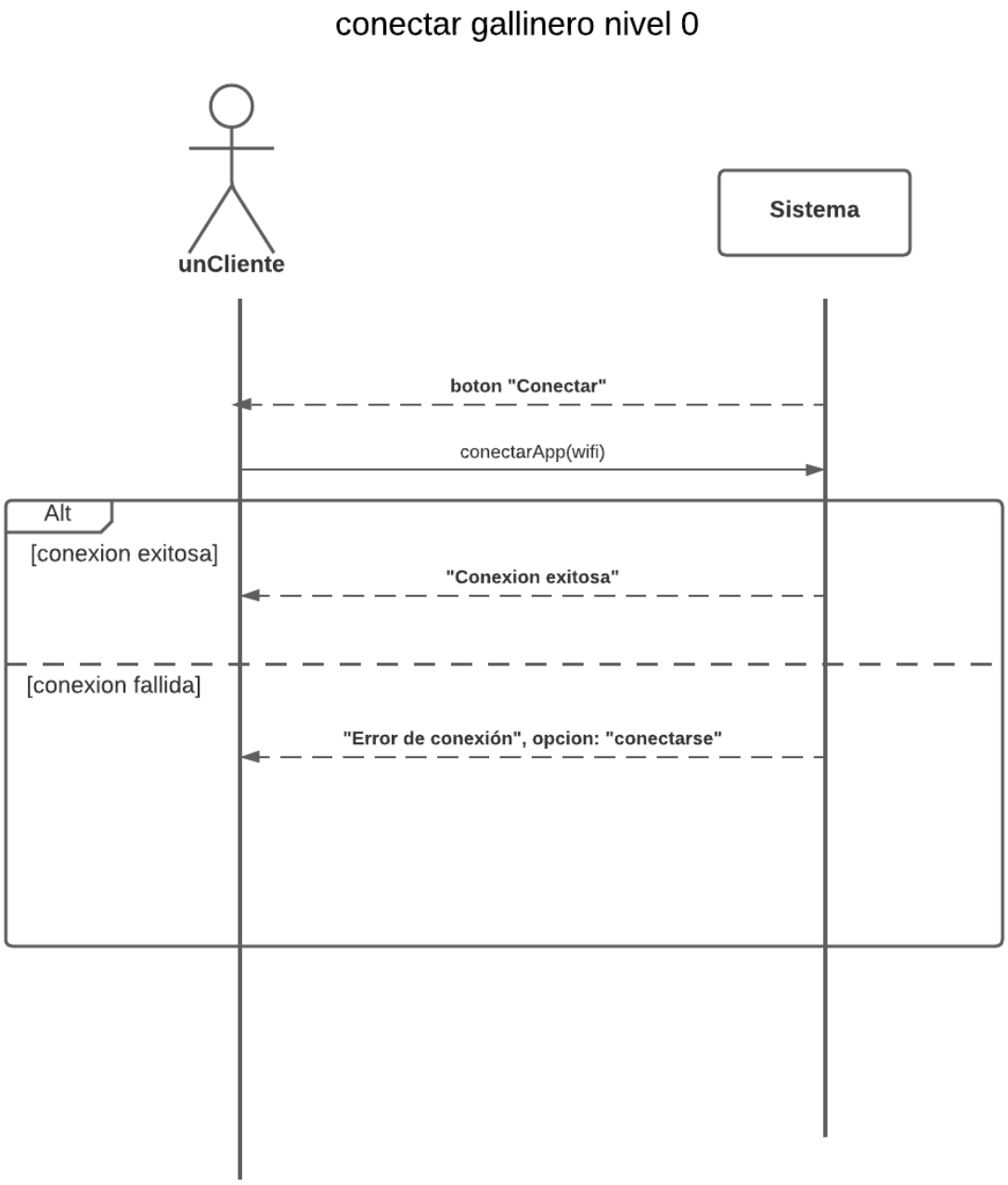

Ilustración 10 : Conectar Gallinero - Diagrama nivel 0

Caso de uso: Gestionar menú principal

Nombre : Gestionar menú principal	
Autor/Fecha: Andrea Navia 24-10-2024	
Descripción: Permite al cliente mediante la aplicación gestionar el gallinero.	
Actor: Cliente	
Precondición:	
Flujo Principal: Cliente 2.Selecciona el botón “Ver estado agua”.	Flujo Principal: Sistema 1.Muestra el título “Chicken check”, con 2 botones rotulados “Ver estado agua” ,“Ver estado alimento” y ”Notificaciones” 3.<<extend>> C.U.S Ver estado agua
Flujo Alternativo: 2.1.Selecciona el botón “Ver estado alimento”.	Flujo Alternativo: 2.2.<<extend>> C.U.S Ver estado alimento
Flujo Alternativo:	Flujo Alternativo: 2.1<<extend>> C.U.S Generar Notificaciones
Postcondiciones:	
Valor medible: eficiencia en gestionar el gallinero, ahorrando tiempo en ir a gestionarlo manualmente.	

Tabla 7: CUS Gestionar menú principal

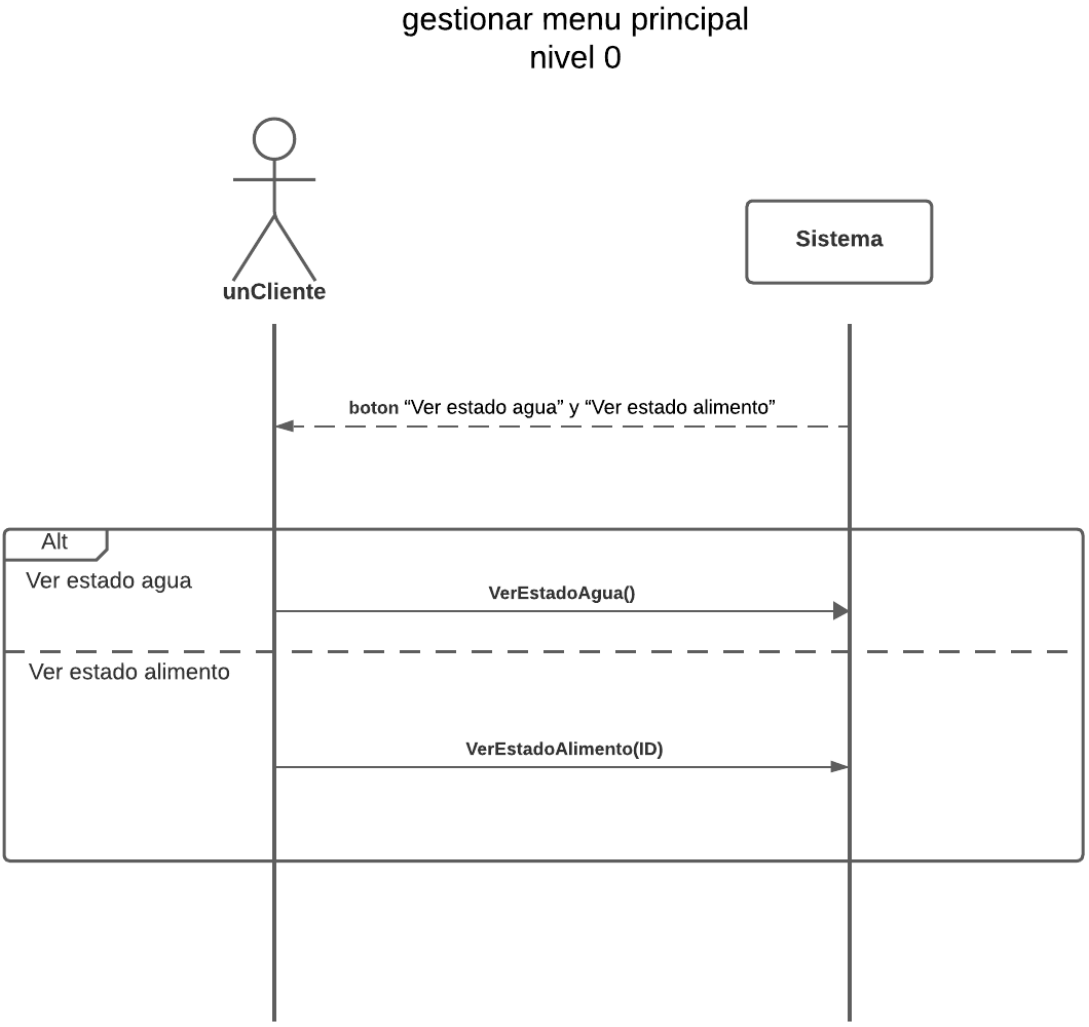

Ilustración 11 : Gestionar menú principal - Diagrama nivel 0

Caso de uso: Ver estado agua

Nombre : Ver estado agua	
Autor/Fecha: Andrea Navia 24-10-2024	
Descripción: Permite al cliente ver y administrar el agua del gallinero	
Actor: Cliente	
Precondición: El sistema contiene información del ph y nivel del agua, además de estar previamente conectado.	
Flujo Principal: Cliente 1.<<incluye>> C.U.S Conectar gallinero 2.Presione el botón suministrar agua	Flujo Principal: Sistema 1.Muestra una vista con los botones suministrar agua y limpiar, además de la información del ph y nivel del agua 3. <<extend>> C.U.S Suministrar Agua
Flujo Alternativo: 2.1.Presiona el botón limpiar agua	Flujo Alternativo: 2.2.<<extend>> C.U.S Limpiar agua
Postcondiciones: Cambia en el sistema el nivel de agua y de ph	
Valor medible: Permite facilidad en ver el estado del gallinero	

Tabla 8: CUS Ver estado agua

ver estado agua nivel 0

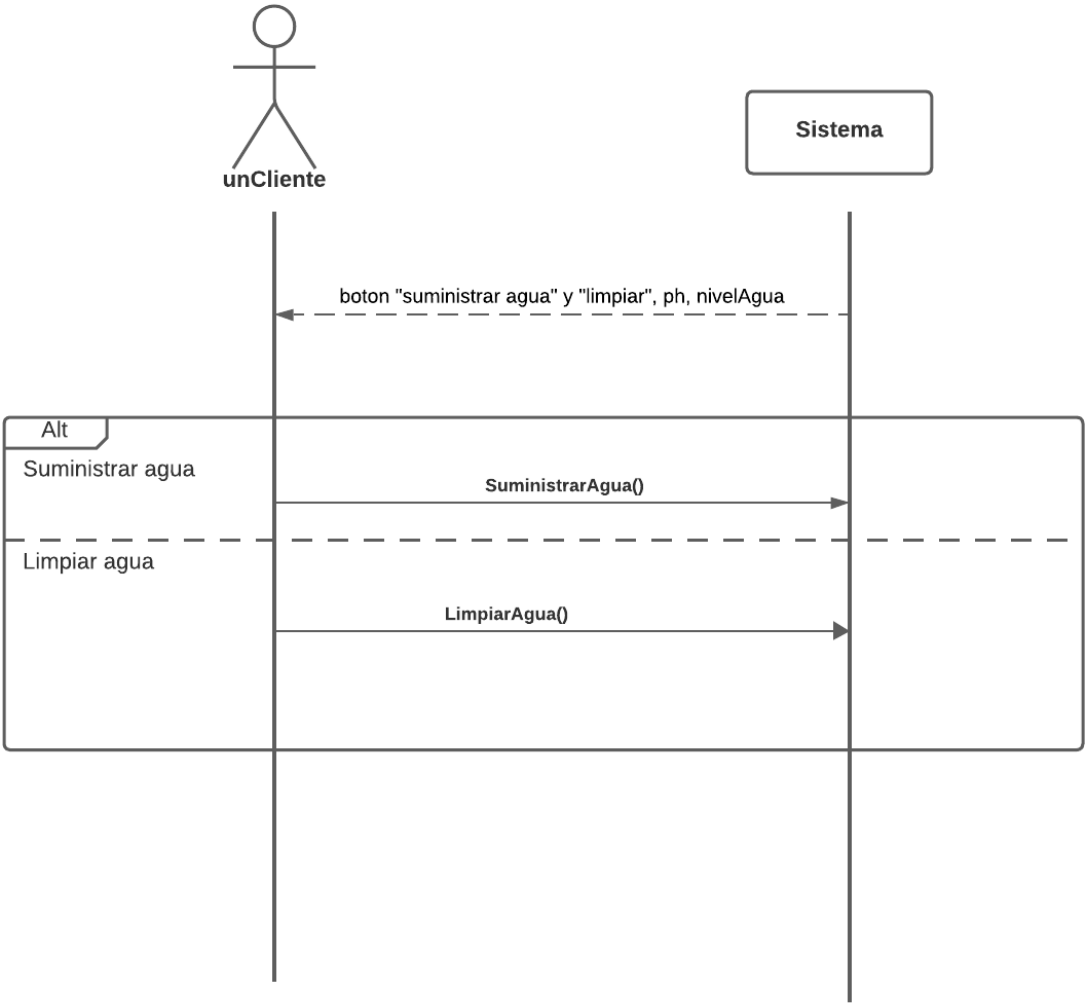

Ilustración 12 : Ver estado agua - Diagrama nivel 0

ver estado agua nivel 1

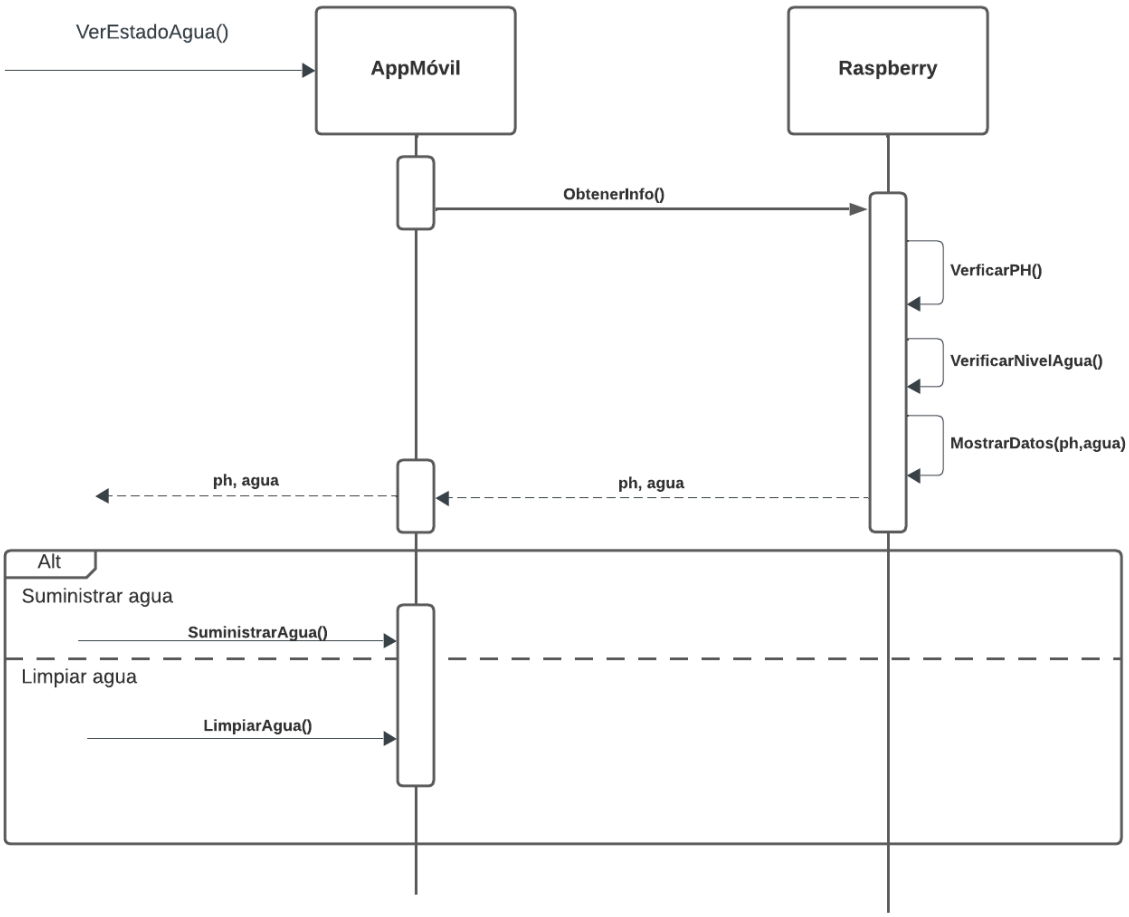

Ilustración 13 : Ver estado agua - Diagrama nivel 1

Caso de uso: Ver estado alimento

Nombre : Ver estado alimento	
Autor/Fecha: Andrea Navia 24-10-2024	
Descripción: Permite al cliente ver y administrar el alimento del gallinero	
Actor: Cliente	
Precondición: El sistema contiene información del nivel de alimento, además de estar previamente conectado	
Flujo Principal: Cliente 1.<<incluye>> C.U.S Conectar gallinero 2. Selecciona el botón suministrar alimento	Flujo Principal: Sistema 1.Muestra una vista con los botones suministrar alimento y la información del nivel de alimento 3. <<extend>> C.U.S Suministrar Alimento
Flujo Alternativo:	Flujo Alternativo:
Postcondiciones: El sistema actualiza el nivel de alimento	
Valor medible: Se gestiona de mejor manera la información del alimento del gallinero.	

Tabla 9: CUS Ver estado alimento

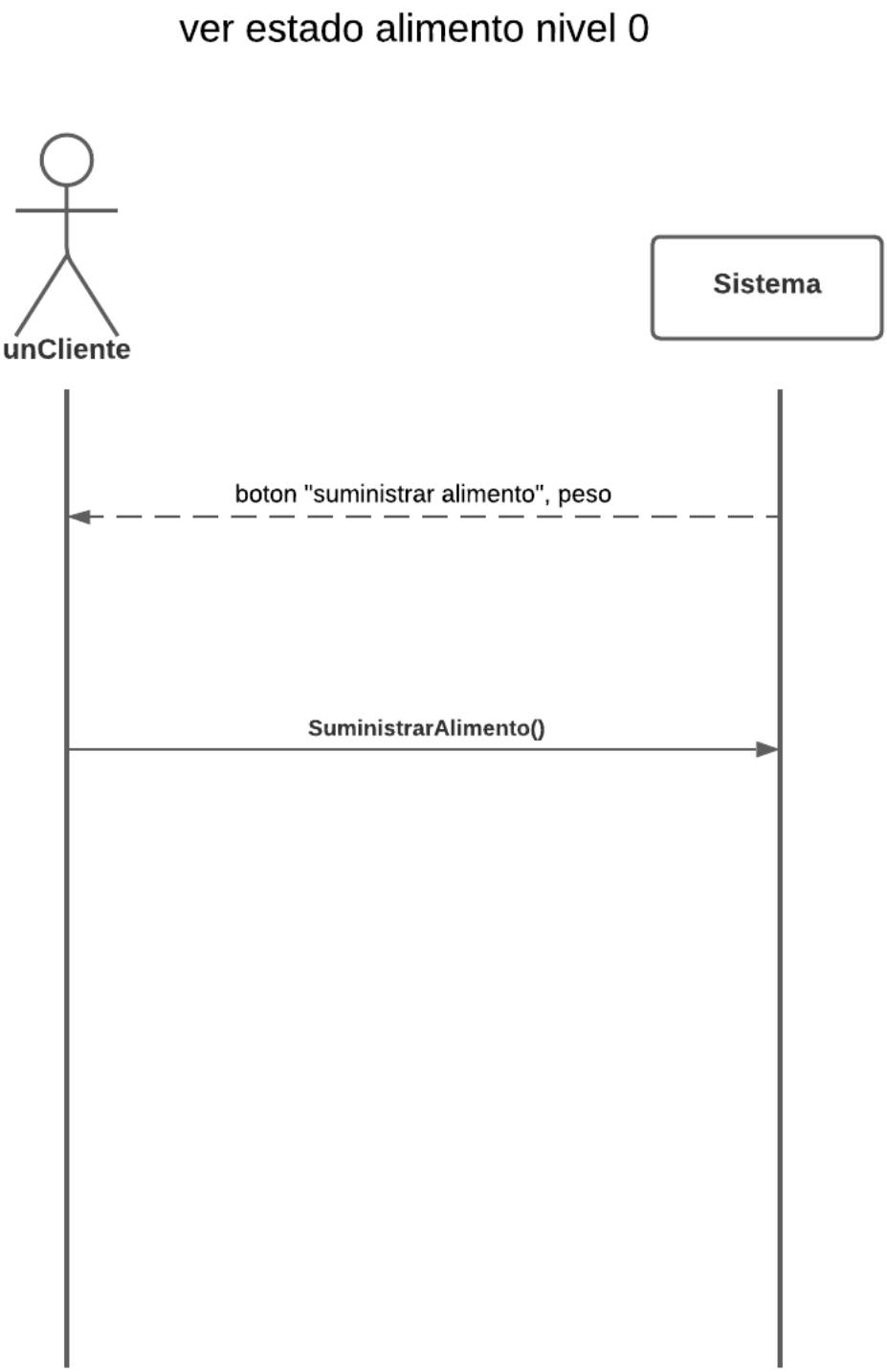

Ilustración 14 : Ver estado alimento - Diagrama nivel 0

ver estado alimento nivel 1

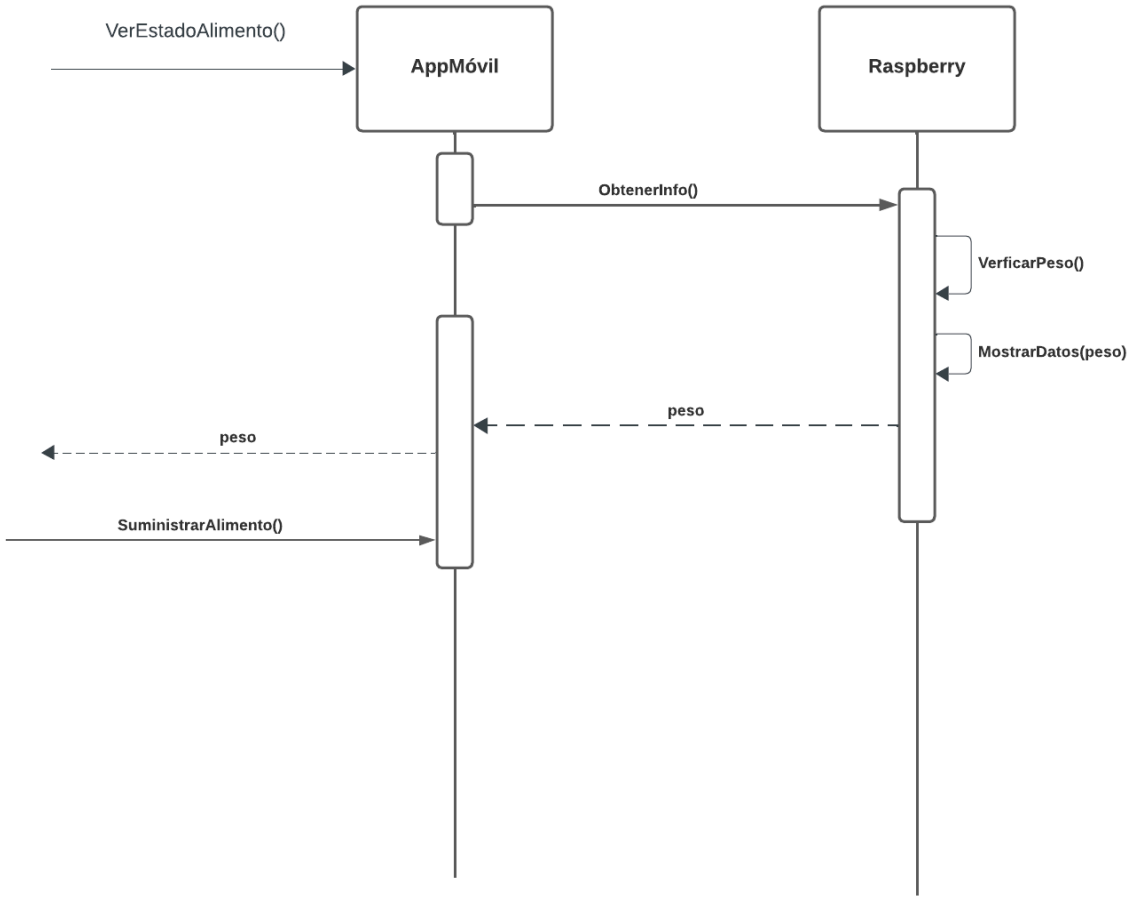

Ilustración 15 : Ver estado alimento - Diagrama nivel 1

Caso de uso: Suministrar agua

Nombre : Suministrar agua	
Autor/Fecha: Andrea Navia 24-10-2024	
Descripción: Permite al cliente rellenar el agua del gallinero de forma remota	
Actor: Cliente	
Precondición: Sensor ultrasónico debe estar conectado y verificado el nivel de agua.	
Flujo Principal: Cliente	Flujo Principal: Sistema 1.<<incluye>> C.U.S Verificar Nivel agua 2.Si es que el nivel de agua esta dentro del umbral predefinido se suministra el agua mediante el sensor 3. Muestra en la vista principal un título “Suministro de agua exitoso”
Flujo Alternativo:	Flujo Alternativo: 2.1El nivel de agua no está dentro del umbral 3.2 Muestra el logo, un botón rotulado “Nivel de agua insuficiente”
Postcondiciones: Se actualiza el nivel de agua en el sistema y en el estado de la vista principal.	
Valor medible: Reduce el tiempo en que al cliente le implicaría hacer esta tarea manualmente.	

Tabla 10: CUS Suministrar agua

suministrar agua nivel 0

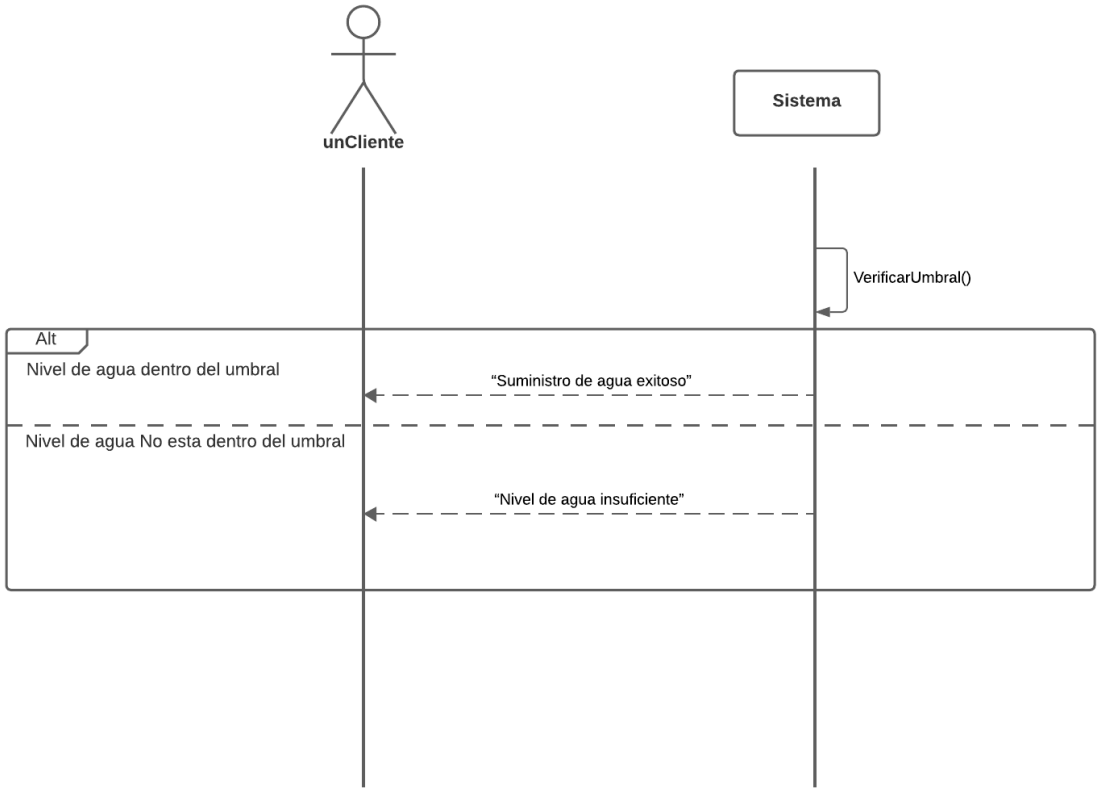

Ilustración 16 : Suministrar Agua- Diagrama nivel 0

suministrar agua nivel 1

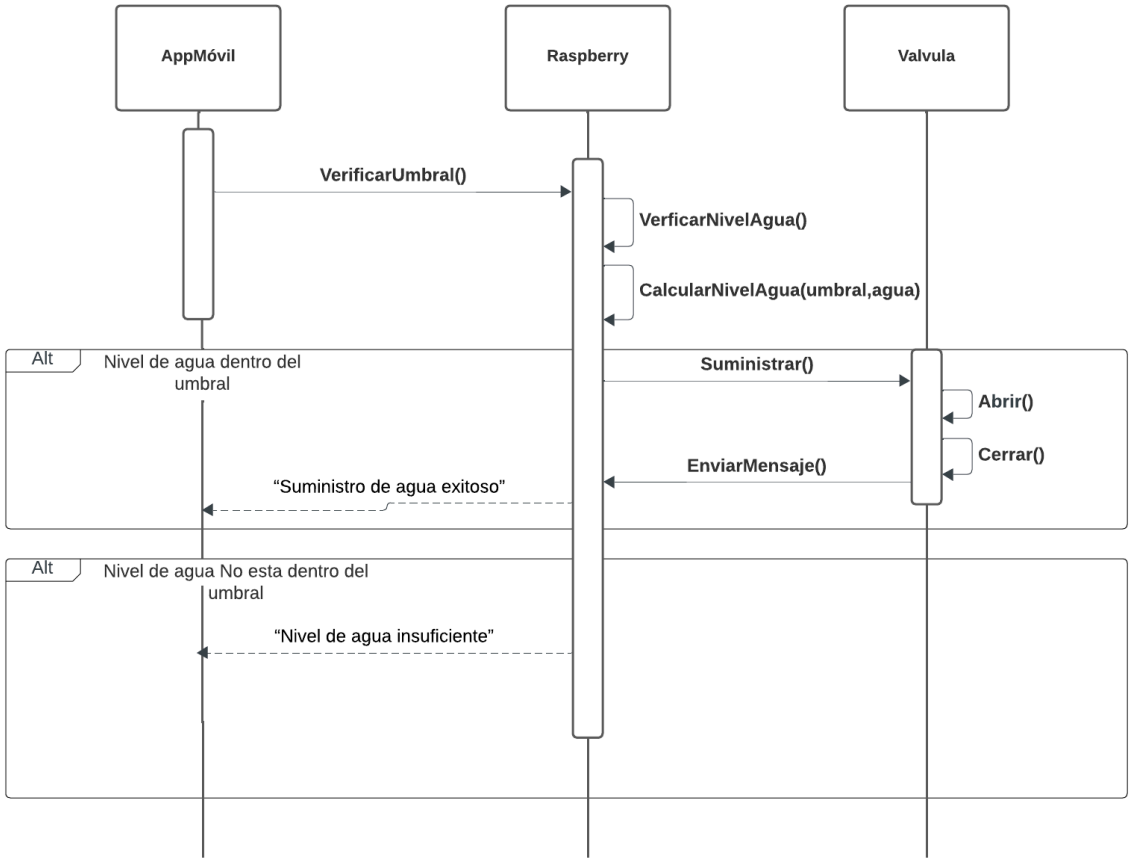

Ilustración 17 : Suministrar Agua- Diagrama nivel 1

Caso de uso: Limpiar Agua

Nombre : Limpiar Agua	
Autor/Fecha: Andrea Navia 24-10-2024	
Descripción: Permite al sistema limpiar el agua si no está en condiciones óptimas.	
Actor: Cliente	
Precondición: Debe de estar verificado el ph del agua	
Flujo Principal: Cliente	Flujo Principal: Sistema 1.<<incluye>> C.U.S Verificar ph 2.El sistema verifica si el ph está dentro del rango predefinido. 3.El ph no está dentro del umbral y el sistema desecha el agua mediante la válvula 4.Muestra en la vista ver estado de agua, “limpieza exitosa”
Flujo Alternativo:	Flujo Alternativo: 3.1 El ph está dentro de los parámetros se muestra el mensaje “Agua en ph normal”
Postcondiciones:	
Valor medible: Reduce el tiempo en que el cliente vaya manualmente a limpiar el agua, haciéndolo de forma automática, verificando correctamente los niveles de ph.	

Tabla 11: CUS Limpiar agua

Limpiar agua nivel 0

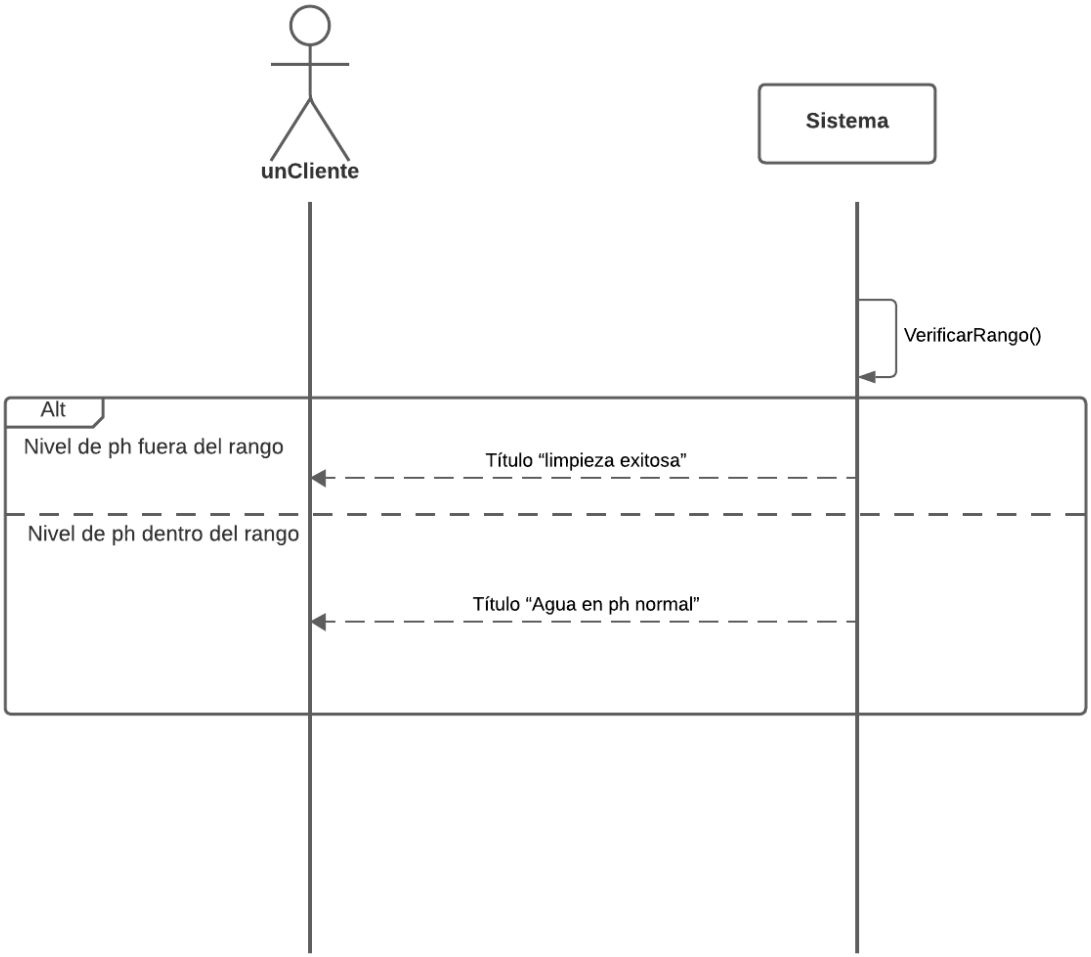

Ilustración 18 : Limpiar agua - Diagrama nivel 0

Limpiar agua nivel 1

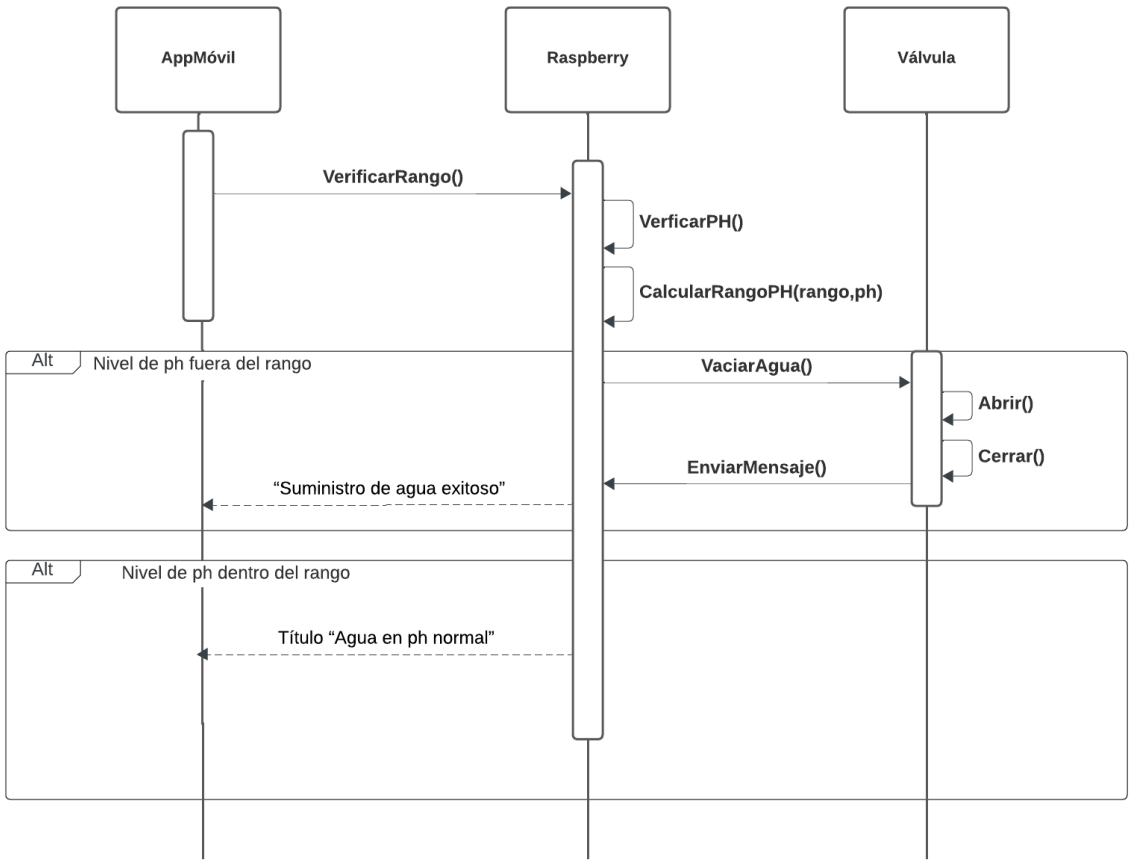

Ilustración 19 : Limpiar agua - Diagrama nivel 1

Caso de uso: Suministrar Alimento

Nombre : Suministrar Alimento	
Autor/Fecha: Ignacio Garrido 24-10-2024	
Descripción: El sistema dispensa automáticamente la cantidad necesaria de alimento a las gallinas.	
Actor: Cliente	
Precondición: El Sistema ya debe estar conectado a la app móvil	
Flujo Principal: Cliente	Flujo Principal: Sistema 1.<<incluye>> C.U.S Verificar Nivel alimento 2. Si es que el nivel de alimento está dentro del umbral predefinido se suministra el alimento mediante el sensor 3.Muestra en la vista un título “Suministro de alimento exitoso”
Flujo Alternativo:	Flujo Alternativo: Flujo Alternativo: 2.1El nivel de alimento no está dentro del umbral 3.2 Muestra un botón rotulado “Nivel de alimento insuficiente”
Postcondiciones: El sistema suministra el alimento.	
Valor medible: Reduce el tiempo en que el cliente deba realizar esta tarea.	

Tabla 12: CUS Suministrar Alimento

Suministrar alimento nivel 0

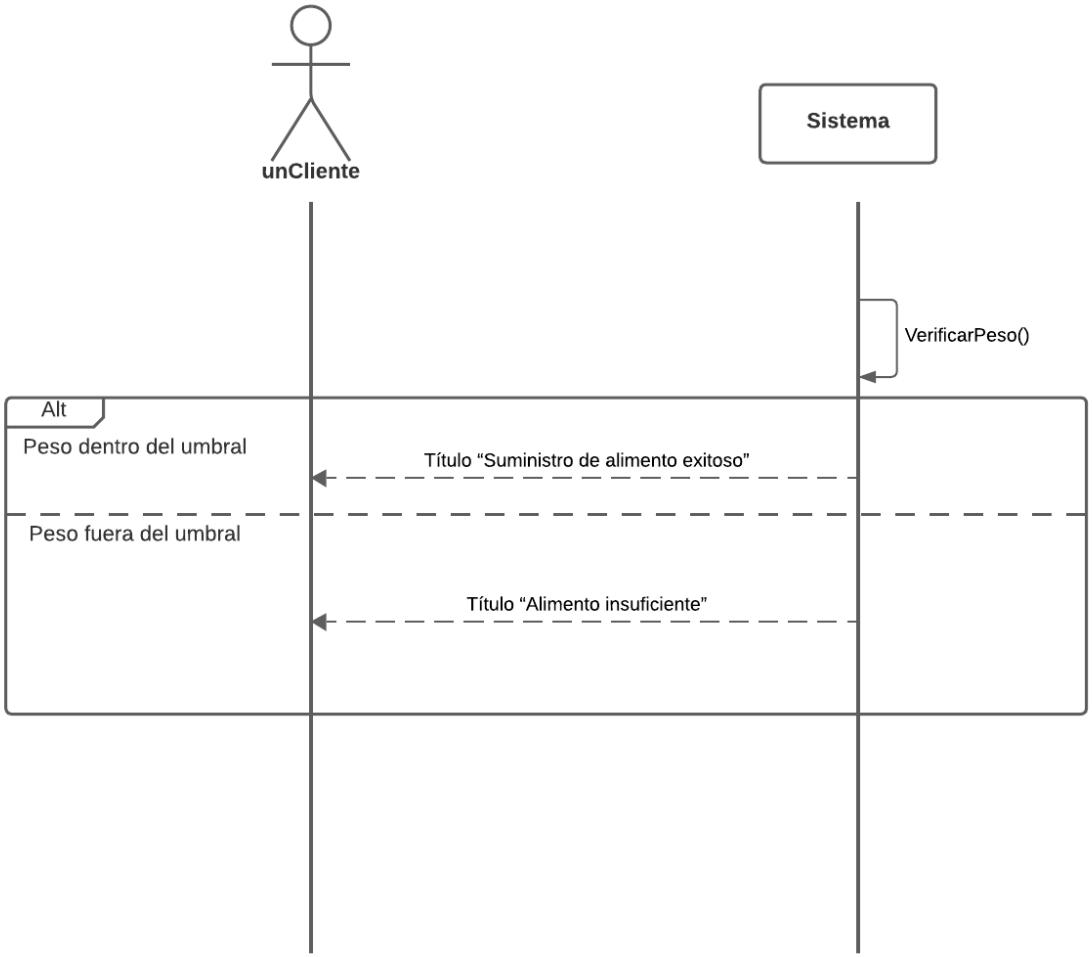

Ilustración 20 : Suministrar alimento - Diagrama nivel 0

Suministrar alimento nivel 1

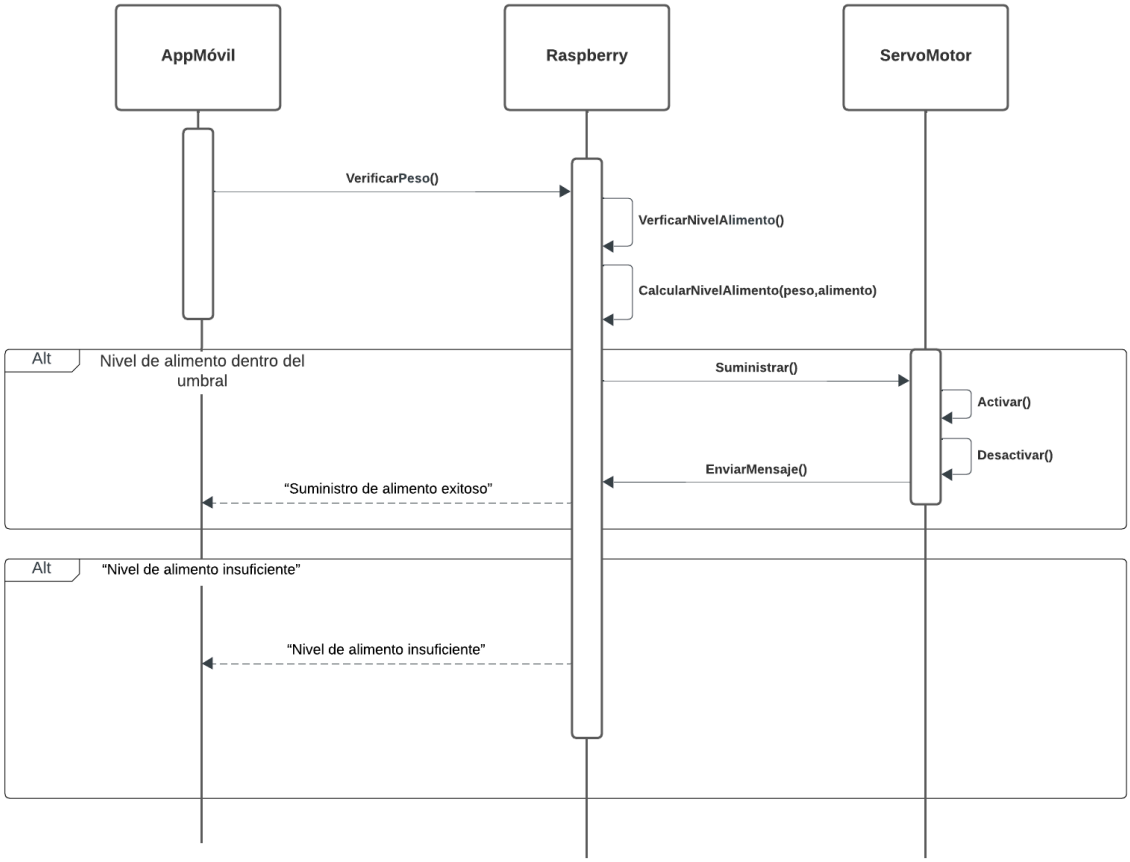

Ilustración 21 : Suministrar alimento - Diagrama nivel 1

Caso de uso: Generar Notificaciones

Nombre : Generar Notificaciones	
Autor/Fecha: Ignacio Garrido 29-10-2024	
Descripción: El sistema permite al cliente recibir una notificación automática en su aplicación móvil cuando el nivel de agua o alimento en el gallinero baja de un umbral preestablecido.	
Actor: Cliente	
Precondición: Debe estar verificado el estado de todos los sensores.	
Flujo Principal: Cliente 3. Selecciona el botón de Notificaciones.	Flujo Principal: Sistema 1. <<incluye>> C.U.S Verificar Nivel Agua 2. El sistema detecta el nivel de agua y si ha caído por debajo del umbral predefinido genera una notificación sobre el bajo nivel de agua y se envía a la aplicación móvil. 3. El sistema muestra la notificación de manera detallada: Nivel de agua, fecha y hora
Flujo Alternativo: 1.3 Selecciona el botón de Notificaciones.	Flujo Alternativo: 1.1 <<incluye>> C.U.S Verificar Nivel alimento 1.2 El sistema detecta el nivel de alimento y si este ha caído por debajo del umbral predefinido genera una notificación sobre el bajo nivel de alimento y se envía a la aplicación móvil. 1.4 El sistema muestra la notificación de manera detallada: Nivel de alimento, fecha y hora
Flujo Alternativo: 1.3 Selecciona el botón de Notificaciones	Flujo Alternativo: 1.1 <<incluye>> C.U.S Verificar Ph 1.2 El sistema detecta el nivel de ph del agua y si este se encuentra por fuera del umbral predefinido genera una notificación sobre el nivel de ph y se envía a la aplicación móvil. 1.4 El sistema muestra la notificación de manera detallada: Nivel de ph, fecha y hora
Postcondiciones: Se cargan las notificaciones al sistema	
Valor medible: El cliente podrá saber de forma rápida y ordenada el estado del gallinero, automático.	

Tabla 13: CUS Generar Notificaciones

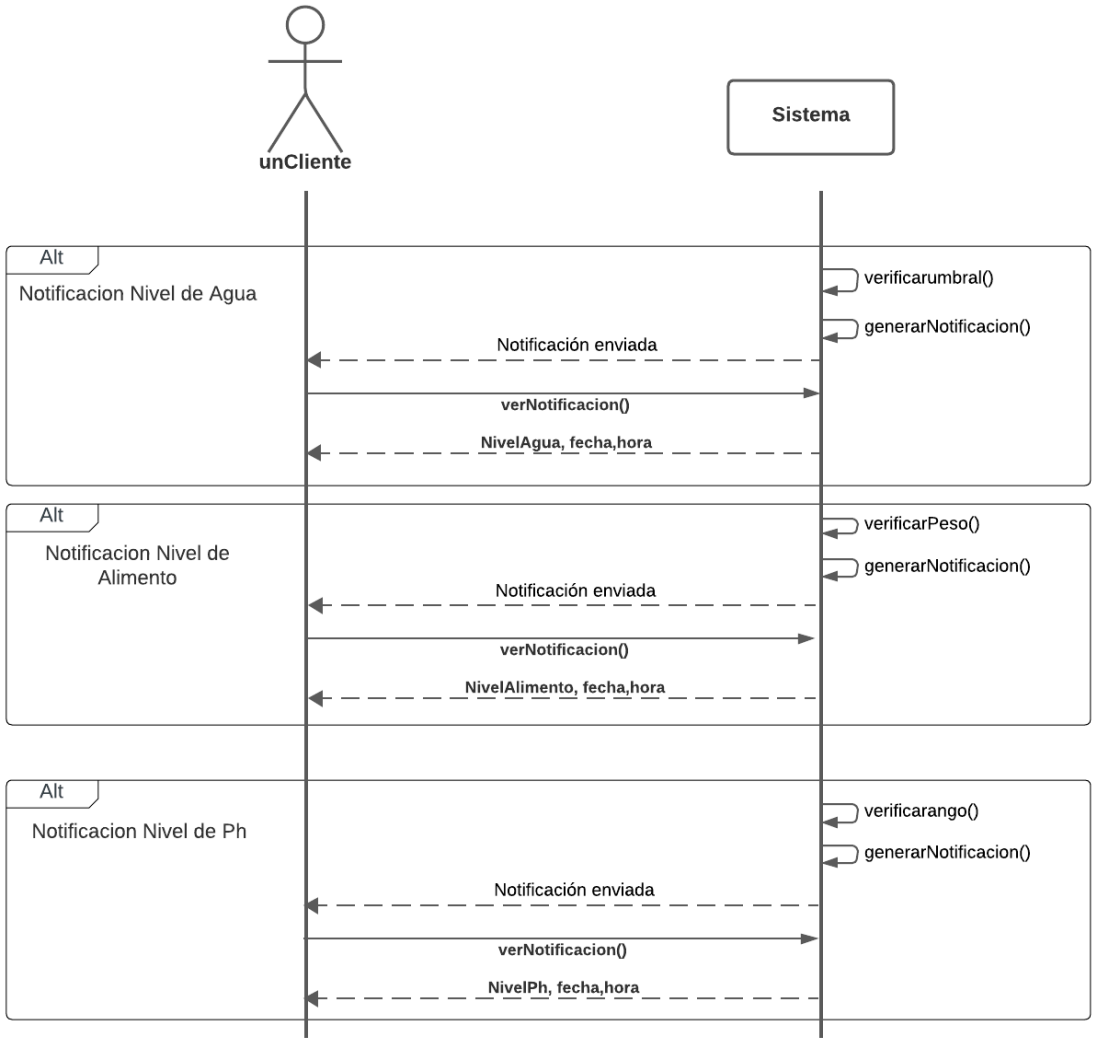

Ilustración 22 : Generar Notificaciones - Diagrama nivel 0

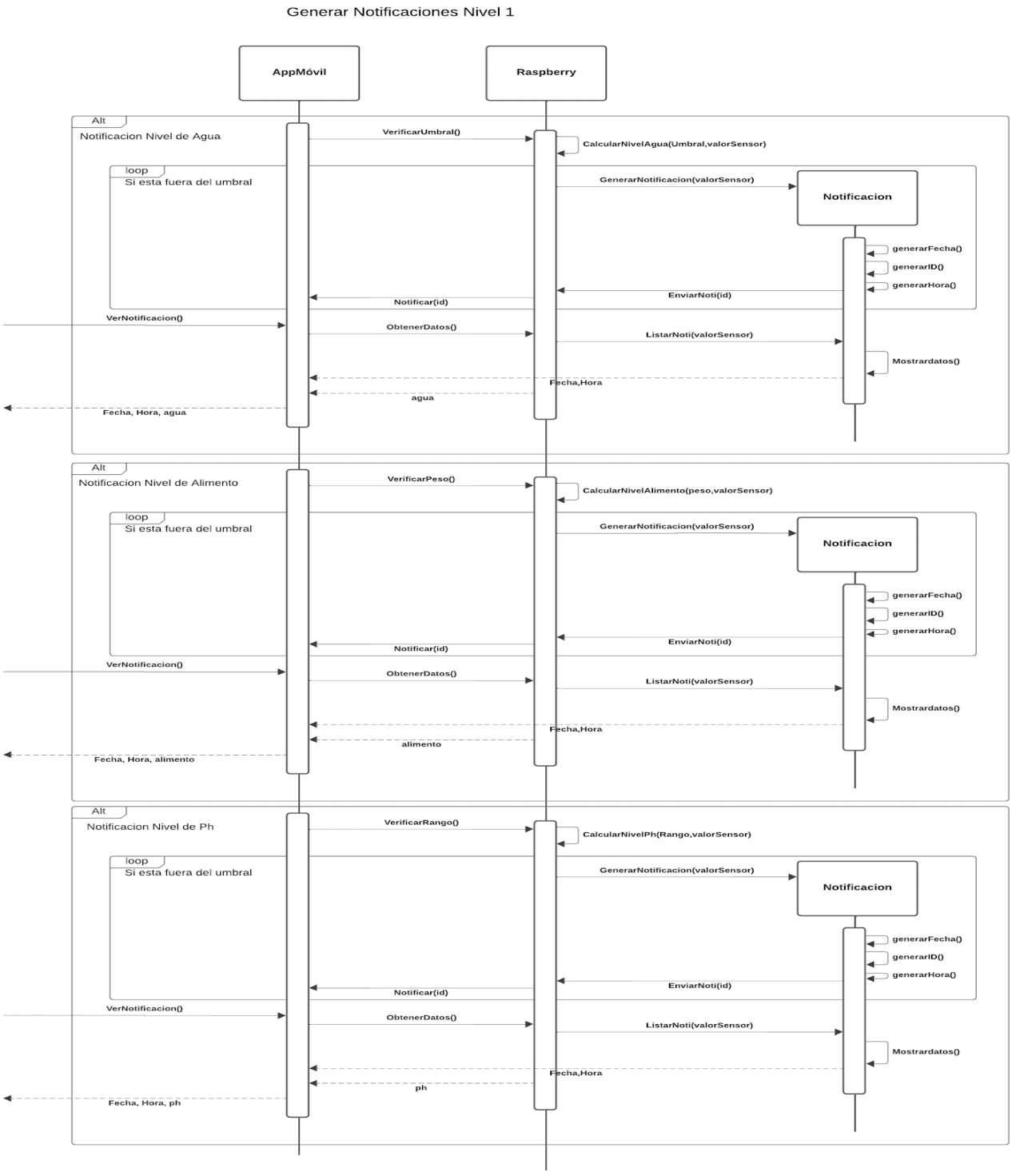

Ilustración 23 : Generar Notificaciones - Diagrama nivel 1

Caso de uso: Verificar ph

Nombre : Verificar ph	
Autor/Fecha: Andrea Navia 24-10-2024	
Descripción: Permite al sistema ver el ph del agua	
Actor: Sensor PH	
Precondición: El sensor debe estar previamente conectado	
Flujo Principal: Sensor PH 2.Envía un valor numérico.	Flujo Principal: Raspberry 1.El sistema verifica el nivel de ph del agua mediante el sensor de ph.
Flujo Alternativo:	Flujo Alternativo:
Postcondiciones: El sistema guarda el valor del sensor	
Valor medible: Agiliza y reduce el tiempo de obtener información directa del sensor.	

Tabla 14: CUS Verificar ph

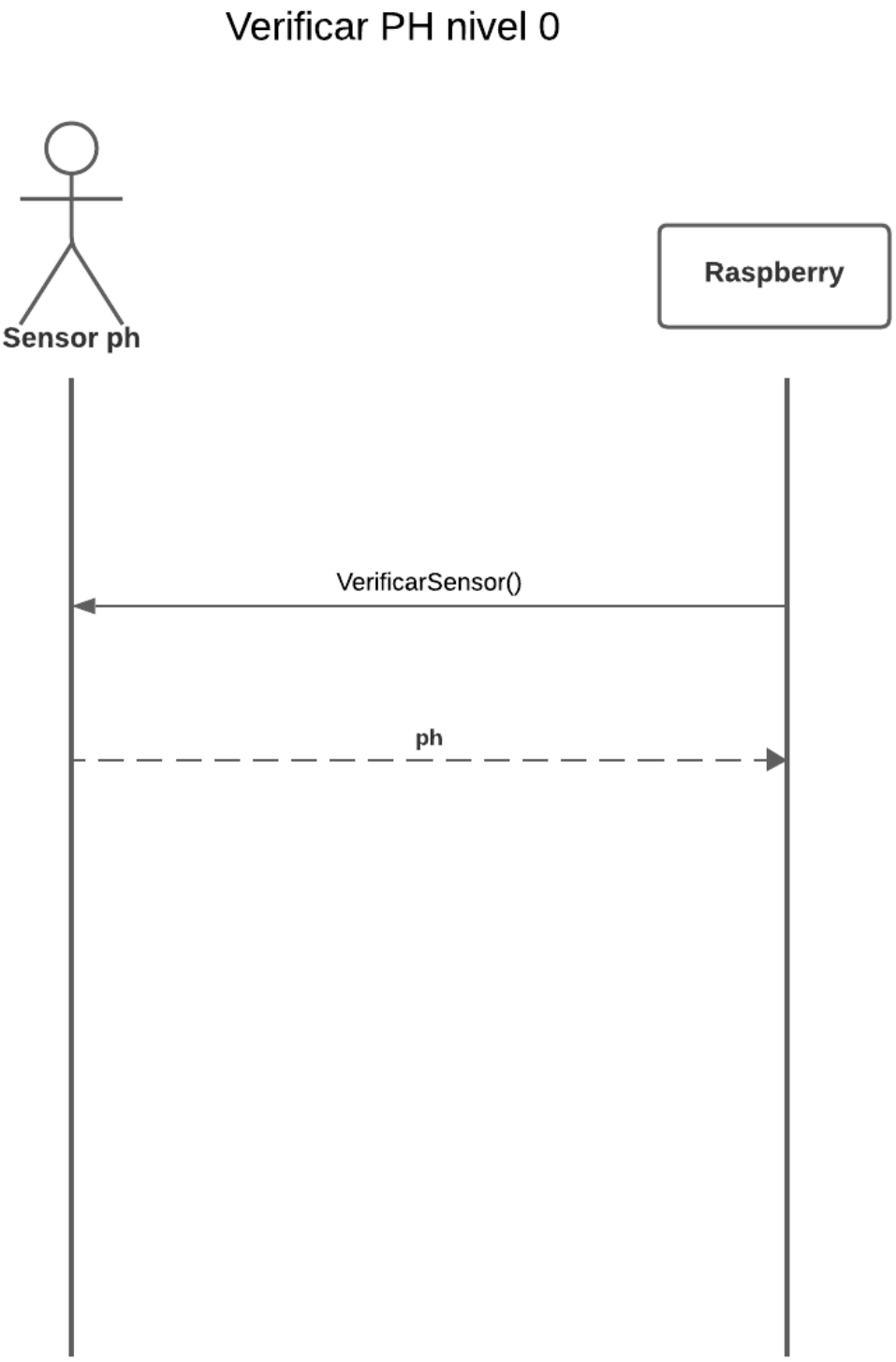

Ilustración 24 : Verificar PH - Diagrama nivel 0

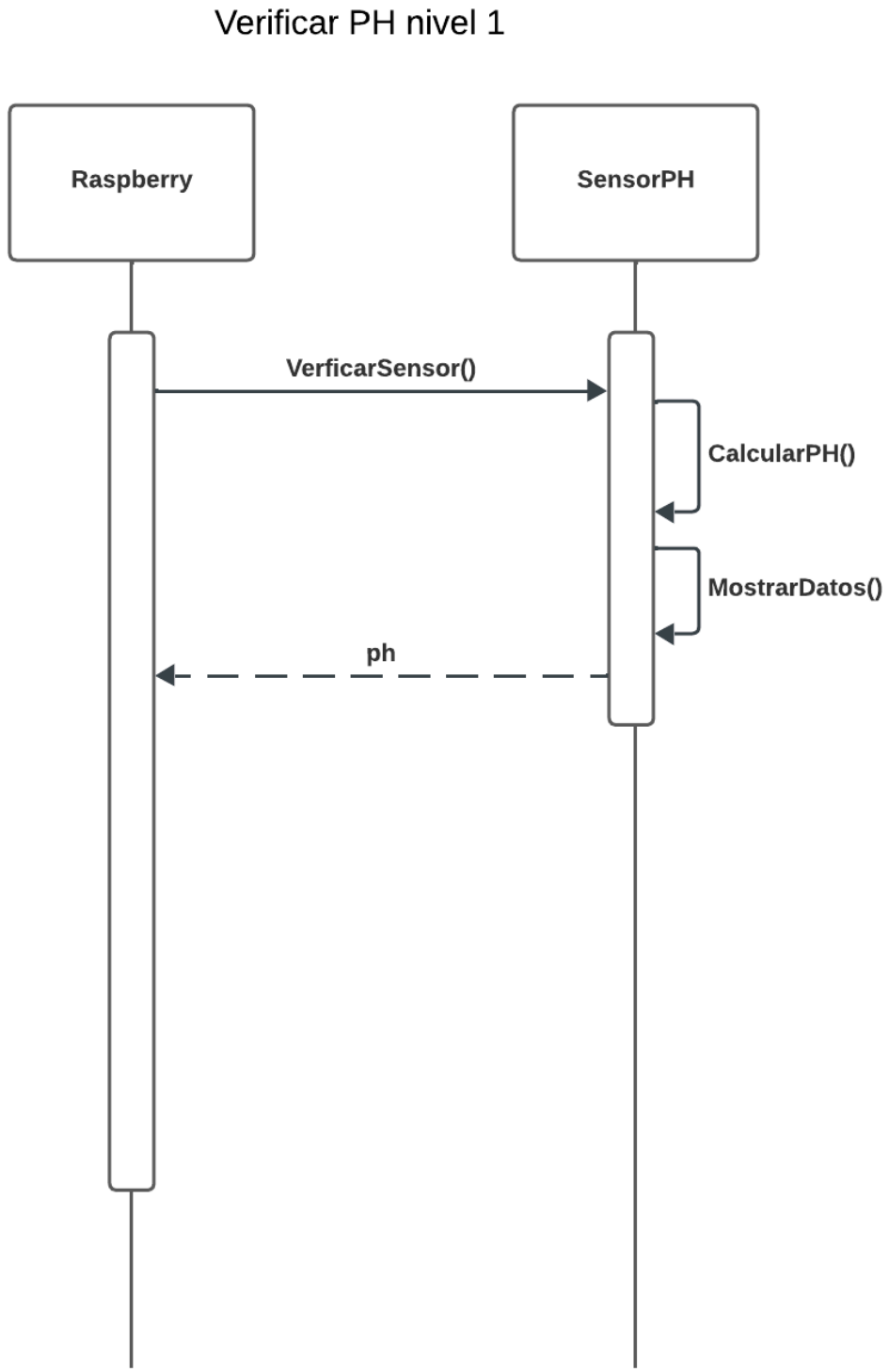

Ilustración 25 : Verificar PH - Diagrama nivel 1

Caso de uso: Verificar Nivel agua

<table>
<tr><td colspan="2">Nombre : Verificar Nivel agua</td></tr>
<tr><td colspan="2">Autor/Fecha: Andrea Navia 24-10-2024</td></tr>
<tr><td colspan="2">Descripción: Permite al sistema saber el nivel de agua.</td></tr>
<tr><td colspan="2">Actor: Sensor ultrasónico</td></tr>
<tr><td colspan="2">Precondición: El sensor debe estar previamente conectado</td></tr>
<tr><td>Flujo Principal: Sensor ultrasónico

2. Envía el valor numérico.</td><td>Flujo Principal: Raspberry
1.El sistema verifica el nivel del agua mediante el sensor ultrasónico.</td></tr>
<tr><td>Flujo Alternativo:</td><td>Flujo Alternativo:</td></tr>
<tr><td colspan="2">Postcondiciones: El sistema guarda el valor del sensor</td></tr>
<tr><td colspan="2">Valor medible: Reduce el tiempo en las demás tareas del sistema</td></tr>
</table>

Tabla 15: CUS Verificar Nivel agua

Verificar Nivel agua nivel 0

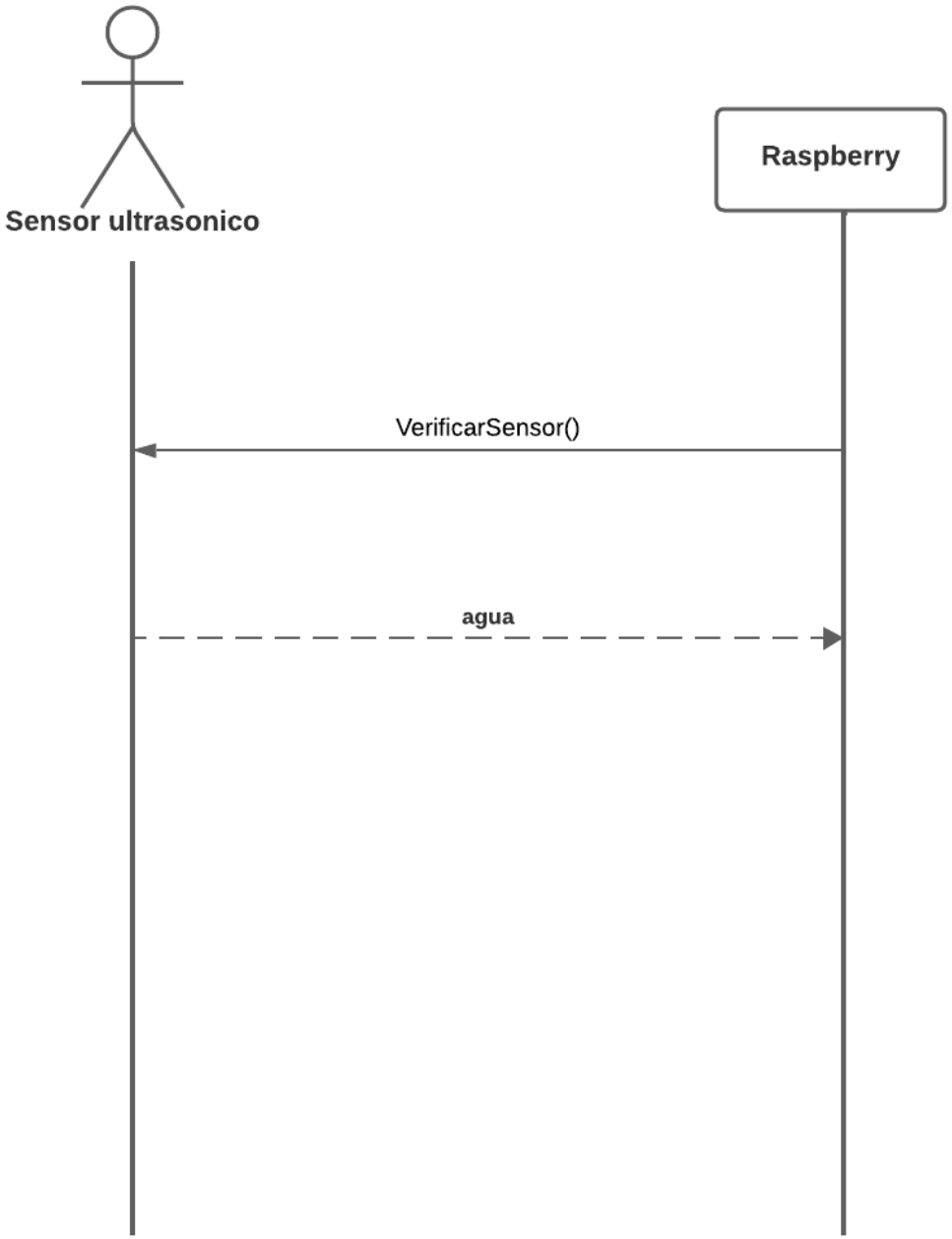

Ilustración 26 : Verificar nivel agua - Diagrama nivel 0

Verificar Nivel agua nivel 1

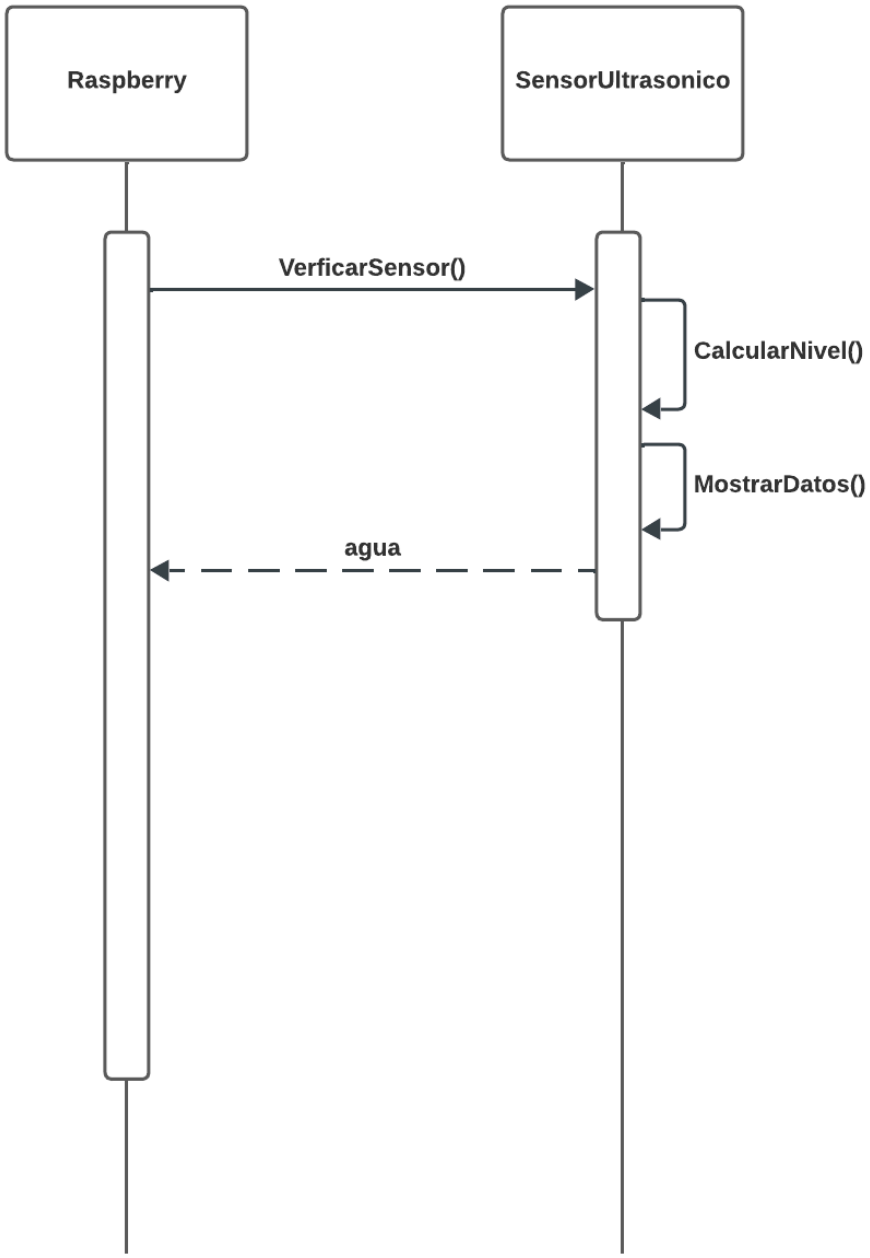

Ilustración 27 : Verificar nivel agua - Diagrama nivel 1

Caso de uso: Verificar Nivel alimento

Nombre : Verificar Nivel alimento	
Autor/Fecha: Andrea Navia 24-10-2024	
Descripción: Permite al sistema saber el nivel del alimento mediante su peso.	
Actor: Sensor de peso	
Precondición: El sensor debe estar previamente conectado	
Flujo Principal: Sensor de peso 2. Envía el valor numérico.	Flujo Principal: Raspberry 1.El sistema verifica el nivel del alimento a través de su peso con el sensor de peso
Flujo Alternativo:	Flujo Alternativo:
Postcondiciones: El sistema guarda el valor del sensor	
Valor medible: Agiliza el proceso de obtener la información de forma rápida	

Tabla 16: CUS Verificar Nivel alimento

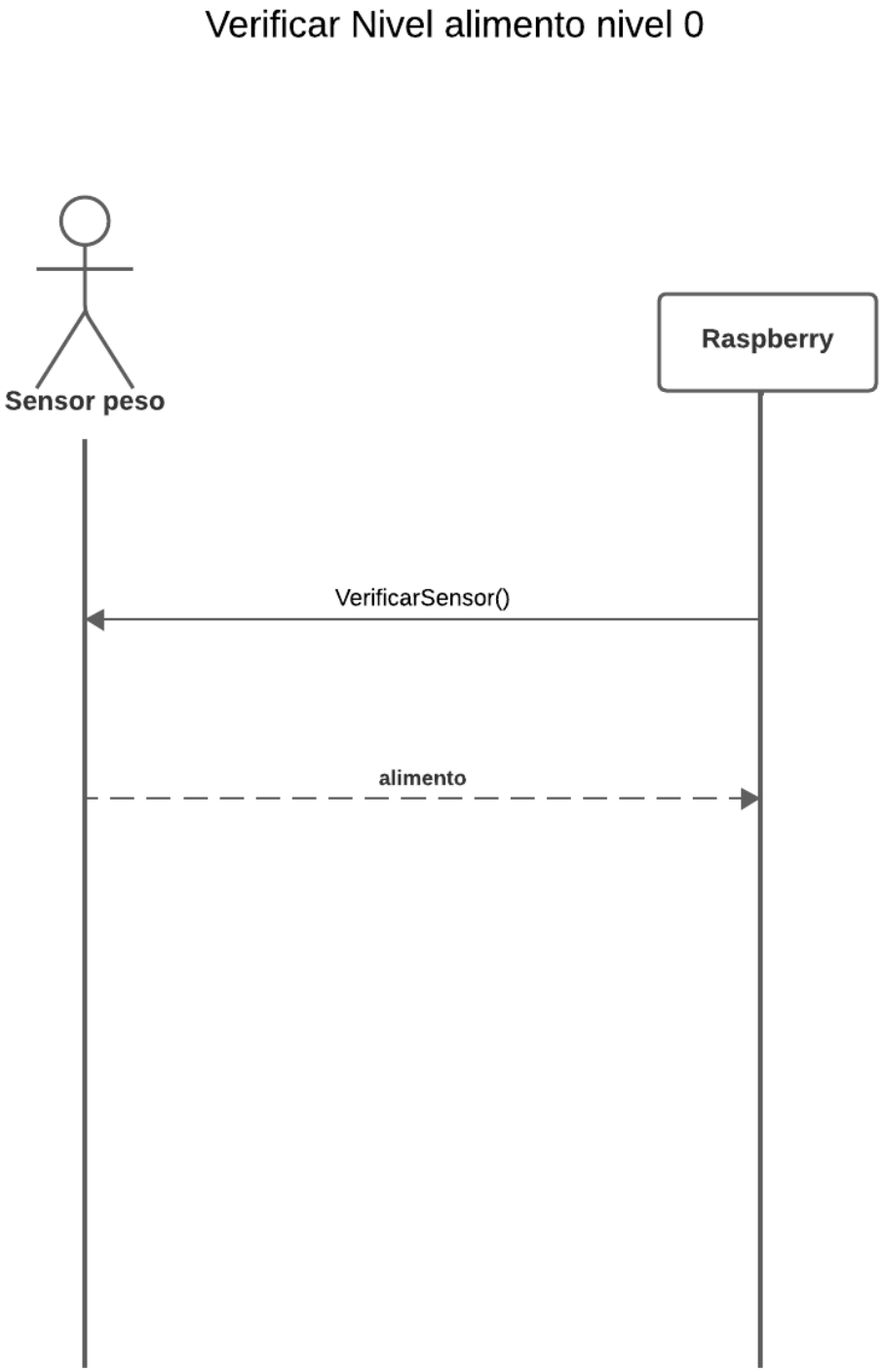

Ilustración 28 : Verificar nivel alimento - Diagrama nivel 0

Verificar Nivel alimento nivel 1

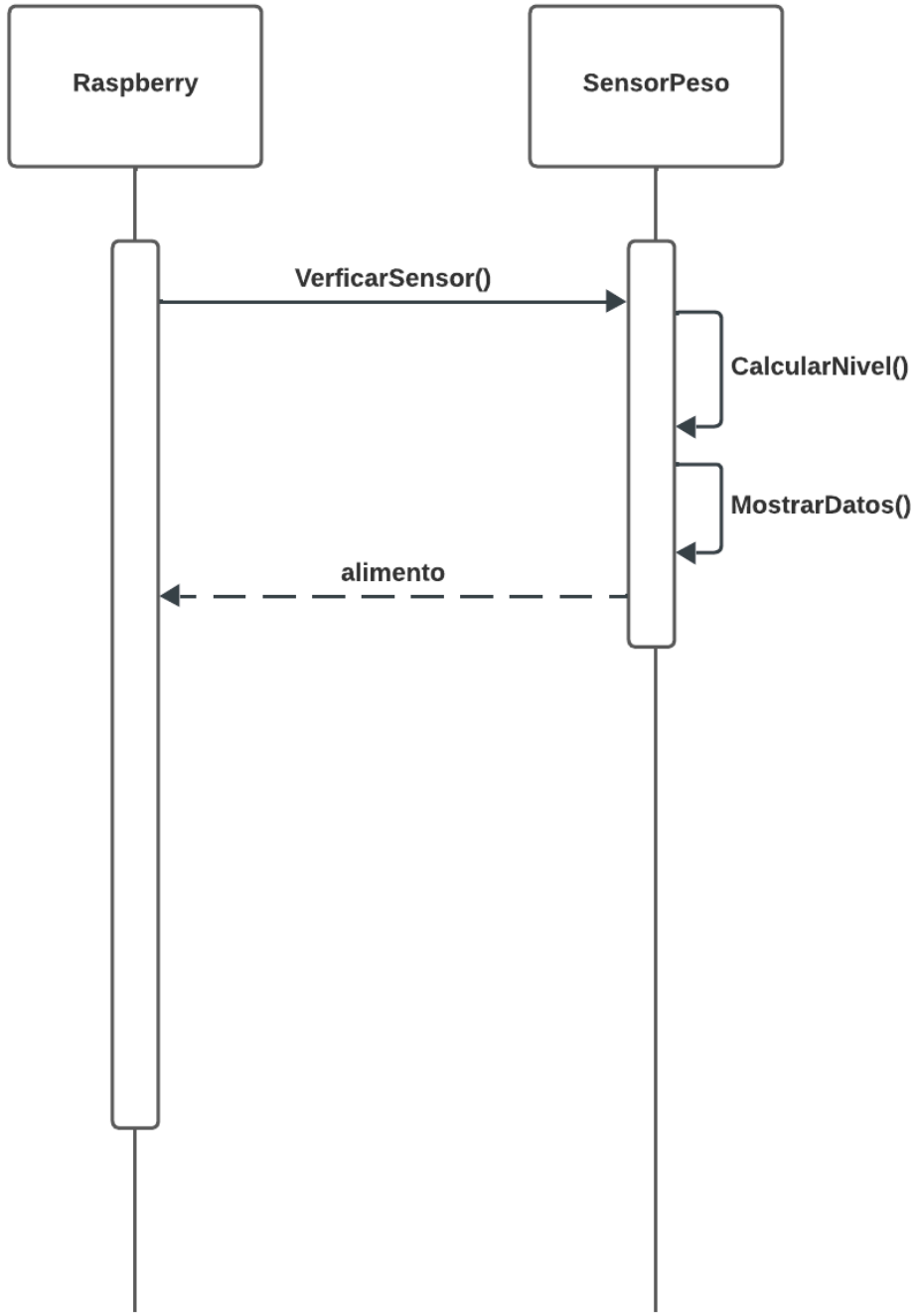

Ilustración 29 : Verificar nivel alimento - Diagrama nivel 1

4.1.4 Diagrama de Clases

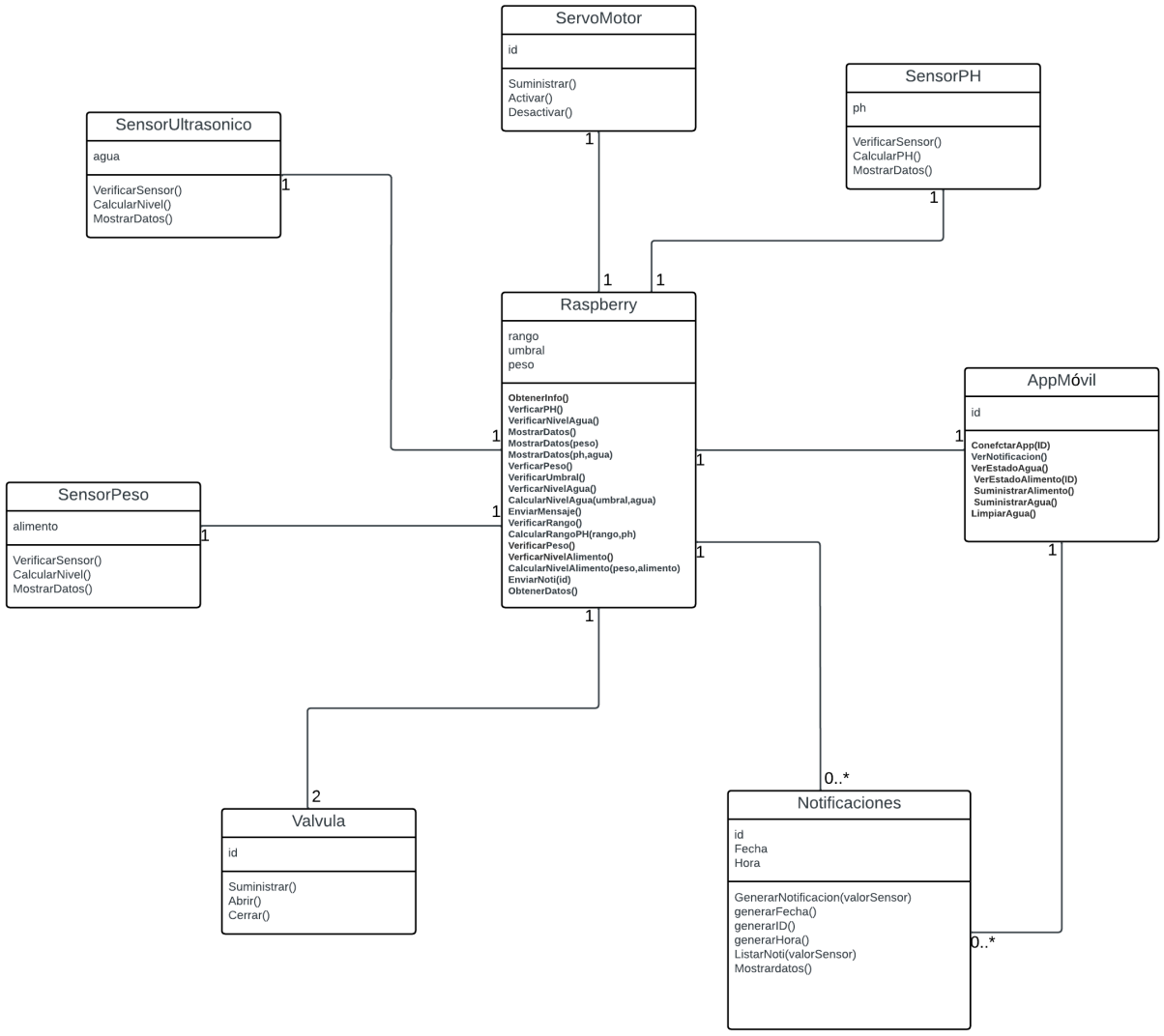

Ilustración 30 : Diagrama de clases

4.2 Herramientas y técnicas

4.2.1 Herramientas

4.2.1.1 Generales

1. Python: Lenguaje de programación versátil y fácil de aprender, utilizado en diversas aplicaciones gracias a sus poderosas bibliotecas.
2. VNC Viewer: Herramienta para acceder y controlar de manera remota el escritorio de otro ordenador, ideal para trabajar con Raspberry Pi 4.
3. Sistema Operativo Linux Ubuntu (CLI): Sistema basado en comandos de texto, utilizado principalmente en servidores para administrar y configurar el sistema.
4. Visual Studio Code: Editor de código ligero y potente, compatible con el desarrollo de aplicaciones web y móviles.

4.2.1.2 Hardware

1. Raspberry Pi 4 Model B: Es una pequeña computadora que puedes usar para hacer proyectos de electrónica o programación. Tiene puertos para conectar otros dispositivos y puede ejecutar sistemas operativos.
2. Protoboard: Es una placa donde puedes conectar los componentes de tus circuitos electrónicos, lo que hace más fácil probar las conexiones.
3. Servomotor: Es un tipo de motor que se puede controlar con precisión para mover cosas, como en robots o mecanismos automáticos.
4. Sensor de pH: Mide cuán ácida y alcalina es la calidad del agua, para saber si está en las condiciones adecuadas de consumo.
5. Sensor de Peso: Este sensor va a medir el peso del alimento actual en la báscula.
6. Válvula Solenoide: Es una válvula que se abre o cierra usando electricidad para controlar el paso del agua.
7. Sensor Ultrasónico: Este sensor mide la distancia a un objeto usando sonido, que vamos a usar para medir el nivel del agua.
8. Resistencia: Es un componente que se usa en circuitos electrónicos para controlar cuánta corriente pasa por un cable, de forma que no se sobrecargue el sistema.
9. Memoria SD: Es una tarjeta que guarda información, para almacenar la S.O del Raspberry y las configuraciones posteriores.
10. Jumper: Son cables que se usan para hacer conexiones entre los componentes de un circuito en el protoboard.

11. Celular: Es un teléfono móvil que puedes usar para controlar o monitorear el proyecto por una aplicación de forma remota.

4.2.1.3 Para el desarrollo del servidor

1. Socket: Mecanismo para enviar y recibir datos entre aplicaciones; el módulo de Python facilita la conexión de red entre servidor y cliente.
2. Raspberry OS: Sistema operativo diseñado para Raspberry Pi, que proporciona la plataforma para ejecutar aplicaciones en este dispositivo.

4.2.1.4 Para el desarrollo de la Aplicación Móvil

1. Kivy: Framework de código abierto en Python para desarrollar aplicaciones interactivas con interfaces gráficas, popular en aplicaciones móviles y de escritorio.
2. Python (con Kivy Widgets): Un "widget" es cualquier componente visual para realizar la interfaz gráfica, como botones, cajas de texto, etiquetas, etc. Algunos widgets comunes incluyen Button, Label, TextInput, BoxLayout, entre otros.
3. Kivy Language (opcional) : es ideal para definir la estructura visual de la interfaz de manera más limpia y rápida, sin mezclarse tanto con la lógica de negocio.
4. Buildozer: Herramienta de automatización que facilita la creación de aplicaciones móviles para Android e iOS usando Kivy.
5. Sistema Operativo de Linux ubuntu: Sistema basado en comandos de texto, utilizado principalmente para administrar y configurar el sistema, especialmente en servidores.

4.2.1.5 Adicionales

5. Redmine: Plataforma para gestionar tareas de proyectos, permite crear, asignar, seguir el progreso y colaborar de forma eficiente.

6. Google Docs: Herramienta capaz de realizar documentos de manera sencilla y en conjunto

7. Discord: Es una plataforma de comunicación que permite a los usuarios conectarse a través de mensajes de texto, voz y video.

8. Whatsapp: Es una aplicación de comunicación por mensajes de texto, principalmente la utilizamos para establecer reuniones, proponer cambios y resolver problemas en conjunto.

9. Git : sistema de control de versiones para el seguimiento de cambios en el código y también utilizamos repositorios externos para la instalación de aplicaciones o dependencias necesarias para el funcionamiento de los componentes.

4.2.2 Técnicas

1. Prueba y error: Para la parte de implementación de todas las cosas lo mejor es la prueba y error, ya que de los errores se aprende. Consideraremos una cantidad de testeos que serán registrados en la WIKI para ver un progreso en nuestro proyecto.

2. División de proyecto: El proyecto lo dividimos con pequeñas tareas más accesibles como se puede ver en el modelo de los C.U.S de contexto. Lo trabajamos de esta manera porque consideramos que es lo más efectivo para atacar un problema grande.

5. Implementación

5.1. Plan de Integración

Se planificó la integración de los diferentes módulos del sistema, asegurando la compatibilidad entre los sensores, el servidor y la aplicación móvil. Se llevaron a cabo pruebas unitarias y funcionales para validar el flujo de datos y las acciones automatizadas.

5.2. Modelo de Implementación

El modelo de implementación se dividió en tres fases principales:

Montaje del Hardware:

Instalación de sensores en los contenedores de alimento y agua.
Configuración de la Raspberry Pi como controlador central.
Conexión de válvulas solenoides y servomotores para el suministro de agua y alimento.

Desarrollo del Software:

Creación de una aplicación móvil para monitorear y controlar el sistema en tiempo real.
Desarrollo de scripts en Python para gestionar la comunicación con los distinto sensores

Pruebas e Iteraciones:
Pruebas de los sensores para medir precisión y consistencia.
Validación del sistema de notificaciones en caso de niveles críticos de agua o alimento.
Ajustes en la interfaz gráfica de la aplicación móvil para mejorar la experiencia del usuario.

5.3. Módulos Implementados

5.3.1. Implementar App para controlar las distintas funciones del proyecto

```python

from kivy.app import App
from kivy.config import Config
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivy.uix.floatlayout import FloatLayout
from kivy.core.window import Window
from kivy.graphics import Color, RoundedRectangle, Rectangle , Line
from kivy.uix.button import ButtonBehavior
from kivy.uix.image import Image
from kivy.uix.popup import Popup
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.textinput import TextInput
from kivy.uix.screenmanager import ScreenManager, Screen
from kivy.uix.scrollview import ScrollView
from kivy.uix.progressbar import ProgressBar
from kivy.clock import Clock
import random

# Configuración de la ventana
Config.set('graphics', 'width', '360')  # Ancho de la ventana
Config.set('graphics', 'height', '640')  # Alto de la ventana
Config.set('graphics', 'resizable', True)  # Permitir redimensionar
Config.write()

class RoundedButton(Button):
    def __init__(self, bg_color=(0.7529, 0.7529, 0.7529, 1), **kwargs):
        super().__init__(**kwargs)
        self.background_normal = ''
        self.background_down = ''
        self.background_color = (0, 0, 0, 0)

        # Personaliza el fondo redondeado con el color proporcionado
        self.bg_color = bg_color
        with self.canvas.before:
            Color(*self.bg_color)
            self.bg_rect = RoundedRectangle(
                size=self.size,
                pos=self.pos,
                radius=[(20, 20), (20, 20), (20, 20), (20, 20)]
            )
        self.bind(size=self.update_rect, pos=self.update_rect)

    def update_rect(self, *args):
        self.bg_rect.size = self.size
        self.bg_rect.pos = self.pos


class ImageButton(ButtonBehavior, Image):
    pass

class MainScreen(Screen):  # Cambié MainScreen a heredar de Screen, no de App
    def __init__(self, **kwargs):
        super(MainScreen, self).__init__(**kwargs)
        Window.clearcolor =  (0.7529, 0.7529, 0.7529, 1)  # Color

        # Crear un FloatLayout para gestionar las posiciones
        layout = FloatLayout()

        # Crear un Label en la parte superior
        label = Label(text="¡Bienvenido a Kivy!", font_size='30sp', color=(2, 0, 0, 1), bold= True)  # Color rojo
        label.size_hint = None, None
        label.size = label.texture_size
        label.pos_hint = {'center_x': 0.5, 'top': 0.9}  # Posición en la parte superior
        layout.add_widget(label)

        # Crear el contenedor amarillo transparente en la parte inferior (50% de la pantalla)
```

```
        # Crear el contenedor amarillo transparente en la parte inferior (50% de la pantalla)
        bottom_half = FloatLayout(size_hint=(1, 0.5))  # 50% de la pantalla de altura
        with bottom_half.canvas:
            Color(1, 1, 0, 0.8)  # Amarillo semi-transparente (r, g, b, alpha)
            self.rect_bottom = Rectangle(size=bottom_half.size, pos=bottom_half.pos)  # Cambiar a Rectangle
        bottom_half.bind(size=self.update_rect_bottom, pos=self.update_rect_bottom)
        bottom_half.pos_hint = {'x': 0, 'y': 0}  # Colocamos en la parte inferior
        layout.add_widget(bottom_half)

        # Crear el contenedor gris con bordes redondeados
        gray_box = FloatLayout(size_hint=(None, None), size=("300dp", "350dp"))
        gray_box.pos_hint = {'center_x': 0.5, 'y': 0.06}

        # Dibujar un fondo gris con bordes redondeados
        with gray_box.canvas:
            Color(1, 1, 1, 1)
            self.rect_gray = RoundedRectangle(size=gray_box.size, pos=gray_box.pos, radius=[(20, 20), (20, 20), (20, 20), (20, 20)])
        gray_box.bind(size=self.update_rect_gray, pos=self.update_rect_gray)
        layout.add_widget(gray_box)

        gray_box.pos = (gray_box.pos[0], gray_box.pos[1] - 50)

        # Crear un label para conectar
        label2 = Label(text="Establecer Conexión", font_size='22sp', color=(2, 0, 0, 1), bold=True)  # Color rojo
        label2.size_hint = None, None
        label2.size = label2.texture_size
        label2.pos_hint = {'center_x': 0.5, 'top': 0.5}  # Posición en la parte superior
        layout.add_widget(label2)

        # Crear el botón de conexión
        button_conect = ImageButton(
            source="conectar.png",  # Ruta de la imagen para el botón
            size_hint=(None, None),
            size=("200dp", "210dp")
        )
        button_conect.pos_hint = {'center_x': 0.5, 'center_y': 0.30}
        button_conect.bind(on_press=self.show_popup)
        layout.add_widget(button_conect)

        # Crear el logo en la parte superior
        image = Image(
            source='logo.png',
            allow_stretch=True,
            keep_ratio=False,
            size_hint=(None, None),  # Desactiva el tamaño relativo
            size=(150, 130)  # Asigna ancho y alto especificos
        )

        # Establecer la posición del logo
        image.pos_hint = {'center_x': 0.5, 'center_y': 0.75}  # Centrar en X y ajustar altura
        layout.add_widget(image, index=len(layout.children))  # Asegúrate de que la imagen esté delante

        self.add_widget(layout)  # Agregar el layout a la pantalla

    def show_popup(self, instance):
        # Crear un BoxLayout para el popup
        popup_layout = BoxLayout(orientation='vertical', padding=10, spacing=10)

        # Crear un campo de entrada para la IP del servidor
        ip_label = Label(text="Dirección IP del servidor:")
        ip_input = TextInput(hint_text="Ingrese la dirección IP", multiline=False)

        # Crear un botón de CONECTAR en el popup
        connect_button = Button(text="Conectar", size_hint=(None, None), size=("300dp", "50dp"))
        connect_button.bind(on_press=lambda x: self.connect_to_server(ip_input.text))  # Llamada al método para conectar
```

```
        connect_button = Button(text="Conectar", size_hint=(None, None), size=("300dp", "50dp"))
        connect_button.bind(on_press=lambda x: self.connect_to_server(ip_input.text))  # Llamada al método para conectar

        # Agregar los elementos al layout del popup
        popup_layout.add_widget(ip_label)
        popup_layout.add_widget(ip_input)
        popup_layout.add_widget(connect_button)

        # Crear el Popup y mostrarlo
        self.popup = Popup(title="Conectar a servidor", content=popup_layout,
                           size_hint=(None, None), size=("340dp", "220dp"))
        self.popup.open()

    def show_connected_screen(self):
        # Cambiar a la pantalla de conexión exitosa
        self.manager.current = "second"  # Asegúrate de que 'second' es el nombre correcto de la pantalla

    def connect_to_server(self, ip_address):
        if ip_address == "1":  # Comprobamos si la IP es "1", solo como ejemplo
            print(f"Conectado a {ip_address}...")
            self.popup.dismiss()  # Cerrar el popup
            self.show_connected_screen()  # Cambiar a la pantalla de conexión exitosa
        else:
            print(f"Dirección IP errónea. Inténtalo de nuevo.")
            # Actualizar el campo del popup para que el usuario ingrese de nuevo
            self.popup.content.children[1].text = ""  # Limpiar el TextInput (si estás usando Kivy)
            # Aquí no terminamos la función con return, solo damos feedback al usuario y seguimos esperando
            self.popup.content.children[1].hint_text = "Ingresa una IP válida"  # Añadir texto de sugerencia para el usuario

    # Métodos para actualizar los tamaños y posiciones de los rectángulos
    def update_rect_bottom(self, instance, value):
        self.rect_bottom.pos = instance.pos
        self.rect_bottom.size = instance.size

    def update_rect_gray(self, instance, value):
        self.rect_gray.pos = instance.pos
        self.rect_gray.size = instance.size


# Pantalla de éxito (pantalla de conexión exitosa)
class SecondScreen(Screen):
    def __init__(self, **kwargs):
        super(SecondScreen, self).__init__(**kwargs)
        layout = FloatLayout()

        image = Image(
            source='logo.png',
            allow_stretch=True,
            keep_ratio=False,
            size_hint=(None, None),  # Desactiva el tamaño relativo
            size=(180, 160)  # Asigna ancho y alto específicos
        )
        # Establecer la posición de la imagen dentro de los límites visibles
        image.pos_hint = {'center_x': 0.5, 'center_y': 0.80}  # Centrar en X y ajustar altura
        layout.add_widget(image, index=len(layout.children))  # Asegúrate de que la imagen esté delante

        # Crear el contenedor amarillo transparente en la parte inferior (50% de la pantalla)
        bottom_half = FloatLayout(size_hint=(1, 0.5))  # 50% de la pantalla de altura
        with bottom_half.canvas:
            Color(0, 0, 1, 0.8)  # Amarillo semi-transparente (r, g, b, alpha)
            self.rect_bottom = Rectangle(size=bottom_half.size, pos=bottom_half.pos)
        bottom_half.bind(size=self.update_rect_bottom, pos=self.update_rect_bottom)
        bottom_half.pos_hint = {'x': 0, 'y': 0}  # Colocamos en la parte inferior
        layout.add_widget(bottom_half)
```

```
        bottom_half.pos_hint = {'x': 0, 'y': 0}  # Colocamos en la parte inferior
        layout.add_widget(bottom_half)

        gray_box = FloatLayout(size_hint=(None, None), size=("300dp", "350dp"))  # Tamaño del rectángulo
        gray_box.pos_hint = {'center_x': 0.5, 'y': 0.06}  # Centramos el contenedor en el eje X y lo movemos en Y
        with gray_box.canvas:
            Color(1, 1, 1, 1)  # Gris (r, g, b, alpha)
            self.rect_gray = RoundedRectangle(size=gray_box.size, pos=gray_box.pos, radius=[(20, 20), (20, 20), (20, 20), (20, 20)])
        gray_box.bind(size=self.update_rect_gray, pos=self.update_rect_gray)
        layout.add_widget(gray_box)
        gray_box.pos = (gray_box.pos[0], gray_box.pos[1] - 50)  # Desplazar el contenedor en Y (50px hacia abajo


        button_to_three = ImageButton(
            source="campana.png",  # Ruta de la imagen para el botón
            size_hint=(None, None),
            size=("100dp", "110dp")
        )
        button_to_three.pos_hint = {'center_x': 0.92, 'center_y': 0.95}
        button_to_three.bind(on_press=self.show_popup)
        layout.add_widget(button_to_three)

        button_to_four = RoundedButton(text="Ver Estado Agua", size_hint=(None, None), size=("250dp", "60dp"),color=(0, 0, 0, 1),bold=True,font_size='18sp')
        button_to_four.pos_hint = {'center_x': 0.5, 'center_y': 0.42}
        button_to_four.bind(on_press=lambda x: self.change_screen("four"))
        layout.add_widget(button_to_four)

        button_to_five = RoundedButton(text="Ver Estado Alimento", size_hint=(None, None), size=("250dp", "60dp"),color=(0, 0, 0, 1),bold=True,font_size='18sp')
        button_to_five.pos_hint = {'center_x': 0.5, 'center_y': 0.28}
        button_to_five.bind(on_press=lambda x: self.change_screen("five"))
        layout.add_widget(button_to_five)

        self.add_widget(layout)

    def change_screen(self, screen_name):
        self.manager.current = screen_name

    def update_rect_bottom(self, instance, value):
        self.rect_bottom.pos = instance.pos
        self.rect_bottom.size = instance.size

    def update_rect_gray(self, instance, value):
        self.rect_gray.pos = instance.pos
        self.rect_gray.size = instance.size

    def show_popup(self, instance):
        # Contenedor principal
        content = BoxLayout(orientation="vertical", spacing=20, padding=20)

        # ScrollView para manejar notificaciones
        scroll_view = ScrollView(size_hint=(1, 0.7))

        # Contenedor de las notificaciones
        notification_list = BoxLayout(orientation="vertical", spacing=30, size_hint_y=None)
        notification_list.bind(minimum_height=notification_list.setter("height"))

        # Agregar notificaciones a la lista
        notifications = [
            "Notificación 1: Sistema actualizado.",
            "Notificación 2: Nuevo mensaje recibido que es muy largo y debe ajustarse al ancho de la ventana.",
            "Notificación 3: La tarea se completó con éxito. Por favor, revise los detalles.",
            "Notificación 4: Error al cargar el archivo. Intente nuevamente o contacte al soporte.",
            "Notificación 5: Se recibió una nueva solicitud con información adicional que debe revisarse detenidamente.",
        ]
        for notif in notifications:
```

```
            "Notificación 5: Se recibió una nueva solicitud con información adicional que debe revisarse detenidamente.",
        ]
        for notif in notifications:
            # Crear un Label con ajuste de texto
            label = Label(
                text=notif,
                size_hint_y=None,
                height=30,  # Establecer una altura mínima para cada notificación
                halign="left",
                valign="middle",
                text_size=(scroll_view.width + 180, None),  # Ajustar el texto al ancho del ScrollView con padding
            )
            label.bind(size=lambda lbl, _: lbl.texture_update())  # Actualizar para que el texto ajuste dinámicamente
            notification_list.add_widget(label)

        scroll_view.add_widget(notification_list)

        # Botón de cerrar dentro del popup
        close_button = Button(
            text="Cerrar",
            size_hint=(None, None),
            size=("150dp", "40dp"),
            pos_hint={"center_x": 0.5},
        )
        close_button.bind(on_press=lambda x: popup.dismiss())

        # Añadir elementos al contenido
        content.add_widget(scroll_view)
        content.add_widget(close_button)

        # Crear el popup
        popup = Popup(
            title="Lista de Notificaciones",
            content=content,
            size_hint=(0.95, 0.7),  # Tamaño relativo a la pantalla
            auto_dismiss=False,  # Requiere acción para cerrarse
        )

        popup.open()



class FourScreen(Screen):
    def __init__(self, **kwargs):
        super(FourScreen, self).__init__(**kwargs)
        layout = FloatLayout()

        # Imagen del logo
        image = Image(
            source='logo.png',
            allow_stretch=True,
            keep_ratio=False,
            size_hint=(None, None),
            size=(180, 160)
        )
        image.pos_hint = {'center_x': 0.5, 'center_y': 0.80}
        layout.add_widget(image, index=len(layout.children))

        # Parte inferior amarilla
        bottom_half = FloatLayout(size_hint=(1, 0.5))
        with bottom_half.canvas:
            Color(1, 1, 0, 0.8)
            self.rect_bottom = RoundedRectangle(size=bottom_half.size, pos=bottom_half.pos)
        bottom_half.bind(size=self.update_rect_bottom, pos=self.update_rect_bottom)
        bottom_half.pos_hint = {'x': 0, 'y': 0}
        layout.add_widget(bottom_half)
```

```
        bottom_half.bind(size=self.update_rect_bottom, pos=self.update_rect_bottom)
        bottom_half.pos_hint = {'x': 0, 'y': 0}
        layout.add_widget(bottom_half)

        # Caja gris flotante
        gray_box = FloatLayout(size_hint=(None, None), size=("300dp", "350dp"))
        gray_box.pos_hint = {'center_x': 0.5, 'y': 0.06}
        with gray_box.canvas:
            Color(1, 1, 1, 1)
            self.rect_gray = RoundedRectangle(
                size=gray_box.size,
                pos=gray_box.pos,
                radius=[(20, 20), (20, 20), (20, 20), (20, 20)]
            )
        gray_box.bind(size=self.update_rect_gray, pos=self.update_rect_gray)
        layout.add_widget(gray_box)

        # Caja de texto para mostrar información con borde negro
        info_box = FloatLayout(size_hint=(None, None), size=("250dp", "180dp"))
        info_box.pos_hint = {'center_x': 0.5, 'center_y': 0.44}
        with info_box.canvas:
            Color(0.9, 0.9, 0.9, 1)
            self.info_background = RoundedRectangle (
                size=info_box.size,
                pos=info_box.pos,
                radius=[(10, 10), (10, 10), (10, 10), (10, 10)]
            )
            Color(0, 0, 0, 1)
            self.info_border = Line(
                rounded_rectangle=(info_box.x, info_box.y, info_box.width, info_box.height, 10),
                width=1.5
            )
        info_box.bind(size=self.update_info_background, pos=self.update_info_background)

        # Etiqueta para mostrar el progreso
        self.label = Label(
            text='Cargando datos...',
            font_size=19,
            size_hint=(None, None),
            size=(info_box.width - 20, 50),
            pos_hint={'center_x': 0.5, 'top': 0.85},
            color=(0, 0, 0, 1)
        )
        info_box.add_widget(self.label)
        layout.add_widget(info_box)

        # Botón para regresar
        back_button = RoundedButton(
            text="Volver",
            size_hint=(None, None),
            size=("250dp", "60dp"),
            color=(1, 1, 1, 1),
            bg_color=(1, 0, 0, 1),
            bold=True,
            font_size='18sp'
        )
        back_button.pos_hint = {'center_x': 0.5, 'center_y': 0.2}
        back_button.bind(on_press=lambda x: self.change_screen("second"))
        layout.add_widget(back_button)

        self.add_widget(layout)

    def on_enter(self):
        # Llamar a la función de medición al entrar en esta pantalla
        self.check_water_level()

    def check_water_level(self):
```

```
    def check_water_level(self):
        try:
            # Obtener distancia del sensor ultrasonico
            distancia = medir_distancia()
            print(f"Distancia medida: {distancia} cm")  # Depuración para confirmar el valor

            # Determinar estado basado en distancia
            estado = "Lleno" if distancia <= 10 else "Vacío"

            # Actualizar etiqueta con la distancia real y el estado
            self.label.text = f"Distancia: {distancia:.2f} cm\nNivel de Agua: {estado}"
        except Exception as e:
            print(f"Error al medir distancia: {e}")
            self.label.text = "Error al obtener distancia"

    def update_info_background(self, instance, value):
        self.info_background.size = instance.size
        self.info_background.pos = instance.pos
        self.info_border.rounded_rectangle = (
            instance.x, instance.y, instance.width, instance.height, 10
        )

    def update_rect_bottom(self, instance, value):
        self.rect_bottom.pos = instance.pos
        self.rect_bottom.size = instance.size

    def update_rect_gray(self, instance, value):
        self.rect_gray.pos = instance.pos
        self.rect_gray.size = instance.size

    def change_screen(self, screen_name):
        self.manager.current = screen_name

    def __init__(self, **kwargs):
        super(FourScreen, self).__init__(**kwargs)
        layout = FloatLayout()

        # Imagen del logo
        image = Image(
            source='logo.png',
            allow_stretch=True,
            keep_ratio=False,
            size_hint=(None, None),
            size=(180, 160)
        )
        image.pos_hint = {'center_x': 0.5, 'center_y': 0.80}
        layout.add_widget(image, index=len(layout.children))

        # Parte inferior amarilla
        bottom_half = FloatLayout(size_hint=(1, 0.5))
        with bottom_half.canvas:
            Color(1, 1, 0, 0.8)
            self.rect_bottom = RoundedRectangle(size=bottom_half.size, pos=bottom_half.pos)
        bottom_half.bind(size=self.update_rect_bottom, pos=self.update_rect_bottom)
        bottom_half.pos_hint = {'x': 0, 'y': 0}
        layout.add_widget(bottom_half)

        # Caja gris flotante
        gray_box = FloatLayout(size_hint=(None, None), size=("300dp", "350dp"))
        gray_box.pos_hint = {'center_x': 0.5, 'y': 0.06}
        with gray_box.canvas:
            Color(1, 1, 1, 1)
            self.rect_gray = RoundedRectangle(
                size=gray_box.size,
                pos=gray_box.pos,
                radius=[(20, 20), (20, 20), (20, 20), (20, 20)]
            )
```

```
            pos=gray_box.pos,
            radius=[(20, 20), (20, 20), (20, 20), (20, 20)]
        )
    gray_box.bind(size=self.update_rect_gray, pos=self.update_rect_gray)
    layout.add_widget(gray_box)

    # Botón redondeado para regresar (rojo)
    back_button = RoundedButton(
        text="Volver",
        size_hint=(None, None),
        size=("250dp", "35dp"),
        color=(1, 1, 1, 1),
        bold=True,
        font_size='18sp',
        bg_color=(1, 0, 0, 1)  # Rojo
    )
    back_button.pos_hint = {'center_x': 0.5, 'center_y': 0.1}
    back_button.bind(on_press=lambda x: self.change_screen("second"))
    layout.add_widget(back_button)

    # Botón redondeado para "Suministrar Agua"
    button_suministrar = RoundedButton(
        text="Suministrar Agua",
        size_hint=(None, None),
        size=("250dp", "35dp"),
        color=(0, 0, 0, 1),
        bold=True,
        font_size='18sp'
    )
    button_suministrar.pos_hint = {'center_x': 0.5, 'center_y': 0.25}
    button_suministrar.bind(on_press=lambda x: print("Suministrando Agua")) # AGREGAR FUNCION
    layout.add_widget(button_suministrar)

    # Botón redondeado para "Limpiar Agua"
    button_limpiar = RoundedButton(
        text="Limpiar Agua",
        size_hint=(None, None),
        size=("250dp", "35dp"),
        color=(0, 0, 0, 1),
        bold=True,
        font_size='18sp'
    )
    button_limpiar.pos_hint = {'center_x': 0.5, 'center_y': 0.186}
    button_limpiar.bind(on_press=lambda x: print("Limpiando Agua")) # AGREGAR FUNCION
    layout.add_widget(button_limpiar)

    # Caja de texto para mostrar información con borde negro
    info_box = FloatLayout(size_hint=(None, None), size=("250dp", "180dp"))
    info_box.pos_hint = {'center_x': 0.5, 'center_y': 0.44}

    with info_box.canvas:
        # Fondo gris claro
        Color(0.9, 0.9, 0.9, 1)
        self.info_background = RoundedRectangle (
            size=info_box.size,
            pos=info_box.pos,
            radius=[(10, 10), (10, 10), (10, 10), (10, 10)]
        )
        # Linea negra para el borde
        Color(0, 0, 0, 1)
        self.info_border = Line(
            rounded_rectangle=(info_box.x, info_box.y, info_box.width, info_box.height, 10),
            width=1.5
        )

    # Actualizar el tamaño y posición del fondo y borde
    info_box.bind(size=self.update_info_background, pos=self.update_info_background)
```

```
        # Actualizar el tamaño y posición del fondo y borde
        info_box.bind(size=self.update_info_background, pos=self.update_info_background)

        # Etiqueta para mostrar el progreso
        self.label = Label(
                text='Distancia: 0.00 cm\nNivel de Agua: 0.00%\nEstado: Desconocido\nEstado de pH: Neutro',
                font_size=19,
                size_hint=(None, None),
                size=(info_box.width - 20, 50),
                pos_hint={'center_x': 0.5, 'top': 0.85},
                color=(0, 0, 0, 1)  # Color negro para el texto
        )



        # Barra de progreso
        self.ph = 0.0  # pH inicial
        self.ph_state = self.determine_ph_state(self.ph)
        self.distance = 0.00  # Distancia inicial en cm (simulación)
        self.max_distance = 10.0  # Distancia máxima en cm
        self.min_distance = 0.0  # Distancia minima en cm
        self.percentage = self.calculate_percentage(self.distance)  # Porcentaje inicial
        self.state = self.determine_state(self.distance)

        self.progress_bar = ProgressBar(max=100, value=0, size_hint=(None, None), width=info_box.width - 20, height=25)
        self.progress_bar.pos_hint = {'center_x': 0.5, 'top': 0.44}

        # Agregar la etiqueta y la barra de progreso a la caja gris
        info_box.add_widget(self.label)
        info_box.add_widget(self.progress_bar)

        # Agregar la caja de texto con la barra de progreso y la etiqueta al layout principal
        layout.add_widget(info_box)

        # Botón para simular el cambio de distancia
        button = Button(text="Actualizar",
                        size_hint=(None, None),
                        size=(100, 50))
        button.pos_hint = {'center_x': 0.5, 'center_y': 0.34}
        button.bind(on_press=self.simulate_distance_change)
        layout.add_widget(button)

        # Actualizar el progreso visual cada segundo
        Clock.schedule_interval(self.update_progress, 1)

        self.add_widget(layout)

    def update_info_background(self, instance, value):
        """
        Actualiza el tamaño y posición del fondo de la caja de texto y el borde negro.
        """
        self.info_background.size = instance.size
        self.info_background.pos = instance.pos
        self.info_border.rounded_rectangle = (
            instance.x, instance.y, instance.width, instance.height, 10
        )

    def change_screen(self, screen_name):
        self.label.text = 'Distancia: 0.00 cm\nNivel de Agua: 0.00%\nEstado: Desconocido'
        self.progress_bar.value = 0  # Resetear la barra de progreso
        self.distance = 0.00  # Reiniciar distancia a cero
        self.percentage = 0.00  # Reiniciar porcentaje a cero
        self.state = 'Desconocido'  # Reiniciar estado
        self.manager.current = screen_name

    def update_rect_bottom(self, instance, value):
        self.rect_bottom.pos = instance.pos
```

```
576
577     def update_rect_bottom(self, instance, value):
578         self.rect_bottom.pos = instance.pos
579         self.rect_bottom.size = instance.size
580
581     def update_rect_gray(self, instance, value):
582         self.rect_gray.pos = instance.pos
583         self.rect_gray.size = instance.size
584
585     def PRUEBA(self, boton, texto):
586         """
587         Actualiza el texto de la caja de información según el botón presionado.
588         """
589         if boton == 1:
590             self.update_information(texto)
591         else:
592             self.update_information("Otro texto o acción especifica")
593
594     def determine_state(self, distance):
595         if distance >= 7:
596             return "Lleno"
597         elif 3 < distance < 7:
598             return "Regular"
599         elif distance <= 3:
600             return "Insuficiente"
601         return "Desconocido"
602
603     def determine_ph_state(self, ph):
604         if 6.5 <= ph <= 7.5:
605             return "Óptimo"
606         elif 5.5 <= ph < 6.5:
607             return "Salubre"
608         elif ph < 5.5 or ph > 7.5:
609             return "Insalubre"
610         return "Desconocido"
611
612
613     def calculate_percentage(self, distance):
614         return max(0, min(100, 100 * (self.max_distance - distance) / (self.max_distance - self.min_distance)))
615
616     def simulate_distance_change(self, instance):
617         # Simula un cambio de distancia
618         self.distance = random.uniform(self.min_distance, self.max_distance)
619         self.percentage = self.calculate_percentage(self.distance)
620         self.state = self.determine_state(self.distance)
621
622         self.ph = random.uniform(5.0, 8.0)  # Por ejemplo, el pH puede variar entre 5.0 y 8.0
623         self.ph_state = self.determine_ph_state(self.ph)
624
625         # Actualiza la etiqueta con la nueva distancia, porcentaje y estado
626         self.label.text = (
627             f'Distancia: {self.distance:.2f} cm\n'
628             f'Nivel de Agua: {self.percentage:.2f}%\n'
629             f'Estado: {self.state}'
630             f'Estado de pH: {self.ph_state}'
631         )
632
633     def update_progress(self, dt):
634         # Actualiza la barra de progreso y la etiqueta
635         self.progress_bar.value = self.percentage
636         self.label.text = (
637             f'Distancia: {self.distance:.2f} cm\n'
638             f'Nivel de Agua: {self.percentage:.2f}%\n'
639             f'Estado: {self.state}\n'
640             f'Estado de pH: {self.ph_state}'
641         )
642
```

```
from kivy.uix.progressbar import ProgressBar
from kivy.uix.label import Label
from kivy.uix.floatlayout import FloatLayout
from kivy.graphics import Color, Line, RoundedRectangle

class FiveScreen(Screen):
    def __init__(self, **kwargs):
        super(FiveScreen, self).__init__(**kwargs)

        layout = FloatLayout()

        # Información inicial de comida
        self.cantidad_comida = 5.00  # Ejemplo de cantidad en gramos, puede cambiar dinámicamente.
        self.estado_comida = "En Proceso"  # Estado puede ser "En Proceso", "Suministrado", etc.

        # Imagen del logo
        image = Image(
            source='logo.png',
            allow_stretch=True,
            keep_ratio=False,
            size_hint=(None, None),
            size=(180, 160)
        )
        image.pos_hint = {'center_x': 0.5, 'center_y': 0.80}
        layout.add_widget(image, index=len(layout.children))

        # Caja gris flotante
        gray_box = FloatLayout(size_hint=(None, None), size=("300dp", "350dp"))
        gray_box.pos_hint = {'center_x': 0.5, 'y': 0.06}
        with gray_box.canvas:
            Color(1, 1, 1, 1)
            self.rect_gray = RoundedRectangle(
                size=gray_box.size,
                pos=gray_box.pos,
                radius=[(20, 20), (20, 20), (20, 20), (20, 20)]
            )
        gray_box.bind(size=self.update_rect_gray, pos=self.update_rect_gray)
        layout.add_widget(gray_box)

        # Botón para "Suministrar Alimento"
        button_suministrar = RoundedButton(
            text="Suministrar Alimento",
            size_hint=(None, None),
            size=("250dp", "43dp"),
            color=(0, 0, 0, 1),
            bold=True,
            font_size='18sp'
        )
        button_suministrar.pos_hint = {'center_x': 0.5, 'center_y': 0.22}
        button_suministrar.bind(on_press=lambda x: self.update_information("Suministrando Alimento"))
        layout.add_widget(button_suministrar)

        # Caja de texto para mostrar información con borde negro
        info_box = FloatLayout(size_hint=(None, None), size=("250dp", "180dp"))
        info_box.pos_hint = {'center_x': 0.5, 'center_y': 0.44}
        with info_box.canvas:
            Color(0.9, 0.9, 0.9, 1)
            self.info_background = RoundedRectangle(
                size=info_box.size,
                pos=info_box.pos,
                radius=[(10, 10), (10, 10), (10, 10), (10, 10)]
            )
            Color(0, 0, 0, 1)  # Línea negra para el borde
            self.info_border = Line(
                rounded_rectangle=(info_box.x, info_box.y, info_box.width, info_box.height, 10),
                width=1.5
            )
```

```
        info_box.bind(size=self.update_info_background, pos=self.update_info_background)

        # Texto dentro de la caja de información
        self.info_label = Label(
            text="Cantidad de Comida: 5.00g\nEstado: En Proceso",
            size_hint=(1, 1),
            color=(0, 0, 0, 1),  # Texto negro
            halign="center",
            valign="middle"
        )
        self.info_label.pos_hint = {'center_x': 0.5, 'top': 1.3}
        self.info_label.bind(size=self.update_label_size)
        info_box.add_widget(self.info_label)

        # Barra de progreso para la cantidad de comida
        self.progress_bar = ProgressBar(
            size_hint=(None, None),
            size=("200dp", "30dp"),
            max=10,  # Valor máximo es 10.00g
            value=self.cantidad_comida  # Valor actual es la cantidad de comida
        )
        self.progress_bar.pos_hint = {'center_x': 0.5, 'center_y': 0.2}
        layout.add_widget(self.progress_bar)

        layout.add_widget(info_box)

        self.add_widget(layout)

    def update_information(self, text):
        """
        Actualiza el texto mostrado en la etiqueta de información.
        """
        self.info_label.text = f"Cantidad de Comida: {self.cantidad_comida:.2f}g\nEstado: {self.estado_comida}"
        self.progress_bar.value = self.cantidad_comida

    def update_label_size(self, instance, value):
        """
        Ajusta el texto de la etiqueta para centrarse correctamente.
        """
        instance.text_size = instance.size

    def update_info_background(self, instance, value):
        """
        Actualiza el tamaño y posición del fondo de la caja de texto y el borde negro.
        """
        self.info_background.size = instance.size
        self.info_background.pos = instance.pos
        self.info_border.rounded_rectangle = (
            instance.x, instance.y, instance.width, instance.height, 10
        )

    def update_rect_gray(self, instance, value):
        self.rect_gray.pos = instance.pos
        self.rect_gray.size = instance.size

    def change_screen(self, screen_name):
        self.info_label.text = "Aquí se mostrará información."
        self.manager.current = screen_name


# Inicialización de la aplicación
class MyApp(App):
    def build(self):
```

```
# Inicialización de la aplicación
class MyApp(App):
    def build(self):
        # Crear el ScreenManager
        sm = ScreenManager()

        # Agregar las pantallas al ScreenManager
        sm.add_widget(MainScreen(name="main"))
        sm.add_widget(SecondScreen(name="second"))
        sm.add_widget(FourScreen(name="four"))
        sm.add_widget(FiveScreen(name="five"))

        return sm


if __name__ == "__main__":
    MyApp().run()




import RPi.GPIO as GPIO
import time

# Configuración del sensor ultrasonico
GPIO.setmode(GPIO.BCM)
TRIG = 2  # Pin Trigger
ECHO = 17  # Pin Echo

GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

def medir_distancia():
    # Asegurarse de que Trigger esté en bajo
    GPIO.output(TRIG, False)
    time.sleep(2)

    # Enviar pulso de 10us al Trigger
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    # Medir tiempo de respuesta del Echo
    while GPIO.input(ECHO) == 0:
        pulso_inicio = time.time()

    while GPIO.input(ECHO) == 1:
        pulso_fin = time.time()

    # Calcular la duración del pulso
    duracion_pulso = pulso_fin - pulso_inicio

    # Calcular distancia (34300 cm/s es la velocidad del sonido)
    distancia = duracion_pulso * 17150  # cm
    distancia = round(distancia, 2)

    return distancia


class FourScreen(Screen):
    def __init__(self, **kwargs):
        super(FourScreen, self).__init__(**kwargs)
        layout = FloatLayout()

        # Imagen del logo
        image = Image(
```

```
        # Imagen del logo
        image = Image(
            source='logo.png',
            allow_stretch=True,
            keep_ratio=False,
            size_hint=(None, None),
            size=(180, 160)
        )
        image.pos_hint = {'center_x': 0.5, 'center_y': 0.80}
        layout.add_widget(image, index=len(layout.children))

        # Parte inferior amarilla
        bottom_half = FloatLayout(size_hint=(1, 0.5))
        with bottom_half.canvas:
            Color(1, 1, 0, 0.8)
            self.rect_bottom = RoundedRectangle(size=bottom_half.size, pos=bottom_half.pos)
        bottom_half.bind(size=self.update_rect_bottom, pos=self.update_rect_bottom)
        bottom_half.pos_hint = {'x': 0, 'y': 0}
        layout.add_widget(bottom_half)

        # Caja gris flotante
        gray_box = FloatLayout(size_hint=(None, None), size=("300dp", "350dp"))
        gray_box.pos_hint = {'center_x': 0.5, 'y': 0.06}
        with gray_box.canvas:
            Color(1, 1, 1, 1)
            self.rect_gray = RoundedRectangle(
                size=gray_box.size,
                pos=gray_box.pos,
                radius=[(20, 20), (20, 20), (20, 20), (20, 20)]
            )
        gray_box.bind(size=self.update_rect_gray, pos=self.update_rect_gray)
        layout.add_widget(gray_box)

        # Caja de texto para mostrar información con borde negro
        info_box = FloatLayout(size_hint=(None, None), size=("250dp", "180dp"))
        info_box.pos_hint = {'center_x': 0.5, 'center_y': 0.44}
        with info_box.canvas:
            Color(0.9, 0.9, 0.9, 1)
            self.info_background = RoundedRectangle (
                size=info_box.size,
                pos=info_box.pos,
                radius=[(10, 10), (10, 10), (10, 10), (10, 10)]
            )
            Color(0, 0, 0, 1)
            self.info_border = Line(
                rounded_rectangle=(info_box.x, info_box.y, info_box.width, info_box.height, 10),
                width=1.5
            )
        info_box.bind(size=self.update_info_background, pos=self.update_info_background)

        # Etiqueta para mostrar el progreso
        self.label = Label(
            text='Cargando datos...',
            font_size=19,
            size_hint=(None, None),
            size=(info_box.width - 20, 50),
            pos_hint={'center_x': 0.5, 'top': 0.85},
            color=(0, 0, 0, 1)
        )
        info_box.add_widget(self.label)
        layout.add_widget(info_box)

        # Botón para regresar
        back_button = RoundedButton(
            text="Volver",
            size_hint=(None, None),
```

```
class FourScreen(Screen):
            size_hint=(None, None),
            size=("250dp", "60dp"),
            color=(1, 1, 1, 1),
            bg_color=(1, 0, 0, 1),
            bold=True,
            font_size='18sp'
        )
        back_button.pos_hint = {'center_x': 0.5, 'center_y': 0.2}
        back_button.bind(on_press=lambda x: self.change_screen("second"))
        layout.add_widget(back_button)

        self.add_widget(layout)

    def on_enter(self):
        # Llamar a la función de medición al entrar en esta pantalla
        self.check_water_level()

    def check_water_level(self):
        try:
            # Obtener distancia del sensor ultrasonico
            distancia = medir_distancia()
            print(f"Distancia medida: {distancia} cm")  # Depuración para confirmar el valor

            # Determinar estado basado en distancia
            estado = "Lleno" if distancia <= 10 else "Vacío"

            # Actualizar etiqueta con la distancia real y el estado
            self.label.text = f"Distancia: {distancia:.2f} cm\nNivel de Agua: {estado}"
        except Exception as e:
            print(f"Error al medir distancia: {e}")
            self.label.text = "Error al obtener distancia"

    def update_info_background(self, instance, value):
        self.info_background.size = instance.size
        self.info_background.pos = instance.pos
        self.info_border.rounded_rectangle = (
            instance.x, instance.y, instance.width, instance.height, 10
        )

    def update_rect_bottom(self, instance, value):
        self.rect_bottom.pos = instance.pos
        self.rect_bottom.size = instance.size

    def update_rect_gray(self, instance, value):
        self.rect_gray.pos = instance.pos
        self.rect_gray.size = instance.size

    def change_screen(self, screen_name):
        self.manager.current = screen_name

    def __init__(self, **kwargs):
        super(FourScreen, self).__init__(**kwargs)
        layout = FloatLayout()

        # Imagen del logo
        image = Image(
            source='logo.png',
            allow_stretch=True,
            keep_ratio=False,
            size_hint=(None, None),
            size=(180, 160)
        )
        image.pos_hint = {'center_x': 0.5, 'center_y': 0.80}
        layout.add_widget(image, index=len(layout.children))

        # Parte inferior amarilla
```

```
        # Parte inferior amarilla
        bottom_half = FloatLayout(size_hint=(1, 0.5))
        with bottom_half.canvas:
            Color(1, 1, 0, 0.8)
            self.rect_bottom = RoundedRectangle(size=bottom_half.size, pos=bottom_half.pos)
        bottom_half.bind(size=self.update_rect_bottom, pos=self.update_rect_bottom)
        bottom_half.pos_hint = {'x': 0, 'y': 0}
        layout.add_widget(bottom_half)

        # Caja gris flotante
        gray_box = FloatLayout(size_hint=(None, None), size=("300dp", "350dp"))
        gray_box.pos_hint = {'center_x': 0.5, 'y': 0.06}
        with gray_box.canvas:
            Color(1, 1, 1, 1)
            self.rect_gray = RoundedRectangle(
                size=gray_box.size,
                pos=gray_box.pos,
                radius=[(20, 20), (20, 20), (20, 20), (20, 20)]
            )
        gray_box.bind(size=self.update_rect_gray, pos=self.update_rect_gray)
        layout.add_widget(gray_box)

        # Caja de texto para mostrar información con borde negro
        info_box = FloatLayout(size_hint=(None, None), size=("250dp", "180dp"))
        info_box.pos_hint = {'center_x': 0.5, 'center_y': 0.44}
        with info_box.canvas:
            Color(0.9, 0.9, 0.9, 1)
            self.info_background = RoundedRectangle (
                size=info_box.size,
                pos=info_box.pos,
                radius=[(10, 10), (10, 10), (10, 10), (10, 10)]
            )
            Color(0, 0, 0, 1)
            self.info_border = Line(
                rounded_rectangle=(info_box.x, info_box.y, info_box.width, info_box.height, 10),
                width=1.5
            )
        info_box.bind(size=self.update_info_background, pos=self.update_info_background)

        # Etiqueta para mostrar el progreso
        self.label = Label(
            text='Distancia: 0.00 cm\nNivel de Agua: Vacío',
            font_size=19,
            size_hint=(None, None),
            size=(info_box.width - 20, 50),
            pos_hint={'center_x': 0.5, 'top': 0.85},
            color=(0, 0, 0, 1)
        )
        info_box.add_widget(self.label)
        layout.add_widget(info_box)

        # Botón para medir la distancia
        button = RoundedButton(
            text="Medir Nivel de Agua",
            size_hint=(None, None),
            size=("250dp", "60dp"),
            color=(0, 0, 0, 1),
            bold=True,
            font_size='18sp'
        )
        button.pos_hint = {'center_x': 0.5, 'center_y': 0.3}
        button.bind(on_press=self.check_water_level)
        layout.add_widget(button)

        self.add_widget(layout)
```

```
1034
1035        def check_water_level(self, instance):
1036            distancia = medir_distancia()
1037            estado = "Lleno" if distancia <= 10 else "Vacío"
1038            self.label.text = f"Distancia: {distancia:.2f} cm\nNivel de Agua: {estado}"
1039
1040        def update_info_background(self, instance, value):
1041            self.info_background.size = instance.size
1042            self.info_background.pos = instance.pos
1043            self.info_border.rounded_rectangle = (
1044                instance.x, instance.y, instance.width, instance.height, 10
1045            )
1046
1047        def update_rect_bottom(self, instance, value):
1048            self.rect_bottom.pos = instance.pos
1049            self.rect_bottom.size = instance.size
1050
1051        def update_rect_gray(self, instance, value):
1052            self.rect_gray.pos = instance.pos
1053            self.rect_gray.size = instance.size
1054
1055
1056    # Integrar métodos importados y funciones de hardware
1057    from peso import hx
1058    from servo import set_angle
1059    from valvula import GPIO, RELAY_PIN
1060    from ph import arduino
1061    from copia8 import medir_distancia
1062
1063    # Modificar la clase 'FourScreen' para agregar funcionalidades al botón de ver estado de agua
1064    def check_water_level(self):
1065        try:
1066            # Obtener distancia del sensor ultrasónico
1067            distancia = medir_distancia()
1068            estado = "Lleno" if distancia <= 10 else "Vacío"
1069            # Actualizar etiqueta con distancia y estado
1070            self.label.text = f"Distancia: {distancia:.2f} cm\nNivel de Agua: {estado}"
1071        except Exception as e:
1072            self.label.text = f"Error al medir distancia: {e}"
1073
1074    def check_ph_status(self):
1075        try:
1076            if arduino.in_waiting > 0:  # Verificar datos disponibles
1077                data = arduino.readline().decode('utf-8').strip()
1078                self.label.text += f"\nEstado de pH: {data}"
1079        except Exception as e:
1080            self.label.text += f"\nError al leer pH: {e}"
1081
1082    # Modificar la clase 'FiveScreen' para agregar funcionalidades relacionadas con el estado de alimento
1083    def check_food_weight(self):
1084        try:
1085            # Leer peso del sensor
1086            weight = hx.get_weight(5)
1087            self.info_label.text = f"Peso: {weight:.2f} gramos\nEstado: Suficiente"
1088            hx.power_down()
1089            hx.power_up()
1090        except Exception as e:
1091            self.info_label.text = f"Error al medir peso: {e}"
1092
1093    def supply_food(self):
1094        try:
1095            set_angle(90)  # Activar el servo para suministrar alimento
1096            self.info_label.text = "Suministrando alimento..."
1097        except Exception as e:
1098            self.info_label.text = f"Error al suministrar alimento: {e}"
1099
```

```
1099
1100  # Modificar la clase 'SecondScreen' para agregar suministro de agua
1101  def supply_water(self):
1102      try:
1103          GPIO.output(RELAY_PIN, GPIO.HIGH)  # Activar relé
1104          Clock.schedule_once(lambda dt: GPIO.output(RELAY_PIN, GPIO.LOW), 5)  # Desactivar después de 5 segundos
1105          self.label.text = "Suministrando agua..."
1106      except Exception as e:
1107          self.label.text = f"Error al suministrar agua: {e}"
1108
1109  # Agregar estas funciones a las clases respectivas en tiempo de ejecución
1110  FourScreen.check_water_level = check_water_level
1111  FourScreen.check_ph_status = check_ph_status
1112  FiveScreen.check_food_weight = check_food_weight
1113  FiveScreen.supply_food = supply_food
1114  SecondScreen.supply_water = supply_water
1115
1116
1117  # Agregar un botón de 'Volver' en la pantalla FiveScreen
1118  def add_back_button(self):
1119      back_button = RoundedButton(
1120          text="Volver",
1121          size_hint=(None, None),
1122          size=("250dp", "60dp"),
1123          color=(1, 1, 1, 1),
1124          bg_color=(1, 0, 0, 1),  # Rojo
1125          bold=True,
1126          font_size='18sp'
1127      )
1128      back_button.pos_hint = {'center_x': 0.5, 'center_y': 0.1}
1129      back_button.bind(on_press=lambda x: self.change_screen("second"))
1130      self.add_widget(back_button)
1131
1132  # Asignar la función a la clase FiveScreen
1133  FiveScreen.add_back_button = add_back_button
1134
1135  # Llamar a la función en la inicialización de FiveScreen
1136  original_init = FiveScreen.__init__
1137
1138  def new_init(self, **kwargs):
1139      original_init(self, **kwargs)
1140      self.add_back_button()
1141
1142  FiveScreen.__init__ = new_init
1143
1144  # Asegurar que el botón de "Volver" esté presente en FiveScreen
1145  def new_init(self, **kwargs):
1146      original_init(self, **kwargs)
1147      # Botón de "Volver"
1148      back_button = RoundedButton(
1149          text="Volver",
1150          size_hint=(None, None),
1151          size=("250dp", "60dp"),
1152          color=(1, 1, 1, 1),
1153          bg_color=(1, 0, 0, 1),  # Rojo
1154          bold=True,
1155          font_size='18sp'
1156      )
1157      back_button.pos_hint = {'center_x': 0.5, 'center_y': 0.1}
1158      back_button.bind(on_press=lambda x: self.change_screen("second"))
1159      self.add_widget(back_button)
1160
1161  # Sobrescribir el método de inicialización para FiveScreen
1162  FiveScreen.__init__ = new_init
1163
```

Ilustración 31 : Interfaz Gráfica

5.3.2 Implementar sensor ultrasónico para medición nivel de agua

```python
# -*- coding: utf-8 -*-

import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
TRIG = 2  # Pin Trigger
ECHO = 17  # Pin Echo

GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

def medir_distancia():
    # Aseg□rate de que Trigger est□ en bajo
    GPIO.output(TRIG, False)
    time.sleep(2)

    # Enviar pulso de 10us al Trigger
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    # Medir tiempo de respuesta del Echo
    while GPIO.input(ECHO) == 0:
        pulso_inicio = time.time()

    while GPIO.input(ECHO) == 1:
        pulso_fin = time.time()

    # Calcular la duraci□n del pulso
    duracion_pulso = pulso_fin - pulso_inicio

    # Calcular distancia (34300 cm/s es la velocidad del sonido)
    distancia = duracion_pulso * 17150  # cm
    distancia = round(distancia, 2)

    return distancia

try:
    while True:
        distancia = medir_distancia()
        print("Distancia:", distancia, "cm")
        time.sleep(1)
except KeyboardInterrupt:
    print("\nSaliendo...")
finally:
    GPIO.cleanup()

```

Ilustración 32 : Sensor ultrasónico

5.3.3 Implementar servomotor para generar la dispensación del alimento

```python
# -*- coding: utf-8 -*-

import RPi.GPIO as GPIO
import time

# Configuracin del GPIO
GPIO.setmode(GPIO.BCM)  # Usar la numeracin BCM
GPIO.setup(26, GPIO.OUT)  # Configurar GPIO 26 como salida

# Configurar PWM en el pin 26 con una frecuencia de 50 Hz
pwm = GPIO.PWM(26, 50)
pwm.start(0)  # Iniciar PWM con ciclo de trabajo 0%

def set_angle(angle):
    """Configura el ngulo del servo (0 a 180 grados)."""
    duty_cycle = 2 + (angle / 18)  # Mapear ngulo a ciclo de trabajo
    GPIO.output(26, True)
    pwm.ChangeDutyCycle(duty_cycle)
    time.sleep(0.5)
    GPIO.output(26, False)
    pwm.ChangeDutyCycle(0)

try:
    # Repetir el movimiento 4 veces
    for _ in range(4):
        # Girar de 0 a 180
        set_angle(0)
        time.sleep(1)
        set_angle(180)
        time.sleep(1)

finally:
    pwm.stop()
    GPIO.cleanup()  # Limpiar la configuracin de los GPIO
```

Ilustración 33 : Servomotor

5.3.4 Implementar sensor de ph mediante arduino conectado a raspberry para lograr identificar nivel de ph

```
void setup() {
  Serial.begin(9600);  // Inicia la comunicación serie
  pinMode(A0, INPUT);  // Configura el pin A0 como entrada para el sensor de pH
}

void loop() {
  int rawValue = analogRead(A0);  // Lee el valor del sensor de pH
  float voltage = rawValue * (5.0 / 1023.0);  // Convierte a voltaje (si estás usando 5V como referencia)

  // Convertir el voltaje a pH
  // Asumimos que 0V corresponde a pH 0 y 5V corresponde a pH 14
  float pH = (14.0 / 5.0) * voltage;

  // Muestra el resultado en el monitor serie
  Serial.print("Raw Value: ");  // Valor analógico bruto
  Serial.print(rawValue);
  Serial.print(", Voltage: ");  // Voltaje calculado
  Serial.print(voltage);
  Serial.print(" V, pH: ");     // Valor de pH
  Serial.println(pH);

  delay(1000);  // Envía datos cada segundo
  }
```

```
import serial

import time

# Configura el puerto serie (ajusta si es necesario)

arduino = serial.Serial('/dev/ttyUSB0', 9600)
time.sleep(2)  # Espera a que el Arduino esté listo

print("Conectado al Arduino. Leyendo datos del sensor de pH...")

while True:

    if arduino.in_waiting > 0:  # Verifica si hay datos disponibles

        data = arduino.readline().decode('utf-8').strip()  # Lee y decodifica los datos

        print(f"Datos recibidos: {data}")
```

Ilustración 34 : Sensor ph

5.3.5 Implementar sensor de peso para medición nivel de alimento

```python
import time
import RPi.GPIO as GPIO
from hx711 import HX711

# Configura los pines GPIO
DT = 5  # Pin de datos (DT) del HX711
SCK = 6  # Pin de reloj (SCK) del HX711

# Configura el modo de los pines GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(DT, GPIO.IN)
GPIO.setup(SCK, GPIO.OUT)

# Crea una instancia del HX711
hx = HX711(DT, SCK)

# Realiza un tare (ajustar a cero)
hx.set_offset(0)
hx.set_scale(2280)  # Este valor de escala puede cambiar dependiendo de tu sensor

# Espera unos segundos para estabilizar el sistema
print("Tareando el sistema...")
time.sleep(2)

# Lee y muestra el peso continuamente
print("Lecturas del sensor de peso:")
while True:
    try:
        # Lee el valor del sensor
        weight = hx.get_weight(5)  # 5 lecturas promedio
        print(f'Peso: {weight} gramos')

        # Resetea el valor a cero
        hx.power_down()
        hx.power_up()

        time.sleep(1)  # Pausa de 1 segundo entre lecturas

    except (KeyboardInterrupt, SystemExit):
        print("Saliendo...")
        GPIO.cleanup()  # Limpia los GPIO al salir del script
        break
```

Ilustración 35 : Sensor peso

5.3.6 Implementar valvula solenoide para dispensar agua

```python
import RPi.GPIO as GPIO
import time

RELAY_PIN = 15

GPIO.setmode(GPIO.BCM)
GPIO.setup(RELAY_PIN, GPIO.OUT)

# Prueba directa para mantener la valvula activa por 5 segundos
try:
    print("Activando valvula")
    GPIO.output(RELAY_PIN, GPIO.HIGH)  # Cambia a LOW si es necesario
    time.sleep(5)

    print("Desactivando valvula")
    GPIO.output(RELAY_PIN, GPIO.LOW)  # Cambia a HIGH si es necesario

except KeyboardInterrupt:
    print("Apagando")
    GPIO.cleanup()

```

Ilustración 36 : Valvula

5.4 Reporte de Revisión

5.4.1. Prueba Número Uno

5.4.1.1. Descripción

En esta primera prueba, se realizó la conexión y prueba inicial de los sensores de ultrasonido utilizando una placa Grove como intermediaria. El objetivo era evaluar el correcto funcionamiento de los sensores al medir niveles de agua y alimento.

5.4.1.2. Resultados obtenidos

El sistema no funcionó como se esperaba debido a la incompatibilidad de la placa Grove con el sistema de control basado en Raspberry Pi. Esto generó errores en las lecturas de los sensores y desconexiones intermitentes.

5.4.1.3 Conclusiones

Se determinó que la placa Grove no era adecuada para este proyecto debido a problemas de compatibilidad. Como resultado, se decidió conectar los sensores directamente a la Raspberry Pi para eliminar posibles intermediarios que causaban fallos.

5.4.1. Prueba Número Dos

5.4.1.1. Descripción

En esta segunda prueba, los sensores de ultrasonido fueron conectados directamente a la Raspberry Pi utilizando una protoboard para facilitar el cableado. Además, se comenzó el desarrollo de la aplicación móvil para monitorear las lecturas de los sensores en tiempo real.

5.4.1.2. Resultados obtenidos

Los sensores de ultrasonido funcionaron correctamente, proporcionando lecturas estables y precisas. La integración con la aplicación móvil fue parcial, permitiendo visualizar los datos básicos de los sensores.

5.4.1.3 Conclusiones

Conectar los sensores directamente a la Raspberry Pi resolvió los problemas de comunicación anteriores. Además, se comprobó que las funcionalidades iniciales de la aplicación móvil eran prometedoras para continuar con su desarrollo.

5.4.1. Prueba Número Tres

5.4.1.1. Descripción

En esta prueba, se conectaron todos los sensores (peso, ultrasonido y pH) directamente a la Raspberry Pi mediante la protoboard. Además, se integraron el servomotor y la válvula solenoide para las pruebas de dispensación de agua y alimento.

5.4.1.2. Resultados obtenidos

Los sensores de peso y pH presentaron problemas en las lecturas, generando valores inconsistentes.
El servomotor funcionó correctamente, pero la válvula solenoide mostró dificultades debido a un voltaje diferente al suministrado por la Raspberry Pi.

5.4.1.3 Conclusiones

Se concluyó que el sensor de pH requería un controlador adicional para operar correctamente, mientras que el sensor de peso necesitaba ajustes de calibración. Asimismo, se identificó la necesidad de utilizar un convertidor de voltaje para la válvula solenoide.

5.4.1. Prueba Número Cuatro

5.4.1.1. Descripción

En esta prueba, el sensor de pH fue conectado a un Arduino para gestionar su operación, y se volvió a probar el sensor de peso tras realizar ajustes en su ensamblaje.

5.4.1.2. Resultados obtenidos

El sensor de pH operó correctamente al estar conectado al Arduino, proporcionando lecturas precisas.
El sensor de peso continuó presentando problemas en la estabilidad de sus lecturas debido a un ensamblaje inadecuado.

5.4.1.3 Conclusiones

El uso de un Arduino para el sensor de pH fue una solución efectiva, pero se necesita una revisión más exhaustiva del ensamblaje del sensor de peso para garantizar su correcto funcionamiento.

5.4.1. Prueba Número Cinco

5.4.1.1. Descripción

En esta etapa final, se integraron todos los componentes del sistema, incluyendo sensores, actuadores y la aplicación móvil. Se realizaron pruebas generales para validar el funcionamiento completo del sistema. Sin embargo, debido a limitaciones de tiempo, no fue posible realizar la prueba de drenaje y limpieza del agua utilizando el desagüe controlado por el sensor de pH y la válvula solenoide.

5.4.1.2. Resultados obtenidos

El sistema operó de manera estable en la mayoría de los aspectos, mostrando lecturas precisas en los sensores de ultrasonido, peso y pH, y actualizando los datos en tiempo real en la aplicación móvil.
El dispensador de alimento funcionó correctamente, y la válvula solenoide para el suministro de agua respondió adecuadamente bajo las condiciones de prueba.
Quedó pendiente la validación del módulo de drenaje y limpieza del agua, específicamente la capacidad de detectar agua insalubre y eliminarla a través del desagüe, por lo que esta funcionalidad no se verificó completamente.

5.4.1.3 Conclusiones

El sistema demostró ser funcional y cumplió con la mayoría de los objetivos planteados,. Sin embargo, la funcionalidad de drenaje y limpieza del agua quedó como un aspecto pendiente que podría abordarse en una fase futura del proyecto.

6. Problemas Encontrados y Soluciones Propuestas

6.1 Problema con la conexión inicial de los sensores

- **Descripción del problema:**
  Los sensores de ultrasonido y peso mostraron lecturas inestables y problemas de comunicación al ser conectados mediante la placa Grove, lo que afectó el correcto funcionamiento del sistema.
- **Solución implementada:**
  Se eliminó la placa Grove y los sensores fueron conectados directamente a la Raspberry Pi mediante una protoboard. Esto elimina intermediarios y permite obtener lecturas estables y precisas.

6.2 Incompatibilidad de voltaje con la válvula solenoide

- **Descripción del problema:**
  La válvula solenoide requería un voltaje superior al proporcionado por la Raspberry Pi, lo que impedía su operación directa.
- **Solución implementada:**
  Se utilizó un convertidor de voltaje para adaptar la salida de la Raspberry Pi a los requerimientos de la válvula, garantizando su funcionamiento adecuado durante las pruebas.

6.3 Inestabilidad en las lecturas del sensor de peso

- **Descripción del problema:**
  El sensor de peso presentó lecturas inconsistentes debido a un ensamblaje inadecuado y problemas en la calibración inicial.
- **Solución implementada:**
  Se realizó un ajuste del ensamblaje y se calibró nuevamente el sensor con métodos más precisos, lo que mejoró significativamente la confiabilidad de las lecturas.

6.4 Limitaciones de tiempo para probar la funcionalidad de drenaje

- **Descripción del problema:**
  No se pudo realizar la prueba del sistema de limpieza y drenaje del agua, específicamente la eliminación de agua insalubre mediante el desagüe, debido al tiempo limitado en la etapa final del proyecto.
- **Solución propuesta:**
  Se dejó planificada esta funcionalidad para una fase futura del proyecto, donde se realizarán pruebas específicas del módulo de drenaje en un entorno controlado para validar su integración con el sensor de pH y la válvula solenoide.

6.5 Dificultades en la conexión remota

- **Descripción del problema:**
  La conexión a la aplicación móvil presentó limitaciones en varias etapas, dificultando el monitoreo y el control remoto del sistema.
- **Solución implementada:**
  Se utilizó la aplicación **RealVNC Viewer** como una alternativa para acceder al sistema de la Raspberry Pi desde dispositivos móviles o computadoras. Esto facilitó el monitoreo y la configuración del sistema en tiempo real.

8. Trabajo a Futuro

El sistema **ChickenCheck** ha cumplido con los objetivos principales del proyecto, pero existen varias oportunidades para ampliar y mejorar su funcionalidad en futuras fases. A continuación, se presentan las áreas clave de trabajo a futuro:

8.1 Implementación Completa del Módulo de Drenaje y Limpieza de Agua

Actualmente, el sistema de drenaje para eliminar agua insalubre no ha sido probado en su totalidad. En futuras iteraciones, se propone:

- Realizar pruebas específicas del sensor de pH en conjunto con la válvula solenoide y el sistema de desagüe.
- Implementar algoritmos que optimicen el uso del agua, minimizando desperdicios y asegurando su calidad.
- Diseñar una infraestructura de desagüe adecuada para entornos reales.

8.2 Mejora en la Conexión y Acceso Remoto

Aunque el uso de **RealVNC Viewer** ha solucionado las limitaciones iniciales, se propone:

- Desarrollar una aplicación móvil más robusta con funcionalidades adicionales como historial de datos y gráficos para analizar patrones de alimentación y consumo de agua.
- Explorar la integración de tecnologías IoT, como servidores en la nube, para permitir un acceso remoto más avanzado y seguro.

8.3 Optimización de Sensores y Componentes

Algunos sensores, como el de peso, podrían beneficiarse de mejoras en su diseño y calibración. Las tareas futuras incluyen:

- Implementar nuevos métodos de calibración más precisos para mejorar la confiabilidad de las lecturas.
- Evaluar alternativas tecnológicas para sensores actuales, buscando mayor precisión y durabilidad.

8.4 Expansión del Sistema para Funcionalidades Adicionales

Para incrementar el valor del sistema, se podrían agregar nuevas características, como:

- Control de temperatura dentro del gallinero mediante sensores y ventiladores automatizados.
- Monitoreo de la salud de las aves utilizando cámaras y análisis de imágenes.
- Incorporación de un sistema para recolección de huevos, complementando las funcionalidades actuales.

8.5 Escalabilidad del Proyecto

El sistema está diseñado para gallineros pequeños, pero se podrían explorar opciones para:

- Ampliar la capacidad del sistema a gallineros de mayor tamaño o comerciales, integrando múltiples sensores y actuadores.
- Adaptar el diseño para otros tipos de animales o usos agrícolas, como invernaderos inteligentes.

Conclusión

El desarrollo del sistema **ChickenCheck** ha demostrado ser una solución viable y funcional para la automatización de la alimentación y suministro de agua en gallineros, cumpliendo con los objetivos principales planteados al inicio del proyecto. Durante el proceso, se enfrentó a diversos desafíos técnicos y logísticos que requirieron ajustes en la planificación y ejecución, pero cada uno de ellos fue abordado con soluciones efectivas que garantizaron el éxito de las funcionalidades principales del sistema.

Además, la aplicación móvil desarrollada proporciona una interfaz amigable e intuitiva para los usuarios, permitiendo el monitoreo remoto en tiempo real y la recepción de notificaciones sobre el estado del gallinero. La incorporación de **RealVNC Viewer** como alternativa de conexión ayuda significativamente la experiencia de monitoreo remoto, resolviendo las limitaciones iniciales de conectividad.

A pesar de los avances, quedan aspectos por perfeccionar, como la funcionalidad de drenaje y limpieza de agua, que no pudo ser completamente implementada debido a restricciones de tiempo. Sin embargo, esto abre la posibilidad de extender el proyecto en fases futuras, permitiendo añadir características que refuercen aún más la eficiencia y alcance del sistema.

El proceso de desarrollo también deja lecciones importantes, como la necesidad de realizar pruebas iterativas en etapas tempranas, lo que permitió detectar y corregir problemas antes de que afectarán el diseño global. La colaboración y el enfoque modular fueron claves.

ChickenCheck representa un avance significativo en la automatización de gallineros, mejorando no solo la eficiencia en la gestión diaria, sino también promoviendo el bienestar de las aves. El sistema está preparado para ser utilizado en entornos reales, y su diseño modular permite futuras mejoras y adaptaciones.

Bibliografía

https://asana.com/es/resources/project-risks

https://www.chickensandyou.com/chicken-feeding-and-diet/

https://cluckchickenfarm.com/daily-chicken-water-intake-how-much-they-need/

https://www.manualslib.es/manual/399784/Raspberry-Pi.html