UNIVERSIDAD DE TARAPACÁ

FACULTAD DE INGENIERÍA

Departamento de Ingeniería en Computación e Informática

Plan de proyecto
Sistema
anti-incendios
"Pyro"

Autores: Martín Castillo
Álvaro Lovera
Isaac Contreras
Miguel Fernández

Asignatura: Proyecto II

Profesor: Diego Aracena

ARICA, 16 de Octubre 2025

1. Historial de Cambios

Fecha	Versión	Descripción	Autor(es)
16/10/25	1.0	Inicio del uso de la plantilla y completación de los datos principales del proyecto.	Martín Castillo, Álvaro Lovera
28/10/25	1.1	Primera entrega del informe hasta la planificación de gestión de riesgo.	Martín Castillo, Álvaro lovera
25/11/2025	1.2	Segunda entrega del informe hasta diseño	Martín Castillo, Álvaro Lovera, Isaac Contreras, Miguel Fernández

Índice

Índice de figuras

Índice de tablas

2. Panorama general

2.1 Resumen del proyecto:

- **Propósito, alcance y objetivos:**

  - Propósito: El presente proyecto aborda la problemática de los incendios, un siniestro de alto riesgo y con un considerable potencial destructivo en entornos urbanos. La rápida propagación del fuego en áreas densamente pobladas puede ocasionar graves pérdidas materiales y, más importante aún, poner en riesgo vidas humanas.

    Por ello, el objetivo principal de esta iniciativa es desarrollar una solución preventiva: un sistema inteligente de detección temprana de incendios. Este sistema se basa en una plataforma Raspberry Pi, la cual integra diversos sensores capaces de identificar los indicios iniciales de un conato de incendio, tales como la presencia de humo, gases o cambios anormales de temperatura. La finalidad es implementar esta tecnología en espacios de uso cotidiano como hogares, oficinas y edificios, para así anticipar la emergencia y permitir una respuesta rápida y eficaz antes de que el fuego se propague.

  - Alcance: El objetivo de este proyecto es la prevención y reducción de incendios en entornos cotidianos. Utiliza sensores que, al detectar un positivo en su monitoreo, alertan a los usuarios para permitir una respuesta y control a distancia, considerando que el tiempo de reacción es vital en una emergencia.

- **Objetivo general:**
  Desarrollar un sistema de monitoreo y control que permita prevenir un incendio y actuar ante ello.

- **Objetivos específicos:**
  - Identificar y desarrollar una problemática haciendo uso de sensores, raspberry.
  - Planificar el diseño del proyecto, por medio de maquetas y modelado 3D.
  - Desarrollar un software capaz de controlar y monitorear los dispositivos instalados.
  - Realizar pruebas del software y sensores que se utilizarán.
  - Analizar resultados de las pruebas para determinar datos relevantes.

- **Suposiciones y restricciones:**

1. El modelo 3D realizado en Unity representa correctamente los escenarios en los que se utilizará el proyecto.
2. Todas las herramientas de trabajo requeridas estarán disponibles para ser usadas dentro del plazo del proyecto.
3. El grupo aprenderá uniformemente y en conjunto los conocimientos básicos para desarrollar el proyecto.
4. El producto final cumplirá con las especificaciones definidas en el plan del proyecto. Las cuáles son las siguientes:
- Entrega del producto terminado con sus requerimientos completados.
- Controlar la fuente eléctrica para el cumplimiento de uno de los objetivos del proyecto, evitar un incendio en el recinto.
- Activar los regadores de agua para controlar el incendio.
- Alertar por medio del software los irregulares que detecten los sensores.

5. Las actividades a realizar se llevarán a cabo en los plazos establecidos previamente, de acuerdo a la carta gantt.

Respecto al usuario

1. Se asume que los usuarios tendrán acceso a dispositivos móviles inteligentes (smartphone).

2. Se asume que los usuarios utilizarán la app para administrar y monitorear el sistema.

Restricciones:

1. **Sensores**: Estamos limitados a ocupar sensores compatibles con raspberry pi
2. **Software**: El principal lenguaje de programación es Python.
3. **Costos**: El máximo de costos extras para el desarrollo es de 20.000 pesos.
4. **Tiempo**: Cada etapa del proyecto tiene un tiempo de plazo definido.

- **Entregables del proyecto:**
  - **Informes**: En estos informes se detalla el progreso que se llevó a cabo en cada etapa en cuanto a los objetivos, resultados, desafíos experimentados. Aportando una visión clara en cuanto al proyecto.
  - **Bitácoras**: Por cada dos reuniones de trabajo se sube a redmine una bitácora informando lo que se avanzó durante la misma
  - **Carta Gantt**: Contienen información clara de las actividades a realizar a lo largo del proyecto.
  - **Modelado 3D**: Representación digital del entorno físico en el cual se basará la maqueta para la identificación de toda la instalación.
  - **Wiki**: Presentación del proyecto y todos sus ámbitos en el desarrollo del proyecto.

Tabla 1: Encargados y encargados.

Entregables	Encargados
Bitacoras	Alvaro Lovera
Carta Gantt	Martin Castillo
Modelado 3D	Martin Castillo, Álvaro Lovera
Maqueta	Martin Castillo
Informe	Alvaro Lovera
wiki	Martin Castillo

3. Organización del proyecto

3.1 Personal y entidades internas:

A cada integrante del equipo de trabajo del proyecto se le asignó una responsabilidad, de la cual estará a cargo para cumplir en el tiempo estipulado. Pero se debe resaltar que aunque existan los responsables de cada eje, cada uno de los integrantes debe auxiliar a cada parte del proyecto, para así, velar una composición digna de sus propias tareas como las de los demás colegas de trabajo.

● Jefe de proyecto: Responsable de la planificación, ejecución y control del proyecto, asegurando que se cumplan los plazos, presupuesto y objetivos. Coordina el equipo, gestiona riesgos y comunica con las partes interesadas.

● Documentador: Genera y gestiona toda la documentación necesaria para el proyecto, incluidos informes, manuales y guías técnicas. Asegura que la información esté actualizada y disponible para el equipo.

- Analista programador: Analiza los requisitos del proyecto, diseña soluciones

técnicas y desarrolla el código necesario. Se encarga tanto del análisis funcional como de la programación y pruebas del software.

● Programador: Encargado de escribir y mantener el código fuente del proyecto, implementar nuevas funcionalidades, realizar pruebas y depurar errores.

● Diseñador: Responsable de la apariencia visual y la experiencia de usuario (UI/UX), creando interfaces atractivas y funcionales que aseguren una interacción eficiente y agradable para el usuario.

3.2 Roles y responsabilidades:

Tabla 2: Roles y responsabilidades.

Rol	Encargado	Involucrado
Jefe de proyecto	Alvaro Lovera	Alvaro Lovera
Documentador	Alvaro Lovera	Alvaro Lovera, Isaac Contreras
Analista programador	Isaac Contreras	Isaac Contreras, Martin Castillo
Programador	Miguel Fernandez	Miguel Fernandez, Martin Castillo
Diseñador	Martin Castillo	Martin Castillo, Alvaro Lovera

3.3 Mecanismos de comunicación:

● **WhatsApp**: Software que facilita las comunicaciones, por medio de mensajes y llamadas, audiovisuales.

● **Discord**: Plataforma que facilita la comunicación audiovisual.

● **Google Drive**: Plataforma que administra la documentación, por medio de carpetas y documentos.

● **Redmine**: Plataforma donde se suben los archivos que forman parte del proyecto.

4. Planificación de los procesos de gestión

4.1 Planificación inicial del proyecto

- **Planificación de estimaciones:**

Tabla 3: Planificación de recursos hardware:

Producto	Cantidad	Costo unidad	Costo total
Notebook	3	500.000	1.500.000
Raspberry pi 4 model B	1	124.600	124.600
Meta quest 3	1	625.000	625.000
grove pi+ starter kit	1	70.000	70.000
Monitor LG	1	85.000	85000
			TOTAl: 2.404.600

Tabla 4: Planificación de recursos software:

Productos	Meses	Costos
Krita	1	0
Unity	6	0
Canva	6	0
Google docs	6	0
Blender	6	0
Python	6	0
C# (Unity)	6	0

- **Planificación de recursos humanos:**

Consideraciones para el proyecto:

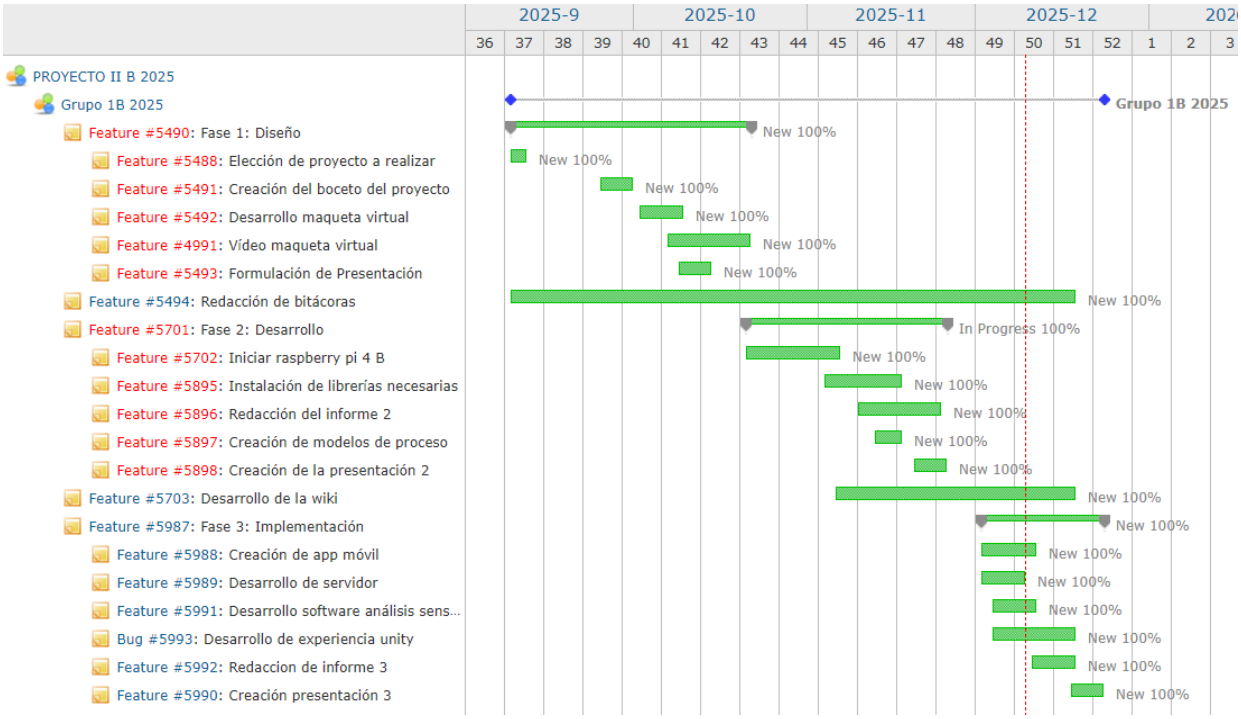
- La fase de inicialización del proyecto comenzará en la segunda semana del ciclo escolar, a partir del Martes 9 de septiembre de 2025.
- La duración total del proyecto es de 4 meses, considerando que se dedicarán 4 horas de Trabajo en clase por semana. Esto da un total de 64 horas de trabajo en clase a lo largo del proyecto.
- Se ha establecido un horario fijo para reuniones semanales, las cuales tendrán una duración de 1 hora y se realizan los días martes y jueves. Esto resulta en 16 horas de trabajo autónomo.

Tabla 5: Planificación de recursos humanos.

Integrantes	Rol	Hora total	Sueldo/Hora	Sueldo Total
Isaac Contreras	Analista programador	64	5.500	352.000
Alvaro Lovera	Jefe de proyecto, Documentador	64	6.200 4.000	652.800
Martín Castillo	Diseñador	64	4.500	288.000
Miguel Fernandez	Programador	64	5.200	332.800
				Total: 1.625.600

4.2 Lista de actividades (carta gantt)

Figura 1: Carta Gantt



- **Actividades de trabajo:**
  - **Redacción de bitácoras:** Se redactan las bitácoras semanales del proyecto, describiendo lo logrado, acuerdos y problemas presentados durante el desarrollo del proyecto.
  - **Desarrollo de la wiki:** Mediante la plataforma redmine se hace una wiki dónde se accede a la información de todo el proyecto, incluido los objetivos, alcance, problemáticas, implementación, etc.

Fase 1:

- **Elección de proyecto a realizar:** Se eligió un proyecto a desarrollar con la tecnología propuesta en clases; RaspBerry Pi.
- **Creación del Boceto del proyecto:** Se utiliza un programa de dibujo para representar el funcionamiento del proyecto mediante fotografías, dibujos y texto.
- **Desarrollo maqueta virtual:** Se utiliza Unity y los lentes de realidad virtual Meta Quest 3 para crear una maqueta virtual basada en el boceto.
- **Video maqueta virtual:** Se graba el funcionamiento de la maqueta

virtual desarrollada a modo de documentación del proyecto.

- **Formulación de presentación:** Se crea una presentación para exponer el proyecto a la clase y recibir retroalimentación.

Fase 2:

- **Iniciar RaspBerry pi 4 B:** Se arma el sistema RaspBerry con un monitor, teclado y mouse para poder instalar el sistema operativo necesario.
- **Instalación de librerías necesarias:** Se instalan las librerías de grovepi+ para hacer la prueba de sensores necesarios para el proyecto.
- **Redacción del informe 2:** Se redacta el informe correspondiente a la fase 2 del proyecto.
- **Creación de modelos de proceso:** Se diseña el funcionamiento del sistema a través de distintos modelos, diagrama de contexto, diagrama de casos de uso, diagrama de clases y diseño de la arquitectura.
- **Creación de la presentación 2:** Se crea la presentación correspondiente al avance realizado en el informe 2 del proyecto.

Fase 3:

- **Creación de app móvil:** Se diseña y crea una aplicación móvil para recibir información de los sensores.
- **Desarrollo de servidor:** Se desarrolla un servidor con Flask en el sistema RaspBerry, para acceder a la información de los sensores desde la red..
- **Desarrollo software análisis sensores:** En la aplicación se debe desarrollar un sistema que interprete y analice los datos de los sensores para detectar amenazas de incendios.
- **Desarrollo experiencia Unity:** .
- **Redacción de informe 3:** .
- **Creación presentación 3:** .

- **Asignación de tiempo:**
  - Redacción de bitácoras: Toda la duración del proyecto.

- Desarrollo de la wiki: 1 mes

Fase 1:

- Elección de proyecto a realizar: 3 días.
- Creación del Boceto del proyecto: 5 días.
- Desarrollo maqueta virtual: 7 días.
- Video maqueta virtual: 14 días.
- Formulación de presentación: 5 días.

Fase 2:

- Iniciar RaspBerry pi 4 B: 2 semanas.
- Instalación de librerías necesarias: 2 semanas
- Redacción del informe 2: 2 semanas
- Creación de modelos de proceso: 4 días.
- Creación de la presentación 2: 5 días

Fase 3:

- Creación de app móvil: 9 días.
- Desarrollo de servidor: 7 días.
- Desarrollo software análisis sensores: 7 días.
- Desarrollo experiencia Unity: 2 semanas.
- Redacción de informe 3: 7 días.
- Creación presentación 3: 5 días.

4.3 Planificación de gestión de riesgos

Para gestionar los riesgos, se identificaron y categorizaron los siguientes tipos de riesgo y sus factores:

Tabla 6: Planificación de gestión de riesgos.

Tipo de riesgo	Descripción
Tecnológico	Retrasos en la entrega o problemas relacionados con

	el hardware o el software. Se reportan numerosos problemas tecnológicos.
Humano	Baja moral del personal, problemas de salud, malas relaciones entre los miembros del equipo y dificultades para encontrar personal disponible.
Herramientas	Resistencia del equipo a utilizar ciertas herramientas, quejas sobre las herramientas de trabajo, y solicitudes de estaciones de trabajo más potentes.
Requerimientos	Frecuentes cambios en los requerimientos, quejas del cliente.
Organizacional	Rumores dentro de la organización y falta de liderazgo por parte de la dirección principal.
Estimación	Incumplimiento de los plazos acordados y dificultades para eliminar defectos reportados.

Para los riesgos latentes que podrían surgir durante el desarrollo del proyecto, se clasificaron en los
siguientes cuatro niveles de impacto:

1. Catastrófico: Impacto crítico que puede poner en riesgo la continuidad o el éxito del proyecto.
2. Crítico: Impacto significativo que requiere recursos adicionales para ser gestionado, pero el proyecto puede continuar.
3. Marginal: Impacto leve que puede retrasar algunos aspectos del proyecto, pero sin afectar gravemente los resultados.
4. Despreciable: Impacto mínimo que no requiere acción inmediata y no afectará de manera relevante el desarrollo del proyecto.

Tabla 7: Probabilidad de ocurrencia de riesgos.

Probabilidad de ocurrencia	Rango de %
Alta	**71%-100%**
Media	**31%-70%**
Baja	**0%-30%**

Tabla 8: Estimación, calificación y acciones remediales para riesgos.

Riesgo	Tipo	Probabilidad de ocurrencia	Nivel de impacto	Acción remedial
Daños/Pérdida de la tarjeta SD	Tecnológico	Alta	2	Realizar copias de seguridad periódicas y tener SD adicionales disponibles para reemplazo inmediato.
Falta de coordinación entre miembros del equipo	Organizacional	Media	3	Establecer reuniones de seguimiento frecuentes y usar herramientas de comunicación efectiva (WhatsApp, Discord, Redmine).

Falta de asistencia de reunión	Humano	Media	3	Amonestar al responsable, y solicitar una penalización adecuada.
Errores en el software de control del sistema.	Requerimientos	Alta	2	Implementar pruebas continuas desde el inicio del desarrollo para detectar y corregir fallos.
Enfermedades del personal	Humano	Baja	4	Redistribuir sus tareas entre los integrantes según sus habilidades.
Incompatibilidad entre los componentes del hardware	Herramientas	Media	3	Verificar la compatibilidad de los componentes antes de comprarlos y realizar pruebas.
Componentes defectuosos	Tecnológicos	Media	2	Comunicarse con el encargado de asignación de componentes para su debido cambio y notificar el problema del componente respectivo.

				Tener componentes de repuesto en caso de salir defectuosos
Salida de un integrante del proyecto	Humano	Baja	2	Reorganización del plan del proyecto para entregar nuevas responsabilidades y roles correspondientes.
Cambio de los requerimientos	Requerimientos	Alta	2	Realizar una reunión con el cliente, en la que se discutirá la viabilidad de los requerimientos y su importancia.
Error de cálculo de costos	Estimación	Media	2	Realizar una reevaluación de los costos del proyecto para garantizar la fiabilidad de los cálculos. Si ocurre más de una vez, hacer cambio de responsable y realizar una amonestación respectiva.

5. Planificación de procesos técnicos

5.1 Requisitos

5.1.1 Requisitos funcionales

RF01	El sistema debe medir en tiempo real la temperatura y humedad del entorno mediante los conjuntos de sensores seleccionados.
RF02	El sistema debe detectar niveles anormales de gases mediante el sensor MQ-2.
RF03	El sistema debe analizar la imagen de la cámara usando colorimetría u otras técnicas para detectar señales visuales de incendio.
RF04	Cuando los sensores detecten niveles peligrosos, el sistema debe generar una alerta automática y enviarla al usuario mediante la app móvil.
RF05	El usuario debe poder visualizar desde la app los valores de temperatura, humedad, gas y estado general de riesgo.
RF06	El sistema debe transmitir la señal en tiempo real de la cámara al smartphone del usuario.
RF07	La app debe permitir al usuario activar manualmente el sistema de aspersores de agua ante una emergencia.
RF08	Si los niveles detectados son críticos y el usuario no responde, el sistema debe activar automáticamente los aspersores para controlar el fuego.
RF09	El sistema debe almacenar un registro de alertas, lecturas anómalas y activaciones de los aspersores para auditoría y revisión.

5.1.2 Requisitos no funcionales

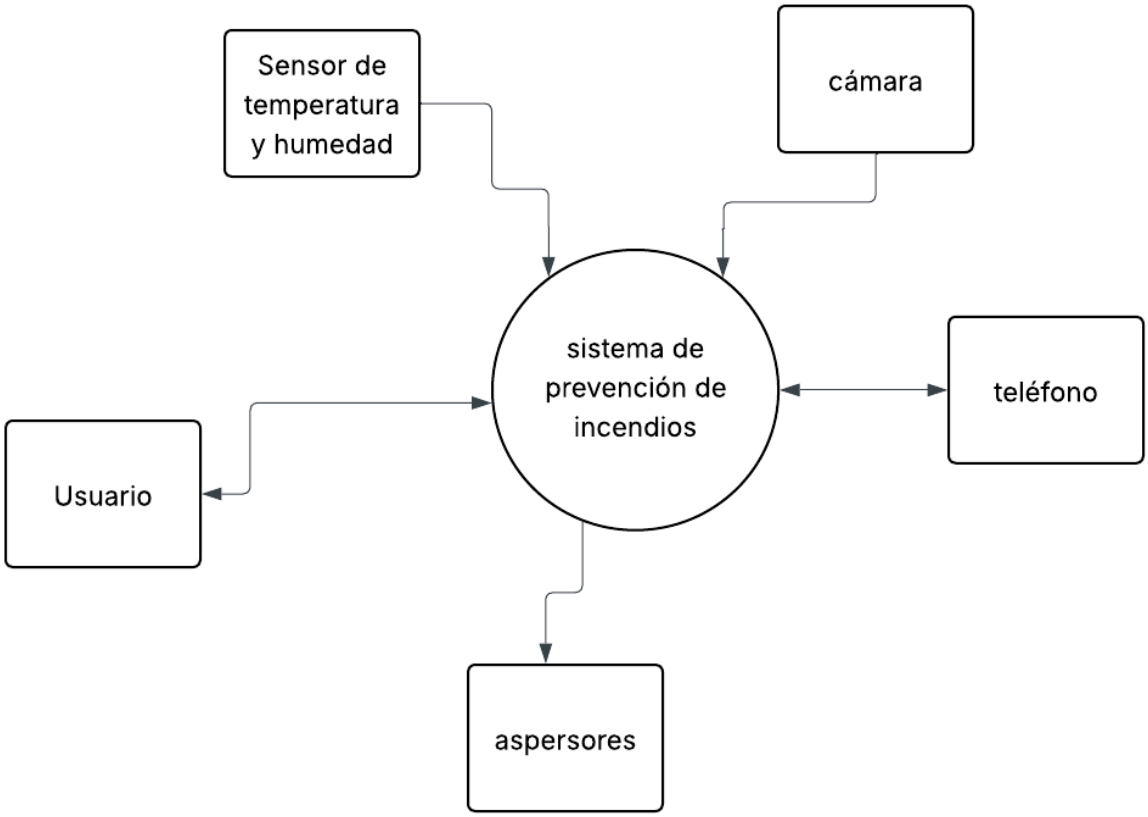
RNF01	La aplicación móvil debe presentar la información de manera clara, amigable e intuitiva para que cualquier usuario pueda interpretar el riesgo sin conocimientos técnicos.
RNF02	El sistema debe procesar datos de sensores y enviar alertas en un tiempo máximo de 1 segundo, garantizando respuestas rápidas ante una emergencia.
RNF03	El sistema debe funcionar de forma continua, garantizando un uptime mínimo del **95%**, especialmente en el monitoreo de sensores.
RNF04	El sistema debe permitir agregar más sensores, habitaciones o cámaras sin requerir rediseños mayores del software.

RNF05	El sistema debe permitir realizar mantenimiento del software y actualización de sensores sin afectar el funcionamiento del monitoreo.
RNF06	El sistema debe continuar funcionando aún cuando falle un sensor, enviando un aviso de sensor desconectado.

5.2 Modelos de Diseño

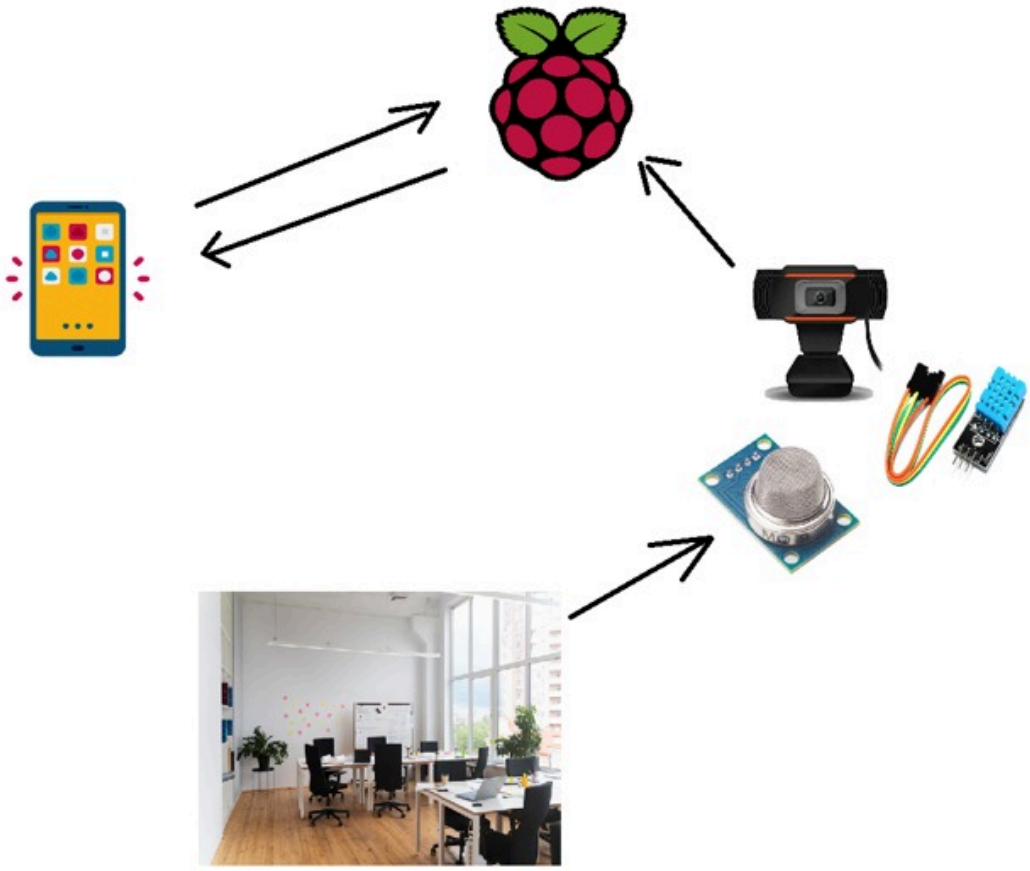
5.2.1 Modelo de contexto:

Figura 2: Modelo de contexto



5.2.2 Arquitectura de la aplicación:

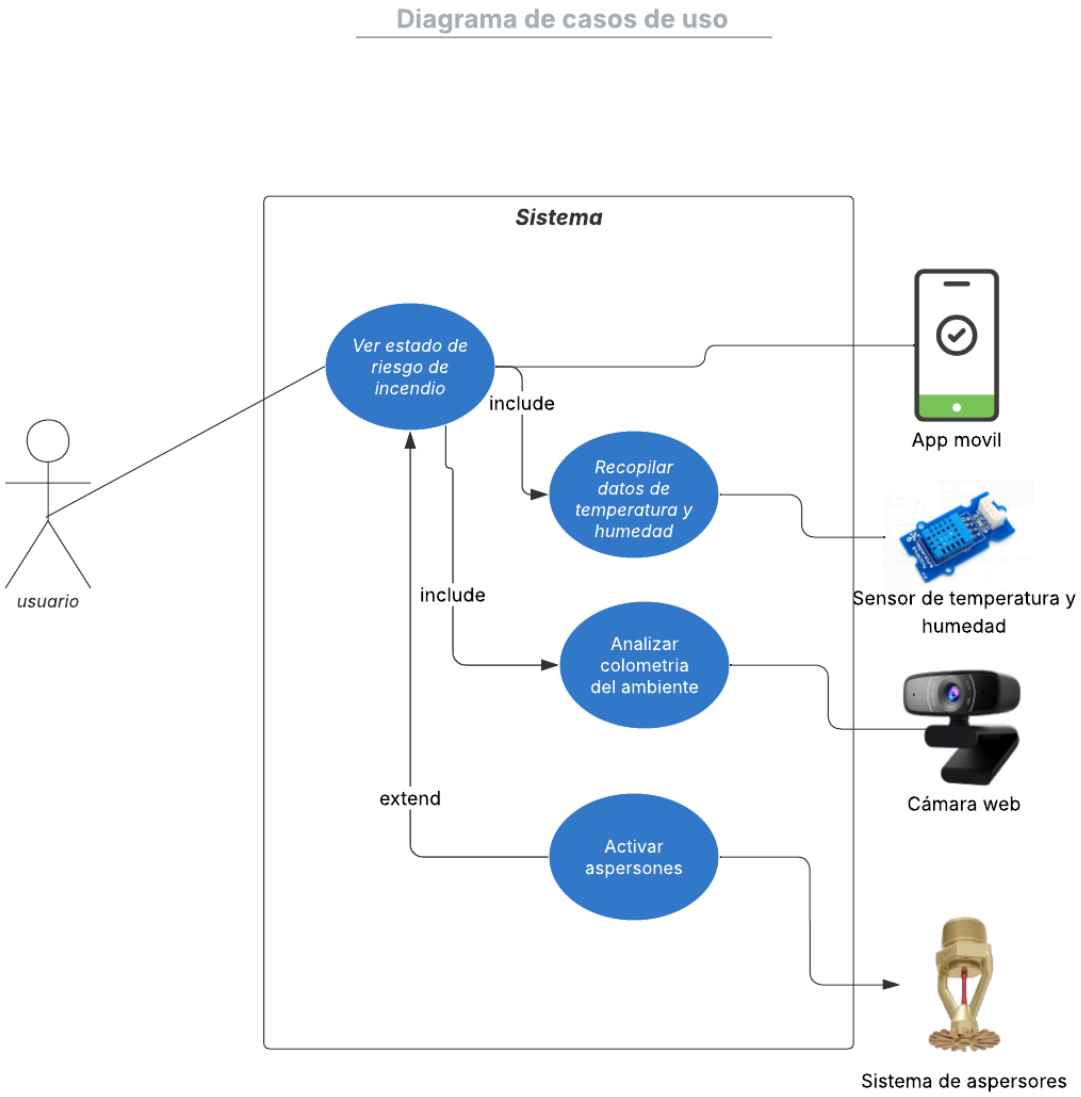
Figura 3: Arquitectura de la aplicación



5.2.2 Modelo de casos de uso:

El diagrama de caso de uso general muestra cómo el usuario interactúa con el sistema para monitorear y reaccionar ante el riesgo. A través de sensores, puede revisar la humedad y el entorno donde está instalado la cámara, recibiendo alertas en caso de niveles críticos de parte de los sensores. Además, el sistema permite usar el sistema de aspersores en caso de una situación crítica.

Figura 4: Modelo de casos de uso

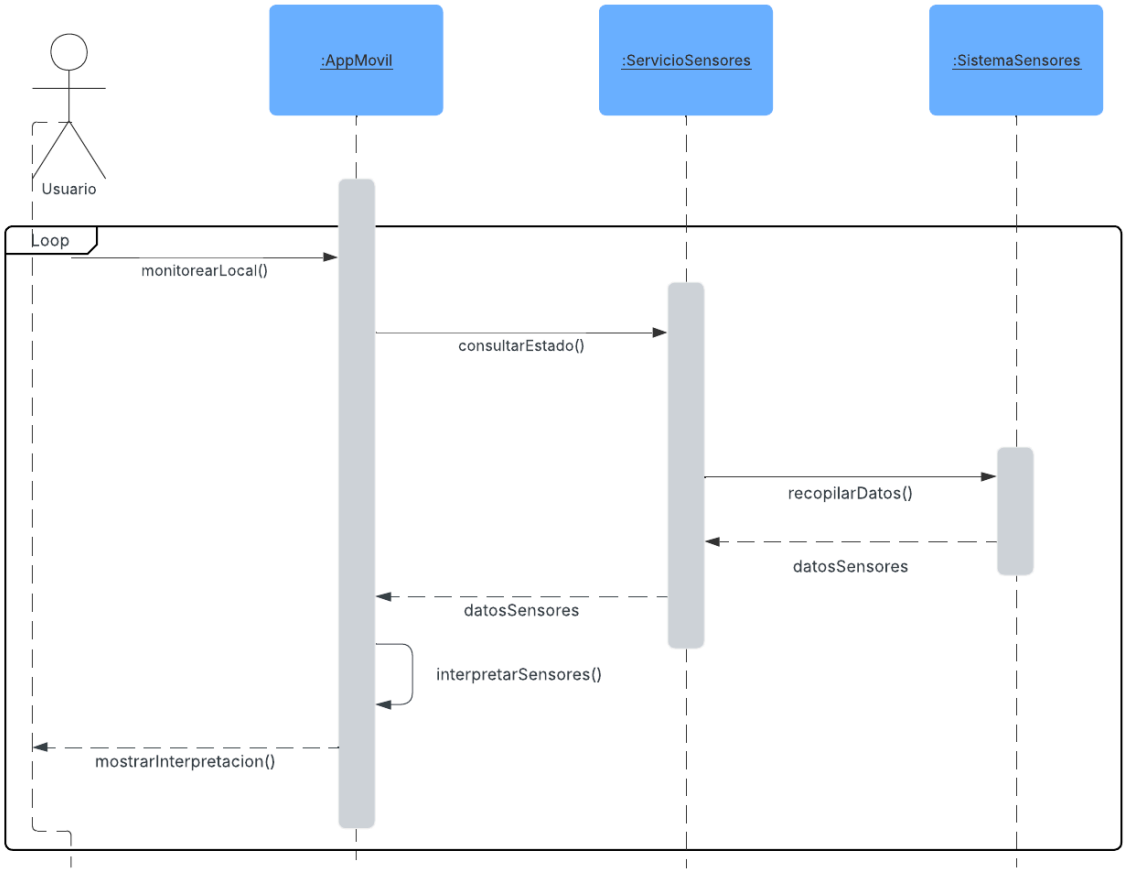


5.2.3 Casos de uso

1) Monitoreo de habitaciones

[Identificador]	**Monitoreo de habitaciones**		
Descripción	El **usuario** desde la **app móvil** podrá monitorear la temperatura del local/oficina en que se encuentra el sistema.		
Actores	**Usuario, Sensor de temperatura y humedad** y **app móvil.**		
Pre condiciones	El **usuario** debe tener un dispositivo móvil.		
Post condiciones	El **usuario** debe haber logrado obtener información clara y amigable sobre el riesgo de incendio del local con el sistema implementado.		
Secuencia Normal	**#**	**Acción (Usuario)**	**Reacción (sistema)**
	1	El usuario abre la aplicación móvil.	
	2		El sistema despliega las habitaciones en las que se encuentran las cámaras y sensores
	3	El usuario selecciona la habitación que desea monitorear	
	4		El sistema muestra la opción de visualizar la cámara de la habitación junto con la información de la temperatura que hay en esa habitación
Excepciones	**#**	**Acción (Usuario)**	**Reacción (sistema)**
	1		No hay conexión con los sensores de la habitación
Rendimiento	El sistema deberá consultar a Raspberry sobre los niveles de temperatura usando el **Sensor de temperatura y humedad**, interpretar esta información y mostrarla de forma clara y amigable al **usuario** a través de la **app móvil**		
Frecuencia	En este caso de uso se espera que se use una cantidad de 2 veces al día.		
Importancia	Vital		
Urgencia	Inmediata		
Comentarios			

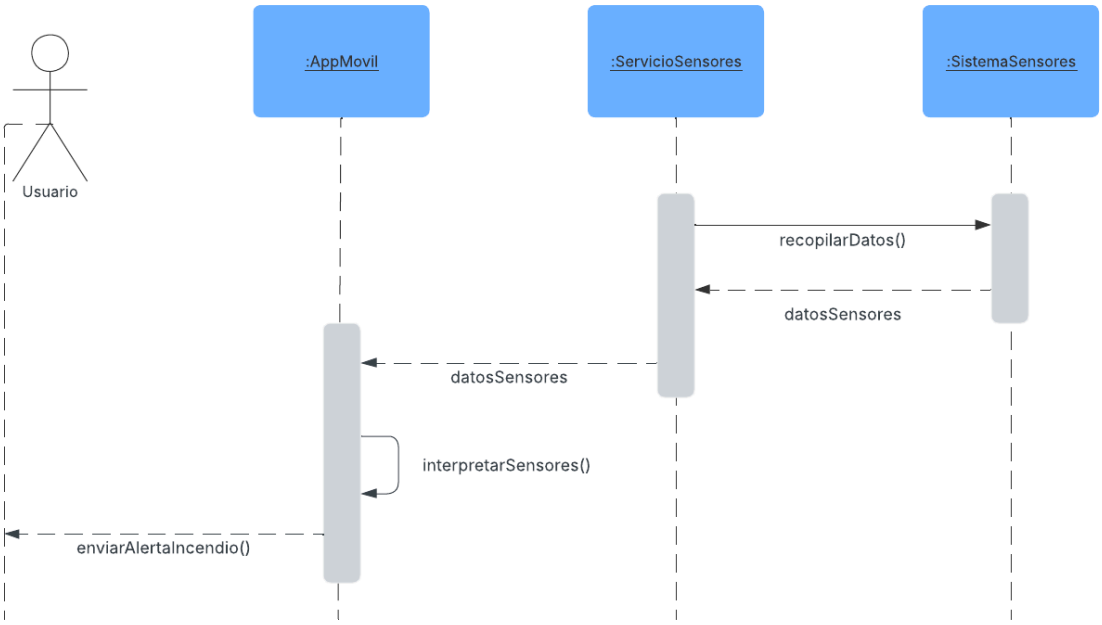
Figura 5: diagrama de secuencia Monitoreo de habitaciones



2) Generar alerta por niveles criticos

[Identificador]	Generar alerta por niveles criticos		
Descripción	Este caso de uso permite generar una alerta por niveles críticos obtenidos de los sensores.		
Actores	Usuario, Sensores		
Pre condiciones	Los sensores deben transmitir datos críticos al sistema dando a entender que algo está fuera de lo común en el sitio monitoreado.		
Post condiciones	El usuario al recibir las alertas debe tomar tomar una acción para combatir la anomalía detectada por los sensores.		
Secuencia Normal	**#**	**Acción (actor)**	**Reacción (sistema)**
	1		Los sensores detectan algo fuera de lo común en sus datos.
	2		Los sensores transmiten la información al sistema
	3		El sistema detecta esto como una alerta y manda un aviso a la app móvil
	4	El usuario recibe la alerta y procede a entrar a la aplicación.	
	5		El sistema le muestra al usuario la información de esta alerta como los niveles de temperatura
Secuencia Alternativa	5	5.1.- El usuario detecta que es un falso positivo y avisa al sistema	
	6		6.1 El sistema recibe la información y procede a recopilar el error.
Rendimiento	El sistema deberá interpretar los datos para llegar a una conclusión y avisar al usuario para que tome acciones.		
Frecuencia	Se espera que este caso de uso suceda en casos de anomalía, puede dar una información crítica pero a la vez puede ser un cambio insignificante, la idea es que el usuario pueda tomar acciones en base a su criterio, se espera que sea medianamente frecuente.		
Importancia	Es de vital importancia		
Urgencia	Inmediata		
Comentarios			

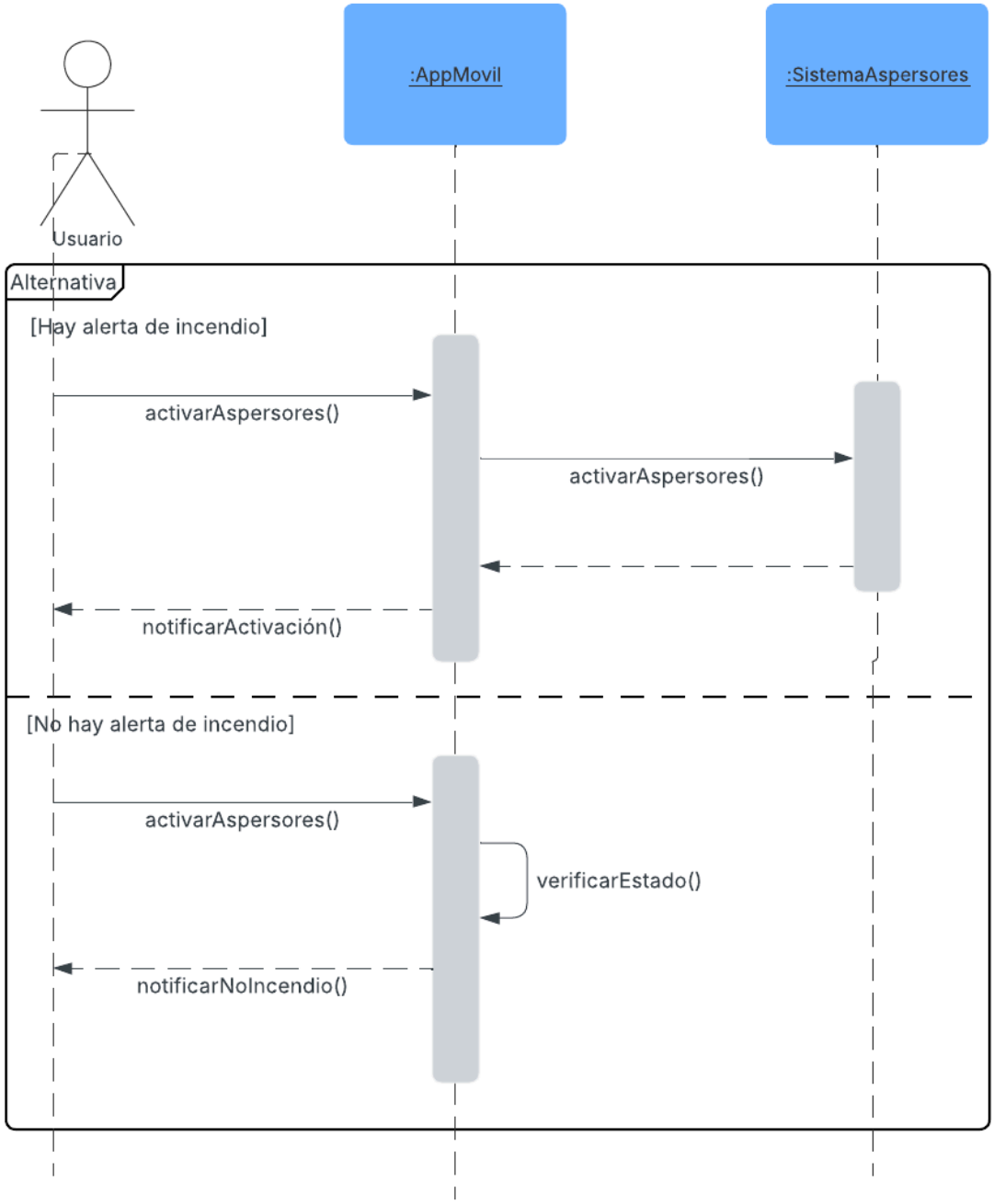
Figura 6: diagrama de secuencia Generar alerta por niveles críticos



3) Activación del sistema de aspersores

[Identificador]	**Activación del sistema de aspersores**		
Descripción	El **usuario** podrá activar el **sistema de aspersores** a través de la **app móvil**		
Actores	**Usuario, sistema de aspersores y app móvil**		
Pre condiciones	El **usuario** debe tener un dispositivo móvil.		
Post condiciones	[Condiciones a cubrir después de terminado el caso de uso]		
Secuencia Normal	**#**	**Acción (Usuario)**	**Reacción (sistema)**
	1		El sistema muestra que el nivel de riesgo en la habitación tiene riesgo de incendio
	2		El sistema ofrece al usuario activar el sistema de aspersores
	3	El usuario activa el sistema de aspersores	
	4		El sistema enciende los aspersores de la habitación
	5		El sistema apaga los aspersores una vez la amenaza se vea reducida
Secuencia Alternativa	5	5.1 El usuario puede apagar los aspersores cuando vea conveniente	
Excepciones	**#**	**Acción (actor)**	**Reacción (sistema)**
	1	El usuario no activa el sistema de aspersores aunque la alerta sea crítica.	El sistema de aspersores se activa automáticamente para evitar accidentes.
Rendimiento	El sistema debe analizar el nivel de riesgo de la temperatura a través de los sensores de Raspberry, dar aviso al **usuario** y mostrar la opción para activarlo, una vez activado, también a través de Raspberry se debe activar el **sistema de aspersores.**		
Frecuencia	En este caso de uso es esperable que se use 1 vez al año.		
Importancia	Importante.		
Urgencia	Hay presión.		
Comentarios			

Figura 7: diagrama de secuencia Activación de sistema de aspersores

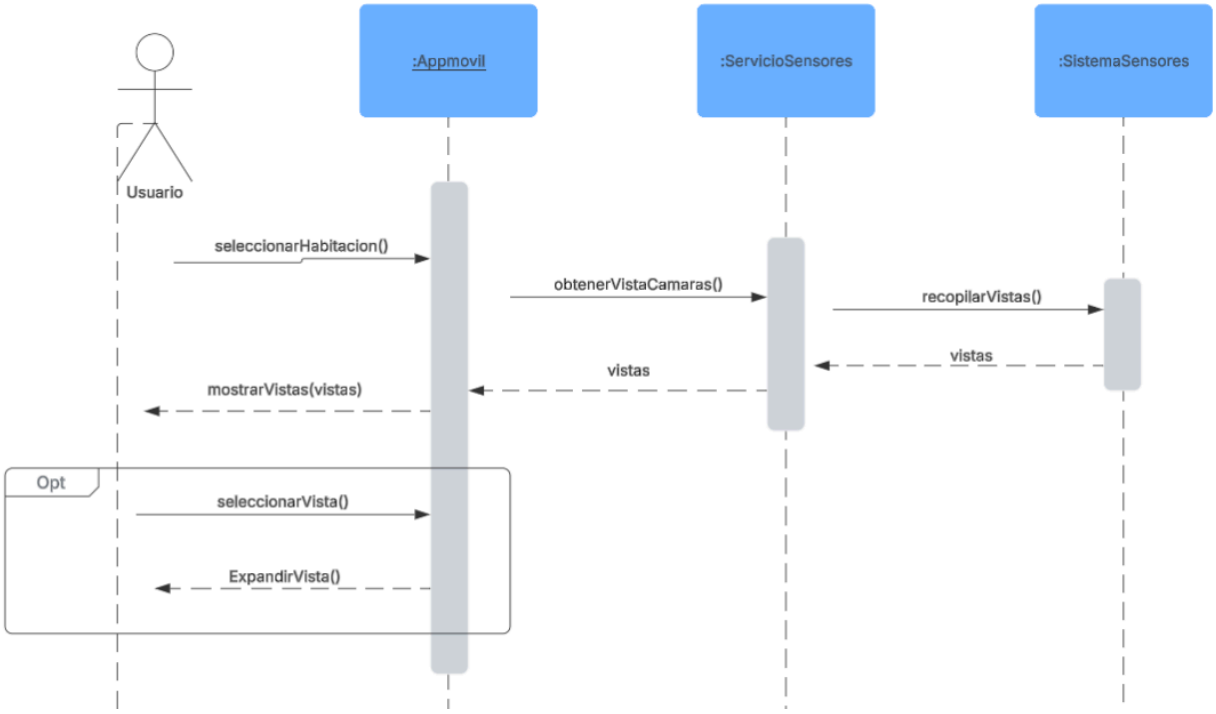


4) monitoreo de la cámara

[Identificador]	Monitoreo de la cámara		
Descripción	El usuario podrá monitorear la colorimetría de la cámara en tiempo real para saber el estado del sitio.		
Actores	Usuario, Camara web		
Pre condiciones	Se espera que el usuario tenga conexión a internet para conectar al servidor con la aplicación móvil.		
Post condiciones	El usuario tendrá una alerta de detección visible del fuego.		
Secuencia Normal	**#**	**Acción (actor)**	**Reacción (sistema)**
	1	El usuario accede a la aplicación.	
	2	El usuario selecciona la habitación que quiere visualizar	
	3		El sistema verifica la conexión con las cámaras.
	4		El sistema logra conectar con el promedio colorimétrico según pixeles rojos de la cámara.
	5	El usuario visualiza la detección de las cámaras.	
	6	El usuario selecciona la cámara de la habitación que desea visualizar	
	7		El sistema expande la visualización de esa cámara.
Secuencia alternativa	3		3.1 El sistema no logra conectar con las cámaras y muestra un mensaje de error en las que no conectaron, con una opción de reintentar la conexión.
	4	● 4.1 El usuario intenta reintentar la conexión.	
Excepciones	**#**	**Acción (actor)**	**Reacción (sistema)**
	p	El usuario no tiene conexión a internet.	Al no tener conexión a internet, el programa se lo hace saber al usuario.
Rendimiento	El sistema debe lograr establecer la conexión con la vista colorimétrica de la cámara, la velocidad de este caso de uso depende de la velocidad y estabilidad de la conexión de		

	red.
Frecuencia	Se estima una alta frecuencia de uso para este componente. La cámara no solo funcionará como un instrumento crítico ante la eventualidad de un incendio, sino que también permitirá al usuario monitorear el estado general del entorno de manera rutinaria.
Importancia	Vital
Urgencia	Puede esperar dependiendo del contexto.

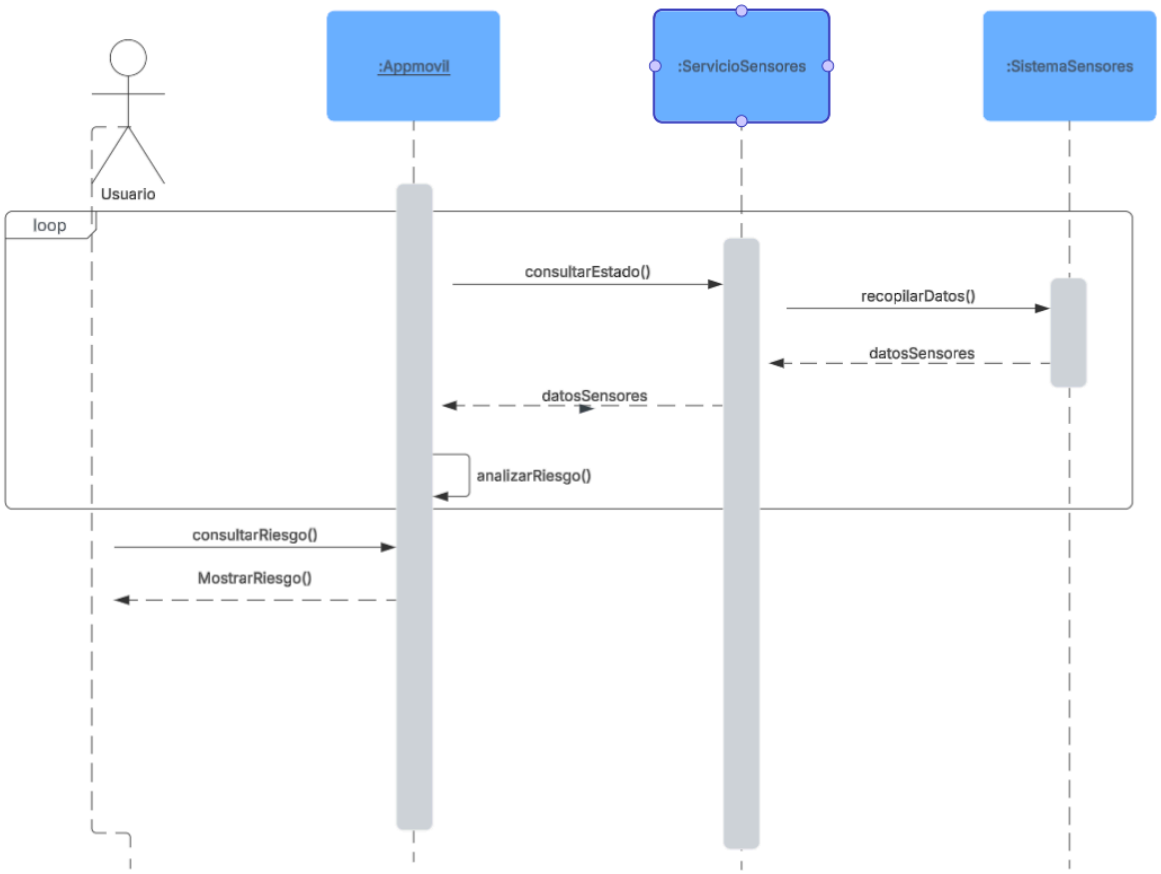
Figura 8: diagrama de secuencia Monitoreo de la cámara



5) ver estado de riesgo a través de la app

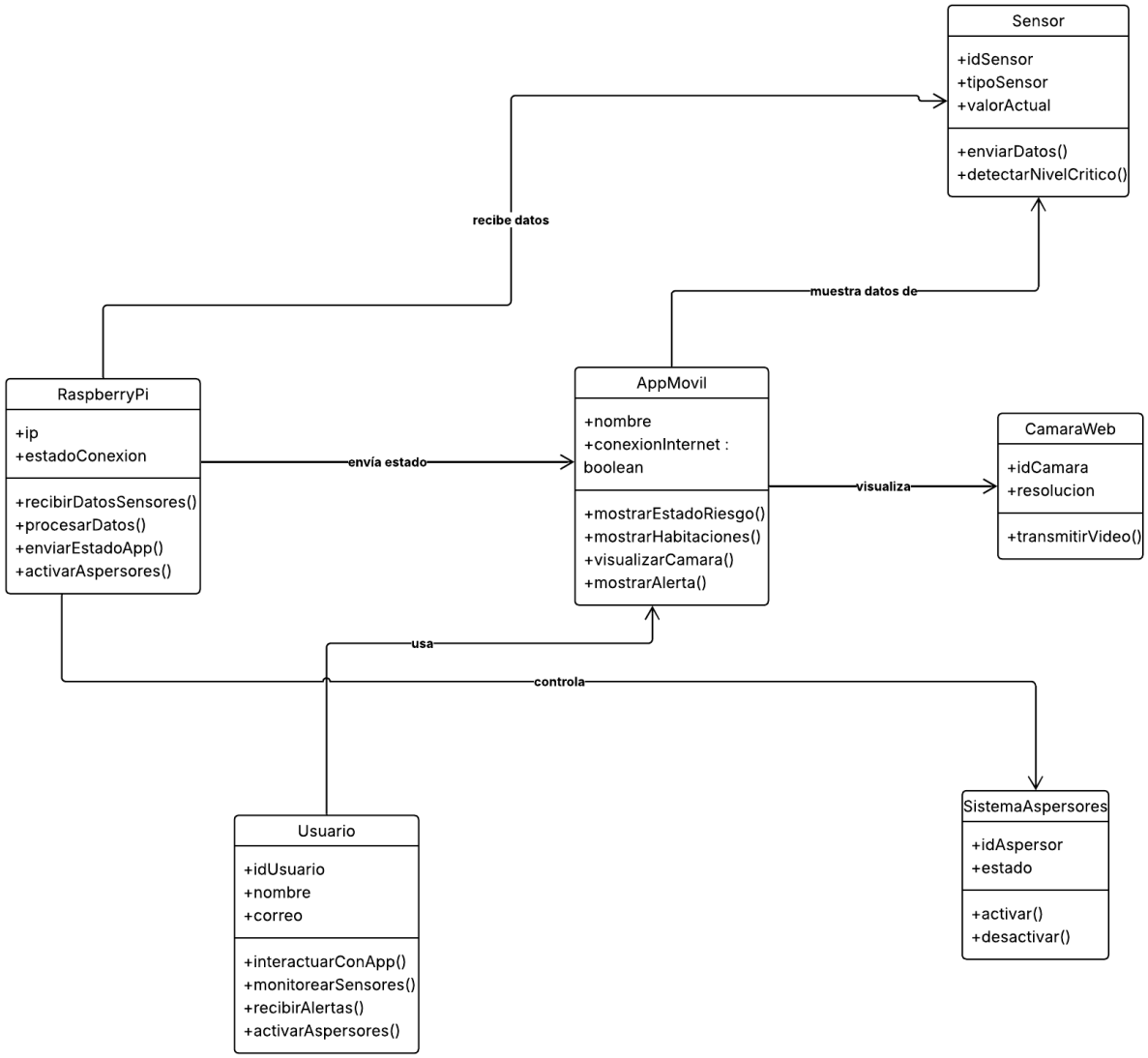
[Identificador]	Ver estado de riesgo a través de la App		
Descripción	Este caso de uso consiste en que la App pueda mostrar el estado de los sensores e interpretarlos para dar a conocer el estado de riesgo del entorno en donde está instalada la cámara.		
Actores	Sensores, App móvil		
Pre condiciones	Para que la App móvil pueda recibir los datos de los sensores debe estar con una conexión a internet y la cámara conectada a corriente.		
Post condiciones	El usuario tendrá una actualización del estado de los sensores y una interpretación del sistema.		
Secuencia Normal	**#**	**Acción (actor)**	**Reacción (sistema)**
	1		El sistema está constantemente recopilando la información de los sensores
	2		El sistema verifica riesgos con la información recopilada de la temperatura y la visualización de la cámara
	3	El usuario consulta el estado de riesgo en todas las habitaciones	
	4		El sistema muestra la información recopilada de las habitaciones
Excepciones	**#**	**Acción (actor)**	**Reacción (sistema)**
	p	En el caso de no tener conexión a internet la app no recibe la información.	El sistema no logra interpretar los diagnósticos ya que no cuenta con una conexión a los sensores.
Rendimiento	El sistema Recibe los datos de forma inmediata y logra generar un estado de riesgo, el cual se envía al usuario.		
Frecuencia	Este caso de uso se espera que sea muy frecuente su uso, ya que no solo mantiene al tanto al usuario sobre la situacion, tambien el sistema recopila la información recibida para		
Importancia	Vital		
Urgencia	Inmediatamente en distintos contextos		

Figura 9: diagrama de secuencia Ver estado de riesgo



5.2.4 Diagrama de Clases:

Figura 10: Diagrama de clases



5.3 Herramientas y técnicas

5.3.1 Herramientas a utilizar:

- Python: Lenguaje de programación que usará el sistema.
- Redmine: Plataforma para compartir avances, planificar y documentar el proceso de desarrollo del sistema.
- Unity: Entorno de desarrollo para crear la maqueta.
- Raspberry Pi 4: Hardware necesario para poder utilizar sensores.

5.3.2 Técnicas a utilizar:

- Carta Gantt: Acompañamiento visual de las tareas a realizar.
- Documentación: Documentos que describen el funcionamiento del proyecto.
- Lluvia de ideas: Técnica que ayuda a buscar ideas de forma rápida y fomentar la participación de todo el equipo.

5.4 Interfaz de usuario

5.4.1 Descripción de la interfaz de usuario

Diseño General y Estilo Visual

La interfaz presenta un diseño moderno en modo oscuro con tarjetas redondeadas para organizar la información, lo que facilita la lectura rápida de métricas clave. Utiliza una codificación de colores semántica para indicar el estado del sistema:

- **Azul Oscuro/Neutro:** Indica un estado de funcionamiento normal y seguro.
- **Rojo Intenso:** Indica una alerta crítica o detección de peligro.

Componentes Principales

La pantalla se divide en cuatro secciones jerárquicas:

- **Cabecera de Estado:** Muestra el nombre de la app, el estado de conectividad (WiFi/Online) y una etiqueta de estado ("En Vivo").
- **Visión en Tiempo Real:** Un contenedor grande reservado para el *feed* de video de la cámara (identificada como Raspberry Pi Cam), permitiendo verificación visual del entorno.
- **Tarjetas de Sensores (Monitorización):**
  - **Temperatura:** Muestra la lectura térmica en grados Celsius (ej. 23.9°C en reposo vs. 55°C en incendio).
  - **Gas / Humo:** Muestra la calidad del aire en partes por millón (PPM). Un valor bajo (12 PPM) es normal, mientras que un valor alto (452 PPM) indica presencia de humo o gas.
  - **Humedad:** Porcentaje de humedad relativa en el ambiente.
- **Panel de Acción:** Un botón inferior grande y accesible para el control manual del sistema, con funciones como "Desactivar Sistema" o "Activar Aspersores".

Comportamiento Dinámico (Normal vs. Emergencia)

La interfaz reacciona a los datos de los sensores:

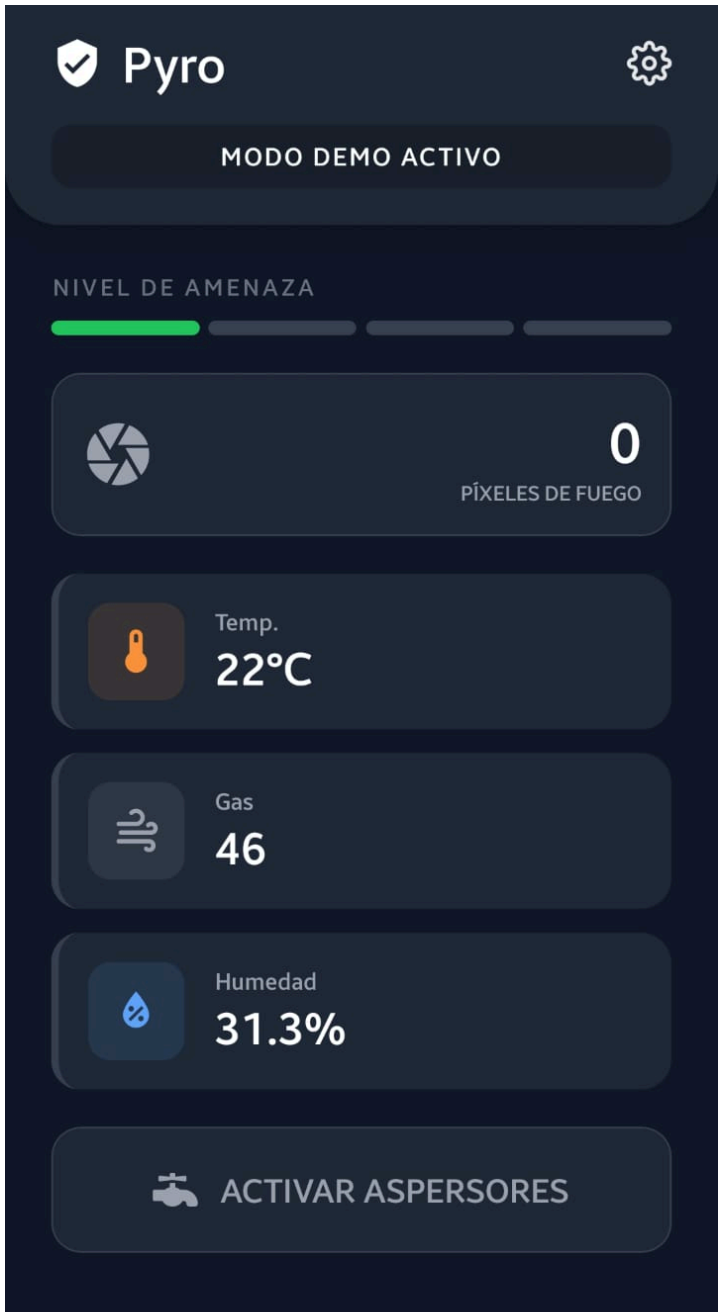
- **En Reposo:** La UI mantiene tonos azules y grises. Los valores son estables y el sistema espera comandos o señales.
- **En Alerta:** La interfaz cambia drásticamente.
  - La cabecera se vuelve roja.
  - Aparece un icono de fuego y el texto "¡FUEGO DETECTADO!".
  - Se despliega una tarjeta de advertencia adicional: "Atención requerida".
  - Las lecturas de temperatura y gas muestran valores elevados consistentes con un incendio.

Esta interfaz está diseñada para ofrecer conciencia situacional

inmediata, permitiendo al usuario distinguir entre seguridad y peligro en una fracción de segundo gracias al cambio de color y los avisos textuales grandes.

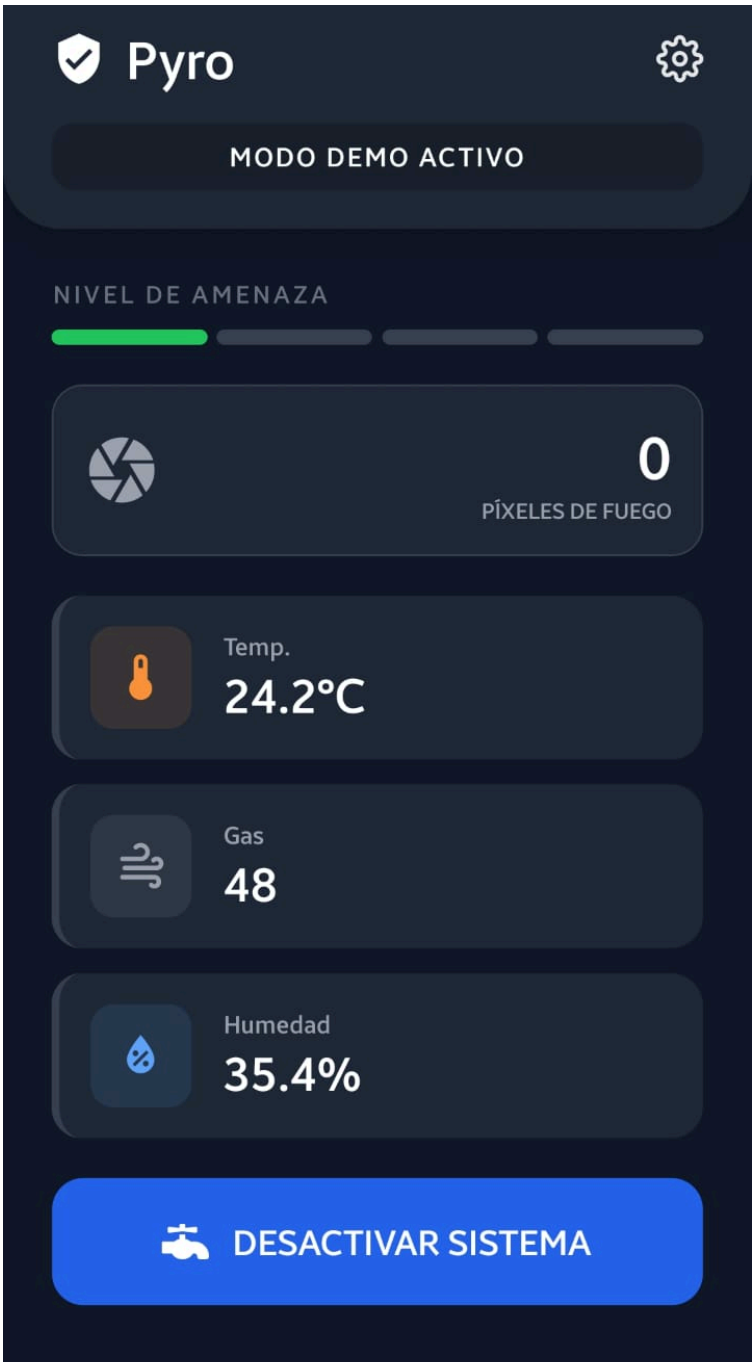
5.4.2 Página principal

Figura 9: IU. Página principal



5.4.3 Aspersores activados

Figura 10: IU. Aspersores activados



5.4.4 Fuego detectado

Figura 11: IU. Fuego detectado

6. Implementación

6.1 Plan de integración

- **Integración de hardware y software:**
  - **Sensores de temperatura:** Verificar que los sensores de temperatura marcan correctamente los niveles de temperatura.
  - **RaspBerry pi 4 B:** Verificar que el servidor funciona correctamente.
  - **Cámara web:** Verificar que la cámara web envía correctamente la información al servidor.
  - **Sensores de humo:** Verificar que los sensores de humo envían correctamente la información al servidor.
- **Integración de la interfaz de usuario:**
  - **Diseño:** Probar la funcionalidad del diseño de la aplicación móvil.
  - **Pruebas de conectividad:** Verificar que la aplicación móvil es capaz de conectarse al servidor de la RaspBerry.
- **Pruebas del sistema completo:**
  - Prueba de niveles de temperatura de riesgo.
  - Prueba de promedio de píxeles rojos en la cámara web.
  - Pruebas de alertas de incendio.

6.2 Descripción de la arquitectura

6.2.1 Nivel de Cliente (Frontend - Mobile)

- **Tecnología:** React Native.
- **Función:** Interfaz de usuario multiplataforma Android que permite el monitoreo y control remoto.
- **Comunicación:** La aplicación consume los datos y envía comandos mediante peticiones HTTP (API REST) hacia el servidor.
- **Características Clave:** Renderizado de alertas en tiempo real, visualización de métricas (Temperatura, Gas, Humedad) y detección de promedio de píxeles rojos en la cámara web.

6.2.2 Nivel de Servidor y Procesamiento (Backend)

- **Tecnología:** Python con el micro-framework Flask.
- **Dispositivo:** Raspberry Pi 4 Model B.
- **Función:** Actúa como servidor web local. Expone endpoints para que la app en React Native pueda consultar el estado de los sensores y recibir órdenes (ej. activar aspersores).
- **Lógica:** Ejecuta scripts de Python para la lectura de sensores Grovepi y análisis de colorimetría en las imágenes capturadas.

6.2.3 Nivel Físico (Hardware & Sensores)

- **Entrada (Input):**
  - Sensor de Gas MQ-2.
  - Sensor de Temperatura y Humedad.
  - Cámara Web para validación visual.
- **Salida (Output):**
  - Sistema de aspersores.

6.3 Modelo de implementación

6.3.1 Objetivo

Proveer una guía técnica y operativa detallada para la instalación, configuración y despliegue del sistema “Pyro” en un entorno, asegurando la correcta integración entre los componentes, el servidor local y la interfaz de usuario móvil.

6.3.2 Fases de Implementación

Fase 1: Instalación Física y Despliegue de Hardware. Esta fase se centra en la ubicación estratégica de los componentes para maximizar la cobertura de detección.

- **Montaje de la Unidad Central:** Fijación de la **Raspberry Pi 4** en una ubicación segura y con acceso a alimentación eléctrica continua.
- **Posicionamiento de Sensores:**
  - Instalación de sensores de gas **MQ-2** y temperatura/humedad en zonas de alto riesgo (ej. cocinas, salas de calderas), preferiblemente en altura para una detección efectiva de humo.
  - Alineación de la **Raspberry Pi Cam** para cubrir el campo visual de la zona monitoreada, garantizando una iluminación adecuada para el algoritmo de colorimetría.
- **Conexión de Actuadores:** Integración del sistema de aspersores mediante relés controlados por los pines GPIO de la Raspberry Pi, asegurando el aislamiento eléctrico adecuado.

Fase 2: Configuración de Software y Red Puesta a punto de la lógica del sistema y la comunicación Cliente-Servidor.

- **Despliegue del Servidor:** Inicio del servicio Flask en la Raspberry Pi y verificación de la exposición de la API en la red local.
- **Vinculación de la App Móvil:** Instalación de la aplicación React Native en el dispositivo del usuario y configuración de la dirección IP del servidor para establecer la comunicación vía HTTP.
- **Parametrización de Alertas:**
  - Definición de umbrales críticos en la interfaz.
  - Calibración de la sensibilidad del algoritmo de detección visual (colorimetría) para evitar falsos positivos por iluminación ambiental.

Fase 3: Pruebas Funcionales y Validación Verificación integral del sistema bajo condiciones controladas.

- **Simulación de Escenarios de Emergencia:**
  - Pruebas de detección térmica acercando fuentes de calor controladas a los sensores.
  - Validación del módulo de visión por computador presentando imágenes o videos de fuego real ante la cámara.
- **Verificación de Respuesta:**
  - Confirmar que la aplicación cambie su interfaz a "Estado de Alerta" (pantalla roja) en tiempo real.
  - Validar la recepción de notificaciones *push* en el móvil.
  - Comprobar la activación física de los aspersores al superar los umbrales establecidos.

6.3.3 Capacitación del Usuario Final

Asegurar la correcta operación del sistema por parte del residente.

- **Manual de Usuario:** Entrega de documentación sobre la interpretación de la interfaz gráfica.
- **Demostración Práctica:** Instrucción sobre el uso del "Panel de Acción" en la app móvil para funciones críticas, como la activación manual de aspersores y el procedimiento para desactivar falsas alarmas.

6.4 Módulos implementados

En esta sección se detallan los componentes de software desarrollados para el funcionamiento integral del sistema "Pyro":

- **Módulo de Sensores (Python/GrovePi):** Este módulo es el encargado de la comunicación directa con el hardware en la Raspberry Pi 4. Implementa scripts en Python para la lectura de los sensores de gas MQ-2 y de temperatura/humedad, de forma que el servidor pueda mandar valores digitales interpretables (PPM, °C y RH).

- **Módulo de Visión Computacional (Colorimetría):** Este componente procesa el flujo de video de la cámara web para detectar patrones visuales de incendio. Utiliza un algoritmo de análisis de píxeles “tipo fuego” para identificar predominancia de tonos rojos y amarillos, sirviendo como un método de validación visual que complementa a los sensores.

- **Servidor Backend (Flask API):** Se implementó un servidor ligero utilizando el micro-framework Flask. Este actúa como un puente de comunicación que permite a la aplicación móvil consultar lecturas de los sensores en tiempo real y enviar comandos, como la activación de los aspersores.

- **Interfaz de Usuario Móvil:** Se desarrolló una aplicación que ofrece un tablero de control. El módulo gestiona las alertas visuales (cambio de interfaz a rojo intenso ante peligro) y permite la interacción manual del usuario con el sistema de seguridad.

- **Código del servidor:**

```
from flask import Flask, jsonify, request

import random, grovepi, math, time, cv2, numpy

from threading import Lock


app = Flask(__name__)


# --- Configuración de Sensores ---

dht_sensor = 4

gas_sensor = 0

grovepi.pinMode(gas_sensor, "INPUT")


# --- Semáforo (Lock) ---

# Evita choques entre lecturas de sensores y peticiones de la App

lock = Lock()


# --- Estado de Aspersores ---

aspersores_activos = False


# --- Configuración de Cámara ---

cap = None


def inicializar_camara():
```

```
    global cap
    if cap is not None:
        cap.release()

    # Usamos backend V4L2 y bajamos resolución para evitar errores de USB/Timeout
    cap = cv2.VideoCapture(0, cv2.CAP_V4L2)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)
    cap.set(cv2.CAP_PROP_FPS, 15)

    if not cap.isOpened():
        print("⚠️ No se pudo abrir la cámara en el intento de inicio.")

# Arrancamos la cámara al principio
inicializar_camara()

# --- Rango de colores (Fuego) ---
lower_fire1 = numpy.array([0, 120, 150])
upper_fire1 = numpy.array([25, 255, 255])
lower_fire2 = numpy.array([25, 100, 150])
upper_fire2 = numpy.array([40, 255, 255])
kernel = numpy.ones((5, 5), numpy.uint8)

def leer_sensores():
    global aspersores_activos, cap
```

```
temp = -1
humidity = -1
gas_value = -1
fire_pixels = 0

with lock: # Bloqueamos para que nadie interrumpa el hardware
    # 1. Lectura de Sensores Físicos
    try:
        [temp, humidity] = grovepi.dht(dht_sensor, 0)
        gas_value = grovepi.analogRead(gas_sensor)
    except IOError:
        # Reintento rápido si falla
        try:
            time.sleep(0.1)
            [temp, humidity] = grovepi.dht(dht_sensor, 0)
        except:
            pass

    # 2. Lectura de Cámara con AUTO-REPARACIÓN
    if cap is not None and cap.isOpened():
        ret, frame = cap.read()

        if ret:
            # Procesamiento de imagen
```

```
            hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
            mask1 = cv2.inRange(hsv, lower_fire1, upper_fire1)
            mask2 = cv2.inRange(hsv, lower_fire2, upper_fire2)
            mask_fire = cv2.bitwise_or(mask1, mask2)
            mask_fire = cv2.morphologyEx(mask_fire, cv2.MORPH_OPEN, kernel)
            fire_pixels = cv2.countNonZero(mask_fire)
        else:
            # Si hay timeout, reiniciamos la cámara
            print("⚠️ Error leyendo frame (TIMEOUT). Reiniciando cámara...")
            inicializar_camara()
    else:
        inicializar_camara()

  return {
    "temperatura": temp,
    "humedad": humidity,
    "valorGas": gas_value,
    "firePixels": fire_pixels,
    "aspersores": aspersores_activos
  }

@app.route("/datos")
def datos():
  data = leer_sensores()
  return jsonify(data)
```

```
@app.route("/control_aspersores", methods=['POST'])
def control_aspersores():
    global aspersores_activos
    content = request.get_json()

    if content and 'accion' in content:
        with lock:
            aspersores_activos = content['accion']
            estado = "ACTIVADOS" if aspersores_activos else "DESACTIVADOS"
            print(f">>> Orden recibida: Aspersores {estado}")

        return jsonify({"estado_aspersores": aspersores_activos})

    return jsonify({"error": "Faltan datos"}), 400

if __name__ == "__main__":
    try:
        app.run(host="0.0.0.0", port=5000, debug=False, threaded=True,
use_reloader=False)
    finally:
            if cap:
                    cap.release()
```

- **Código principal de la aplicación (index.js):**

```
import { Feather, FontAwesome5, MaterialCommunityIcons } from
```

```
'@expo/vector-icons';
import AsyncStorage from '@react-native-async-storage/async-storage';
import React, { useEffect, useRef, useState } from 'react';
import { ActivityIndicator, Alert, Modal, SafeAreaView, ScrollView, StatusBar,
StyleSheet, Text, TextInput, TouchableOpacity, View } from 'react-native';
// 1. IMPORTAR NOTIFICACIONES
import * as Notifications from 'expo-notifications';

const STORAGE_KEY = '@server_ip';

// CONFIGURACIÓN DE UMBRALES
const THRESHOLD = {
  TEMP_WARN: 35,
  TEMP_CRIT: 50,
  GAS_WARN: 150,
  GAS_CRIT: 400,
  FIRE_PIXELS: 4000
};

// 2. CONFIGURAR COMPORTAMIENTO (Que suene aunque la app esté abierta)
Notifications.setNotificationHandler({
  handleNotification: async () => ({
    shouldShowAlert: true,
    shouldPlaySound: true,
    shouldSetBadge: false,
  }),
});

export default function App() {
  // --- CONFIG ---
  const [serverIp, setServerIp] = useState('');
  const [showSettings, setShowSettings] = useState(false);
  const [tempIp, setTempIp] = useState('');
  const [isIpLoaded, setIsIpLoaded] = useState(false);

  // --- ESTADOS ---
  const [riskLevel, setRiskLevel] = useState(0);
  const [riskMessage, setRiskMessage] = useState("Sistema estable");

  // Estado para el MODO DEMO (Simulación)
  const [demoMode, setDemoMode] = useState(false);

  const [sensorData, setSensorData] = useState({
    temperatura: 0,
    humedad: 0,
    valorGas: 0,
    firePixels: 0
```

```
  });

  const [sprinklersActive, setSprinklersActive] = useState(false);
  const [connected, setConnected] = useState(false);

  const intervalRef = useRef(null);
  const lastCameraRequestRef = useRef(0);

  // 3. REFERENCIA PARA EL ANTI-SPAM DE NOTIFICACIONES
  const lastNotificationTime = useRef(0);

  // 4. PEDIR PERMISO DE NOTIFICACIONES AL INICIAR
  useEffect(() => {
    async function requestPermissions() {
      const { status } = await Notifications.requestPermissionsAsync();
      if (status !== 'granted') {
        Alert.alert('Permiso denegado', 'No podremos avisarte si hay fuego si no
activas las notificaciones.');
      }
    }
    requestPermissions();
  }, []);

  // Cargar IP
  useEffect(() => {
    const loadIp = async () => {
      try {
        const savedIp = await AsyncStorage.getItem(STORAGE_KEY);
        if (savedIp) setServerIp(savedIp);
      } catch (e) { console.error(e); }
      finally { setIsIpLoaded(true); }
    };
    loadIp();
  }, []);

  // Bucle de Datos (Lectura constante o SIMULACIÓN DEMO)
  useEffect(() => {
    // Si no hay IP y no estamos en modo demo, no hacemos nada
    if (!serverIp && !demoMode) return;

    const fetchData = async () => {

      // --- LOGICA DEL MODO DEMO ---
      if (demoMode) {
        // Generar datos falsos para las fotos del informe
        const randomChance = Math.random();
        // 40% de probabilidad de mostrar situación de PELIGRO
```

```
            const isDanger = randomChance > 0.6;

            const fakeData = {
                // Si es peligro, temp > 50, si no, temp ~20
                temperatura: isDanger ? (52 + Math.random() * 10).toFixed(1) : (22 +
Math.random() * 3).toFixed(1),
                humedad: (30 + Math.random() * 15).toFixed(1),
                // Si es peligro, gas > 400, si no, gas ~50
                valorGas: isDanger ? (450 + Math.random() * 100).toFixed(0) : (40 +
Math.random() * 20).toFixed(0),
                // Si es peligro, muchos pixeles de fuego
                firePixels: isDanger ? (4500 + Math.random() * 1000).toFixed(0) : 0
            };

            setSensorData({
                temperatura: parseFloat(fakeData.temperatura),
                humedad: parseFloat(fakeData.humedad),
                valorGas: parseFloat(fakeData.valorGas),
                firePixels: parseFloat(fakeData.firePixels)
            });

            setConnected(true); // Engañar a la UI para que diga "CONECTADO"
            return; // Salir aquí para no intentar conectar de verdad
        }
        // --- FIN LOGICA DEMO ---

        const now = Date.now();
        const shouldRequestCamera = (now - lastCameraRequestRef.current) > 0;

        let url = `http://${serverIp}:5000/datos`;
        if (shouldRequestCamera) {
            url += '?camara=1';
            lastCameraRequestRef.current = now;
        } else {
            url += '?camara=0';
        }

        try {
            const controller = new AbortController();
            const timeoutId = setTimeout(() => controller.abort(), 3000);

            const response = await fetch(url, { signal: controller.signal });
            clearTimeout(timeoutId);

            if (response.ok) {
                const json = await response.json();
                const safeParse = (val) => (val === "Couldn't read" || val === undefined) ?
```

```
0 : parseFloat(val);

                setSensorData(prev => ({
                    temperatura: safeParse(json.temperatura),
                    humedad: safeParse(json.humedad),
                    valorGas: safeParse(json.valorGas),
                    firePixels: shouldRequestCamera ? safeParse(json.firePixels) :
prev.firePixels
                }));

                // Sincronizar estado real de aspersores desde el servidor
                if (json.aspersores !== undefined) {
                    setSprinklersActive(json.aspersores);
                }

                setConnected(true);
            } else {
                setConnected(false);
            }
        } catch (error) {
            setConnected(false);
        }
    };

    fetchData();
    intervalRef.current = setInterval(fetchData, 2000);
    return () => clearInterval(intervalRef.current);
  }, [serverIp, demoMode]); // Agregamos demoMode a dependencias

  // CEREBRO INTELIGENTE + NOTIFICACIONES
  useEffect(() => {
    if (!connected) return;

    const { firePixels, valorGas, temperatura } = sensorData;
    let score = 0;
    let reasons = [];

    if (temperatura > THRESHOLD.TEMP_CRIT) { score += 2; reasons.push("Calor
Extremo"); }
    else if (temperatura > THRESHOLD.TEMP_WARN) { score += 1;
reasons.push("Alta Temperatura"); }

    if (valorGas > THRESHOLD.GAS_CRIT) { score += 2; reasons.push("Humo
Denso"); }
    else if (valorGas > THRESHOLD.GAS_WARN) { score += 1;
reasons.push("Gases"); }
```

```
        if (firePixels > THRESHOLD.FIRE_PIXELS) { score += 3; reasons.push("Fuego Visible"); }

        // Determinar Nivel
        let currentLevel = 0;
        let message = "Ambiente Seguro";

        if (score >= 5) { currentLevel = 3; message = `PELIGRO: ${reasons.join(" + ")}`; }
        else if (score >= 3) { currentLevel = 2; message = `Alerta: ${reasons.join(" y ")}`; }
        else if (score >= 1) { currentLevel = 1; message = `Precaución: ${reasons[0]}`; }

        setRiskLevel(currentLevel);
        setRiskMessage(message);

        // 5. LÓGICA DE DISPARO DE NOTIFICACIÓN
        if (currentLevel >= 2) {
            sendEmergencyNotification(currentLevel, message);
        }

    }, [sensorData, connected]);

    // 6. FUNCIÓN PARA ENVIAR LA NOTIFICACIÓN
    const sendEmergencyNotification = async (level, bodyText) => {
        const now = Date.now();
        if (now - lastNotificationTime.current > 60000) {

            await Notifications.scheduleNotificationAsync({
                content: {
                    title: level === 3 ? "🚨 ¡PELIGRO - PYRO DETECTÓ FUEGO! 🚨" : "⚠️ Pyro: Advertencia del Sistema",
                    body: bodyText,
                    sound: true,
                    priority: Notifications.AndroidNotificationPriority.HIGH,
                    data: { data: 'goes here' },
                },
                trigger: null, // Enviar inmediatamente
            });

            lastNotificationTime.current = now;
        }
    };

    // --- LOGICA DEL BOTÓN DE ASPERSORES ---
    const toggleSprinklers = async () => {
        if (!connected) {
            Alert.alert("Pyro Offline", "Conecta el sistema primero");
            return;
```

```
        }

        const newState = !sprinklersActive;
        // Cambio optimista (UI instantánea)
        setSprinklersActive(newState);

        // Si estamos en MODO DEMO, solo mostramos la alerta simulada
        if (demoMode) {
            Alert.alert("Modo Demo", newState ? "Aspersores ACTIVADOS (Simulado)" :
"Aspersores APAGADOS (Simulado)");
            return;
        }

        try {
             const response = await fetch(`http://${serverIp}:5000/control_aspersores`, {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ accion: newState }),
            });

            if (!response.ok) throw new Error("Error server");

        } catch (error) {
            console.error(error);
            // Revertir si falló
            setSprinklersActive(!newState);
            Alert.alert("Error", "No se pudo enviar la señal a Pyro Server");
        }
    };

    // --- UI ---
    const saveSettings = async () => {
        if(tempIp.length > 0) {
            setServerIp(tempIp);
            setConnected(false);
            await AsyncStorage.setItem(STORAGE_KEY, tempIp);
        }
        setShowSettings(false);
    };

    const getThemeColor = () => {
        if (!connected) return '#374151';
        switch(riskLevel) {
            case 3: return '#EF4444';
            case 2: return '#F97316';
            case 1: return '#EAB308';
            default: return '#1F2937';
```

```
    }
  };

  if (!isIpLoaded) return <View style={styles.loadingContainer}><ActivityIndicator
size="large" color="#3B82F6" /></View>;

  return (
    <SafeAreaView style={[styles.container, { backgroundColor: '#111827' }]}>
      <StatusBar barStyle="light-content" backgroundColor={getThemeColor()} />

      <View style={[styles.header, { backgroundColor: getThemeColor() }]}>
        <View style={styles.headerContent}>
          <View style={styles.titleRow}>
            {riskLevel >= 2
              ? <MaterialCommunityIcons name="fire-alert" size={28} color="white"
/>
              : <MaterialCommunityIcons name="shield-check" size={28}
color="white" />
            }
            <Text style={styles.headerTitle}>Pyro</Text>
          </View>
          <TouchableOpacity onPress={() => { setTempIp(serverIp);
setShowSettings(true); }}>
            <Feather name="settings" size={24} color="white" style={{ opacity: 0.9
}} />
          </TouchableOpacity>
        </View>
        <View style={styles.statusBanner}>
          <Text style={styles.statusText}>
            {connected ? (demoMode ? "MODO DEMO ACTIVO" :
riskMessage.toUpperCase()) : "SISTEMA DESCONECTADO"}
          </Text>
        </View>
      </View>

      <ScrollView contentContainerStyle={styles.scrollContent}>

        <View style={styles.riskMeterContainer}>
          <Text style={styles.sectionTitle}>NIVEL DE AMENAZA</Text>
          <View style={styles.riskSteps}>
            <View style={[styles.riskStep, riskLevel >= 0 ?
{backgroundColor:'#22C55E'} : {}]} />
            <View style={[styles.riskStep, riskLevel >= 1 ?
{backgroundColor:'#EAB308'} : {}]} />
            <View style={[styles.riskStep, riskLevel >= 2 ?
{backgroundColor:'#F97316'} : {}]} />
            <View style={[styles.riskStep, riskLevel >= 3 ?
```

```
{backgroundColor:'#EF4444'} : {}]} />
                </View>
            </View>

            <View style={styles.section}>
              <View style={[styles.visualCard, riskLevel === 3 ?
styles.visualCardDanger : {}]}>
                <View style={styles.visualContent}>
                  <MaterialCommunityIcons
                    name={sensorData.firePixels > 100 ? "fire" : "camera-iris"}
                    size={40}
                    color={sensorData.firePixels > 100 ? "#EF4444" : "#9CA3AF"}
                  />
                  <View>
                    <Text style={styles.visualValue}>{connected ?
sensorData.firePixels : "--"}</Text>
                    <Text style={styles.visualLabel}>PÍXELES DE FUEGO</Text>
                  </View>
                </View>
              </View>
            </View>

            <View style={styles.grid}>
              <SensorCard
                icon="thermometer" title="Temp."
                value={connected ? sensorData.temperatura + "°C" : "--"}
                color="#FB923C" warning={sensorData.temperatura >
THRESHOLD.TEMP_WARN}
              />
              <SensorCard
                icon="weather-windy" title="Gas"
                value={connected ? sensorData.valorGas : "--"}
                color="#9CA3AF" warning={sensorData.valorGas >
THRESHOLD.GAS_WARN}
              />
              <SensorCard
                icon="water-percent" title="Humedad"
                value={connected ? sensorData.humedad + "%" : "--"}
                color="#60A5FA"
              />
            </View>

            <TouchableOpacity
              style={[
                styles.actionButton,
                sprinklersActive ? styles.btnActive : styles.btnInactive,
                !connected && { opacity: 0.5 },
```

```
                    riskLevel === 3 && !sprinklersActive && { borderColor: '#EF4444',
borderWidth: 2 }
                ]}
                onPress={toggleSprinklers}
                disabled={!connected}
            >
                <FontAwesome5 name="faucet" size={24} color={sprinklersActive ?
"white" : "#9CA3AF"} />
                <Text style={[styles.btnText, sprinklersActive ? {color:'white'} :
{color:'#9CA3AF'}]}>
                    {sprinklersActive ? 'DESACTIVAR SISTEMA' : 'ACTIVAR
ASPERSORES'}
                </Text>
            </TouchableOpacity>

        </ScrollView>

        <Modal visible={showSettings} animationType="slide" transparent={true}>
            <View style={styles.modalOverlay}>
                <View style={styles.modalContent}>
                    <Text style={styles.modalTitle}>Configuración Pyro</Text>

                    <Text style={{color:'#9CA3AF', marginBottom:5}}>Dirección IP
Raspberry:</Text>
                    <TextInput
                        style={styles.input}
                        value={tempIp}
                        onChangeText={setTempIp}
                        placeholder="Ej: 192.168.1.90"
                        placeholderTextColor="#6B7280"
                        keyboardType="numeric"
                    />

                    {/* --- BOTON MODO DEMO --- */}
                    <TouchableOpacity
                        style={[styles.input, {
                            backgroundColor: demoMode ? '#10B981' : '#374151',
                            borderColor: demoMode ? '#10B981' : '#4B5563',
                            alignItems: 'center',
                            justifyContent:'center'
                        }]}
                        onPress={() => {
                            setDemoMode(!demoMode);
                            if(!demoMode) {
                                Alert.alert("Modo Demo Activado", "Los datos mostrados son
simulados para demostración.");
                            }
```

```
                        setShowSettings(false);
                    }}
                >
                    <Text style={{color: 'white', fontWeight: 'bold'}}>
                        {demoMode ? "DESACTIVAR MODO DEMO" : "ACTIVAR
SIMULACIÓN (DEMO)"}
                    </Text>
                </TouchableOpacity>

                <View style={styles.modalButtons}>
                    <TouchableOpacity onPress={() => setShowSettings(false)}
style={styles.btnCancel}>
                        <Text style={styles.btnCancelText}>Cerrar</Text>
                    </TouchableOpacity>
                    <TouchableOpacity onPress={saveSettings} style={styles.btnSave}>
                        <Text style={styles.btnSaveText}>Guardar IP</Text>
                    </TouchableOpacity>
                </View>
            </View>
        </View>
      </Modal>
    </SafeAreaView>
  );
}

const SensorCard = ({ icon, title, value, color, warning }) => (
  <View style={[styles.card, warning && { borderColor: color, borderWidth: 1,
backgroundColor: color + '10' }]}>
    <View style={[styles.iconContainer, { backgroundColor: color + '20' }]}>
      <MaterialCommunityIcons name={icon} size={28} color={color} />
    </View>
    <View>
      <Text style={styles.cardLabel}>{title}</Text>
      <Text style={styles.cardValue}>{value}</Text>
    </View>
    {warning && <Feather name="alert-circle" size={16} color={color}
style={{position:'absolute', top:10, right:10}} />}
  </View>
);
```

6.5 Reporte de revisión

Tras completar la fase de integración, se realizaron pruebas funcionales para validar la fiabilidad del sistema bajo los escenarios definidos en el plan de pruebas:

- **Validación de Sensores:** Se comprobó que el sensor MQ-2 detecta correctamente concentraciones de gas/humo superiores a los 400 PPM, disparando la alerta en la aplicación. Además, el sensor de temperatura mostró una precisión adecuada al registrar incrementos ante fuentes de calor controladas.

- **Prueba de Conectividad y Alertas:** Se verificó la estabilidad del servidor Flask, confirmando que la aplicación móvil recibe notificaciones *push* y actualiza el estado de "En Vivo" de manera consistente dentro de la red local.

- **Análisis de Falsos Positivos:** Durante las pruebas del módulo de visión, se calibraron los umbrales de colorimetría para evitar que se interprete fuego erróneamente (Ej: objetos de color rojo/amarillo, luces led), optimizando la sensibilidad del algoritmo.

Conclusión

Durante esta etapa del desarrollo se logró profundizar de manera significativa en la planificación del proyecto, complementando el avance anterior con la elaboración de modelos conceptuales que representan con mayor precisión su funcionamiento. La definición detallada de los casos de uso, junto con el establecimiento de una arquitectura preliminar del sistema, proporciona una base metodológica robusta para la siguiente fase. En consecuencia, la transición hacia la etapa de implementación podrá abordarse con mayor coherencia, control y eficiencia. Considerando los resultados obtenidos en la planificación y modelación realizadas, se evidencia que el proyecto avanza de manera consistente y se encuentra bien encaminado hacia el logro de sus objetivos.

Referencias

[1] Falabella, "Meta Quest 3 512GB Almacenamiento Realidad Mixta". Disponible en: https://www.falabella.com/falabella-cl/product/140394966/Meta-Quest-3-512GB-Almacenamiento-Realidad-Mixta./140394970. [Accedido: 28 de Octubre de 2025].

[2] Centro Europeo de Postgrado (CEUPE), "Riesgos de un proyecto: qué son, tipos y cómo gestionarlos". Disponible en: https://www.ceupe.com/blog/riesgos-de-un-proyecto.html. [Accedido: 28 de Octubre de 2025].

[3] Smartsheet, "Tipos comunes de riesgos en la gestión de proyectos". Disponible en: https://es.smartsheet.com/content/project-risk-types. [Accedido: 28 de Octubre de 2025].

[4] Yuvaraj R, Senthil K, Sunil Arjun B, Krishnan Murugesan, Suresh Vellaiyan y Nguyen Van Minh "Real-time fire detection and suppression system using YOLO11n and Raspberry Pi for thermal safety applications", Case Studies in Thermal Engineering Volume 75, Noviembre 2025. Disponible en: https://www.sciencedirect.com/science/article/pii/S2214157X25014194.