Diseño e implementación de un sistema de seguridad para mitigar el hurto en las bodegas de insumos ubicadas en las plantaciones bananeras a cargo de ASBAMA

JOSÉ CARLOS OLIVA RIVERA

BAYRON TOVAR OLIVERA

RAVEN ADAILTON CALERO MANIGUA

UNIVERSIDAD MAGDALENA

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA ELECTRÓNICA

SANTA MARTA, COLOMBIA

2021

Diseño e implementación de un sistema de seguridad para mitigar el hurto en las bodegas de insumos ubicadas en las plantaciones bananeras a cargo de ASBAMA

JOSÉ CARLOS OLIVA RIVERA

BAYRON TOVAR OLIVERA

RAVEN ADAILTON CALERO MANIGUA

Trabajo presentado como requisito parcial para optar al título de:

INGENIERO ELECTRÓNICO

Director:

Ing. (Esp.) Carlos Eduardo Cabas Meriño

Universidad del Magdalena

Facultad de Ingeniería

Programa de Ingeniería Electrónica

Santa Marta, Colombia

2021

NOTA DE ACEPTACIÓN

_____

_____

_____

_____

_____

_____

_____

_____

Firma del Presidente del Jurado

_____

Firma del Jurado

_____

Firma del Jurado

Santa Marta, 12 de abril de 2021

DEDICATORIA

Dedico especialmente este proyecto a mis padres, Wilson Tovar Olivera y Diladis Olivera Benítez y hermanos Brandon Tovar Olivera y Braulio Tovar Olivera, los cuales me han dados las enseñanzas y consejos para poder crecer en el ámbito personal y profesional. Sin embargo, cabe mencionar que mi proceso de formación académica y crecimiento personal tendría un camino diferente si no hubiese sido por la ayuda de personas tan especiales como mi novia Camila Peláez y amigos Julia Rivas, Darwin Sánchez, José Oliva y Raven Calero. Y en general a todos los que de alguna manera han aportado a mi formación.

BAYRON TOVAR OLIVERA

DEDICATORIA

A mis padres, Carlos Miguel Oliva Vergara y Luz Stella Rivera Salgado, por afirmar e impulsar incondicionalmente mis ideas. A mi hermana, María Angelica Oliva Rivera. Por aconsejarme en todo el camino. A mis amigos por todo el apoyo que me brindaron en el proceso. A mis compañeros Bayron Tovar y Raven Calero que no solo estuvieron en la realización de este proyecto, sino que ayudaron a formar mi camino. A todos los que me inspiraron e impulsaron en mi crecimiento personal y profesional. Para Angela Martínez por todo, Siempre.

JOSÉ CARLOS OLIVA RIVERA

DEDICATORIA

Dedico especialmente este proyecto a mis padres, Miladys Esther Manigua Navarro y Efren Enrique Calero Guardiola, a mi hermano Efren Esneider Calero Manigua y a mi tío Rafael Manigua Navarro, por sus enseñanzas y dedicaciones, ellos con cada apoyo y enseñanza que me han dado contribuyeron en gran parte a la realización de este logro, lográndome formar en el ámbito personal como en el ámbito profesional. En el camino conocí a varias personas que también aportaron con un grano de profesionalismo como lo son José Carlos Oliva Rivera y Bayron Tovar Olivera grandes profesionales el día de hoy.

RAVEN ADAILTON CALERO MANIGUA

AGRADECIMIENTOS

Agradecidos principalmente con Dios, que nos regaló la sabiduría para llevar a cabo este proyecto, también agradecidos con el Centro de Innovación y Emprendimiento del CIE, que nos apoyó con la gestión del proyecto y gracias a sus retroalimentaciones logramos mejorar en el desarrollo del proyecto, gracias a que logramos resolver dudas a lo largo del camino, le damos gracias a todas las personas que hicieron parte del proyecto como lo fueron compañeros que alguna vez nos dieron sus puntos de vista. Seguidamente le damos gracias a nuestro director el ingeniero especialista Carlos Cabas Meriño por su gran apoyo al resolver dudas y por su supervisión en cada etapa de la documentación y desarrollo del prototipo.

Agradecemos a los padres y familiares que nos aportaron su apoyo y enseñanzas lo cual fue clave para seguir adelante en el desarrollo del prototipo y en el alcance de nuestras metas de nuestra carrera universitaria.

Por último, queremos agradecer a los ingenieros y docentes que hicieron parte de nuestro proceso de formación con cada saber que nos impartieron, lo cual nos fue de gran ayuda para realizar este gran proyecto de grado.

LISTA DE CONTENIDO

Pág.

LISTA DE TABLAS

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE ILUSTRACIONES

Pág.

LISTA DE ANEXOS

Pág.

RESUMEN

La intención de la práctica de innovación se basa en el diseño e implementación de un sistema de seguridad, encapsulado en una estación de control que, tome datos a través de sensores dispuestos óptimamente dentro del recinto. Este en su ejecución tiene como prioridad la prevención de los robos de insumos en las bodegas ubicadas en las fincas bananeras que están asociadas a ASBAMA. Por lo que, es capaz de enviar notificaciones al personal encargado (seguridad privada) de cualquier incidente. El sistema cuenta con sensores de registro de movimiento y elementos magnéticos, encargados de activar la vigilancia del perímetro, cuyo funcionamiento reside en el uso de una cámara ubicada estratégicamente, acompañada de una rutina interna de alarma con posibilidad de desactivación por medio de identificación RFID, mientras el usuario tenga credenciales activas. En caso de no poseerlas este enviará alertas tanto a los encargados del monitoreo del área, como a toda persona avalada por la administración.

La seguridad del espacio cimenta su funcionamiento en elementos como sensores de movimiento, sensor de puerta, módulo de RFID, Raspberry pi, cámara, lámpara, módulo GSM y un sistema de alarmas robusto, con interfaz para su administración. Cuando se registre un ingreso por parte del encargado o por cualquier intruso, los sensores de movimientos y/o el sensor de puerta detectarán dicho ingreso (sea autorizado o no). Posteriormente se procesarán estos datos para tomar decisiones al respecto. Si este determina que, en efecto se trata de un agente externo, enviará, mensajes de texto a teléfonos activos por el administrador a manera de alerta. Sin embargo, si el sujeto que ingresa a la bodega es el encargado, tendrá un tiempo corto, el cual es ajustable para desactivar el sistema con su etiqueta RFID. Esto quiere decir que, el o los encargados serán de las pocas personas acreditadas a portar una tarjeta. Asimismo, todo el sistema lo acompaña una base de datos con los registros de usuarios, teléfonos, configuración e ingresos, además de, una interfaz de administración donde se visualiza toda la data y las evidencias recolectadas con el tiempo.

El propósito de la práctica de innovación es diseñar e implementar un sistema de alarma con un punto de procesamiento de información dentro de la bodega, que prevenga el robo de insumos en las fincas bananeras que están asociadas a ASBAMA, capaz de notificar cualquier incidente por medio de mensajes de texto al personal previamente asignado. El sistema está formado por sensores de movimiento, cámara de vigilancia, lector RFID, sensor magnético de puerta, módulo GSM, una Raspberry pi y una lámpara. En un ingreso normal, la persona tiene cierto tiempo para desactivar la alarma antes que se active y envíe el mensaje de alarma;

no obstante, siempre se llevará registros de datos, fotográfico y de video que podrán ser visualizados en el aplicativo web. En caso contrario, que el ingreso no sea autorizado, los sensores de movimiento y el sensor de puerta serán capaces de detectar dicho ingreso y, enviar la señal a la Raspberry, la cual se encargará de continuar con su rutina de alarma basado en las decisiones que tome a partir de estos datos.

Todo el proyecto está enfocado a la innovación tecnológica e interconexión de dispositivos, implementado con programación en Python con ayuda del paradigma orientado a objetos y multiprocesos en cuanto a la aplicación del hardware se refiere. En el caso de la interfaz administrativa, se elaborará con ayuda de un framework que ayude con las tareas del backend del aplicativo, adicionalmente, el uso y conexión con la base de datos, y el despliegue de una página web para la vista de registros, evidencias, usuarios y configuración. Esto quiere decir que, la información procesada y obtenida se podrá visualizar en tiempo real en el aplicativo web que se diseña con características específicas.

Este prototipo permitirá funciones como:

•Registro de personal autorizado para el ingreso a la bodega.

•Identificación del usuario que ingreso al recinto, en caso de alguien ya registrado en el sistema.

•Identificación de señal de alerta, en caso de una persona no registrada ni autorizada.

•Reconocimiento de la hora exacta y día que se presentó el ingreso a la bodega ya sea autorizado o no.

•Permitirá acceder a la descarga del video que se almacenó, durante el tiempo que el sistema evidenció movimiento en la bodega y visualización de fotos.

•Cambio de resolución de captura de foto y video.

•Selección de usuarios a quien se les dará aviso.

•Configuración de los parámetros de la aplicación mediante interfaz administrativa.

•Encender y desactivar el sistema.

•Selección y visualización de usuarios y sus registros individuales.

•Identificación a través de etiquetas RFID.

Palabras claves: innovación, sistema, sensores, identificación, seguridad, administración, bodegas y Asbama.

ABSTRACT

The intention of the innovation practice is based on the design and implementation of a security system, encapsulated in a control station that collects data through optimally available sensors within the enclosure. This in its execution has as a priority the prevention of theft of supplies in the warehouses located in the banana farms that are associated with ASBAMA. Therefore, it is capable of sending notifications to the personnel in charge (private security) of any incident. The system has motion recording sensors and magnetic elements, responsible for activating perimeter surveillance, whose operation resides in the use of a strategically located camera, accompanied by an internal alarm routine with the possibility of deactivation by means of RFID identification, as long as the user has active credentials. In case of not having them, it will send alerts both to those in charge of monitoring the area, as well as to any person endorsed by the administration.

The security of the space bases its operation on elements such as motion sensors, door sensor, RFID module, Raspberry pi, camera, lamp, GSM module and a robust alarm system, with interface for its administration. When an entry is registered by the manager or by any intruder, the movement sensors and / or the door sensor will detect said entry (whether authorized or not). Subsequently, this data will be processed to make decisions about it. If this determines that it is indeed an external agent, it will send text messages to active phones by the administrator as an alert. However, if the person entering the warehouse is the person in charge, they will have a short time, which is adjustable, to deactivate the system with their RFID tag. This means that the person in charge will be one of the few people accredited to carry a card. Likewise, the entire system is accompanied by a database with user, phone, configuration and income records, as well as an administration interface where all the data and evidence collected over time is displayed.

The purpose of the innovation practice is to design and implement an alarm system with an information processing point inside the winery, which prevents the theft of supplies in the banana farms that are associated with ASBAMA, capable of reporting any incident by means of of text messages to previously assigned personnel. The system consists of motion sensors, surveillance camera, RFID reader, magnetic door sensor, GSM module, a Raspberry pi and a lamp. In a normal admission, the person has some time to deactivate the alarm before it activates and sends the alarm message; however, data, photographic and video records will always be kept that can be viewed in the web application. Otherwise, if the entry is not authorized, the motion sensors and the door sensor will be able to detect said entry and send the

signal to the Raspberry, which will be in charge of continuing with its alarm routine based on the decisions. that you take from this data.

The entire project is focused on technological innovation and device interconnection, implemented with Python programming with the help of the object-oriented and multiprocessing paradigm in terms of hardware application. In the case of the administrative interface, it will be developed with the help of a framework that helps with the tasks of the application's backend, additionally, the use and connection with the database, and the deployment of a web page to view the records, evidences, users and configuration. This means that the information processed and obtained can be viewed in real time in the web application that is designed with specific characteristics.

This prototype will allow functions such as:

• Registration of authorized personnel to enter the winery.

• Identification of the user who entered the venue, in the case of someone already registered in the system.

• Identification of the alert signal, in the case of a person not registered or authorized.

• Recognition of the exact time and day that the entry to the winery was presented, whether authorized or not.

• It will allow access to the download of the video that was stored, during the time that the system evidenced movement in the warehouse and viewing of photos.

• Change of photo and video capture resolution.

• Selection of users who will be notified.

• Configuration of application parameters through administrative interface.

• Power on and off the system.

• Selection and display of users and their individual records.

• Identification through RFID tags.

Keywords: innovation, system, sensors, identification, security, administration, warehouses and Asbama.

INTRODUCCIÓN

La necesidad surge por parte de los propietarios de las plantaciones bananeras, donde ASBAMA como intermediario, planea desplegar soluciones. Las cuales, tienen como objetivo disminuir considerablemente el hurto de insumos, con el fin de, evitar pérdidas al recomprar estos y como consecuencia directa e indirecta de la instancia fallida misma de los elementos robados. Esta situación genera incertidumbre al no encontrar un procedimiento definitivo a lo largo del tiempo, que mitigue los efectos y el propio hecho de la contravención. El contexto se agrava al entender que, varias de estas fincas son, por su definición y liquidez, pequeñas productoras que no cuentan con los recursos económicos mínimos para darle fin a dicho problema, provocando irremediablemente en una cantidad importante de casos, el cese de las actividades en las plantaciones. Por consiguiente, los empresarios están en la obligación de tomar decisiones que converjan en buscar soluciones innovadoras que, resulten técnicamente posible y económicamente viable.

Sin lugar a duda este inconveniente afecta directamente a la región del Magdalena y la Guajira. Puesto que, estos departamentos producen una alta cantidad de banano de tipo exportación, la cual requiere altos índices de calidad para su aceptación en el mercado internacional. Por consiguiente, durante el proceso de siembra y cosecha, se debe contar con el terreno, personal, maquinaria, insumo, sistema de riego y demás elementos del ciclo de trabajo, en las mejores condiciones. Según información suministrada por ASBAMA, los robos se presentan cada 15 días, llegando a un límite de incidencia semanal, donde se evidencian los fines de semana como los días de mayor actividad. Esto último, por obvias razones, no permite invertir a las plantaciones con materiales en el mejor de sus estados para la producción.

Es preciso decir que, refiriéndose al momento actual, la tecnología evoluciona a pasos agigantados, a tal punto de encontrar industrias agrícolas enteras, que implementan técnicas innovadoras, a partir de soluciones totalmente disruptivas en sus cultivos, para buscar la eficiencia de los sistemas que apunta al crecimiento de su economía, conociendo en tiempo real sus datos y necesidades, de forma segura y robusta. Es por este motivo que resulta redundante afirmar que todos los procesos, incluyendo en los que se enfoca esta práctica, serán optimados gracias a estos avances. De los cuales se esperan impactos que sean capaces de erradicar tales siniestros, como los que se presentan en la actualidad en las zonas bananeras del Magdalena y la Guajira.

El diseño del sistema, considerado para la bodega de insumos, se piensa no solo para reducir drásticamente los gastos que se generan en eventos como los mencionados con anterioridad, sino que, funcionan perfectamente como refuerzo positivo en segundo plano para la producción y la calidad del producto. Lo cual se ve reflejado directamente en la economía de la bananera. Además, llevar el registro de incidencias de cualquier tipo dentro de la locación, permite, en gran medida, llevar el control de las acciones ejecutadas, mediante su sistema de trazado digital. En otras palabras, de ser necesario podrán ser monitoreadas las operaciones que se efectúen en el lugar.

ALCANCES Y LIMITACIONES

El proyecto inicial ofrecido por la empresa de asociación de bananeros ASBAMA era trabajar en un prototipo que ayudará a monitorear y brindar seguridad a los aspersores de las fincas asociadas a ASBAMA, teniendo en cuenta que en el momento de su presentación de problemática estos tenían costos de $50.000-$250.000, en un robo cada finquero evidenciaba la pérdida de 3 o más aspersores dejándole perdidas de alrededor de $500.000 en una semana, sin embargo, lograron evitar este problema al cambiarlos por unos desechables los cuales costaban de $5.000-$15.000, por lo tanto lleva a implementar una nueva solución ofrecida a la seguridad y monitoreo de la bodega de insumos descartando así la anterior que iba dirigida a la seguridad de los aspersores, el proyecto, a pesar de las modificaciones, destaca por tener unos alcances muy exactos y necesarios como lo es el control de acceso y la vigilancia a la bodega de insumos evitando los robos en este sitio, lo que reduce en gran medida gastos de reinversión. El sistema consta de sensores de movimiento, cámaras de vigilancia, control de acceso por RFID, una alarma la cual se debe desactivar con la identificación RFID sino se desactiva acudirá a enviar mensaje de alerta a la persona designada de la finca.

Por lo anterior evidenciamos que el proyecto cumple con los criterios de seguridad que debe tener un recinto como lo es la bodega de insumos tanto el control de acceso es controlado como las distancias que pueden reconocer el movimiento los sensores para activar las cámaras.

El proyecto presenta varias limitaciones, la principal es el corto presupuesto por lo que se opta por usar un sensor de puerta como simulador de la puerta de seguridad, el personal autorizado después de que el sistema reconozca su identificación podrá pulsar el pulsador para obtener acceso en caso de que sea satisfactorio el sistema enviará un visto bueno activando el sistema para simular el accionamiento de la puerta por medio del pulsador.

Al ser un proyecto basado en la seguridad de la bodega tenemos como limitante la seguridad por fuera del recinto por lo que se recomienda seguridad privada que pueda recibir el mensaje o en su defecto autoridades de la zona, para evitar hurtos en caso de que pueda registrarse un ingreso de una persona que tiene en su poder un reconocimiento RFID de personal autorizado sin él serlo y por desconocimiento del sistema se active la alarma y nadie pueda llegar a evitar el hurto.

IMPACTO ESPERADO DEL PROYECTO

El proyecto tiene como finalidad brindarles alternativas y/o herramientas de seguridad a las fincas asociadas a ASBAMA, sin importar que tan grande o pequeña estas sean. El proyecto está directamente enfocado en las bodegas centrales de cada una de estas, donde se albergan los insumos pertenecientes a las plantaciones, entre las que cabe resaltar las herramientas de trabajo y otros tipos de objetos de considerable valor que ayudan al desarrollo del producto.

Resulta indispensable para la gerencia de las plantaciones, generar sistemas que puedan brindar seguridad en los recintos que han sido afectados notablemente por estos siniestros, que involucran, indiscutiblemente, a sus mismos trabajadores (desleales) y personal externo que saca provecho del hurto de los materiales que se aseguran dentro del local. Con ayuda de funcionarios de ASBAMA se calcula una cifra que ronda entre los COP $500.000 y los COP $2.000.000 (quinientos mil a dos millones de pesos colombianos) como perdidas quincenales a causa de los robos, teniendo en cuenta que este es un promedio realizado por ellos basándose en las fincas que tienen adscritas, que, además, no se añaden a este valor las pérdidas del producto que resulta afectado por la falta del insumo; puesto que, esta estimación funge como de coste incalculable.

Para garantizar completamente la seguridad del recinto, es necesaria la ayuda de personal de seguridad privada o de autoridades como caso exclusivo de acompañamiento. El prototipo tendrá simulaciones que se centra en evitar gastos por reinversión, con ayuda de su sistema de registros para la detección y aseguramiento de la información provocada por la intromisión. Con lo cual los finqueros de pequeñas, medianas y grandes bananeras asociadas a ASBAMA, verán reflejados mayores ingresos y tranquilidad en sus negocios, gracias a la prevención o en su defecto eliminación de los hurtos hacia sus materiales.

A partir de todo el proceso realizado se espera obtener resultados del desarrollo de esta práctica que, si bien son ajenos al problema, indirectamente dan otras soluciones en ámbitos un poco diferentes:

Incentivar a los maestros y estudiantes que se interesen en el tema a la creación de proyectos propios donde utilicen el prototipo para darle solución a sus problemas. Ya que, estos dispositivos son versátiles, bastante potentes y económicos.

El proyecto está intencionado a ser un proceso que busca la relación entre la tecnología y el campo, e inclusive la apropiación social de personal de seguridad en zonas rurales por medio del uso integral de dispositivos electrónicos.

Este proyecto será motivación para estudiantes, recién graduados, pequeñas y medianas empresas para que se atrevan a emprender nuevos retos e innovar para dar soluciones viables económicamente. Además de que, las Universidades y empresas vean emprendiendo a las personas y, se animen a invertir aún más en ideas que no se materializan por falta de recursos económicos.

Desarrollar todo el potencial de las herramientas dadas para presentar un prototipo funcional para la empresa, y aprovechar las líneas futuras para optimizar en gran medida, la interfaz administrativa y la aplicación que serán desarrolladas a lo largo de esta práctica.

MARCO TEÓRICO

Dada la necesidad de seguridad que se encuentra en las fincas bananeras por su gran expansión territorial, se opta por recurrir a grandes inversiones de personal de seguridad para asegurar tanto el perímetro de expansión como las diferentes zonas de interés (bodega de insumos, entradas y salidas de la finca).

Este personal de seguridad demanda grandes cantidades de inversión que tienden a ser imposibles de costear para propietarios de fincas pequeñas y medianas, con lo cual se ve la necesidad de emplear sistemas automatizados de seguridad en los diferentes puntos de acceso a la finca o zonas de almacenamiento de insumos y herramientas, por lo que nosotros planteamos la seguridad de la bodega de insumos que normalmente también se encuentran en este lugar las herramientas y otros objetos usados en el diario de la finca.

Esto con el fin de disminuir el costo a mediano y largo plazo de personal de seguridad usando la ***tecnología***[1] (entendiéndose por tecnología la ciencia aplicada a la resolución de problemas concretos. Constituye un conjunto de conocimientos científicamente ordenados, que permiten diseñar y crear bienes o servicios que facilitan el monitoreo de recintos brindando seguridad y registro de estos lugares. Ha ayudado a la solución de los problemas de la sociedad desarrollando toda clase de dispositivos como por ejemplo ***sensores***[2], los sensores son dispositivos electrónicos que están capacitados para detectar acciones o estímulos generados por una variable física o química entregándolas de forma eléctrica. Con el uso de estos se pueden registrar diferentes variables como lo son: la distancia, velocidad, dirección, luminosidad, humedad, temperatura, movimiento, etc.)

Generando en nosotros las ganas de ***diseñar***[3] (diseñar es elegir entre un abanico muy grande de opciones cuál es la que construiremos. Esta es una forma clara y conveniente de entender las responsabilidades que el buen diseño implica) e ***implementar***[4] (poner en funcionamiento o aplicar métodos, medidas, etc., para llevar algo a cabo) un ***sistema***[5] (un sistema es un módulo ordenado de elementos que se encuentran interrelacionados y que interactúan entre sí) electrónico el cual ayudará al monitoreo, registro y seguridad de la bodega de insumos brindándole la

[1] https://concepto.de/tecnologia/ , Autor: María Estela Raffino. De: Argentina. Para: Concepto.de. Disponible en: https://concepto.de/tecnologia/. Consultado: 18 de abril de 2020.
[2] http://paolaguimerans.com/openeart/2018/05/05/que-son-los-sensores/ , Consultado 18 de abril de 2020.
[3] http://www.mordecki.com/mordecki.com/que-es-disenar/, Autor: Daniel Mordecki, Consultado 18 de abril de 2020.
[4] https://www.wordreference.com/definicion/implementar , Consultado 18 de abril de 2020.
[5] https://definicion.de/sistema/ , Autor: Julián Pérez Porto, Consultado 18 de abril de 2020.

reducción en personal de seguridad dirigida a este espacio, debemos tener en cuenta que este sistema tendrá sensores por ejemplo el sensor de movimiento con el cual se busca identificar el acceso que se tuvo a la bodega de insumos para así activar el sistema interno de la bodega ya que se consta de un sistema externo que cumple la función de ingreso y registro de la bodega.

Este sistema logrará funcionar a la perfección gracias a la ***programación***[6] (se entiende como programación de un dispositivo electrónico la parte no visible que ejecuta las acciones y genera respuestas a variables monitoreadas ya sea para almacenarlas o transmitirlas) que se le debe hacer tanto al sistema interno y externo que conjuntamente cumplen con un sistema robusto de seguridad, brindándole a las personas interesadas el monitoreo de quien ha entrado a la bodega de insumos, seguridad por lo que solo personal autorizado debidamente registrado anteriormente puede ingresar a la cabina y en caso de que el acceso sea autorizado el sistema interno logrará captar el video de la persona que ingreso y que hizo dentro de la bodega de insumos con lo cual se logra evitar posibles anomalías en la prestación del servicio de seguridad.

Cabe resaltar que a pesar de ser un sistema de seguridad robusto necesita el acompañamiento de personal de seguridad alrededor el cual se encargará de en caso de un intento de ingreso no autorizado ser la persona que se dirija a la bodega de insumos a revisar si es una persona con acceso autorizado a la cual se le están presentando fallas con su credencial de acceso o si es una persona que quiere burlar el sistema para hacer un hurto de la bodega de insumos.

[6] https://definicion.de/programacion/ , Autor: Julián Pérez Porto y Ana Gardey, Consultado 18 de abril de 2020.

ANTECEDENTES

Con el pasar de los años la inseguridad se ha vuelto “pan de todos los días” y se ha visto reflejado en las estadísticas asociadas a esta, como referencia tenemos el hurto que viene afectando a muchas personas y empresas del país, con lo cual se viene presentando problemas de hurtos en fincas bananeras por lo cual se ha dispuesto en crear un sistema robusto y a la vez accesible para el bolsillo de todos los finqueros del país, principalmente a las fincas asociadas a ASBAMAS que se encuentran en la región caribe de Colombia.

Evidenciando estos casos de hurto y las estadísticas presentada por los dirigentes de ASBAMAS se busca realizar un sistema recompilando principios y funciones de diferentes proyectos asociados para lograr un sistema robusto y rentable teniendo en cuenta que las fincas asociadas a esta empresa pueden ser pequeñas, medianas y grandes.

En Septiembre del año 2017, Oscar Gabriel Fuentes Lanfor y José Fernando Pérez Pérez, durante una conferencia en panamá exponen la “Implementación de un sistema de seguridad independiente y automatización de una residencia por medio del internet de las cosas” el cual se basa en el internet de las cosas teniendo en cuenta el monitoreo de casas por medio de cámaras de videos y sensores de movimiento que logran encender hasta 18 luces y 2 puertas a distancia, teniendo la facilidad de monitorear estos recintos por medios web, con lo cual evidencian el uso de sensores pasivos infrarrojos HC-SR501 que permite controlar este sistema de control de manera confiable, con un nivel preciso de aceptación [1].

En el año 2019, Mónica Paulina Gutiérrez Díaz y Rodrigo Villegas Téllez, muestran “Sistema de control de acceso basado en tecnología Arduino y RFID” el cual es una de las bases de implementación de nuestro proyecto por la forma que logran generar un acceso confiable por medio de RFID el cual permite tanto controlar quien puede ingresar y quien no y a su vez es capaz de mantener un registro de las entradas al recinto con lo cual nos sirve a nosotros como principal plus para implementar en otro tipo de placa electrónica que nos pueda brindar mayor confianza y mayores ventanas de posibilidades de almacenamiento y tecnología [2].

Carlos Andrés González Godoy y Octavio José Salcedo Parra, en un artículo presentado a la revista virtual de la Universidad Católica del Norte publican un sistema de seguridad basado en sensores de movimiento PYR, este registra datos cada 10 segundos enviando una captura de la imagen tomada por las cámaras al correo electrónico autorizado, siendo un sistema de seguridad aceptable que depende de la iluminación del lugar puesto que esto ayuda al funcionamiento del

sensor, la programación de este sistema es implementada en una placa Raspberry 2 modelo B lo cual puede ser un sistema de aprendizaje para mejorar el sistema robusto que se planea desarrollar [3].

En el mes de diciembre del año 2006, en la Universidad Técnica de Ambato dos estudiantes de la carrera de ingeniería electrónica y comunicaciones Geovanni Brito y Cecilia Izurieta, desarrollaron un sistema de vigilancia y monitoreo por medio de cámaras pi, las cuales brindaban acceso en tiempo real y desde cualquier lugar por medio del internet, con el cual lograban evitar intentos de robo o en caso registrar el hurto que se dio, el proyecto se llama “Diseño de un Sistema de Seguridad mediante Cámaras IP para la Empresa Proalpi de la Ciudad de Píllaro” [4].

El estudiante Iván Pérez de la carrera de ingeniería mecatrónica de la Universidad Tecnología Equinoccial, desarrolla la tesis “Desarrollo de un sistema de seguridad para parqueaderos basados en tecnología RFID” buscando implementar el reconocimiento por radiofrecuencia para controlar el ingreso y la salida de estos parqueaderos [5].

Con los anterior expuesto se evidencia que la mayor similitud entre los proyectos es garantizar la seguridad de los recintos utilizando todo tipo de tecnologías funcionales, que en nuestro caso se pretende realizar un sistema robusto por lo cual se debe realizar un conjunto de estas funciones para lograrlo, como lo es elegir la mejor placa electrónica que nos ayude a evidenciar que funciona de manera correcta la programación realizada para el sistema, los siguientes requisitos se van tomando de acuerdo a lo necesitado que en nuestro caso es controlar el acceso y monitorear lo que pase dentro de estos recintos por lo cual se llegó a la elección de cámaras para almacenamiento de video dentro del recinto, sensores de movimientos dentro del recinto para la activación del sistema y para controlar el acceso una puerta robusta con ingreso por RFID el cual nos ayudará a controlar y monitorear los ingresos al recinto que en nuestro caso es la bodega de insumos de la finca.

1 DESARROLLO TECNOLÓGICO

1.1 DESCRIPCIÓN DEL PROBLEMA

Entre las dificultades que se presentan en las bananeras de la región, existe uno muy común y es el robo de insumos y suministros, los cuales son usados a diario para la producción del fruto y otros derivados. Estas plantaciones bananeras cuentan con una bodega para el almacenamiento de estos insumos, entre los cuales podemos encontrar vástagos, fertilizantes, palas, picos, cajas, bolsas, etc. El hurto de estos, crea un detrimento en las finanzas de este sector, lo que ha llevado a la quiebra a pequeños finqueros.

Uno de los impedimentos principales es la falta de tecnología aplicada a la agricultura y la seguridad. Normalmente estas plantaciones están alejadas de la comunidad, donde es muy costoso implementar algunas soluciones tecnológicas que sean asequibles para el dueño de la finca, además la señal telefónica no es buena y la mayoría de las fincas no cuentan con internet. Los altos costos de soluciones tecnológicas a estos problemas, también son un impedimento para la disminución de estos los hurtos.

1.2 DESCRIPCIÓN DE LA SOLUCIÓN

La solución planteada a esta problemática es un sistema de seguridad a través de una estación de control, conformada por sensores de movimiento, cámara, sensor magnético de puerta, tecnología RFID, módulo GSM y centro de procesamiento. La puerta de la bodega tendrá un sensor magnético que nos ayudará a saber cuándo abren y cierran la puerta. Cuando haya un ingreso forzado (que no se desactive la alarma), el módulo GSM enviará un mensaje o llamada al vigilante para que este acuda al lugar, dicha alarma se desactivará con una tarjeta RFID. Cada ingreso a la bodega será grabado y almacenado para futuras revisiones, por si ocurre algún impase. Los sensores de movimiento también darán aviso y activará algunas funciones del sistema con el fin de evitar estos hurtos.

1.3 LIBRERIAS UTILIZADAS

1.3.1 SERIAL

Esta librería encapsula el acceso para el puerto serie. Proporciona backend para Python que se ejecuta en Windows, osx, Linux, bsd e ironpython. Hacemos uso de esta librería para la conexión del sim800l, el cual nos permitirá la transferencia de información por el tx y rx de nuestras raspberry y el sim800l.

1.3.1.1 PRINCIPALES CARACTERÍSTICAS

•Acceso a la configuración del puerto a través de las propiedades de Python.

•Soporte para diferentes tamaños de bytes, bits de parada, paridad y control de flujo con RTS / CTS y / o Xon / Xoff.

•Trabajar con o sin recibir tiempo de espera.

•Archivo como API con "leer" y "escribir" ("readline", etc. también es compatible).

•Los archivos de este paquete son 100% Python.

•El puerto está configurado para transmisión binaria. Sin eliminación de bytes NULL, traducción CR-LF, etc. (que muchas veces están habilitadas para POSIX). Esto hace que este módulo sea universalmente útil.

•Compatible con la biblioteca io

1.3.2 TIME

En la librería time hay funciones para obtener la fecha y/o hora de distintos tipos de relojes; para obtener información relacionada con nuestro huso horario; para convertir fechas y/o horas entre distintos tipos; para validar y aplicar formatos de fechas y/o horas y para detener durante un tiempo la ejecución de un programa. En la ejecución del sistema se lleva un almacenamiento de la evidencia fotográfica, tal evidencia es guardada y nombrada con la fecha y hora en la cual se presenta una incidencia en el sistema, esta fecha y hora nos la brinda esta librería.

1.3.3 SUBPROCESS

Esta librería nos permite invocar procesos desde Python y comunicarse con ellos: enviar datos a la entrada (stdin) y recibir la información de salida (stdout). Además, esperar a que el proceso finalice o bien terminarlo cuando queramos, y obtener el valor de retorno. Resulta ideal y es el método recomendado para ejecutar comandos del sistema operativo o lanzar programas (en lugar de la tradicional os.system ()) y opcionalmente interactuar con ellos. La forma más sencilla de ejecutar un comando o invocar un proceso es con la función run (). Entre las funciones más importantes en nuestro código es la de matar procesos por encontrarse algún error o estar en un ciclo infinito, entre otras.

1.3.4 OS

Este módulo proporciona una forma portátil de utilizar la funcionalidad dependiente del sistema operativo. Nos permite leer o escribir un archivo, manipular rutas, también si queremos leer todas las líneas en todos los archivos en la línea de comando, además para crear archivos, directorios temporales y para el manejo de archivos y directorios de alto nivel. Es usada esta librería en el código principalmente para crear directorios y modificar los nombres de carpetas y archivos.

1.3.5 PICAMERA

Esta librería nos proporciona una interfaz Python pura para el módulo de cámara raspberry pi. Nos permite tomar fotografías, videos, transmisión, etc. además, podemos renderizar, codificar, y modificar colores entre otras más funciones.

1.3.6 REQUESTS

Nos permite enviar solicitudes HTTP con Python sin necesidad de tanta labor manual, haciendo que la integración con los servicios web sea mucho más fácil. No es necesario agregar manualmente consultas a las urls o de convertir información a formularios para realizar una solicitud post. Todo esto es logrado gracias a la buena integración de urllib3en requests.

1.4 MODULOS

1.4.1 SIM800L

Ilustración 1. Módulo SIM800L

Fuente: Extraído de https://descubrearduino.com/sim800l-gsm/

Este es un módulo que utiliza tecnología GSM para la comunicación con otros dispositivos por medio de voz, texto y datos. Es un módulo cuatribanda que permite agregar funcionalidades avanzadas de comunicación a través de la red celular, además se requiere un microcontrolador para controlarlo.

1.4.1.1 CARACTERÍSTICAS MÁS IMPORTANTES DEL SIM800L

-Alimentación entre 3.7v y 4.4v

-Consumo medio de 350mA con picos de 2ª

-Pines con lógica de 2.8v

-Detección de la velocidad del puerto

-Solo trabaja con tecnología 2G.

-Puertos TX/RX

Se necesita una fuente de poder confiable y sin variaciones para obtener un buen rendimiento de este módulo, ya que se han realizado varias pruebas con diferentes baterías, cargadores y reguladores, con el fin de que este módulo trabaje correctamente. Hemos llegado a la conclusión de usar una batería Li-Po de 3.7v a 800mAh en paralelo con un capacitor de 2200 µF a 50v, lo que nos permitirá suportar los picos de corriente que este módulo requiere en ciertas ocasiones en particular. La simcard que lleve este debe contar con mensajes y llamadas, y en su

defecto datos móviles. Como este dispositivo es controlado por comandos AT y se requiere de una interfaz (micro controlador) para ejecutar los comandos que lo harán funcionar.

En términos de GPRS tiene las siguientes características:

- Velocidad máxima de transmisión 85.6 Kbps

- Protocolo TCP/IP en chip

- Codificación: CS-1, CS-2, CS-3 y CS-4

- Soporta USSD

- Soporta reloj en tiempo real (RTC)

- Velocidades de transmisión serial desde 1200bps hasta 115 200 bps

Entre los comandos AT más usados en esta tecnología encontramos:

Tabla 1. Lista de comandos AT más usados para la comunicación del SIM800L

Código	Función
AT	Para verificar si el módulo SIM800L está funcionando adecuadamente
AT+CGSN	Para visualizar el IMEI del chip utilizado
AT+CSCS?	Para saber el tipo de texto configurado, por defecto es GSM
AT+CSCS="XXX"	Configurar el tipo de texto
AT+CMGF?	Ver el formato de un mensaje, ya sea PDU(0) o SMS(1)
AT+CMGS=XXXXXXX	Para enviar un SMS
AT+CMGL=ALL	Sirve para ver todos los mensajes que nos han llegado al SIM
ATD04455XXXXXXXX	Sirve para hacer una llamada a cualquier teléfono móvil
ATA	Sirve para contestar una llamada
ATH	Sirve para colgar una llamada

Fuente: Extraído de https://www.estudioelectronica.com/wp-content/uploads/2018/09/ISTD-034.pdf

1.4.1.2 CONFIGURACIÓN E INTERCONEXIÓN CON EL MÓDULO SIM800L.

Se necesita desactivar uno de los servicios que viene activo por defecto en la Raspberry Pi que usa el puerto serie de nuestra placa, para que solo se use este puerto para el módulo GSM:

```
sudo systemctl stop serial-getty@ttyAMA0.service
```

sudo systemctl disable serial-getty@ttyAMA0.service

Ahora por medio del editor Nano, modificaremos el archivo cmdline.txt, ya que este restringe la comunicación con el módulo GSM.

Escribimos *sudo nano /boot/cmdline.txt*

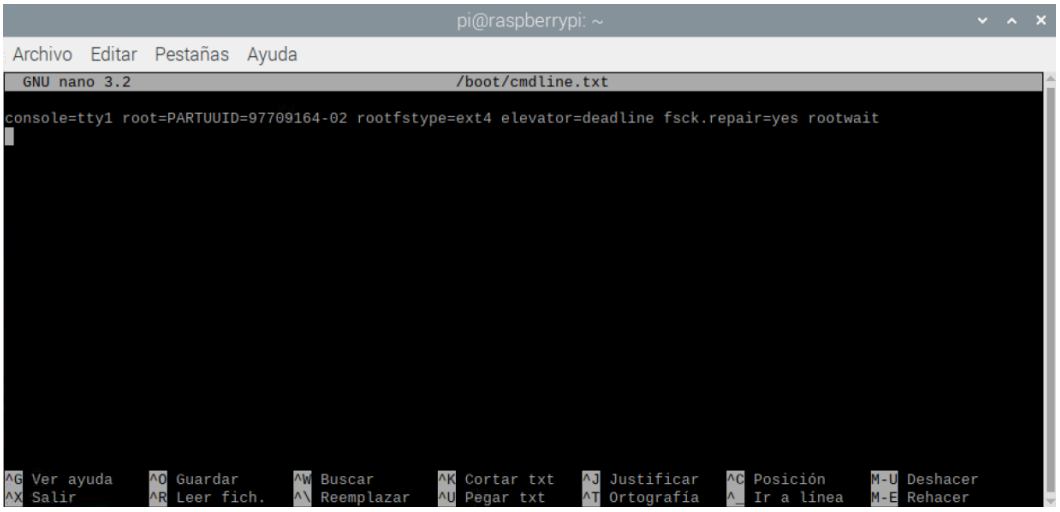
En nuestro caso, este archivo viene con el siguiente código por defecto:

console=serial0,115200 console=tty1 root=PARTUUID=97709164-02 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait quiet splash plymouth.ignore-serial-consoles

El cual reemplazaremos por:

console=tty1 root=PARTUUID=97709164-02 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait

En este último se le borra *quiet splash plymouth.ignore-serial-consoles* para que nos permita la comunicación serial.



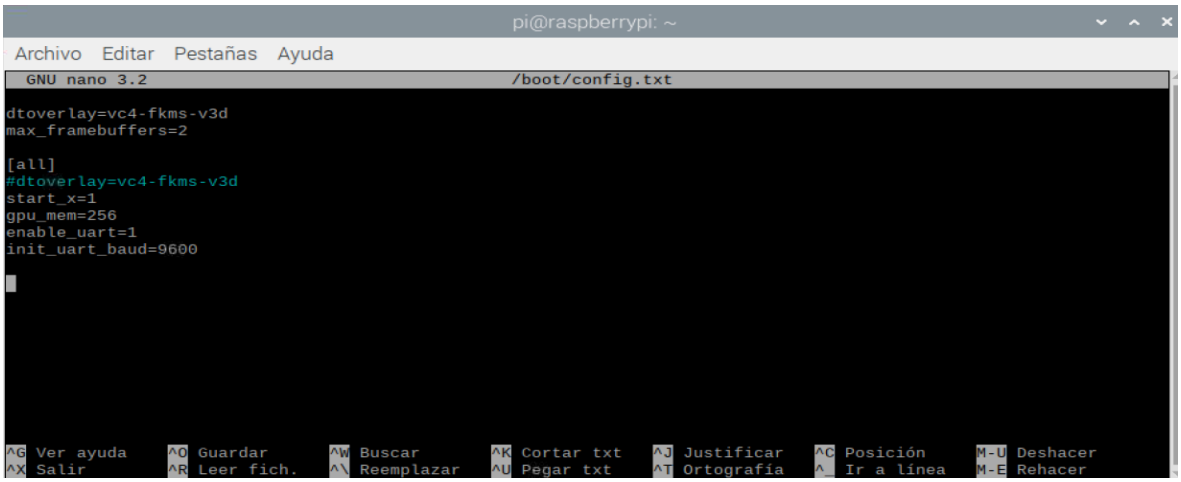
Ahora para configurar el archivo de inicio y ajustar el reloj del UART, editaremos el archivo config.txt digitando:

sudo nano /boot/config.txt

En este archivo debemos adicionar estas líneas al final del texto

enable_uart=1

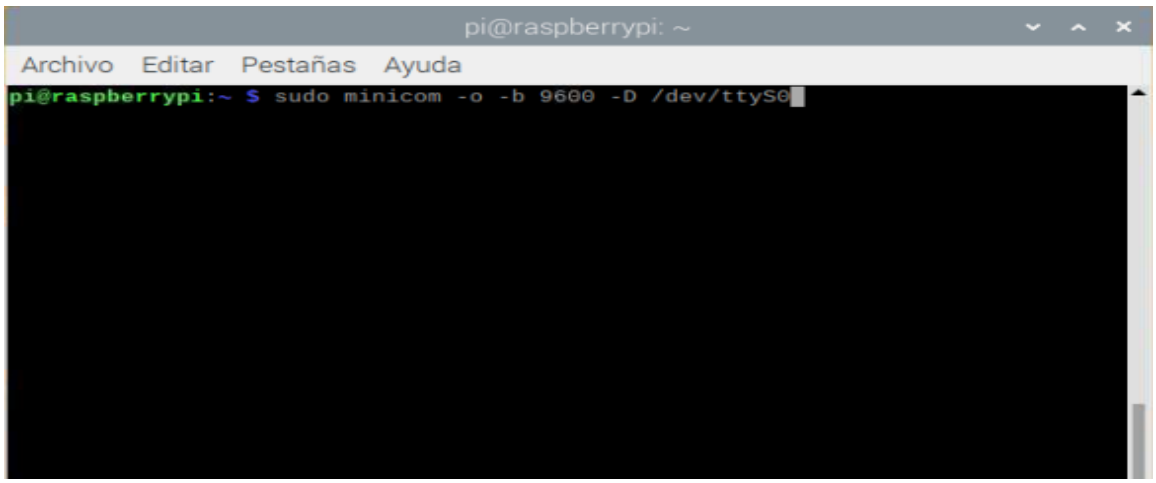
init_uart_baud=9600



Guardamos y finalmente reiniciaremos la tarjeta.

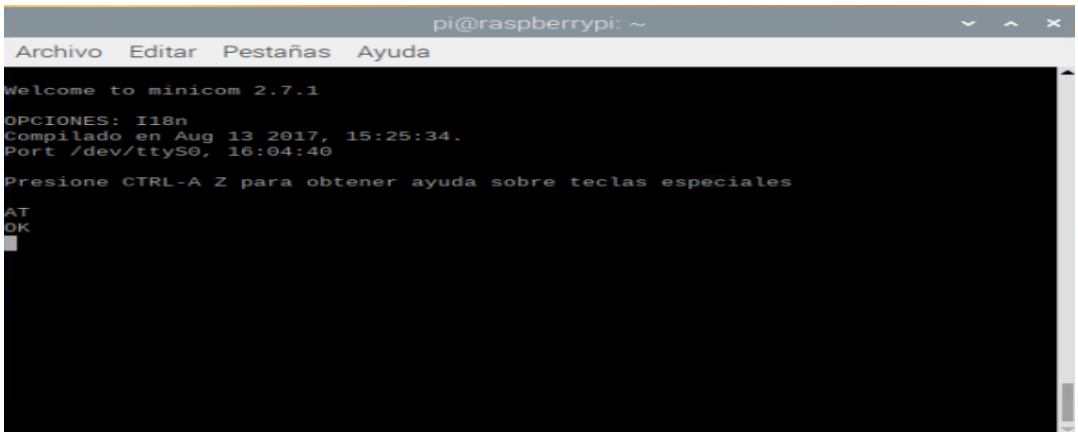
Ahora procedemos a instalar Minicom que nos permitirá la configuración de algunas condiciones del puerto serie. Con el comando *sudo apt-get install minicom* lo habremos instalado y procedemos a ejecutar el siguiente comando:

sudo minicom –o –b 9600 –D /dev/ttys0



Con los comandos AT antes mencionado podemos verificar el correcto funcionamiento de la comunicación con del GSM.

Al escribir AT en la interfaz que nos brinda minicom, nos debe dar con respuesta OK y así sabremos que el GSM está listo para su funcionamiento.



Ya con los pasos anteriores tenemos nuestro modulo listo para la comunicación.

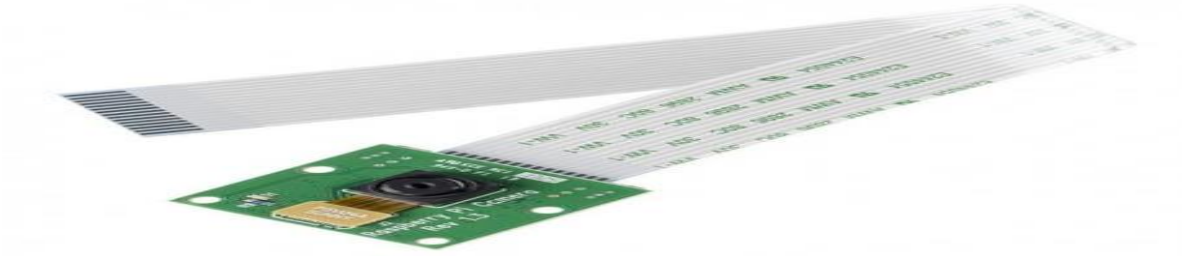
A continuación, tenemos el código necesario para que, en conjuntos con los demás módulos, el SIM800L pueda funcionar de forma correcta dentro del sistema.

VER ANEXO 2

Primeramente, en el código se hace la activación y configuración de los valores iniciales del módulo, como lo son a cuantos baudios se establecerá la comunicación, al igual que el sistema queda a la espera que el módulo se conecte a la red de Tigo. Seguido de esto, tenemos la ejecución de los códigos AT necesarios para que nuestro dispositivo envíe los mensajes necesarios.

1.4.2 CÁMARA

Ilustración 2. Módulo de la cámara para la Raspberry Pi



Fuente: Tomado de https://www.bigtronica.com/centro/sistemas-mini-pc/raspberry-pi/915-camara-raspberry-pi-v13-5mp-1080p-5053212009151.html

Este módulo nos permite grabar y tomar fotos para procesarlos en la Raspberry pi. Esta cámara es de 5 Megapíxeles con un lente de foco fijo. Es capaz de tomar imágenes de 2592 x 1944, y también es compatible con el formato de video 1080p30, 720p60 y 640x480p60/90. Se conecta al Raspberry Pi por medio de un pequeño conector en la parte superior de la tarjeta y utiliza la interfaz dedicada CSi, diseñado especialmente para la conexión de cámaras.

1.4.2.1 CARACTERÍSTICAS

- 1,4 µm X 1,4 µm píxeles con tecnología OmniBSI de alto rendimiento (alta sensibilidad, baja diafonía, ruido bajo)
- Tamaño óptico de 1/4 “
- Funciones de control de imagen automáticas:
  - Control automático de exposición (AEC)
  - Balance de blancos automático (AWB)
  - Filtro de banda automático (ABF)
  - Calibración del nivel de negro automático (ABLC)
- Controles programables para la velocidad de fotogramas, AEC / AGC 16 zonas / posición / control de peso, espejo y lado, recorte, ventanas, y el panorama
- Puerto de vídeo digital (DVP) Interfaz de salida en paralelo
- 32 bytes de memoria programable una sola vez incorporada (OTP)

1.4.2.2 ESPECIFICACIONES TÉCNICAS

- Resolución: 1/4 5M
- Apertura: 2.9
- Longitud focal: 3.29
- FOV: 65 grados
- Tipo de sensor: OmniVision OV5647 Color CMOS QSXGA (5 megapíxeles)
- Tamaño del sensor: 3.67 x 2.74 mm (formato 1/4″)
- Cantidad de píxeles: 2592 x 1944
- Tamaño de píxel: 1.4 x 1.4 um
- Lente: f = 3.6 mm, f / 2.9
- Ángulo de visión: 54 x 41 grados
- Campo de visión: 2.0 x 1.33 m a 2 m
- Lente SLR de fotograma completo equivalente: 35 mm
- Video: 1080p a 30 fps con códec H.264 (AVC)
- Video de hasta 90 fps en VGA
- Tamaño de la placa: 25 x 24 mm (sin incluir el cable flexible)

1.4.2.3 CÓDIGO

Para visualizar el código dirigirse a VER ANEXO 3

El código anterior es el que se encarga del funcionamiento la cámara y está en el archivo cámara.py. Entre sus funciones está la de activar y desactivarla con unas

configuraciones de grabación estipuladas desde el aplicativo web, la librería Picamera nos permite todos los cambios de resolución, renderización, entre otros. A su vez, guarda las fotos y videos en carpetas que tienen de nombre la fecha y la hora en la cual se registró el ingreso a la bodega, estos valores lo obtenemos gracias a libraría Time.

1.4.3 SENSOR DE PUERTA

El sensor magnético sirve para puertas y ventanas. El mecanismo del sensor MC-38 es normalmente cerrado (NC) y envía un 1 lógico cuando ambas partes del sensor están en contacto y 0 cuando están separadas. Este sensor nos servirá como bandera para saber si la puerta de la bodega la abre o la cierran. Es de vital importación que la instalación de este se haga de forma precisa, porque solo detectará cambios de estado si la separación entre las barras está entre 15 y 25 mm

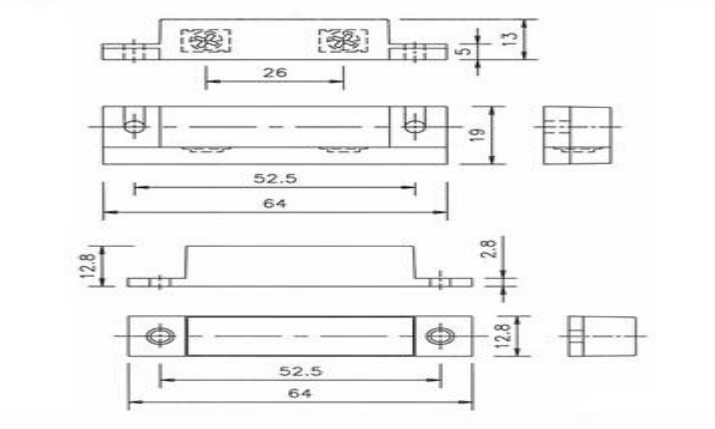
Ilustración 3. Sensor magnético de puerta

Fuente: Tomado de https://www.carrod.mx/products/sensor-magnetico-para-puerta-con-cable-para-arduino-mc-38

1.4.3.1 ESPECIFICACIÓN Y CARACTERÍSTICAS

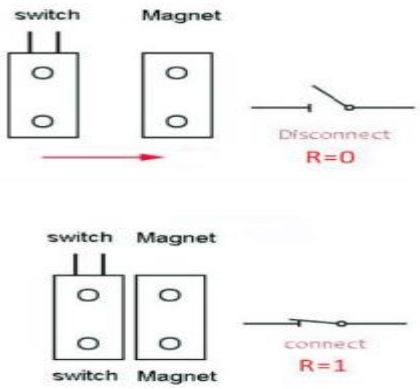
- Modelo: MC-38
- Voltaje de alimentación máxima: 100V
- Corriente máxima: 0.5 A
- Potencia nominal: 3 W
- Mecanismo: Normalmente Cerrado (NC)
- Distancia de activación mínima: 15 mm
- Distancia de activación máxima: 25 mm
- Largo del cable: 25 cm
- Dimensiones: 27 mm x 14 mm x 8 mm
- Peso: 16 gramos
- Color: Blanco

Ilustración 4. Dimensiones del sensor de puerta



Fuente: Tomado de https://www.domodesk.com/516-sensor-de-puerta--ventana-inalambrico.html

Ilustración 5. Sistema de funcionamiento del sensor de puerta



Fuente: Tomado de https://www.domodesk.com/516-sensor-de-puerta--ventana-inalambrico.html

1.4.4 LECTOR DE TARJETAS RFID

La identificación por radiofrecuencia RFID basa su funcionamiento en comunicaciones inalámbricas entre un emisor y un lector a través de ondas de radio. Estas son comúnmente aplicadas en establecimientos comerciales y/o privados para garantizar que sus productos sean retirados con los permisos correspondientes o, por el contrario, no permitir el ingreso de personas o bienes a estos. La premisa es simple, su labor es estrictamente transferir información entre emisor y receptor. El primero puede encontrarse de dos tipos, pasivo o activo, donde la principal diferencia radica en tener o no fuente de alimentación. Los tags pasivos,

sufren de alcance limitado, tomando como aproximado entre 4 y 6 metros. Por su parte, los activos pueden radiar entre 2 y 10 veces más que las anteriores.

Para lograr la comunicación, el receptor o como se llamará en adelante, lector, emite una señal continua con un determinado alcance, cuando el tag entra en contacto con la RF (no se necesita contacto físico entre los dispositivos), envía información que, hace referencia a la identidad de un objeto en concreto. La cual, es interpretada por el lector, basando su programación en cierto protocolo.

Las tarjetas RFID, también llamados etiquetas pasivas o transpondedores, son chips que funcionan con la propia señal del lector. Puesto que, alimentan el circuito mediante la antena. Su fin radica en enviar una respuesta, que cuenta con el criterio de ser el código de identificación único del tag. Los RFID que predominan en el mercado actual, son las correspondientes a las bandas LF y HF debido a las ventajas que estos ofrecen como:

- Lecturas más rápidas, sin perder la precisión.
- Mejoras en el flujo y la reducción potencial de tiempo.
- Ayuda a conocer rápidamente los elementos.
- Se Integran mejor con múltiples tecnologías y así previenen el robo de bienes.

Para lograr un estándar es necesario abordar protocolos de comunicación específicos para la comunicación entre el lector y la tarjeta, especificar el formato de los datos y certificar que el desarrollo pueda operar con dispositivos de distintos fabricantes.

En resumen, la conceptualización del funcionamiento del sistema es simple, la tarjeta RFID contiene los datos de identificación del objeto, la cual genera señal al contacto con radiofrecuencia, cuya información es captada por un lector RFID encargado de leer y transpilar la información a un formato conocido por el sistema. Es importante remarcar que este tiene dos componentes fundamentales, La etiqueta (tarjeta, tag, transpondedor) compuesta por un chip, un transductor y una antena, los cuales guardan información, transforman la señal a radio y transmiten los datos de la tarjeta, además de energizar el circuito, respectivamente [tipos de tarjetas]. El lector, que integra una antena, un transceptor y un decodificador.

1.4.5 PROTOCOLO WIEGAND 26.

Wiegand es un estándar enfocado en lectores de tarjetas. Normalmente se usa para conectar un mecanismo de lectura de tarjeta al resto de un sistema de control de acceso. Básicamente el sistema utiliza tres cables, de los que uno es la conexión

común a tierra y, los dos restantes asumen el rol de la transmisión de datos. Generalmente llamados DATA0 y DATA1, nombrados en algunos casos como D0 y D1 respectivamente. Cuando no se envían datos, tanto DATA0 como DATA1 se elevan al nivel de voltaje alto, también llamado uno lógico. Cuando se envía un 0, el DATA0 tiende a un cero lógico mientras que el cable DATA1 permanece en su estado inicial. En caso contrario, DATA1 envía un voltaje bajo mientras que DATA0 permanece en un uno lógico.

El protocolo de comunicaciones utilizado se conoce como protocolo Wiegand. El formato Wiegand original tenía un bit de paridad, 8 bits de código de instalación, 16 bits de código de identificación y un bit de paridad final para un total de 26 bits. Sabiendo que, el primer bit de paridad se calcula con los 12 primeros bits del código y el bit de paridad final con los 12 bits finales.

1.5 SISTEMA DE SEGURIDAD ASBAMA

La realización de esta solución se define en la implementación de dos partes que resultan fundamentales para la evolución y ejecución de las actividades. Estas enmarcan, a rasgos generales la administración y respuesta a los incidentes que puedan o no ocurrir en cualquier escenario previamente pensado y evaluado por el equipo de trabajo basado en los requerimientos de los encargados por parte de la organización Asbama. Estas partes llevan el nombre de AsbamaAdmin y Security respectivamente, son ejecutadas al tiempo y se comunican entre sí por medio de un servicio que se conecta con una misma base de datos.

Para entender el funcionamiento de ambas partes de la aplicación es necesario segmentar y extender por separado. Puesto que, aunque se basen en el mismo lenguaje su aplicación es distinta en casi su totalidad. Por ende, su naturaleza es disímil. Esto quiere decir que el enfoque es diferente en el sentido de que AsbamaAdmin corre el software de la interfaz administrativa y Secutiry la aplicación, que se comunica con los módulos que toman datos del exterior y opera con ellos para realizar su operación y posteriormente generar interactividad con el usuario final.

1.6 ASBAMAADMIN.

Este apartado describe la interfaz administrativa de la solución, la cual está pensada para que el usuario encargado pueda visualizar el registro de ingresos generales e individuales a la instalación. Esto quiere decir que, se encontrará un home con las listas de los registros totales con su respectiva fecha, usuario, y descripción. Es importante resaltar que dicha tabla soporta el registro de cualquier ingreso no autorizado. Asimismo, se puede visualizar individualmente en el perfil del usuario toda la actividad que ha realizado. La finalidad de llevar un control estricto es tener

total transparencia en las transacciones que se efectúan, y reducir el saque de cualquier elemento que pertenezca a la organización.

Implícitamente, se debe incluir un sistema de usuarios, el cual permita tener el registro de cada uno, para modificarlos, crearlos, leerlos, pero no borrarlos. Adicionalmente una sección donde se puedan visualizar y, que además posea un buscador. Cada usuario tendrá tanta información como sea requerida, donde es obligatorio tener nombre de usuario, tarjeta relacionada, numero de contacto, contraseña y rol asignado. Este último surge de la necesidad de mostrar información basado en la posición del cliente. Debido que, tener todas las bondades de la interfaz en manos de cualquiera persona, podría incurrir en problemas de seguridad, por el nivel de administración y configuración que posee esta. Por lo tanto, se implementa un sistema de roles, inspirado en los encontrados en los CRM-ERP's del mercado, el cual determinará el nivel de acceso de cada usuario a las características, información y distribución del sistema.

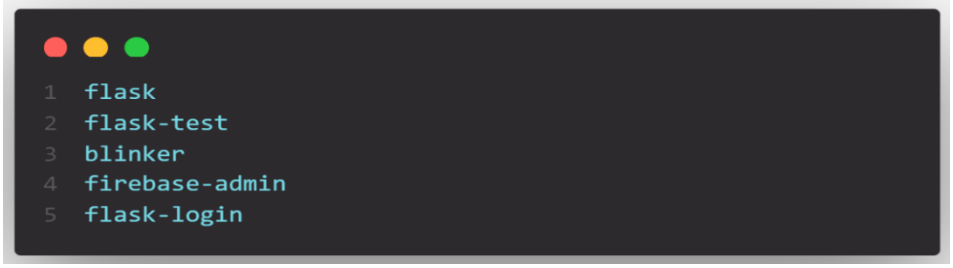
Como se mencionó con anterioridad, es imperativo que cada usuario posea un número de teléfono. Ya que, resulta en información útil para la sección Security en caso de enviar alertas de intrusos en la plantación. El servicio toma los teléfonos que necesita no basado en el rol asignado, si no en una característica especial de cada documento, que establece si el usuario está apto o no para recibir el mensaje. Indistintamente del rango en la organización o en la aplicación de la persona.

El sistema en su totalidad está protegido ante errores HTTP, bien sea por parte del servidor o del usuario, los más comunes son el error 404 y 500, que resultan ser Not Found e Internal Server Error, respectivamente. Este manejador en caso de incurrir en alguno, redirige la plataforma a una sección que muestra el identificador del error, su nombre e información acerca de este mismo.

Por ser una aplicación que se encarga de la administración del sistema, se añade un apartado donde se muestra toda la configuración de Security donde podemos modificar datos como el encendido del sistema, los tiempos de espera, la resolución de las cámaras, entre otros detalles. Es importante denotar que, solo estará disponible para los usuarios con un rol específico.

Para ahondar en la implementación de AsbamaAdmin, es necesario conocer e instalar las dependencias dispuestas en un archivo del repositorio como se muestran en la Figura 1. Este se ejecutará en un sistema Linux con una distribución basada en Debian optimizada para correr en Raspberry, llamada RASPBIAN. En primera instancia es necesario instalar Python globalmente en el sistema [6], particularmente se utilizó en su versión 3.7. Así mismo, se requiere de su gestor de paquetes PIP [7]. Para ajustar las necesidades específicas que requiere para su funcionamiento se crea un ambiente virtual con ayuda de Virtualenv [8], instalado con el gestor.

Figura 1. Requerimientos de AsbamaAdmin



Fuente: Propia.

La aplicación que sirve el sistema está escrita basando sus componentes en Flask [9], a razón de ser más liviano que sus contrapartes. Esto principalmente se debe a que se trata de un framework, que no requiere de herramientas particulares para funcionar. Sin embargo, no cuenta con una capa de abstracción para base de datos (ORM) [10], ni para validación de formularios, tecnologías de autenticación abiertas, entre otras herramientas relacionadas a un framework, por lo que de ser necesario se puede añadir bibliotecas externas que proporcionen funciones comunes. En otras palabras, esto no significa que tenga funcionalidad limitada, por el contrario, esto busca 'mantener el sistema simple pero extensible', resolviendo en una herramienta potente que permite tener todo lo que sea necesario y dimitir de lo que sobre.

Para conectar la aplicación con una base de datos se hace uso de firebase-admin [11], el cual es un SDK de administración que permite interactuar directamente con Firebase desde un entorno que posea ciertos privilegios para ejecutar acciones como leer y escribir datos, acceder a los recursos de Google Cloud Platform como Cloud Storage y base de datos de Firestore asociadas a proyectos Firebase. Este último es el utilizado en la actual implementación de la interfaz administrativa, por lo que es completamente necesario instalar la herramienta de línea de comandos conocida como gcloud [12] que permite por medio de la consola inicializar la conexión, con un proyecto específico relacionado con una cuenta de Google y dotar a la aplicación con las autenticaciones necesarias. Con la finalidad de, que dentro del sistema se pueda crear y administrar los recursos disponibles en la nube.

1.6.1 NOSQL

Flask en sus herramientas, no cuenta con un sistema de bases de datos u ORM predefinido en su entorno que, en cortas palabras, ayude a convertir la información almacenada en la DB a un sistema orientado a objetos. Aunque a priori suene como desventaja para la aplicación, realmente brinda la posibilidad de manejar cualquier base de datos, gracias a la bondad de ser extensible, resultando necesario solo la implementación del sistema seleccionado. Es decir, que se puede hacer uso de bases de datos SQL, llenas de tablas relacionadas con campos estructurados (filas)

que resulta no ser favorable en concreto para este tipo de aplicación. Sin embargo, se puede manejar con ayuda de extensiones como SQLAlchemy [13], que hace las veces de ORM para bases de datos MySQL o SQLite ejemplos de las más utilizadas, Asimismo, se puede utilizar NoSQL, que cuenta con composición abierta y flexible a distintos tipos de datos, teniendo la ventaja de no necesitar recursos excesivos para ejecutarse. Además, no requiere una tabla fija como en su contraparte más común y es altamente escalable a bajos costos en cuanto a hardware se refiere. Refiriéndose a la desplegada por Google, conocida con el nombre de Firestore, el cual es una base de datos orientada a documentos.

1.6.2 FIRESTORE

Firestore maneja el concepto de colecciones que hace referencia directa a una tabla, se puede apreciar una comparativa entre distintos tipos de base de datos en la Tabla 2, donde precisamente se tiene en las dos últimas columnas, las distinciones más comunes entre una base de datos relacional y Cloud Firestore.

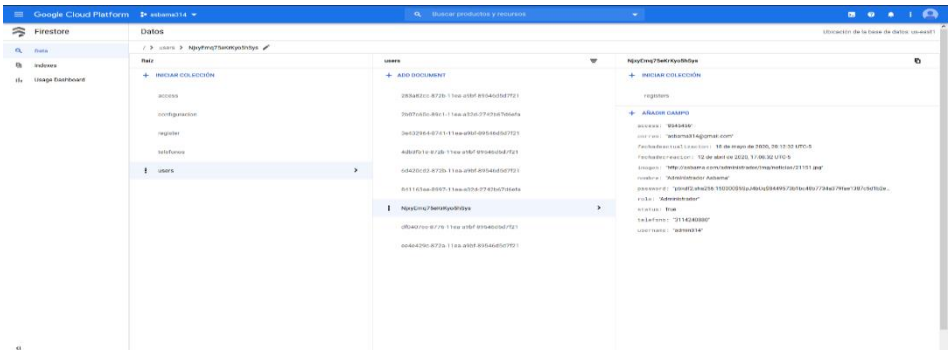
Tabla 2. Tabla comparativa entre distintas bases de datos

Concepto	Datastore	Firestore	Relacional
Categoría de Objetos	Tipo	Grupo de colecciones	Tabla
Un Objeto	Entidad	Documento	Fila
Dato individual	Propiedad	Campo	Columna
Id único	Llave	Id del Documento	Llave Primaria

Fuente: Propia

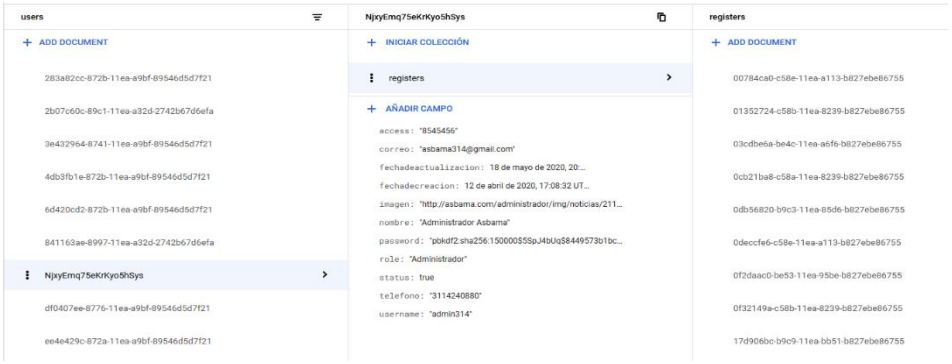
Dentro de cada documento, se pueden anidar más colecciones, lo cual es pertinente en este escenario, a razón de ser preciso para guardar registros globales en una colección externa a los usuarios y, locales en cada documento de la colección ‘users’, donde se consignan individualmente esta información. En la Figura 2, se puede divisar la distribución en la consola de las distintas colecciones utilizadas por esta solución. Además, en la Figura 3, se aprecia el anidado de los antes mencionados dentro del documento de un usuario.

Figura 2. Consola de Google donde se consignan los datos del aplicativo.



Fuente: Propia

Figura 3. Colección anidada dentro de un documento.



Fuente: Propia

1.6.3 CONFIGURACIÓN DE GCLOUD

En ese orden e ideas, se instala el Google Cloud SDK en un sistema Linux, donde el resultado esperado radica en obtener un comando por terminal con el nombre gcloud, que permita la autenticación con la consola de Google. Por lo que se procede a descargar la opción optima pata el sistema operativo, como se muestra en la Figura 4.

Figura 4. Paquetes de instalación de gcloud.

Plataforma	Paquete	Tamaño	Suma de verificación SHA256
Linux de 64 bits (x86_64)	google-cloud-sdk-301.0.0-linux-x86_64.tar.gz	78.3 MB	4773911e9f8fb7fe3b915e1f24ebe957436474253688820a9f2990d7399bd3b8
Linux de 32 bits (x86)	google-cloud-sdk-301.0.0-linux-x86.tar.gz	76.5 MB	9dd34f130459c9dd3f8a0c5d74439245d58c27e7f3d5dda11ae7e6f3ea59209a

Fuente: Propia

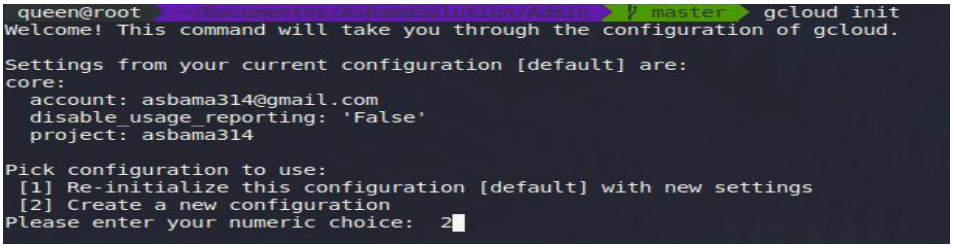
De fallar la instalación por este método se tiene como alternativa el uso de curl [14] por la línea de comandos, para un Linux de 64 bits, como se puede entrevé en la Figura 5.

Figura 5. Comando por terminal para la instalación de gcloud.

Fuente: Propia

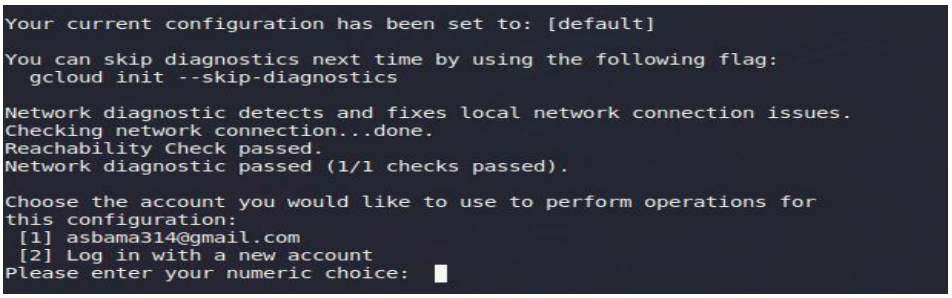
Se continua con la respectiva inicialización con ayuda del comando gcloud init [15] que en su práctica ejecuta varias tareas de configuración del SDK, en las que se incluye la autorización de las herramientas utilizadas para acceder Google Cloud con credenciales de una cuenta específica. En las Figuras 6, 7, 8 y 9 se precisa la serie encargada de ajustar la disposición del ambiente.

Figura 6. Selección de la configuración a utilizar.



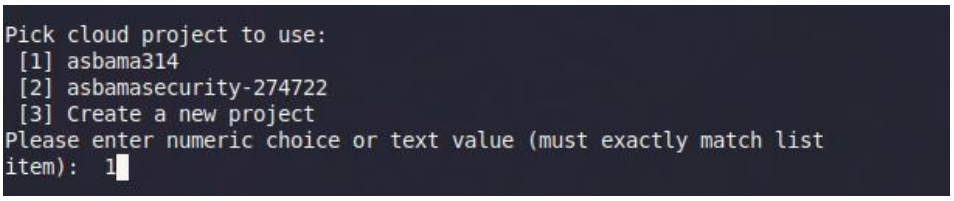
Fuente: Propia

Figura 7. Selección de la cuenta asociada a la consola deseada



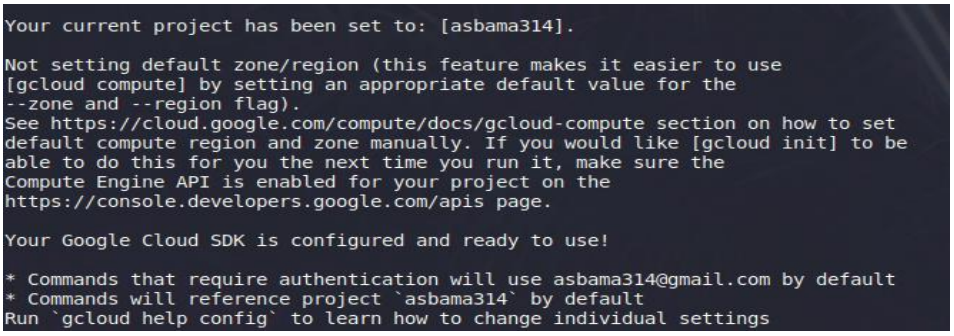
Fuente: Propia

Figura 8. Selección del proyecto a utilizar



Fuente: Propia

Figura 9. Mensaje de asignación correcta de la selección de proyecto, e información de los pasos siguientes para la autenticación de la aplicación.



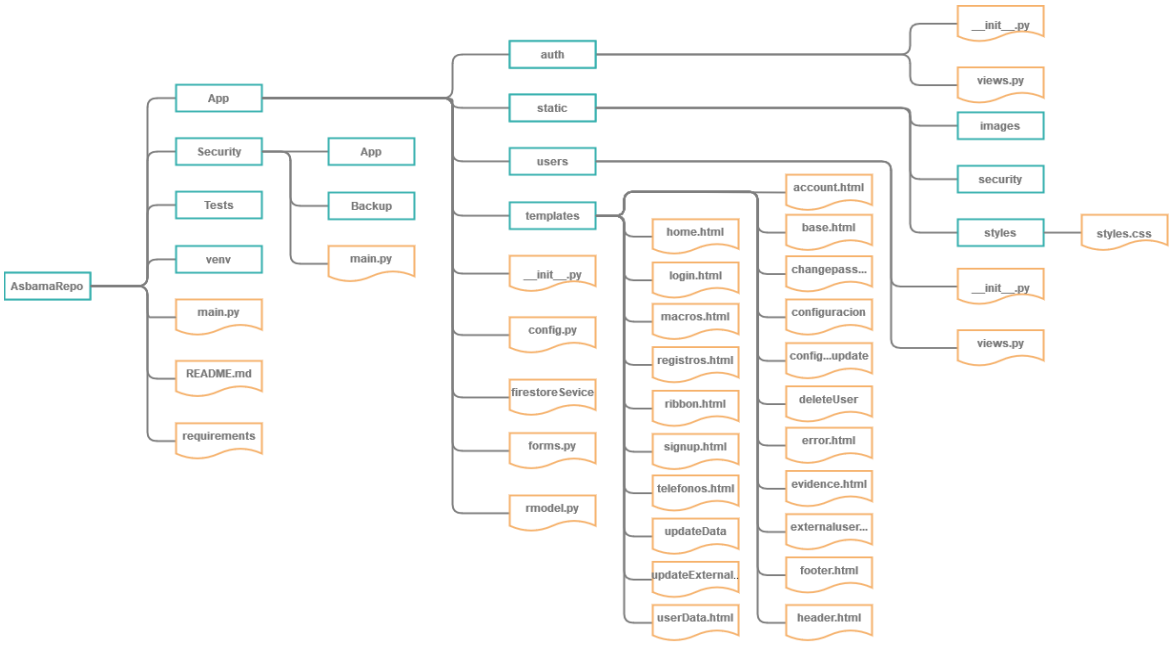
Fuente: Propia

Con ayuda del comando `gcloud auth login` se autentica el proyecto especificado en la previa inicialización con el Google Cloud SDK. Sin embargo, para hacer uso de la base de datos, o bien la Api [16] de Firestore, se requiere de un último paso de autenticación, basado en un comando similar al anterior, `gcloud auth application-default login`, con la finalidad de comunicar el servidor local con la Db en la nube.

1.6.4 INICIALIZACIÓN DE LA APLICACIÓN

En aras de ganar flexibilidad y escalabilidad en el proyecto, se opta por implementar y garantizar una configuración estándar para los distintos ambientes de trabajo (desarrollo, producción) de la aplicación. En la Figura 10, se distingue la disposición de los distintos componentes del proyecto, que se unen y comunican entre ellos dependiendo de los requerimientos, tomando como base la raíz del proyecto.

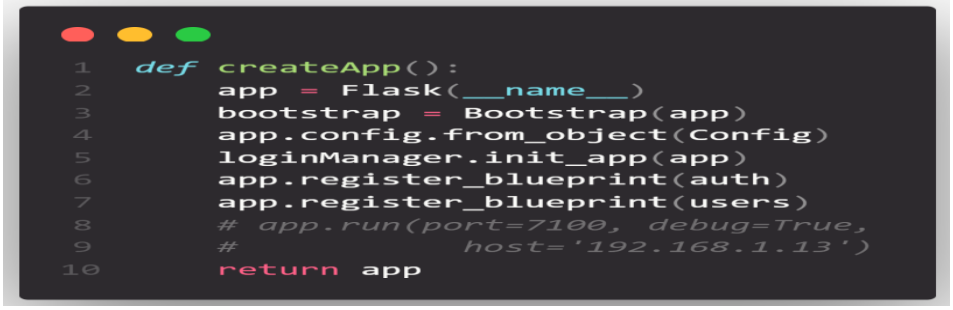
Figura 10. Diagrama de componentes de la aplicación.



Fuente: Propia

La primera necesidad de este tipo de distribución radica en generar una carpeta que haga las veces de App Factory para AsbamaAdmin. El significado se puede resumir en un método que es capaz de generar una aplicación y retornarla con una configuración especifica. El directorio App es realmente un paquete, por lo que cuenta con un archivo de inicialización que lleva el nombre de `__init__.py`, este se ejecuta para disponer al servicio del módulo todas las configuraciones necesarias para su funcionamiento, entre los que se ven los distintos blueprint [17] para autenticación y transacciones de información de usuario, disposición de la data del consumidor una vez se haya logeado en el sistema y la misma sesión del navegador. En la Figura 11, se entrevé el método encargado de crear el objeto App, configurarlo e inicializar el login mánager. El archivo `config.py` en su haber tiene consignada la llave de seguridad y la posibilidad de anexar más opciones de configuración. Además, en la Figura 12, se muestra el método que se encarga de inicializar el servicio para el inicio de sesión y hacer los Queries necesarios para el usuario pueda ser logeado correctamente y utilizar el sistema.

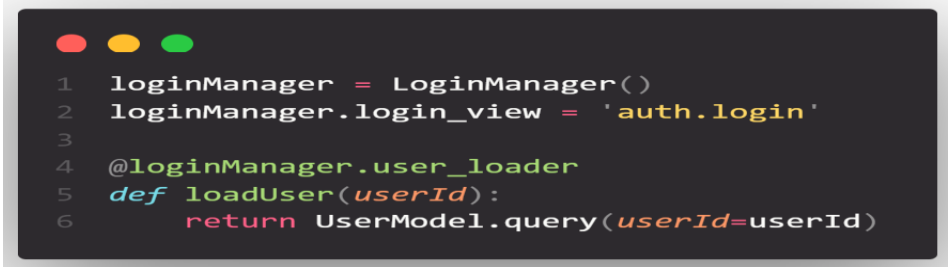
.Figura 11. Método encargado de la inicialización de la aplicación.



```
def createApp():
    app = Flask(__name__)
    bootstrap = Bootstrap(app)
    app.config.from_object(Config)
    loginManager.init_app(app)
    app.register_blueprint(auth)
    app.register_blueprint(users)
    # app.run(port=7100, debug=True,
    #             host='192.168.1.13')
    return app
```

Fuente: Propia

Figura 12. Método encargado de cargar el registro de inicio de sesión del usuario.

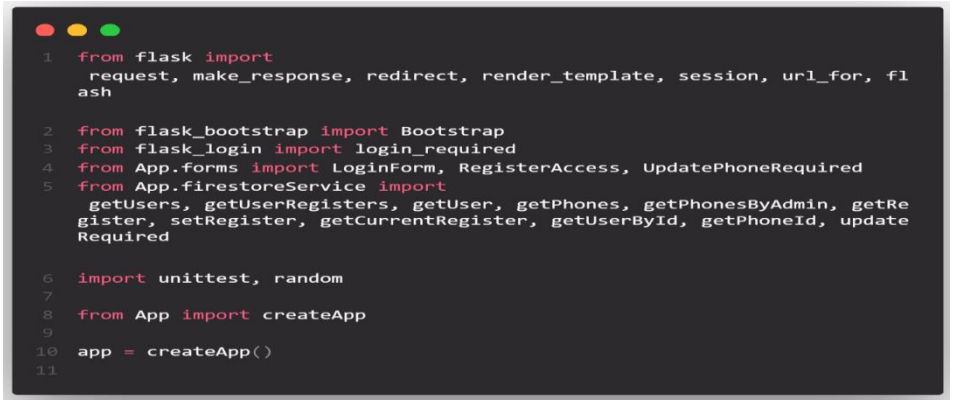


```
loginManager = LoginManager()
loginManager.login_view = 'auth.login'

@loginManager.user_loader
def loadUser(userId):
    return UserModel.query(userId=userId)
```

Fuente: Propia

Luego de importar los paquetes que requiere la interfaz de usuario e instanciar la aplicación en el archivo `main.py` del sistema, que se encarga de renderizar las páginas que no están asociadas a las autenticaciones, ni a las vistas de los usuarios, como se muestra en la Figura 13. Se implementa un método llamado `index` que funge como punto de entrada del sistema. En otras palabras, cuando se llama la raíz del aplicativo, se ejecuta este, puesto que, posee la ruta raíz '/'.

Figura 13. Paquetes necesarios e instancia de la aplicación en main.

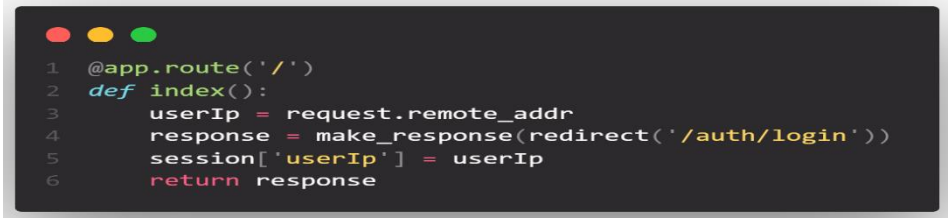


```
from flask import
 request, make_response, redirect, render_template, session, url_for, fl
ash

from flask_bootstrap import Bootstrap
from flask_login import login_required
from App.forms import LoginForm, RegisterAccess, UpdatePhoneRequired
from App.firestoreService import
 getUsers, getUserRegisters, getUser, getPhones, getPhonesByAdmin, getRe
gister, setRegister, getCurrentRegister, getUserById, getPhoneId, update
Required

import unittest, random

from App import createApp

app = createApp()

```

Fuente: Propia

Index dentro de su ejecución se encarga de tomar la IP del usuario para añadirla a la sesión, con el fin de tener una identificación fija. Además, crea y retorna una respuesta que, intuitivamente redirige a la pestaña de login, la cual tiene su lógica propia dentro de una vista de su respectivo blueprint. En la Figura 14, se puede apreciar la implementación de este, asimismo se vislumbra que es uno de los pocos métodos que no requieren el decorador '@login_required' [18] que valide el inicio de sesión del consumidor.

Figura 14. Ruta raíz de la interfaz administrativa.



```
@app.route('/')
def index():
    userIp = request.remote_addr
    response = make_response(redirect('/auth/login'))
    session['userIp'] = userIp
    return response
```

Fuente: Propia

1.6.5 BLUEPRINT

Para entender correctamente el flujo de trabajo de la aplicación es necesario tener claridad conceptual de la 'arquitectura' misma del proyecto. AsbamaAdmin se fracciona virtualmente en tres grandes bloques. En los que se reparten los comportamientos principales del sistema, la autenticación de este, y la capa de gestión de usuarios. Los podemos describir en los componentes main, auth y users respectivamente. Se plantea de esta forma con la intención de desacoplar funcionalmente las capas de la aplicación. En otras palabras, descentralizar el código.

El éxito de la disposición planteada se basa en la implementación de aplicaciones modularizadas con blueprint [17]. Flask usa este concepto para crear sus componentes y soportar los patrones que en este se encuentren. Esto facilita el mantenimiento, refactorización y evolución del código cuando crecen.

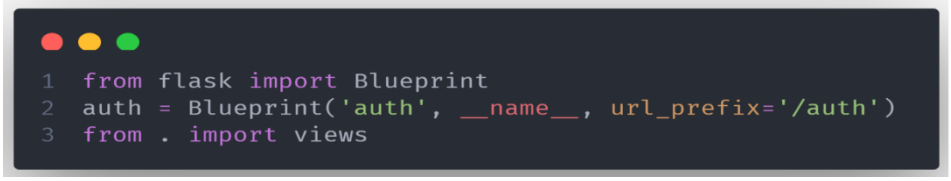
Un blueprint es similar en concepto e implantación a una aplicación Flask. Sin llegar a ser propiamente una. Busca extenderla más no competir contra ella. Al hacer el llamado, se dispone un prefijo o bien, subdominio, que hace referencia directa a la responsabilidad específica del componente, resulta imperativo tener muy claro que un Blueprint no es una aplicación. Aunque resulte común el concepto para algunas personas. Es normal confundirlo y errar en su concepción. Flask registra operaciones que se ejecutan cuando son firmadas por la aplicación. Asocia funciones en varios niveles de las vistas al enviar peticiones y cuando se hace el llamado a un 'endpoint' en particular.

Al realizar la solicitud a la raíz de la aplicación, este redirige a la endpoint de logeo de usuarios.

1.6.6 LOGIN

El blueprint encargado de autorizar al usuario es el conocido como ‘auth’ que lleva un prefijo con este mismo nombre como se aprecia en la Figura 15. Dentro de sus vistas se puede encontrar el endpoint **login**. En principio este le renderiza al usuario un template con un formulario simple para inicio de sesión. Este le provee al consumidor campos donde pueda ingresar su nombre de usuario y su contraseña.

Figura 15. Blueprint de autenticación de usuarios.



Fuente: Propia

Para visualizar el HTML renderizado en primera instancia dirigirse a ANEXO 4.

Figura 16. Contexto enviado como respuesta

Fuente: Propia

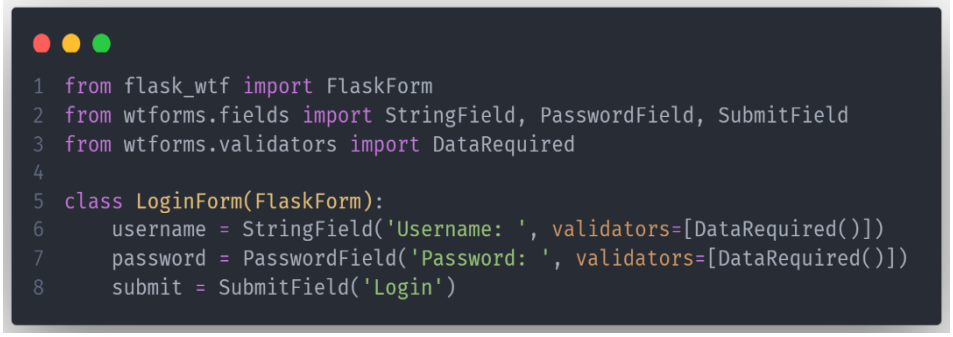
1.6.7 FLASK FORMS

Al observar el contexto enviado en la respuesta del servidor hacia el navegador —Figura 17— y renderizado a través del gestor de plantillas, se nota la aparición en escena de un módulo importante para la composición de este servicio. Normalmente, el código se vuelve difícil de mantener y leer cuando se trabaja con datos que requieren ser enviados por una vista del navegador. Para facilitar su manejo, Flask y su comunidad proveen alternativas al momento de requerir el uso

de formularios dentro de una aplicación. Las cuales están diseñadas para proporcionar una administración simple de este proceso [19].

WTForms, además de ser una de las opciones mejor aceptada por la comunidad. Dirige su implementación hacia la facilidad para el desarrollador. En el sentido de ser similar a la estructura con la que se trabaja en Flask. La ruta, luego de requerir el uso del módulo, apreciamos en un ejemplo preciso, el manejo de la herramienta para su usabilidad en la aplicación, con el montaje del formulario referente al login de los usuarios en la Figura 18.

Figura 17. Clase que extiende desde FlaskForm.

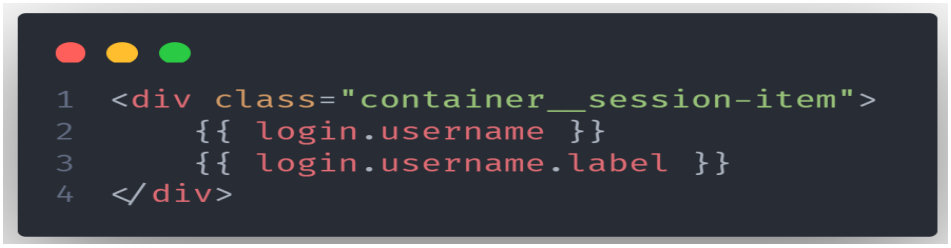


```
1 from flask_wtf import FlaskForm
2 from wtforms.fields import StringField, PasswordField, SubmitField
3 from wtforms.validators import DataRequired
4
5 class LoginForm(FlaskForm):
6     username = StringField('Username: ', validators=[DataRequired()])
7     password = PasswordField('Password: ', validators=[DataRequired()])
8     submit = SubmitField('Login')
```

Fuente: Propia

La clase FormLogin extiende directamente de FlaskForm. Dentro de la primera se pueden encontrar referencia a tres propiedades —username, password y submit— que, hacen alusión a los campos que verá el usuario al renderizarse el template de inicio de sesión. Los dos primeros son, específicamente, los inputs donde el usuario ingresará los datos requeridos, siendo de tipo string y password respectivamente. Cabe resaltar que ambos están siendo validados, lo que significa, a grandes rasgos, que son campos obligatorios. El tercero es un botón de tipo submit que, al disparar su evento, lanza la petición dirigida hacia el mismo endpoint de login. En la Figura 19. Se aprecia que el objeto enviado en el contexto es el encargado directo de insertar y guardar los valores, y realizar acciones con estos mismos.

Figura 18. Atributos del objeto que hace referencia al login.



```
1 <div class="container__session-item">
2     {{ login.username }}
3     {{ login.username.label }}
4 </div>
```

Fuente: Propia

Al llegar la petición nuevamente al endpoint, se valida si el objeto de login a enviado información al servidor. De ser esto correcto, obtiene los datos recibidos de la petición y procede a hacer uso del servicio, para consultar estos datos alojados en la Firestore. El método preciso para esta operación es precisamente 'getUser'. Este recibe como argumento, el nombre de usuario, con el fin de realizar consultas relacionadas con este en la base de datos. Programáticamente, con ayuda del SDK se trata de acceder a la colección de usuarios, para obtener las coincidencias con el username. Luego de validar la existencia del usuario, como se aprecia en la Figura 19, se obtiene la contraseña codificada alojada en la base de datos. (Más adelante se explicará con mayor claridad que significa que la contraseña esté en un hash). Esto es puramente necesario para, con las bondades del método 'check_password_hash', validar si la contraseña brindada por el usuario corresponde a la guardada con anterioridad. En caso de no encontrar usuarios o directamente tropezar con inconsistencias en la contraseña, se dispara un flash. Y se retornara el template de inicio de sesión nuevamente, con los mensajes de error respectivos para cada caso. De otro modo, se tomará como login exitoso, y se procede a almacenar los datos del usuario utilizando una instancia de la clase 'UserData' la cual recibe en su constructor todos los datos provenientes del esquema de la base de datos.

Figura 19. Método del firestore service que se encarga de consultar un usuario por su username.

```
def getUser(username):
    users = db.collection('users').where('username', '==', username).get()
    for user in users:
        return user
```

Fuente: Propia

Para visualizar el método que se ejecuta cuando se hace un llamado al endpoint Login dirigirse a VER ANEXO 5.

Como se puede apreciar en ANEXO 6. La clase 'UserData' es realmente un modelo que, hace referencia directa a los datos previstos por el usuario. Por tanto, como se considera en el ANEXO 7. Se busca setearlos cuando se hacen peticiones a estos y amoldarlos para su correcta utilización en el contexto mismo de la aplicación.

Para visualizar la clase UserData dirigirse a VER ANEXO 6.

Para visualizar la clase UserModel dirigirse a VER ANEXO 7.

Al guardar correctamente los datos en un UserData se procede a enviar estos directamente como argumentos al UserModel. La razón de ser de este actuar, es para el logeo del usuario. Puesto que, este es el tipo de dato solicitado por la función 'login_user' para guardar el registro del usuario. Finalmente, se flashea un mensaje de confirmación y se redirige a la pestaña home del usuario.

1.6.8 JINJA

Un concepto que, también, es importante para el flujo de trabajo con este framework es Jinja. Aunque se había mencionado en apartados anteriores no se había hecho hincapié en entender que es, para que sirve y por qué, en la práctica es útil para el desarrollo de aplicaciones de esta índole. Jinja, particularmente en su versión 2, toma como función el ser el motor de plantillas que respeta todas las funciones de Python [20]. Inspirado en el sistema de plantillas de Django, Además de agregar ejecución en ambientes aislado para aplicaciones donde la robustez y la seguridad son imperativos, provee a los desarrolladores un conjunto de herramientas 'easy to use' poderosas pero amigables con el ambiente de trabajo.

Algo fundamental para la escalabilidad de este tipo de aplicaciones, desarrolladas con estas herramientas o similares es la posibilidad de utilizar herencia en los templates. Pues ayuda tanto al rendimiento en tiempo de ejecución, como al mantenimiento y la legibilidad del código. Por estas razones se decide implementar este módulo, para encargarse de toda la renderización. Siguiendo el principio de una sola responsabilidad. Se le delega a este mismo.

1.6.9 BASE

Al hacer la redirección hacia 'home', se debe ejecutar el código de dicho endpoint. Este se puede apreciar en el Anexo 8. Primeramente, toma la ip del usuario proveniente de la sesión misma, posteriormente se hace uso del Firestore Service para proveer de todos los registros de acceso a la bodega.

En el Anexo 9. Se observa la lógica necesaria para obtener dichos registros. Se realiza una petición a la base de datos con el fin de acceder a la colección donde son almacenados, para así obtenerlos filtrados por la fecha de creación de la incidencia descendientemente. Se realiza un pequeño procesamiento a los datos obtenidos. Que, busca mostrarse más amablemente al usuario. Al retornar la información al endpoint se retorna una lista de objetos de tipo modelos de registro (RegistrosModel), como se aprecia en la Figura 20.

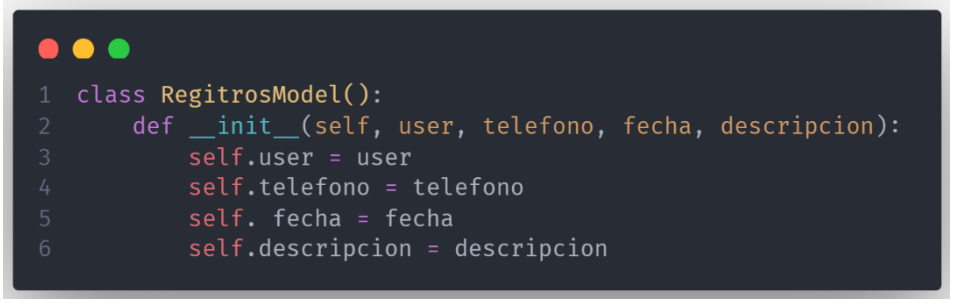
Al finalizar su ejecución, home crea su contexto, y lo envía al gestor de plantillas para que se encargue de generar y renderizar el resultado que el usuario final verá

en él navegador con la información proporcionada en la meta data desestructuradas.

Cuando es llamado la plantilla de home por el motor esta nota (Anexo 10) en primera instancia que debe extender de otro template llamado base (Anexo 11). Este último, es llamado por todos los template que requiera de una implementación previa. Esta nos describe las buenas prácticas descritas por HTML5, es decir, dispone al motor información como el DOCTYPE, las cabeceras con todos los Lang la metadata, el título, los links de css fuentes y favicons. Además, en el body incluye la invocación del header, los flashes, el ribbon el contenido que es donde se albergan el resto de templates y el footer. Además de los scripts mínimos para el desarrollo del prototipo.

En este punto, con todos los datos previos cargados, solo resta llenar o extender los bloques mencionados. Es decir, con ayuda de Jinja extendemos, por ejemplo, el titulo para mostrar algo similar a 'Asbama | Bienvenido' dinámicamente. E implementados el contenido del bloque con los datos provistos por el contexto.

Figura 20. Modelo de registros



Fuente: Propia

1.6.10 PÁGINAS DE ERROR

Es importante soportar los errores que puedan ocurrir mientras se realiza una petición HTTP, lo más común es empezar a manejar los que coexisten en la familia 400, como el 404 not Found, o los de la familia 500, como el Internal server error. Esto da claridad en la dirección que debe tomar el programa para resolver estos. Estos nos hablan de errores al hacer las peticiones del lado del cliente y a los fallos del lado del servidor. Flask facilita demasiado realizar el código que se ejecuta al disparar estos eventos. En el Anexo 12. Vemos ejemplos de cómo se implementan. En palabras muy generales, estos se acercan en gran medida a la arquitectura de un endpoint. Estos, en vez de recibir una ruta como decorador, reciben '@app.errorhandler(numero)', donde número hace referencia directa al código de error. Luego, en el método mismo solo se envía un contexto y se renderiza un template de error (Anexo 13).

1.6.11 FLASHES

Es una buena práctica en la creación de aplicaciones o interfaces graficas poder retroalimentar al usuario de lo que está pasando en los tiempos muestro y en los resultados de ejecuciones dentro de estas mismas. Puesto que, de no ser así resulte en una mala experiencia y finalmente abandono de esta. Flask, provee una manera extremadamente sencilla para dar solución a este pequeño detalle. Un pequeño sistema de flasheo. Este permite guardar un mensaje al final de cualquier solicitud, y leerla desde la siguiente. Esta solución tiene una limitante, el tamaño no debe ser demasiado grande.

Al momento de tomar la decisión de retroalimentar al usuario se utiliza la función vista en la función que se aprecia en la Figura 21. Y se renderiza en el template base. Este dato se observa fácilmente en la Figura 22.

Para obtenerlos se hace uso de la función 'get_flashed_messages', este retorna una tupla con la categoría, es decir si es de error o de success y, mensaje mismo. Posteriormente, se utiliza la macro provista en el Anexo 14. Para renderizar el flash.

Figura 21. Función para generar una retroalimentación al usuario con ayuda de Flash.

```
flash('Usario: {}, Ah iniciado sesion con exito'.format(user.username.title()))
flash("Contraseña incorrecta, vuelva a intentarlo.", 'error')
flash("El usuario {}, No has sido encontrado.".format(username), 'error')
```

Fuente: Propia

Figura 22. Renderización de los mensajes.

```
{% for category, message in get_flashed_messages(with_categories=true) %}
    {{ macros.render_flashes(category, message) }}
{% endfor %}
```

Fuente: Propia

1.6.12 MACROS

Para tener consistencia, es imperativo conservar ciertos principios, como el mencionado anteriormente (Responsabilidad única) que ayuda a la legibilidad y la

coherencia del código, para amoldar el desarrollo a las mejores prácticas posibles. Uno de estos, que sin duda es de gran importancia, es el DRY, que hace referencia a las siglas en ingles a 'dont repeat yourself' o en español, tan sencillo como, **no te repitas**. Este se puede implementar en las plantillas, abstrayendo el fragmento de código necesario, en lo que, en otros leguajes hace símil a los componentes compartidos. Conocido en Flask como macros. Dentro de estas encontramos código que se repite varias veces a lo largo del desarrollo del proyecto. En otras palabras, proporcionan un modo de modularizar fragmento de código a manera de una simple función que es llamada desde otra parte del proyecto. En el Anexo 15 se evidencia un ejemplo de una macro creada para esta implementación que se encarga de renderizar la data de los usuarios albergados en un carrusel de perfiles.

1.6.13 ROLES DE SEGURIDAD

Teniendo presente las políticas del negocio y que no todos los que tengan acceso y perfiles dentro de la plataforma de seguridad van a poder utilizar toda la carga dispuesta en la aplicación. Se decide por implementar un sistema de autorización sencillo basado en roles, específicamente en dos roles principales. Administrador y Operario. Estos segmentan la usabilidad de la aplicación modificando los niveles de acceso que pueda tener o no una persona en particular. El primer cambio se evidencia en la distinción que se tiene, como convención, al crear un nuevo perfil de usuario. Generalmente en las personas que fungen el rol de administrador. Se aprecia el apellido seguido del número de identificación 314. En el caso de los operarios, la convención es determinar las iniciales del nombre, separadas por punto y seguidas del apellido. Se especifica doe314 o j.doe, a manera de ejemplo, para el rol de administrador y operario respectivamente.

1.6.14 AUTENTICACIÓN DE USUARIOS

Si bien es sabido que la restricción para la mayoría de los endpoint de la aplicación es el inicio de sesión por parte del usuario. También se debe controlar en el front que información está disponible para que persona dependiendo, no solo del login, si no del nivel de autorización que este posea en la aplicación. El cambio más evidente es la pérdida de funciones que puede ejecutar el usuario. Puesto que, de no estar autorizado en las pestañas que lo requieran de la aplicación, no se mostrará la información pedida, y mostrará la opción de regresar al menú de inicio de la aplicación. De estarlo, se habilitan funciones principalmente del ribbon de sesión.

El ribbon o listón, es un acompañante del header original, donde se le brinda al usuario la oportunidad de ejecutar ciertas acciones. Esta idea es tomada directamente de las funcionalidades que poseen los CRM modernos para mostrar las opciones disponibles si modificar la barra de navegación. Se menciona este

apartado pues es donde se muestran las opciones de administración y visualización de la aplicación (puede encontrar el código en el Anexo 16). Además, sufre grandes cambios que depende de la autorización de los usuarios. Los administradores tienen acceso a todos los apartados que dispone la aplicación, desde la modificación, creación y visualización de usuarios, hasta registro telefónico y administración completa del sistema.

Figura 23. Visualización del ribbon en la aplicación para usuarios autorizados.

Fuente: Propia

Figura 24. Visualización del ribbon en la aplicación para usuarios no autorizados.

Fuente: Propia

1.6.15 USUARIOS

El siguiente núcleo de esta aplicación es el de usuarios, pues gran parte de la implementación se basa en lo que estos pueden realizar o ver. Incluido que acciones puede tomar el sistema basado en sus datos y en su configuración.

Es preciso arrancar diciendo que, solo un usuario con el rol de administrador es capaz de crear un usuario o modificar su rol. Él debe llenar un formulario donde se encuentran los datos necesarios para el mapeo de este en el sistema. La implementación de esta operación es muy parecida a la vista en el login de usuarios. La diferencia radica en dos puntos importantes. En este caso, el sistema no se encarga de consultar en los usuarios ya existentes, por el contrario, este crea el usuario y lo firma en la base de datos. Además, no solo afecta la colección de 'users' sino que también crea un nuevo documento en la colección de teléfonos.

1.6.16 CODIFICACIÓN DE CONTRASEÑAS

Para garantizar un punto de seguridad en el sistema, nunca se deben guardar datos sensibles en crudo en las tablas o colecciones. Es por esto qué se pretende inyectar a la base de datos información que se encuentre cifrada. En este caso en particular se implementa, proveniente del módulo **werkzeug.security** los métodos encargados de generar un hash para la contraseña y otro que se encargue de realizar una aserción [21] con el dato en crudo proveniente del usuario y la contraseña encriptada de la base de datos.

Figura 25. Contraseña hasheada en la base de datos.

Fuente: Propia

Figura 26. Se aprecia la codificación y la aserción de contraseñas

```
updatePassword(user, generate_password_hash(password))
if(check_password_hash(getUser(username).to_dict()['password'], password)):
    flash('Contraseña Actualizada con Exito!!')
    return redirect(url_for('home'))
```

Fuente: Propia

En el siguiente enlace se encuentra el repositorio donde se evidencian todos los fragmentos de código necesarios para la implementación de esta iniciativa.

https://github.com/joseoliva762/AsbamaAdminRepo

1.7 ASBAMA SECURITY

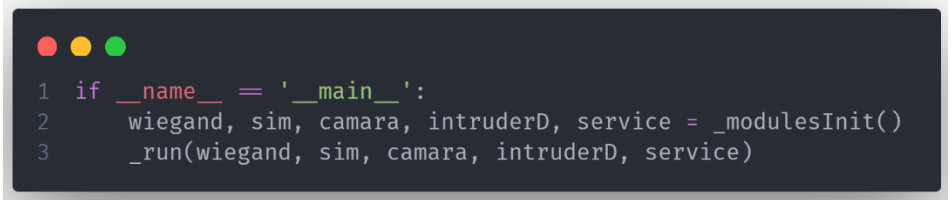
Si bien AsbamaAdmin es el encargado del manejo de datos, administración del sistema y gestión de usuarios. El núcleo de todo el proyecto sigue siendo el sistema de seguridad que maneja toda la interacción con los sensores y toma las decisiones basadas en las acciones realizadas en cada ambiente. O bien, petición que se haga basado en las operaciones que se realicen. Esta iniciativa porta un nombre asociado a su participación, Asbama Security.

Al igual que segmento anterior, Security basa su construcción en la orientación a objetos. Sin ser directamente uno. Básicamente, se estructura de esta manera para evitar código repetitivo y poco escalable. Apuntando a tener bajo acoplamiento entre las partes de este.

La aplicación cuenta con 6 paquetes y un fichero principal. Entre estos se puede encontrar los controladores de la cámara, de los hilos, de interrupciones, de la comunicación vía comandos at, protocolo Wiegand y el servicio. El orquestador principal de todas las transacciones es el archivo main. Es aquí donde se alojan las funciones que tienen como responsabilidad tomar decisiones basados en las acciones realizadas por los usuarios del sistema.

En la Figura 27, se observa el arranque del sistema. El cual tienen dos funciones principales muy sencillas. Por un lado, se obtiene la inicialización de los módulos y por otro la ejecución del runtime del sistema de seguridad.

Figura 27. Método para arrancar la aplicación desde consola.

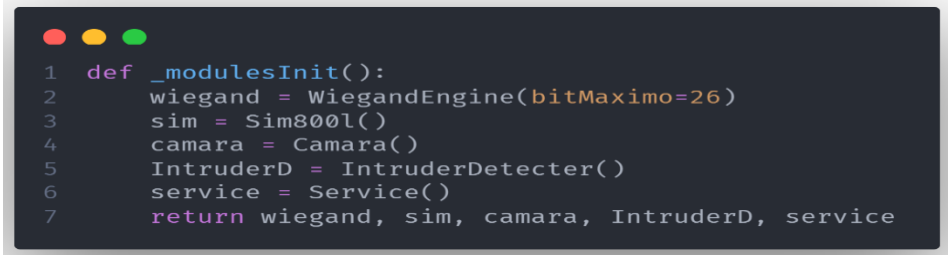


```
if __name__ == '__main__':
    wiegand, sim, camara, intruderD, service = _modulesInit()
    _run(wiegand, sim, camara, intruderD, service)
```

Fuente Propia.

En la definición de la función privada que lleva el nombre ‘modulesInit’ se busca generar una instancia de las clases necesarias para la ejecución del runtime de dicha aplicación, en la Figura 28, se aprecia la construcción de cada uno de los objetos que pide como argumento es método de ejecución.

Figura 28. Función encargada de instanciar las clases que se necesitan para correr la rutina de alarma.



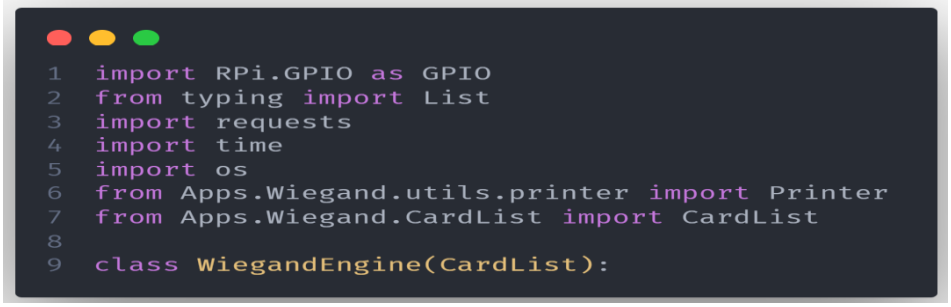
```
def _modulesInit():
    wiegand = WiegandEngine(bitMaximo=26)
    sim = Sim800l()
    camara = Camara()
    IntruderD = IntruderDetecter()
    service = Service()
    return wiegand, sim, camara, IntruderD, service
```

Fuente: Propia

El primero que se observa es la creación del objeto Wiegand. Quien en su interior consta de los métodos y atributos necesarios para cumplir con la tarea de gestionar el protocolo que lleva el mismo nombre del módulo.

La clase que contiene el motor del controlador del sensor que utiliza el protocolo de comunicación Wiegand –como se aprecia en la Figura 29– extiende de la clase, llamada ‘CardList’, que contiene las propiedades de las tarjetas que se leerán. En la Figura 30, se ve sencillamente que la finalidad de esta es proveer un atributo que guarde el código en crudo que arroja la lectura del tag, la longitud máxima del protocolo, la máscara y el identificador. Asimismo, implementa dos métodos que, cumplen la función de añadir un bit leído y, retornar el valor de la codificación. Esto hace referencia a que, el atributo ‘code’ realmente es una propiedad que cuenta con un setter llamado ‘AddBitToCardList’ y un getter que lleva el nombre 'getCardList’.

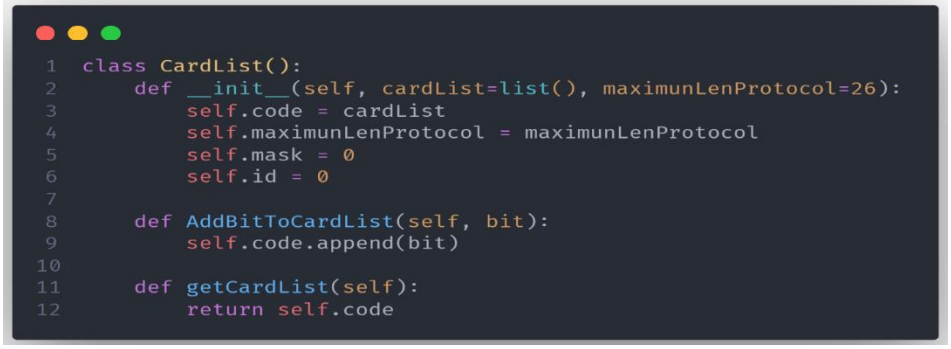
Figura 29. Importaciones y herencia en la construcción del motor del protocolo Wiegand.



```python
import RPi.GPIO as GPIO
from typing import List
import requests
import time
import os
from Apps.Wiegand.utils.printer import Printer
from Apps.Wiegand.CardList import CardList

class WiegandEngine(CardList):
```

Fuente: Propia

Figura 30. Clase de la que extiende el motor del protocolo Wiegand.



```python
class CardList():
    def __init__(self, cardList=list(), maximunLenProtocol=26):
        self.code = cardList
        self.maximunLenProtocol = maximunLenProtocol
        self.mask = 0
        self.id = 0

    def AddBitToCardList(self, bit):
        self.code.append(bit)

    def getCardList(self):
        return self.code
```

Fuente: Propia

1.7.1 WIEGAND ENGINE

El motor que le da vida al protocolo vive en la clase que lleva el mismo nombre. El constructor de esta recibe como parámetros los números correspondientes a los cuatro pines comúnmente necesarios para el funcionamiento del sensor. Estos hacen referencia a los pines que alimentan a la cadena de lectura de código. Bit uno y cero. Además, encargados de activar o desactivar el led y el beeper. Estos por defecto están seteados con los valores 23, 24, 17, 27 respectivamente. El constructor recibe un valor más llamado bit máximo. Este es encargado de determinar cuál es la carga útil del protocolo. Es decir, este será de 26 o 32 bits. Por defecto, este está ajustado al Wiegand 26.

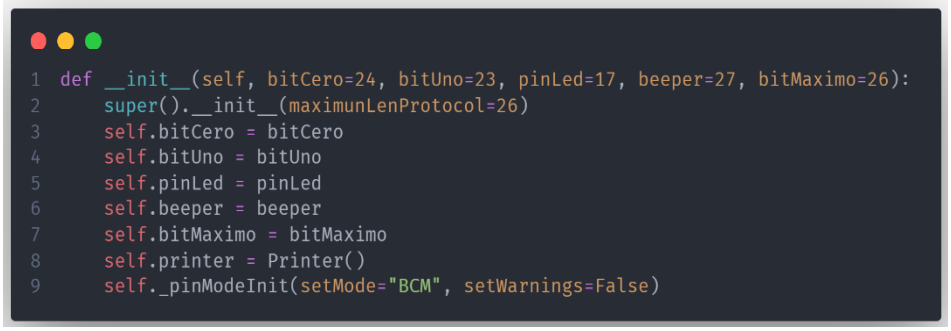
Los pasos por seguir se basan en fijar los números de los pines en atributos de la clase misma, así como, llamar a las super clases para extender las funcionalidades del motor. Finalmente, como se aprecia en la Figura 31, se hace un llamado a un

método privado que se encarga de inicializar todos los pines que anteriormente se mencionaron.

Para la inicializar los puertos primero se define como estos son leídos. Basado en el paquete 'RPi.GPIO', existen dos formas de numerar los pines de entrada y salida de una raspberry. El sistema **BOARD** que hacen referencia a de pin en el encabezado de la placa. Se utiliza principalmente para que inequívocamente funcione el hardware. En otras palabras, no se necesita re-cablear su conector, ni cambiar el código. Por otra parte, está el sistema de numeración **BCM** que se refiera a los números del canal en el Broadcom SOC, por esto siempre debe revisarse con diagrama el número del canal y el pin en la RPi. Por facilidad y conveniencia en las conexiones del prototipo, se opta por implementar esta última.

Es importante anotar que se setean las advertencias en falso puesto que varios scripts hacen uso de este módulo para ejecutarse. Como se precisa en la Figura 32. Es necesario llamar a un método privado que recibe dos valores como argumentos. El canal y el estado hacen referencia al número del pin y como este será configurado en el sistema. Salida o entrada. Para lograr esto se hace uso del método **setup** el cual recibe los mismos valores, pero, como valores constantes enumerados dentro del módulo tercero. Como en el caso de la numeración de la placa, se emplea un condicional ternario para especificar el modo en que serán configurados los puertos.

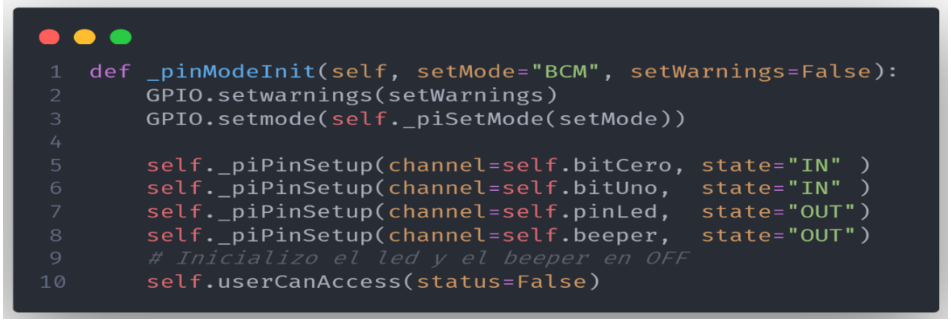
Figura 31. Constructor de la clase Wiegand Engine.



```
def __init__(self, bitCero=24, bitUno=23, pinLed=17, beeper=27, bitMaximo=26):
    super().__init__(maximunLenProtocol=26)
    self.bitCero = bitCero
    self.bitUno = bitUno
    self.pinLed = pinLed
    self.beeper = beeper
    self.bitMaximo = bitMaximo
    self.printer = Printer()
    self._pinModeInit(setMode="BCM", setWarnings=False)
```

Fuente: Propia

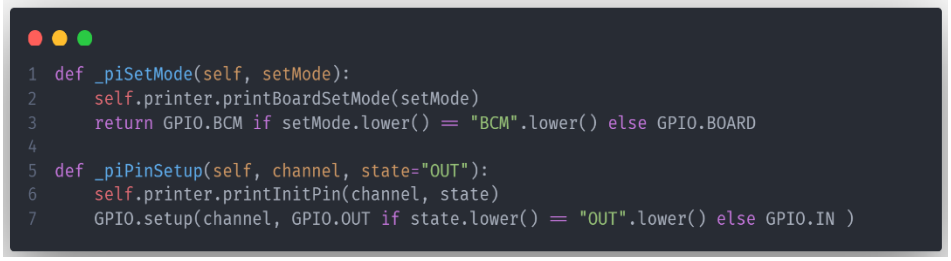
Figura 32. Método encargado de la inicialización de los puertos para el Wiegand 26.



```
def _pinModeInit(self, setMode="BCM", setWarnings=False):
    GPIO.setwarnings(setWarnings)
    GPIO.setmode(self._piSetMode(setMode))

    self._piPinSetup(channel=self.bitCero, state="IN" )
    self._piPinSetup(channel=self.bitUno,  state="IN" )
    self._piPinSetup(channel=self.pinLed,  state="OUT")
    self._piPinSetup(channel=self.beeper,  state="OUT")
    # Inicializo el led y el beeper en OFF
    self.userCanAccess(status=False)
```

Fuente: Propia

Figura 33. Métodos encargados de establecer la numeración de la placa y establecer los pines.



```
def _piSetMode(self, setMode):
    self.printer.printBoardSetMode(setMode)
    return GPIO.BCM if setMode.lower() == "BCM".lower() else GPIO.BOARD

def _piPinSetup(self, channel, state="OUT"):
    self.printer.printInitPin(channel, state)
    GPIO.setup(channel, GPIO.OUT if state.lower() == "OUT".lower() else GPIO.IN )
```

Fuente: Propia

1.7.2 SIM800L

El siguiente módulo en requerir de una instancia es el que controla todo el comportamiento de repuesta del sistema mediante comandos AT en el sistema global de comunicaciones para móviles (toda la información fue provista en componentes previos del documento). Para establecer la comunicación con el módulo se requiere establecer un puerto de conexión que por defecto está seteado como el ‘ttys0’. Para, como se observa en la Figura 34, se debe inicializar con ayuda del paquete serial con el fin de obtener un número específico. Para hacer uso del módulo, se requiere pasarle como argumento la ruta completa del puerto, los baudios y el timeout.

Es necesario ceñirse a un protocolo cuando se quiere enviar un mensaje a un destinatario específico. Cuando se requiere se debe llamar al método ‘sms’ como se muestra en la Figura 35, este internamente ejecuta la serie de comandos necesarios para enviar un mensaje.

Según la documentación y lo mencionado en componentes anteriores, luego de haber enviado los comandos y haber inscrito el número del destinatario, con su respectivo encode. Se es posible enviar el mensaje. Es importante recordar que toda la comunicación ser hace a través del número registrado al momento de inicializar el puerto serie. Con ayuda del método de escritura (write) que recibe el código AT y lo codifica según el encoder enviado como parámetro a la función que lo invoca.

Figura 34. Inicialización de la clase Sim800L.

```
class Sim800l:
    def __init__(self, puerto = "ttyS0"):
        print("Inicializando SIM800L ... ")
        self.puerto = puerto
        self.phone = self._serialInit()
        print("Inicializado SIM800L [OK]")

    def _serialInit(self, baudios = 9600, timeout=1):
        return serial.Serial('/dev/{}'.format(self.puerto), baudios, timeout=timeo
```

Fuente: Propia

Figura 35. Métodos encargados de establecer la comunicación con el destinatario, mediante comandos AT.

```
def sms(self, mensaje, numeroDeTelefono, encode='utf-8'):
    self._sendSMS("AT\r", encode, 1)
    self._sendSMS("AT+CMGF=1\r\n", encode, 1)
    self._sendSMS('AT+CMGS="+57{}"\r\n'.format(numeroDeTelefono), encode, 2)
    self._sendSMS(mensaje+chr(26), encode, 4)
    print("Mensaje enviado")

def _sendSMS(self, codigoAT, encode, seg):
    self.phone.write(codigoAT.encode(encode))
    time.sleep(seg)
```

Fuente: Propia

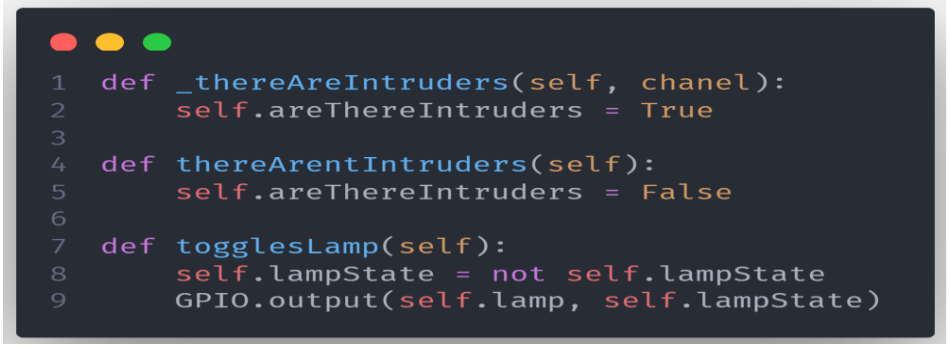
1.7.3 INTRUDER DETECTER

Teniendo en cuenta que tres de los módulos solo envían impulsos al sistema cuando captan un fenómeno en específico. Además, existe una salida digital que controla el encendido de la iluminación. Se decide encapsular su funcionamiento en uno solo. Como su nombre lo indica, la funcionalidad de este es basada enteramente en detectar cuando hay un intruso en la propiedad. Como se muestra en la inicialización (Anexo 17), su constructor recibe 6 argumentos de entrada, que hacen referencia a

los pines 12, 13, 6, y 16 que son la entrada del sensor de puerta, el sensor de movimiento uno y dos respectivamente, y la salida digital para la iluminación del recinto. Los parámetros restantes indican el modo que se leerán los pines en la placa y la bandera para deshabilitar las advertencias del sistema (información disponible en módulos anteriores).

Al momento de asignar los valores a los atributos de la clase, se añade una propiedad de control conocidas con el nombre 'areThereIntruders'. Este se configura en primera instancia como falsa, pues, no es más que una bandera para determinar se encontró o no presencia de algún ente en el local. (La configuración de los pines es exactamente igual que en módulos anteriores). Esta, a pesar de ser una propiedad cuenta con una particularidad. Cuenta con dos setters. Aunque suene redundante al principio, ayuda a la legibilidad en el runtime de la aplicación. La peculiaridad surge de que si bien a pesar de ser lo mismo, pero con valores opuestos como se muestra en la Figura 36, uno de ellos es privado y el otro no lo es. Ya que, al ejecutarse se puede decidir externamente cuando ya no hay un intruso, es decir que se puede invocar implícitamente cuando se toma esta decisión. Sin embargo, es el sistema quien decide cuando existe una intromisión. Por lo tanto, es el quien hace el llamado al método, cuando lo crea estrictamente necesario.

Figura 36. Setters de las banderas que indican cuando existe un intruso. Y el disparador de la iluminación de la zona acordada.



```
def _thereAreIntruders(self, chanel):
    self.areThereIntruders = True

def thereArentIntruders(self):
    self.areThereIntruders = False

def togglesLamp(self):
    self.lampState = not self.lampState
    GPIO.output(self.lamp, self.lampState)
```

Fuente: Propia

1.7.4 SERVICIO

Si bien esta sección no es una aplicación para la administración. Si debe, imperativamente, hacer consultas y crear registros en la misma base de datos que se utiliza para la visualizar y configurar el contenido en AsbamaAdmin. Es por esta razón por lo que, se requiere la instancia de un servicio que facilite la comunicación con la DB. En la Figura 37, se puede apreciar la inicialización y la conexión realizada

con la base de datos de Google. Todo el poder de comunicación queda inmerso en un simple atributo, llamado **db**.

Nota: podrá encontrar todos los métodos utilizados por Secutiry para consultar a la base de datos en el anexo llamado comunicación con la DB desde Asbama Security.

Figura 37. Inicialización de la clase encargada de la conexión con la base de datos.

```
class Service:
    def __init__(self):
        if (not len(firebase_admin._apps)):
            credential = credentials.ApplicationDefault()
            firebase_admin.initialize_app(credential, {
                'projectId': 'asbama314',
            })
        self.db = firestore.client()

```

Fuente: Propia

Al iniciar la rutina de sistema de alerta, se entra en un bucle infinito, al cual se le denomina, para esta implementación en particular como ‘runtime loop’. Dentro de este se busca en primera línea proteger la ejecución contra excepciones provenientes de la API de Google o de la práctica misma. En cualquiera de los casos, se limpian todos los puertos de y se realiza una nueva instancia de los módulos previamente mencionados. El contenido de este método puede encontrarlo en la Figura 38.

Posteriormente se establece un tiempo de reconocimientos, que no es más que un multiplicador del tiempo de espera del sistema. Conjuntamente, se habilitan las interrupciones de los sensores. Se necesita recordar que todas las funcionalidades de los sensores que, habilitan y deshabilitan la bandera que determina la existencia de un intruso viven dentro del contexto del intruder detecter. Cuando se habilita la lectura de estos, por debajo solo se activa la rutina de atención a interrupción, que espera directamente que alguno de estos envíe un pulso para levantar la bandera ‘areThereIntruders’. En la Figura 39. Se puede observar el método que se encarga de habilitar la interrupción en estos puertos en específico.

Figura 38. Función encargada de reiniciar el sistema

```python
def sistemRestart():
    GPIO.cleanup()
    time.sleep(0.3)
    wiegand, sim, camara, IntruderD, service = _modulesInit()
    return  wiegand, sim, camara, IntruderD, service
```

Fuente: Propia

Figura 39. Método para habilitar la interrupción de los sensores de movimiento y el sensor de puerta.

```python
def enable_interrupt(self):
    time.sleep(0.3)
    GPIO.add_event_detect(self.door, GPIO.FALLING)
    GPIO.add_event_detect(self.moveOne, GPIO.FALLING)
    GPIO.add_event_detect(self.moveTwo, GPIO.FALLING)
    GPIO.add_event_callback(self.door, self._thereAreIntruders)
    GPIO.add_event_callback(self.moveOne, self._thereAreIntruders)
    GPIO.add_event_callback(self.moveTwo, self._thereAreIntruders)

```

Fuente: Propia

El paso inmediatamente siguiente en la rutina de alarma es la creación e inmersión en un nuevo ciclo, que para efectos prácticos se denomina ‘execution loop’. Al entrar en este, el sistema entra en una consulta constante a la bandera que se encarga de determinar la existencia de algún intruso. De ser el caso, lo primero que el sistema determina correcto y hace, es consultar la base de datos. En la Figura 40. Se muestra el método utilizado para hacer la petición.

Figura 40. Método para obtener la configuración del sistema desde la base de datos.

```python
def getConfiguration(self):
    return self.db.collection('configuracion').document('configuraciongeneral').get()
```

Fuente: Propia

La tarea del método asociado al servicio es muy sencilla. Este busca acceder a la colección donde está almacenado un documento con la configuración general del sistema. Y obtener para devolverá a quien lo requiera. Con la información obtenida, el bucle de ejecución requiere tres valores que son indispensables para continuar

con la rutina de atención a alertas. Estado del sistema, resolución y tiempo de espera son los datos que se necesita para avanzar. Al obtener estos datos correctamente. Se procede a validar si el sistema, según la configuración de usuario, está encendido. En caso de no estarlo, se regresa directamente al loop de ejecución. No sin antes configurar la bandera del intruder detecter en falso. En caso contrario, el sistema determina la presencia de un intruso en el área, por lo que empieza a tomar acciones al respecto. Lo primero que hace es detener las interrupciones con el método que se muestra en la Figura 41. Y encender la iluminación del recinto.

Figura 41. Función encargada de deshabilitar la atención a las interrupciones proveniente de los sensores de movimiento y el sensor de puerta.

```
1  def disable_interrupt(self):
2      time.sleep(0.3)
3      GPIO.remove_event_detect(self.door)
4      GPIO.remove_event_detect(self.moveOne)
5      GPIO.remove_event_detect(self.moveTwo)
```

Fuente: Propia.

Con todo el entorno listo para determinar quién ha disparado el sistema, la rutina entra en nuevo bucle conocido como ‘intruder loop’ en este se encuentra el núcleo de todo el sistema de seguridad.

Al plantear la idea de la construcción y la implementación del sistema de seguridad. Se estableció que, desde un inicio se va a capturar una imagen, y se grabará un video de por lo menos 3 veces el tiempo de espera. Sin embargo, lograr esto es un reto. Puesto que, esta ejecución de la cámara debe ser no bloqueante. Por lo que debe ejecutarse en un hilo distinto a donde corre el intruder loop. Para lograr eso se crea, como se aprecia en la Figura 42, una clase que heredad directamente de **multiprocessing**, sin embargo, este solo sería una especie de clase abstracta que solo será heredada por otras clases. Como la llamada a cámara. Figura 43.

Figura 42. Clase para crear un multiproceso.

```python
import multiprocessing

# Clase Abstracta.
class MyThread(multiprocessing.Process):
    def __init__(self, threadName):
        super().__init__(name=threadName, target=self.runThread)

    def runThread(self):
        # Debe ser sobre escrita en el extension.
        pass
```

Fuente: Propia

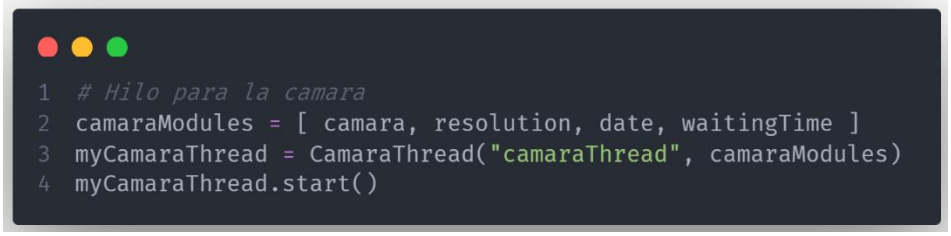
Figura 43. Clase que estructura la ejecución del hilo encargada de manejar la cámara.

```python
from Apps.Camara.camara import Camara
from datetime import datetime
from Apps.Hilos.myThread import MyThread

class CamaraThread(MyThread):
    def __init__(self, threadName, modules):
        super().__init__(threadName=threadName)
        self.camara = modules[0]
        self.resolution = modules[1]
        self.date = modules[2]
        self.recordingTime = int(modules[3])

    def runThread(self):
        # Start
        self.camara.getPacket(self.resolution, self.date, self.recordingTime)
```

Fuente: Propia

El proceso que ejecuta la cámara recibe como parámetro el nombre del hilo y un contexto de ejecución, como se ve en la Figura 44. La clase misma, como en todos los casos hereda las configuraciones de la clase abstracta creada con anterioridad. Esta solo recibe el nombre del hilo como parámetro, el resto de la ejecución del constructor se le encarga de generar los atributos con los datos obtenidos en el contexto enviados desde intruder loop. Todas las clases que sean hijas de **myThread**, deben sobrescribir el método **runTread**, en su implementación se debe alojar el código que se desee correr cuando se haga el llamado.

Figura 44. Llamado a ejecutarse el proceso que maneja la cámara.



```
# Hilo para la camara
camaraModules = [ camara, resolution, date, waitingTime ]
myCamaraThread = CamaraThread("camaraThread", camaraModules)
myCamaraThread.start()
```

Fuente: Propia

Se aprecia que la ejecución de este se hace mediante el método start, quien hace el llamado al método mencionado en el párrafo anterior. Cuando esto pasa, se hace uso de la instancia del controlador de la cámara para obtener el paquete deseado (foto y video). Getpacket, como se muestra en la Figura 45, obtiene la resolución de la cámara que esta por defecto en 720x560. Sin embargo, esta llega en el contexto de ejecución como un valor proveniente del módulo administrativo de AsbamaAdmin. De igual modo, recibe un tiempo de grabación que es afectado por el factor multiplicativo mencionado al inicio de esta sección. La decisión tomada con respecto a cuánto tiempo se destina a la grabación fue por lo menos tres veces el tiempo máximo de lectura del sensor RFID. Teniendo estos datos bien definidos, se procede a tomar la foto y grabar el video. Para capturar una imagen se hace uso de un módulo llamado 'picamera', este recibe las características que mínimas necesarias para la configuración de la cámara. Es decir, la resolución, el preview y la ruta donde será almacenada la captura. El método que se ejecuta se observa en la Figura 46. Además, en la Figura 47 se aprecia cómo se obtiene la ruta de guardado de la información.

Figura 45. Método encargado de ejecutar el controlador de la cámara.

```
def getPacket(self, resolucion = "720×560", date = datetime.now(), recordingTime = 30):
    widht, heith = self.getResolucion(resolucion)
    recordingTime *= self.waitingTime
    self._camFoto(widht, heith , self.waitingTime, date)
    self._camVideo(widht, heith, recordingTime, date)

def getResolucion(self, resolucion):
    res = resolucion.split("x")
    return int(res[0]), int(res[1])
```

Fuente: Propia

Figura 46. Captura de la foto al recinto.

```python
def _camFoto(self, widht, heith , seg, date):
    print("Tomando foto ... ")
    with picamera.PiCamera() as picam:
        picam.resolution = (widht, heith)
        picam.start_preview()
        time.sleep(seg)
        ruta = self.getRuta(date)
        picam.capture(ruta)
        picam.stop_preview()
        picam.close()
        print("Foto guardada en la ruta: {} [OK]".format(ruta))
```

Fuente: Propia

Figura 47. Método para obtener la ruta donde se guardará la información.

```python
def getRuta(self, date = datetime.now(), extension = "jpg" ):
    enrutado = "{}\'{}+00:00\'/".format(self.ruta, str(date))
    try:
        os.mkdir(enrutado)
    except FileExistsError:
        print("Directorio encontrado")
    finally:
        return "{}\'{}+00:00\'.{}".format(enrutado, str(date), extension)
```

Fuente: Propia

Para grabar el video, se realiza un proceso similar al mencionado en la captura de la imagen. La diferencia radica en cambiar la extensión de la ruta a ‘h264’ y utilizar el método **start_recording**, además este debe esperar el tiempo pactado al inicio de la ejecución y luego detener el preview y la grabación. Puede encontrar la implementación de este método en la Figura 48.

Figura 48. Método encargado de iniciar la grabación.

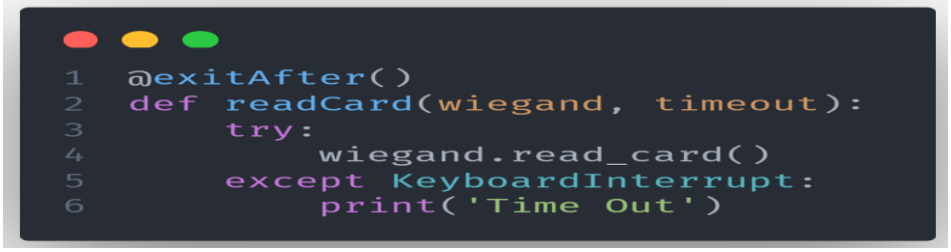
```python
def _camVideo(self, widht, heith , recordingTime, date):
    print("Grabando ... ")
    with picamera.PiCamera() as picam:
        picam.resolution = (widht, heith)
        picam.start_preview()
        ruta = self.getRuta(date, extension = "h264")
        picam.start_recording(ruta)
        totalTime = int(recordingTime)
        picam.wait_recording(totalTime)
        picam.stop_recording()
        picam.stop_preview()
        print("Video guardado en la ruta: {} [OK]".format(ruta))
```

Fuente: Propia

Debido a que la práctica del proceso encargado de controlar la cámara es **No Bloqueante**, se puede continuar con la ejecución del hilo principal. El siguiente paso es iniciar ejecución de la funcionalidad que se encarga de obtener los datos de la tarjeta de identificación, con el fin de determinar si se trata de un usuario validado o no. El comportamiento de este es un poco diferente al del resto componentes macro del sistema. Puesto que, se requiere no solo que se espere al usuario por la información del tag, sino que también tenga un tiempo límite de lectura. La primera idea fue montar un proceso que ejecutara y esperara que el usuario se identificara. O con ayuda del parámetro timeout salir del proceso. Si este fuere el caso, al salir del proceso, el mismo continuaría ejecutándose. Por lo que se debería validar la vida de este y tomar acciones pertinentes. Esta solución fue tempranamente descartada porque, al ejecutarse de este modo, el scope del mismo estaría fuera del hilo principal, ocasionando que la configuración de los pines este fuera de su alcance. Es decir que, al momento de iniciar las interrupciones, este no las detecte. Es por eso que se decide implementar una función que, su vida dependa de las decisiones que pueda tomar un decorador [22]. El método llamado readCard, como se ve en la Figura 49, recibe como parámetro la instancia que controla el sensor basado el protocolo wiegand26 y un timeout que es sacado directamente de la configuración del sistema. Al correr esta función se hace el llamado al método **read_card**, que podemos ver en la Figura 50. Este esta manejado por un bloque try. También se aprecia que esta denotado por el decorador **exitAfter**.

Como se aprecia en la Figura 51 y, aunque se vea un poco abstracto, irracional y no tan fácil de entender. Realmente es muy sencillo saber que está pasando realmente. En resumen, se establece un timeout como flotante y se instancia un Timer del paquete threading. Este recibe tres argumentos básicamente. El temporizador, una función de cierre y un arreglo de argumentos. Al ser un hilo separado, la función principal continúa con su ejecución y llama el método start para iniciar el conteo. Posteriormente y, protegido por un bloque try. Se ejecuta la función encargada de leer la tarjeta. Con una salvedad. Si la lectura es satisfactoria este se culmina correctamente y devuelve el valor esperado. Pero, de no ser así y, el temporizador termina su ejecución primero. Este ejecuta el **callback** que se le paso como parámetro anteriormente. De modo que, este simplemente interrumpe la ejecución del hilo y lanza una excepción que es inmediatamente manejada, terminado o cancelando el mismo timer. Y continuando con el **ciclo del intruder**.

Figura 49. Método para ejecutar la lectura de las tarjetas.



```
@exitAfter()
def readCard(wiegand, timeout):
    try:
        wiegand.read_card()
    except KeyboardInterrupt:
        print('Time Out')
```

Fuente: Propia

Figura 50. Método del controlador que inicia la lectura.

```
def read_card(self, sleepTime=0.001):
    self.id = 0
    self.mask = 0
    self.code = list()
    self.togglesAccess()
    try:
        self.enable_interrupt()
    except RuntimeError:
        print("Interrupcion previamente encendida")
    print("Leyendo ... ")
    while(len(self.code) != self.maximunLenProtocol):
        time.sleep(sleepTime) #esperando lectura de tarjeta
    self.disable_interrupt()
    self.getIdFromCode()
    return(self.id)
```

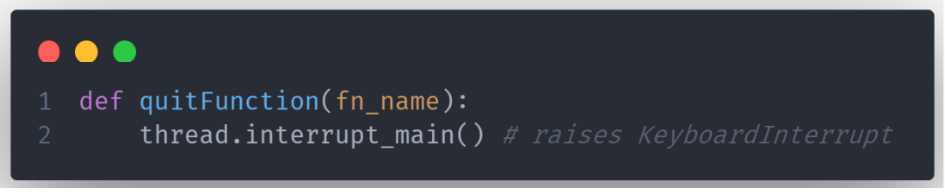
Fuente: Propia

Figura 51. Decorador que controla el timeout del método read_card.

```
def exitAfter():
    def outer(fn):
        def inner(*args, **kwargs):
            timeout = float(args[1])
            timer = threading.Timer(timeout, quitFunction, args=[fn.__name__])
            timer.start()
            try:
                result = fn(*args, **kwargs)
            finally:
                timer.cancel()
            return result
        return inner
    return outer
```

Fuente: Propia

Figura 52. Callback para terminar la ejecución.



```
def quitFunction(fn_name):
    thread.interrupt_main() # raises KeyboardInterrupt
```

Fuente: Propia

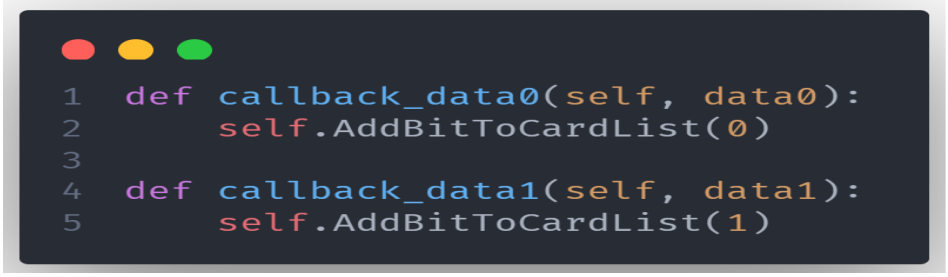
Al correr el método de lectura de tarjetas este debe primeramente limpiar los valores y mostrarle al usuario que ya puede colocar la tarjeta en el sensor. Luego de esto enciende las interrupciones con el método encargado de hacerlo. Figura 53. Además, cuando esto pasa y, la interrupción ya está encendida, se lanza un error. Es por esto que se protege dentro de un bloque try. Luego entra a un ciclo que espera a que el buffer del código este lleno en su totalidad. Puesto que, las interrupciones son las que, como se muestra en la Figura 54. Son las encargadas de llenar esta lista de datos. Al culminar esta operación exitosamente se deshabilitan las interrupciones (Figura 55) y, se obtiene el identificador de la tarjeta. La implementación de esta lógica se aprecia en la Figura 56.

Figura 53. Método para activar las interrupciones en los puertos del sensor RFID.

```
def enable_interrupt(self):
    time.sleep(0.3)
    GPIO.add_event_detect(self.bitCero, GPIO.FALLING)
    GPIO.add_event_detect(self.bitUno, GPIO.FALLING)
    GPIO.add_event_callback(self.bitCero, self.callback_data0)
    GPIO.add_event_callback(self.bitUno, self.callback_data1)
```

Fuente: Propia

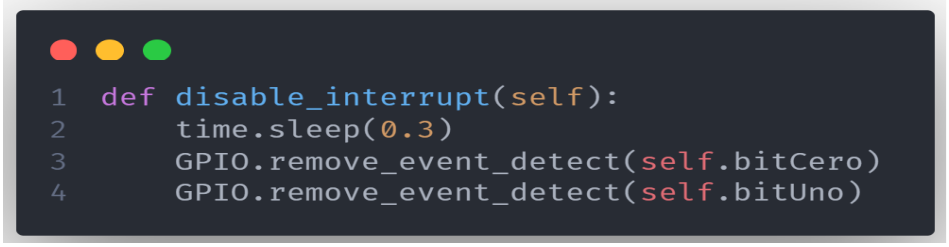
Figura 54. Callbacks de las interrupciones del lector de tarjetas.



```
def callback_data0(self, data0):
    self.AddBitToCardList(0)

def callback_data1(self, data1):
    self.AddBitToCardList(1)
```

Fuente: Propia

Figura 55. Método encargado de deshabilitar las interrupciones en los puertos del sensor RFID.



```
def disable_interrupt(self):
    time.sleep(0.3)
    GPIO.remove_event_detect(self.bitCero)
    GPIO.remove_event_detect(self.bitUno)
```

Fuente: Propia

Figura 56. Método encargado de convertir la colección de bits en un identificador único.

```
def getIdFromCode(self):
    self.code.pop( self.maximunLenProtocol - 1 )
    self.code.pop( 0 )
    try:
        for bit in self.code:
            if(bit == 1):
                self.id = self.id + (1 << ( ( self.maximunLenProtocol - 3 ) - self.mask))
            self.mask += 1
        print("Code ID:\n", self.code)
        print("User ID: ", self.id)
    except:
        print('Error obteniendo el Card ID... ')
```

Fuente: Propia

Luego de obtener la información del identificador de la tarjeta. Se debe cotejar con la información inscrita en la base de datos y así saber si se trata de un usuario del sistema o si simplemente es un intruso. Como se muestra en la Figura 57. El método del servicio recibe como parámetro el identificador leído por el sensor RFID. Esto se hace por que posteriormente se hace una consulta a la colección de usuarios para obtener el que, específicamente, tenga en su metadata en el campo **Access** el mismo identificador. Esto sea cual fuere el caso me retorna una lista de documentos. La cual, para ser un usuario validado se requiere que este tenga valores y que solo tenga uno. De ser así se retorna el valor de la confirmación y el usuario mismo. En caso contrario se envía el valor de la validación y un usuario indefinido.

Se procede a validar que el usuario este validado y que el id sea un valor real. De ser esto favorable, solo se procede a crear un registro con ayuda del método de la Figura 58 y a configurar en 0 el identificador único.

Para la creación del registro es necesario obtener un id para saber cuál se trata. Se procede a buscar en la colección de usuarios el documento con el consumidor específico en su colección de registros, para crear un nuevo documento identificado con el id creado recientemente. Luego se configura un diccionario con los valores enviados en la descripción, el identificador de usuario y, la fecha de creación del reporte. Asimismo, se crea un nuevo documento con el mismo identificador en la colección de registros global.

Figura 57. Método encargado de validar la existencia de la tarjeta en un usuario de la base de datos.

```python
def wiegandIdValidate(self, rfid):
    users = self.db.collection('users').where('access', '==', str(rfid)).get()
    users = [ user for user in users ]
    validate = True if (len(users) > 0) and (len(users) == 1) else False
    if( validate ):
        for user in users:
            return validate, user
    else:
        return validate, 'undefined'
```

Fuente: Propia

Figura 58. Creación de registro para usuario validado.

```python
def createRegister(self, description, userId, date ):
    registerId = self.getNewId()
    accessRef = self.db.collection('users').document(userId).collection('registers').document(str(registerId))
    information = {
        'descripcion': description,
        'user': userId,
        'fechadecreacion': date
    }
    accessRef.set(information)
    accessGlobalRef = self.db.collection('register').document(str(registerId))
    accessGlobalRef.set(information)
```

Fuente: Propia

En caso de no tratarse de un usuario validado, se crea un reporte en la base de datos, pero solo en la colección de registros globales, identificado como un usuario indefinido. Esto se logra con la implementación de método descrito en la Figura 59. Posteriormente, el sistema debe crear una alerta a todos los administradores u operarios que estén configurado en la pestaña de administración de teléfonos de AsbamaAdmin. Para esto, se hace una consulta a la base de datos –Figura 60–

donde, primeramente, se obtienen todos los documentos de la colección donde se guardan los teléfonos. Seguidamente, se crea una lista temporal a modo de placeholder, para recorrer los documentos obtenidos y consultar el usuario, mediante la llave **user**, con el fin de crear un modelo de usuarios y agregarlo en la lista temporal. Para finalmente retornar esta. Puede apreciar en la Figura 61 el modelo mencionado anteriormente.

Figura 59. Registro de usuarios indefinidos.

```python
def undefinedRegister(self, description, undefinedId, date):
    registerId = self.getNewId()
    accessGlobalRef = self.db.collection('register').document(str(registerId))
    accessGlobalRef.set({
        'descripcion': description,
        'user': undefinedId,
        'fechadecreacion': date
    })
```

Fuente: Propia

Figura 60. Método del servicio encargado de consultar por los teléfonos en la base de datos.

```python
def getPhones( self ):
    phones = self.db.collection('telefonos').get()
    phoneTemplateData = list()
    for phone in phones:
        user = self.getUserById(phone.to_dict()['user'])
        model = PhoneModel(user, phone.to_dict()['telefono'], user.to_dict()['role'], phone.to_dict()['fechadeactualizacion'], phone.to_dict()['requerido'])
        phoneTemplateData.append(model)
    return phoneTemplateData
```

Fuente: Propia

Figura 61. Modelo de usuarios para teléfonos.

```python
class PhoneModel():
    def __init__(self, user,telefono, role, update, required):
        self.user = user
        self.telefono = telefono
        self.role = role
        self.update = update
        self.required = required

```

Fuente: Propia

Luego de tener la lista de los usuarios con sus respecticos teléfonos se procede a crear un hilo para enviar el mensaje de alerta a cada una de las personas que estén autorizadas para recibir esta información. Es por esto por lo que se crea un contexto que recibe como parámetros el servicio, la instancia del sim800 y la lista de usuarios. Esto se aprecia en la Figura 62. Luego de recibir esos datos en la clase que controla la ejecución del proceso que enviara la información, este se ejecuta con el método start. Que a su vez llama el método runThread que se observa en el Anexo 18. En este se observa que recorre la lista de usuarios basados en el modelo de teléfonos y, solo a los cuentan con la bandera de requeridos se les arma un mensaje personalizado y se envía con ayuda del método **sms** de la clase sim800 explicada con anterioridad. Finalmente, con ayuda del método join, se espera que el proceso termine su ejecución.

Figura 62. Llamado e inicio del proceso encargado de enviar los mensajes.

```
# Hilo para el Sim800L
phones = service.getPhones()
simModules = [ service, sim, phones]
mySimThread = Sim800Thread("simThread", simModules)
mySimThread.start()
mySimThread.join()
```

Fuente: Propia

Para ambos casos, se debe esperar que el proceso de la cámara termine su ejecución. Y luego validar que el proceso haya culminado satisfactoriamente, de no ser así, se debe matar el proceso como se muestra en la Figura 63.

Figura 63. Validar y matar el proceso de la cámara si es necesario.

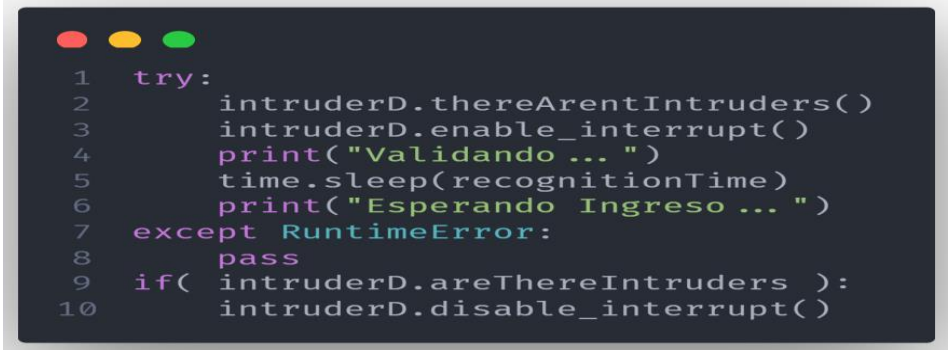
```
myCamaraThread.join()
if(myCamaraThread.is_alive()):
    subprocess.call(['kill', '-9', '{}'.format(myCamaraThread.pid)])
```

Fuente: Propia

Finalmente, se aplica una lógica básica para determinar si aún existe un intruso en la zona. Puede encontrar información de la implementación en la Figura 64. En primera medida se setea el valor de la intromisión en falso y se enciende las interrupciones del intruder. Posteriormente se duerme el sistema por unos pocos segundos. Como el intruder depende de las interrupciones y no del ciclo del hilo principal, si en este tiempo ocurre alguna anomalía la bandera del intruder cambiará

su está a verdadero. Por lo que se valida este mismo. Si aún hay intrusos se apagan las interrupciones y el **intruder loop** vuelve a iniciar y seguir la sucesión de pasos que se explicó en este apartado. En caso contrario la ejecución termina satisfactoriamente y se retorna al **excecution loop** a esperar que exista una irrupción nuevamente.

Figura 64. Validación de existencia de usuarios.



```
try:
    intruderD.thereArentIntruders()
    intruderD.enable_interrupt()
    print("Validando ... ")
    time.sleep(recognitionTime)
    print("Esperando Ingreso ... ")
except RuntimeError:
    pass
if( intruderD.areThereIntruders ):
    intruderD.disable_interrupt()
```

Fuente: Propia

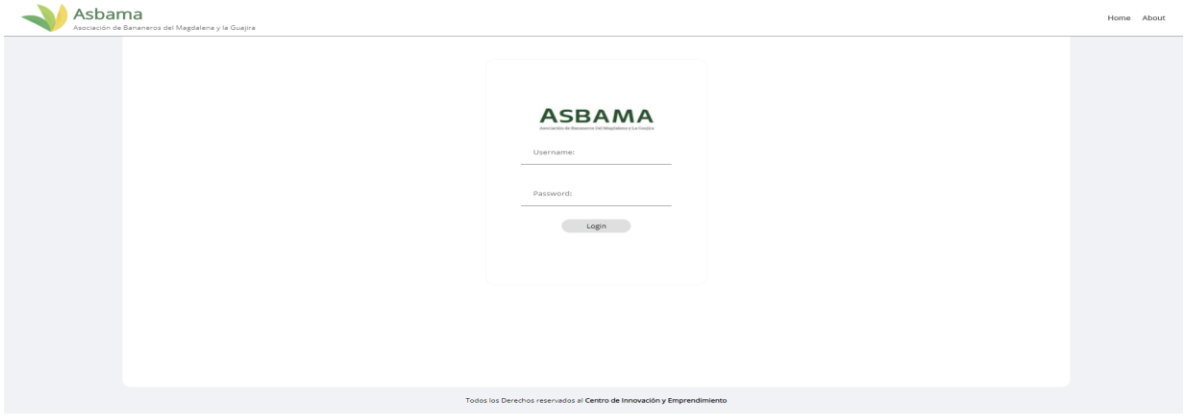
Nota: En el Anexo 19. Se aprecia todo el flujo de trabajo del sistema de alarma mostrado en este módulo. Con toda su lógica y sus protecciones a posibles errores.

1.8 APLICATIVO WEB

Para contar con un sistema innovador, confiable y que ofrezca una mejor visualización y almacenamiento de los datos, se creó el siguiente aplicativo web. Este es capaz de administrar los perfiles de usuarios, almacenar evidencia fotográfica y modificar parámetro del sistema de seguridad que se está implementando. A continuación, describiremos las diferentes interfaces del aplicativo.

Se cuenta con una pantalla de inicio para ingresar los datos del usuario, dependiendo que usuario sea el que ingrese, este tendrá algunas funciones más que otros. Por ejemplo, un perfil de administrador podrá modificar todos los parámetros del sistema, activar o desactivar el sistema, administración de usuarios, activar y desactivar a quienes se les dará aviso al presentarse alguna incidencia, parámetros de la cámara, entre otros.

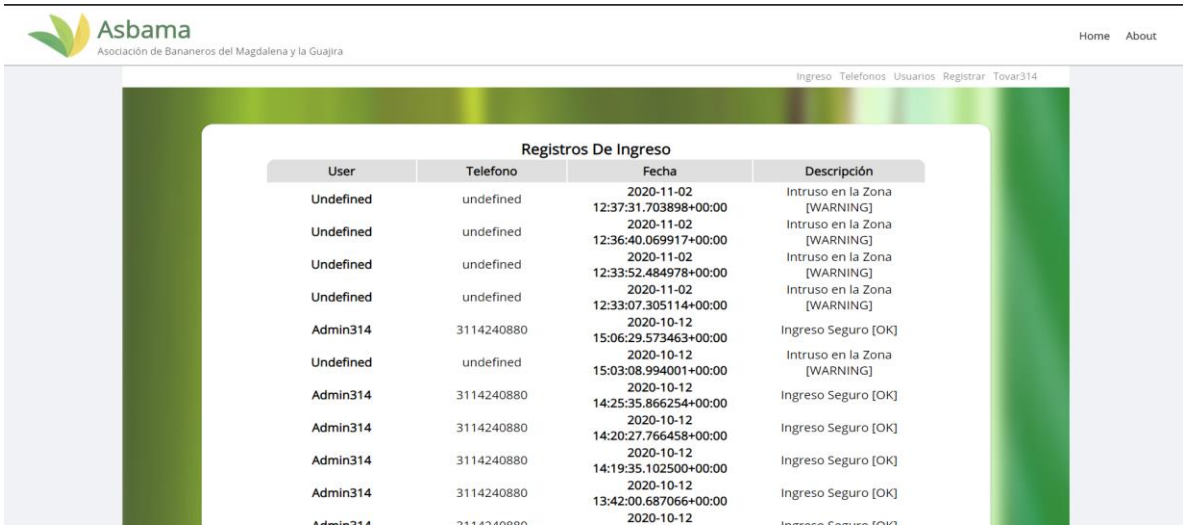
Figura 65. Pantalla de inicio del aplicativo Web.



Fuente: Propia

Al ingresar los datos del usuario seguimos al Home, en esta ventana evidenciamos en tiempo real cada ingreso que se haga a la bodega, sea legitimo o ilegitimo. En el registro de ingreso se discrimina el usuario que ingresó, el teléfono, la fecha y hora, además, si el ingreso es seguro o hay un intruso. Cada uno de estos registros nos permitirá visualizar una foto y un video del interior de la bodega, justo en el preciso momento del ingreso a ella.

Figura 66. Home aplicativo Web.



Fuente: Propia

En la barra de opciones, nos da la posibilidad de ver todos los datos del perfil, cambiar la contraseña, ver y modificar todos los parámetros del sistema y cerrar sesión.

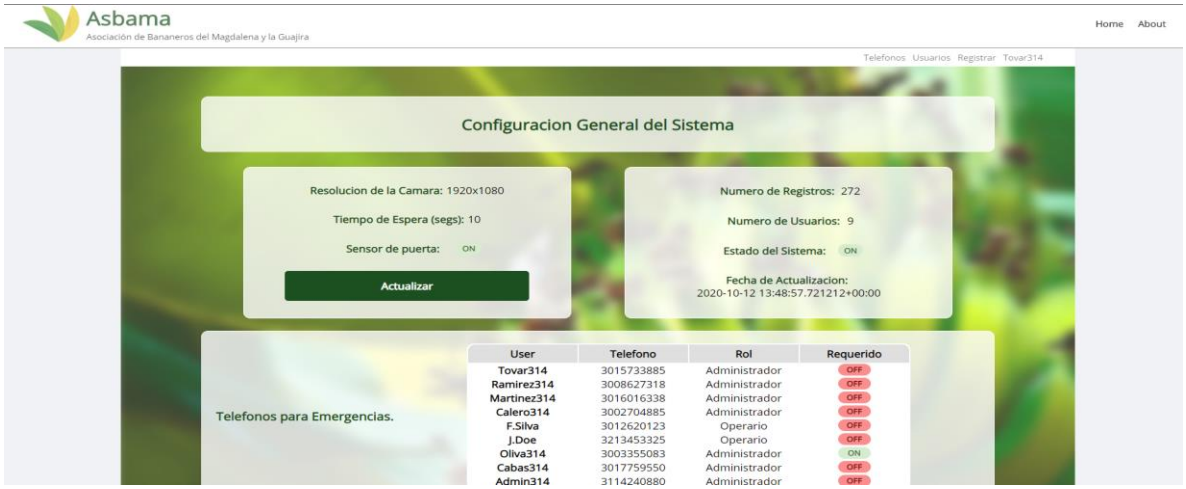
Figura 67. Despliegue barra de opciones.

Fuente: Propia

Como lo mencionamos anteriormente, al ingresar a los Datos de Configuración tendremos todos las condiciones del sistema para modificarlos, también ver el número de usuarios y la cantidad de registros a la fecha.

- Modificar la resolución de los registros fotográficos.
- El tiempo de espera para que se vuelva a activar el sistema.
- Activar o desactivar el sensor de puerta.
- Ver el número de registro a la fecha.
- Cantidad de usuarios registrados.
- Activar o desactivar el sistema.
- Ver la última actualización realizada al sistema.
- Seleccionar los usuarios a los que se les dará aviso por mensaje de texto.

Figura 68. Vista de configuración general del sistema.



Fuente: Propia

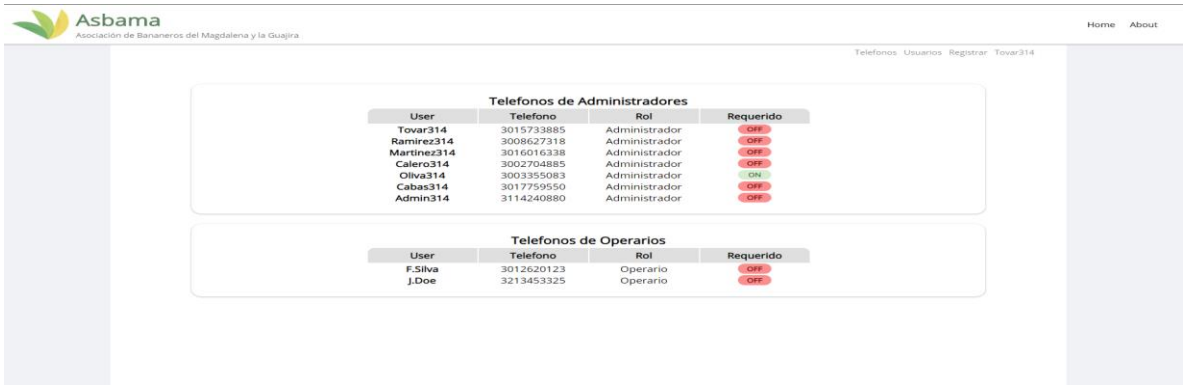
1.8.1 PERFIL DE USUARIO

Figura 69. Vista de información de usuario.

Fuente: Propia

1.8.2 USUARIOS REGISTRADOS EN EL SISTEMA

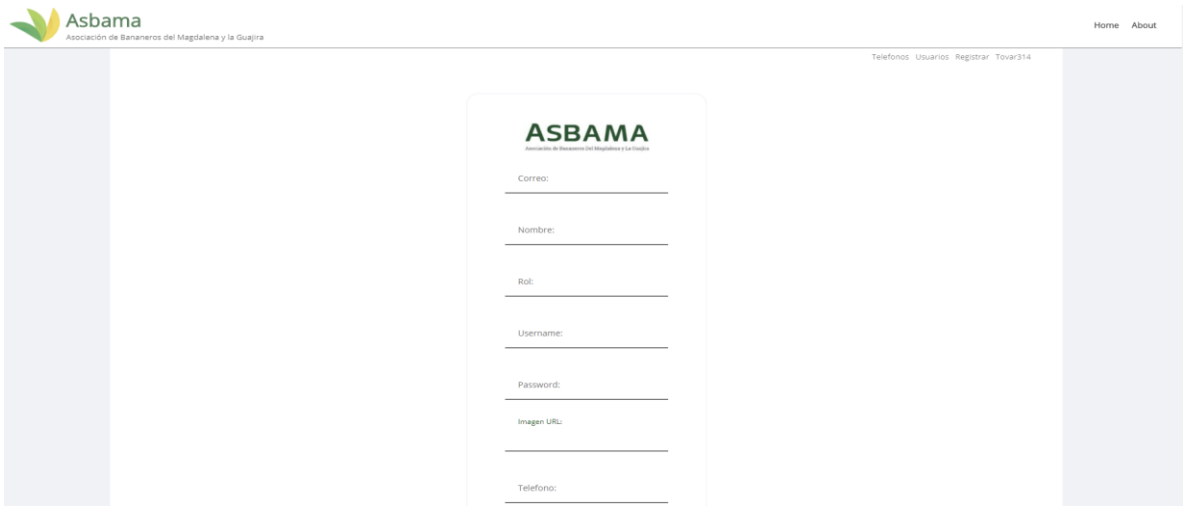
Figura 70. Vista de usuarios registrados en el sistema.



Fuente: Propia

1.8.3 CREACIÓN DE USUARIOS

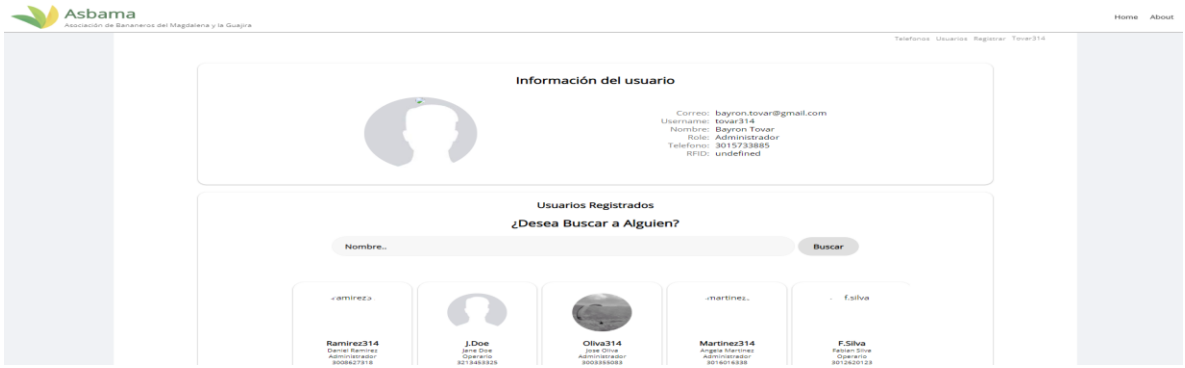
Figura 71. Vista de creación de usuario.



Fuente: Propia

1.8.4 VISUALIZACIÓN DE USUARIOS

Figura 72. Vista general de usuarios.



Fuente: Propia

1.8.5 RESPONSIVE DESIGN DEL APLICATIVO WEB

El responsive design o diseño adaptativo es la técnica que se usa en la actualidad para tener una aplicación web o página web adaptada para cada dispositivo, es decir, celulares, tablets, computadores y televisores [23].

Algo muy necesario en la actualidad para que el producto puede llegar a más usuarios y por ende sea más eficiente el producto final.

Se evidencia el diseño responsivo de las diferentes vistas HTML en el siguiente apartado.

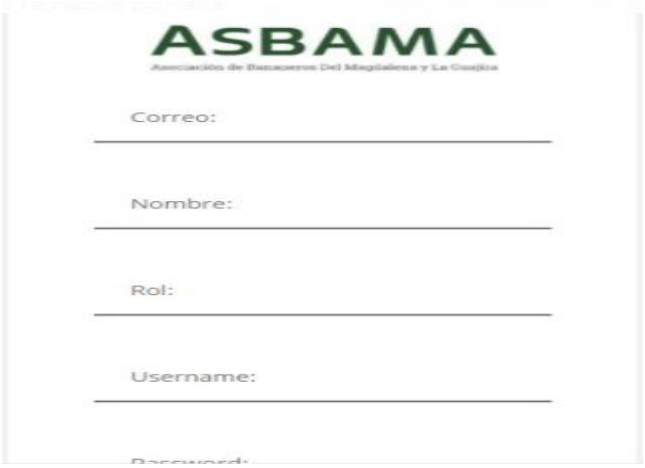
Figura 73. Responsive design vista Registro de ingreso.

Fuente: Propia

Figura 74. Responsive design vista información de usuario.

Fuente: Propia

Figura 75. Responsive design creación de usuario.



Fuente: Propia

Figura 76. Responsive design vista de usuarios registrados.

Fuente: Propia

Figura 77. Responsive design vista configuración general del sistema.

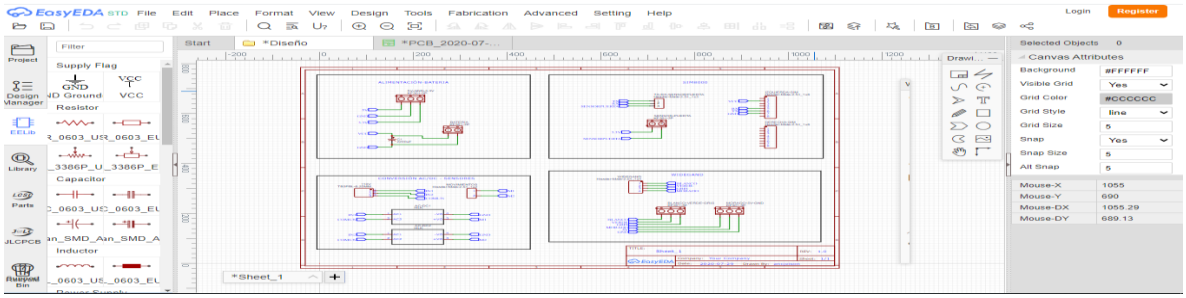
Fuente: Propia

1.9 EASYEDA SIMULATOR

EasyEDA Simulator es un software que te permite realizar de manera gratuita y sin ninguna restricción simulaciones de circuitos y circuitos impresos PCB. Una de las facultades de este software es que se utiliza desde una página web y no necesita instalar ningún plugin adicional, lo cual lo hace asequible al usuario poder realizar un circuito rápido sin un equipo muy potente [24].

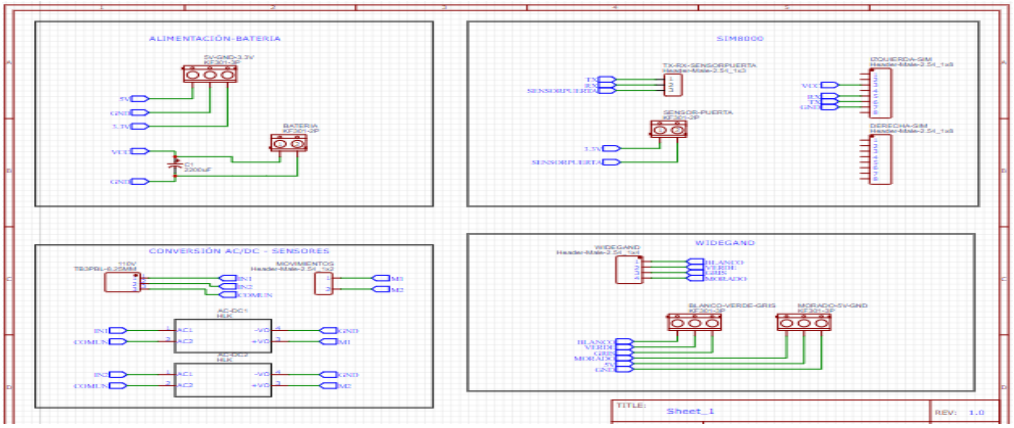
Se optó por realizar los diseños de los circuitos esquemáticos e impresos (PCB) del circuito para efectos de estética y para evitar fallos en la conexión que se evidenciaron en las pruebas realizadas en la protoboard.

Ilustración 6. Entorno de desarrollo de EasyEDA Simulator.



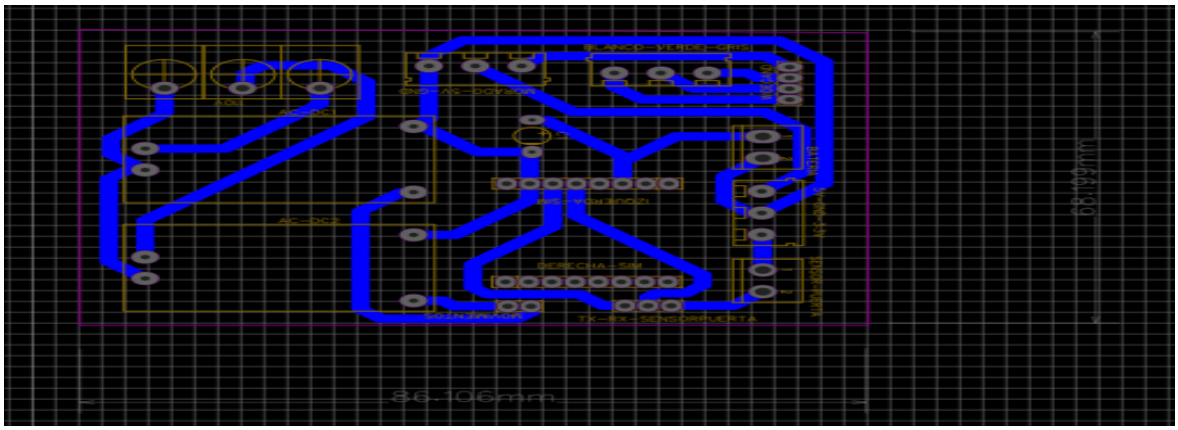
Fuente en línea: https://easyeda.com/editor

Ilustración 7. Diseño de circuitos esquemáticos realizados en EasyEDA Simulator.



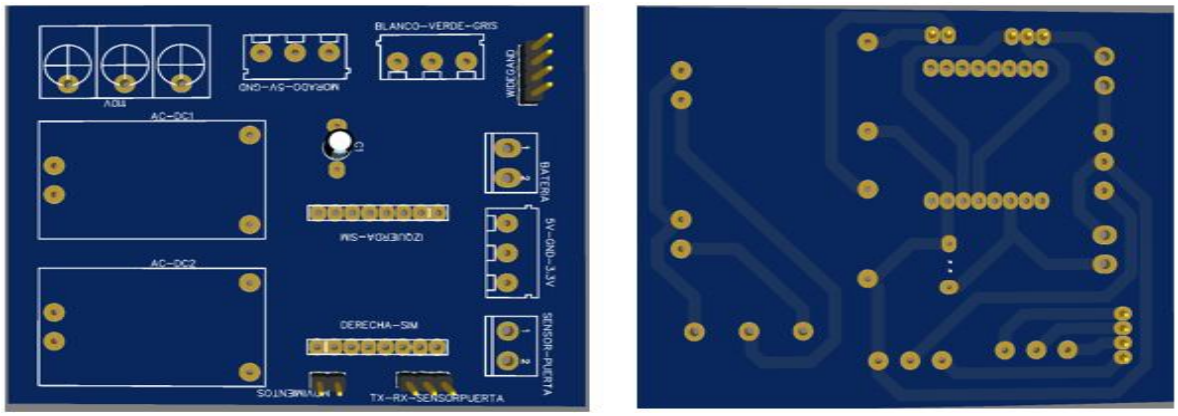
Fuente: Propia.

Ilustración 8. Diseño de circuito impreso PCB realizado en EasyEDA Simulator.



Fuente: Propia.

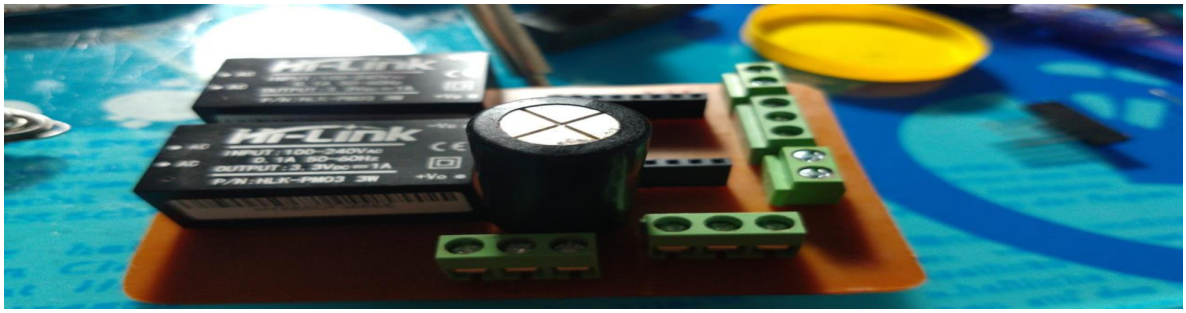
Ilustración 9. Diseño de circuito impreso (PCB) visualización 3D.



Fuente: Propia.

El desarrollo de la PCB se realizó en una maquina CNC para mayor optimización del espacio y una mejor estética del circuito implementado, teniendo como resultado la siguiente placa.

Ilustración 10. Circuito implementado en una PCB.



Fuente: Propia

2 LÍNEAS FUTURAS

Para una futura optimización del sistema se sugiere hacer cambios a los conectores que se utilizan en la PCB de 5V-GND-3.3V, reemplazando la bornera KF301 que consta de 3 pines por un conector de 3 pines macho para que se facilite la conexión de la Raspberry pi con la placa de circuito impreso.

También se recomienda implementar una forma para descargar el capacitor de entrada que tiene una capacitancia de 2200 μf a 50V, Esto con el objetivo de evitar posibles interferencias con el funcionamiento de los sensores de movimiento que nos envían un pulso.

Una de las principales mejoras al sistema a futuro es la implementación de otro tipo de cámara para lograr generar los videos de grabación en otro formato para acceder a estos sin la necesidad de descargarlos y convertir el formato, logrando que los videos generados por la grabación del sistema puedan ser vistos online desde la interfaz administrativa, con lo cual se puede tener estos videos en una base de datos.

También se busca trasladar el sistema para un servidor en la nube y evitar la carga que tiene la raspberry pi para que esta se dedique exclusivamente al sistema de seguridad teniendo mejores respuestas.

Teniendo en cuenta que el sistema de seguridad está enfocado principalmente a la entrada y vigilancia de la bodega de insumos, además del corto presupuesto que se obtuvo para el prototipo, no es posible implementar una forma de evitar los saqueos de materiales, insumos y demás herramientas. Por lo que se sugiere a lo largo del escrito seguridad privada o entes de seguridad de la zona, los cuales son alertados por la alarma vía SMS que es enviada a las personas designadas en la interfaz administrativa.

Sin embargo, el sistema puede obtener una mejora para implementar refiriéndose al monitoreo de los materiales que están dentro de la bodega, puede ser implementado código RFID en estos materiales para que el usuario que está dentro de la bodega registre en la interfaz administrativa el material a sacar y que este sistema solo permita sacar dichos materiales, en caso de que se encuentre algún material demás también se emita una alarma vía SMS al personal designado.

CONCLUSIONES

En consideración a los requisitos propuestos para el desarrollo de este proyecto de innovación y emprendimiento se lograron obtener nuevos conocimientos en el área de programación y de innovación, los cuales fueron primordiales para la implementación del prototipo y del aplicativo web.

A su vez presentamos inconvenientes en la implementación del prototipo físico lo cual nos llevó a crear mecanismos de simulación para evidenciar posibles errores y mejoras, estas se lograban efectuar en las diferentes etapas de validación del proyecto con lo cual buscábamos generar mayor confianza, robustez y veracidad de la información.

Este prototipo busca ser escalable para un futuro, creando un sistema de seguridad aún más robusto que pueda servir de ayuda para fortalecer conocimientos en programación y desarrollo de software a futuros jóvenes investigadores, innovadores y emprendedores.

REFERENCIAS Y BIBLIOGRAFÍA

[1] M. P. Gutiérrez Diaz y R. Villegas Téllez, Sistema de control de acceso basado en tecnologia arduino y RFID (Tesis), JÓVENES EN LA CIENCIA vol.7, 2019.

[2] O. G. Fuentes Lanfor y J. F. Pérez Pérez, Implementación de un sistema de seguridad independiente y automatización de una residencia por medio del internet de las cosas (Tesis), Panamá: IEEE, 2017.

[3] C. A. González Godoy y O. J. Salcedo Parra, «Sistema de seguridad para locales comerciales mediante Raspberry Pi, cámara y sensor PIR (Articulo),» *Revista virtual de la Universidad Católica del Norte.*

[4] G. Brito y C. Izurieta, Diseño de un Sistema de Seguridad mediante Cámaras IP para la empresa Proalpi de la Ciudad de Píllaro (Tesis), Universdad Técnica de Ambato.

[5] I. Pérez, Desarrollo de un sistema de seguridad para parqueaderos basados en tecnología RFID (Tesis), Quito: Universidad Tecnológica Equinoccial .

[6] C. García Muelas , «http://openaccess.uoc.edu/,» 2015. [En línea]. Available: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/40187/6/cgmuelasTFC0115memoria.pdf.

[7] E. Lázaro, «neoguias.com,» 27 mayo 2019. [En línea]. Available: https://www.neoguias.com/como-instalar-pip-python/.

[8] dreamhost, «help.dreamhost.com,» [En línea]. Available: help.dreamhost.com/hc/es/articles/115000695551-Instalar-y-usar-virtualenv-con-Python-3. [Último acceso: 2021].

[9] J. Domingo Muñoz, «openwebinars.net,» 17 Noviembre 2017. [En línea]. Available: https://openwebinars.net/blog/que-es-flask/.

[10] J. A. Muro, «deloitte,» 2021. [En línea]. Available: https://www2.deloitte.com/es/es/pages/technology/articles/que-es-orm.html.

[11] Google , «firebase.google,» [En línea]. Available: firebase.google.com/docs/admin/setup?hl=es-419#:~:text=El%20SDK%20de%20Admin%20es,ejecutar%20acciones%20c

omo%20las%20siguientes%3A&text=Accede%20a%20recursos%20de%20Google,con%20tus%20proyectos%20de%20Firebase..

[12] Google, «cloud.google,» [En línea]. Available: https://cloud.google.com/sdk/gcloud?hl=es-419.

[13] SQLALCHEMY, «sqlalchemy,» [En línea]. Available: https://www.sqlalchemy.org/.

[14] Hostinger, «hostinger.es,» 9 Diciembre 2020. [En línea]. Available: https://www.hostinger.es/tutoriales/comando-curl.

[15] Google, «gcloud.google,» [En línea]. Available: https://cloud.google.com/sdk/gcloud/reference/init?hl=es-419.

[16] Y. Fernández, «xataka,» [En línea]. Available: https://www.xataka.com/basics/api-que-sirve.

[17] flask.palletsprojects, «flask.palletsprojects.com,» [En línea]. Available: https://flask.palletsprojects.com/en/1.1.x/blueprints/.

[18] flask-login.readthedocs, «flask-login.readthedocs.io,» [En línea]. Available: https://flask-login.readthedocs.io/en/latest/.

[19] flask.palletsprojects, «flask.palletsprojects,» [En línea]. Available: https://flask.palletsprojects.com/en/1.1.x/patterns/wtforms/.

[20] Uniwebsidad, «Uniwebsidad.com,» [En línea]. Available: https://uniwebsidad.com/libros/django-1-0/capitulo-4/usando-el-sistema-de-plantillas.

[21] P. Galvo Garrido, «eustat.eus,» [En línea]. Available: https://www.eustat.eus/documentos/datos/ct_03_c.pdf.

[22] apsl.net, «apsl.net,» 31 Agosto 2009. [En línea]. Available: https://www.apsl.net/blog/2009/08/31/decoradores-en-python/.

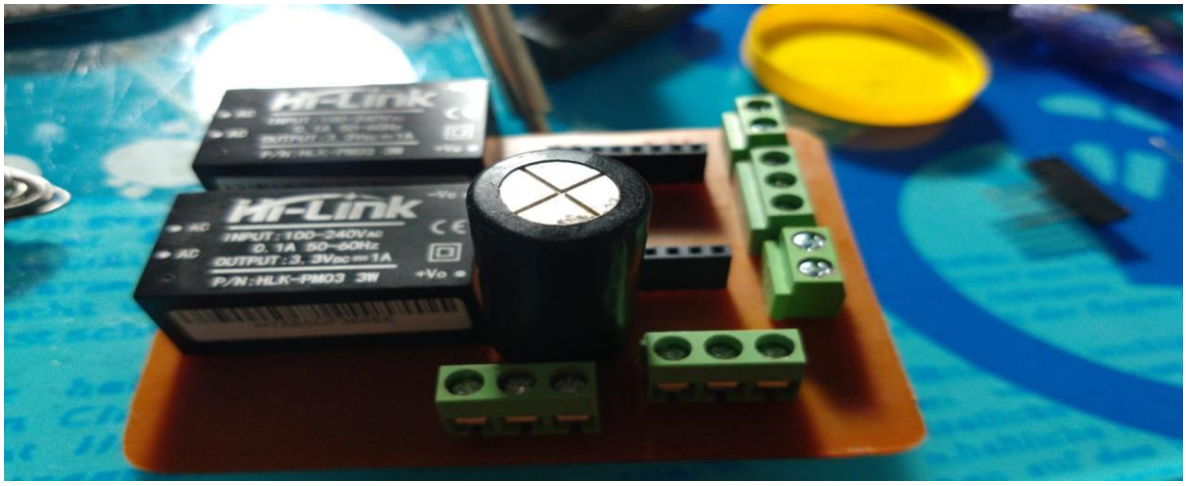
[23] w3schools, «w3schools.com,» [En línea]. Available: https://www.w3schools.com/html/html_responsive.asp.

[24] redeszone, «redeszne.net,» 19 Febrero 2016. [En línea]. Available: https://www.redeszone.net/2016/02/19/conoce-easyeda-un-completo-software-de-simulacion-de-circuitos-y-diseno-de-pcb-online/.

ANEXOS

ANEXO 1. Registro fotográfico.

Figura 78. Circuito impreso en la PCB.



Fuente: Propia

Figura 79. Caja Principal de la instalación del prototipo.

Fuente: Propia

ANEXO 2. Código para el funcionamiento conjunto del SIM800L.

```
Import serial
       import time
       import subprocess
       import os
       class Sim800l:
           def __init__(self, puerto = "ttyS0"):
               print("Inicializando SIM800L...")
               self.puerto = puerto
               self.phone = self._serialInit()
               print("Inicializado SIM800L [OK]")
           def _serialInit(self, baudios = 9600, timeout=1):
               return serial.Serial('/dev/{}'.format(self.puerto),baudios,
               timeout=timeout)
           def getPhone(self):
               return self.phone
           def sms(self, mensaje, numeroDeTelefono, encode='utf-8'):
               self._sendSMS("AT\r", encode, 1)
               self._sendSMS("AT+CMGF=1\r\n", encode, 1)
               self._sendSMS('AT+CMGS="+57{}"\r\n'.format(numeroDeTelefono), encode, 2)
               self._sendSMS(mensaje+chr(26), encode, 4)
               print("Mensaje enviado")
           def _sendSMS(self, codigoAT, encode, seg):
               self.phone.write(codigoAT.encode(encode))
               time.sleep(seg)
           def sim800Response(self, mensaje="CMTI"):
               while True:
                   try:
                       response = str(self.phone.readline())
                       print(response)
```

```
                    if mensaje in response:
                        break
                except:
                    print ("Ha ocurrido una excepcion")
                    print ('-'*60)
                    subprocess.call(['sudo','systemctl','stop','serial-
    getty@{}.service'.format(self.puerto)])
```

Fuente: Propia

ANEXO 3. Código para el funcionamiento de la cámara.

```
import time
    import picamera
    import os
    import subprocess
    from datetime import datetime
    from Apps.Hilos.convertThread import ConvertThread
    class Camara:
        def __init__(self, ruta = "../App/static/Security/Evidencias/", waitingTime =
            print("Inicializando CAMARA...")
            self.waitingTime = waitingTime
            self.ruta = ruta
            print("Inicializada CAMARA [OK]")
        def getPacket(self, resolucion = "720x560", date = datetime.now(), recordingTime
            widht, heith = self.getResolucion(resolucion)
            recordingTime *= self.waitingTime
            self._camFoto(widht, heith , self.waitingTime, date)
            self._camVideo(widht, heith, recordingTime, date)
        def getResolucion(self, resolucion):
            res = resolucion.split("x")
            return int(res[0]), int(res[1])
```

```
def _camFoto(self, widht, heith , seg, date):
            print("Tomando foto...")
            with picamera.PiCamera() as picam:
                picam.resolution = (widht, heith)
                picam.start_preview()
                time.sleep(seg)
                ruta = self.getRuta(date)
                picam.capture(ruta)
                picam.stop_preview()
                picam.close()
                print("Foto guardada en la ruta: {} [OK]".format(ruta))
        def getRuta(self, date = datetime.now(), extension = "jpg" ):
            enrutado = "{}\'{}+00:00\'/".format(self.ruta, str(date))
            try:
                os.mkdir(enrutado)
            except FileExistsError:
                print("Directorio encontrado")
            finally:
                return "{}\'{}+00:00\'.{}".format(enrutado, str(date), extension)
        def _camVideo(self, widht, heith , recordingTime, date):
            print("Grabando...")
            with picamera.PiCamera() as picam:
                picam.resolution = (widht, heith)
                picam.start_preview()
                ruta = self.getRuta(date, extension = "h264")
                picam.start_recording(ruta)
                totalTime = int(recordingTime)
                picam.wait_recording(totalTime)
                picam.stop_recording()
                picam.stop_preview()
                print("Video guardado en la ruta: {} [OK]".format(ruta))
```

Fuente: Propia

ANEXO 4. HTML renderizado en primera instancia.

```
{% extends "base.html"  %}
{% import 'bootstrap/wtf.html' as wtf %}
{% block title %}
    {{super()}}
    Login
{% endblock %}
{% block content%}
    {% if userIp %}
        <div class="form__container">
            <img
                class="logo"
                src="{{ url_for('static',
filename='images/asbama.jpg') }}"
                alt="Asbama Logo">
            <form
                action="{{ url_for('auth.login') }}"
                method="POST"
                class="form__container-session"
                role="form">
                <div class="container__session-item">
                    {{ login.username }}
                    {{ login.username.label }}
                </div>
                <div class="container__session-item">
                    {{ login.password }}
                    {{ login.password.label }}
                </div>
                <div class="container__session-submit">
                    {{ login.submit }}
                </div>
            </form>
        </div>
    {% else %}
        <a href="{{ url_for('index') }}"> Algo ha salido
mal, Reload </a>
    {% endif %}
{% endblock %}
```

Fuente: Propia

ANEXO 5. Método que se ejecuta cuando se hace un llamado al Endpoint Login.

```
@auth.route('/login', methods=['GET', 'POST'])
def login():
    userIp = session.get('userIp')
    login = LoginForm()
    context =  {
        'userIp': userIp,
        'login': login    }
    if ( login.is_submitted() ):
        username = (login.username.data).lower()
        #session['username'] =  login.username.data#request.form.get('username')
        password = login.password.data
        userDoc = getUser(username)
        if userDoc is not None:
            passwordFromDb = userDoc.to_dict()['password']
            if check_password_hash(pwhash=passwordFromDb, password=password):
                userData = UserData(
                    username=username,
                    password=password,
                    id=userDoc.id,
                    correo=userDoc.to_dict()['correo'],
                    nombre=userDoc.to_dict()['nombre'],
                    role=userDoc.to_dict()['role'],
                    imagen=userDoc.to_dict()['imagen'],
                    telefono=userDoc.to_dict()['telefono'],
                    access=userDoc.to_dict()['access'],
                    fechadeactualizacion=userDoc.to_dict()['fechadeactualizacion']
)
                user = UserModel(userData)
                login_user(user)
                flash('Usario: {}, Ah iniciado sesion con
exito'.format(user.username.title()))
                session['username'] = username
                return redirect(url_for('home'))
            else:
                flash("Contraseña incorrecta, vuelva a intentarlo.", 'error')
        else:
            flash("El usuario {}, No has sido encontrado.".format(username), 'error')
    return render_template('login.html', **context)
```

Fuente: Propia

ANEXO 6. Clase UserData.

```
class UserData():
    def __init__(
        self, username,
        password, id = '',
        correo = '', nombre= '',
        role='', imagen='',
        telefono='', access='',
        fechadeactualizacion=''
    ):
        self.username = username
        self.password = password
        self.id = id
        self.correo = correo
        self.nombre = nombre
        self.role = role
        self.imagen = imagen
        self.telefono = telefono
        self.access = access
        self.fechadeactualizacion=fechadeactualizacion
```

Fuente: Propia

ANEXO 7. Clase UserModel.

```
class UserModel(UserMixin):
    def __init__(self, userData):
        self.username = userData.username
        self.password = userData.password
        self.id = userData.id
        self.correo = userData.correo
        self.nombre = userData.nombre
        self.role = userData.role
        self.imagen = userData.imagen
        self.telefono = userData.telefono
        self.access = userData.access
        self.fechadeactualizacion=userData.fechadeactualizacion
```

Fuente: Propia

ANEXO 8. Endpoint de Home.

```
@app.route('/home', methods=['GET'])
@login_required
def home():
    userIp = session.get('userIp')
    registros = getRegister()
    context = {
        'userIp': userIp,
        'registros': registros,
        'registroCode': 314,
        'background': chargeBackgruound()
    }
    return render_template('home.html', **context
```

Fuente: Propia

ANEXO 9. Servicio para obtener los registros de ingreso a la bodega.

```
def getRegister():
    registros = db.collection('register').order_by(
        'fechadecreacion',
        direction=firestore.Query.DESCENDING).get()
    registerTemplateData = list()
    for registro in registros:
        if(str(registro.to_dict()['user']) != 'undefined' ):
            user = getUserById(str(registro.to_dict()['user']))
            model = RegitrosModel(
                user.to_dict()['username'],
                user.to_dict()['telefono'],
                registro.to_dict()['fechadecreacion'],
                registro.to_dict()['descripcion'] )
        else:
            model = RegitrosModel(
                str(registro.to_dict()['user']),
                str(registro.to_dict()['user']),
                registro.to_dict()['fechadecreacion'],
                registro.to_dict()['descripcion'])
        registerTemplateData.append(model)
    return registerTemplateData
```

Fuente: Propia

ANEXO 10. Template del endpoint Home.

```
{% extends "base.html"  %}
{% import "macros.html" as macros %}
{% import 'bootstrap/wtf.html' as wtf %}

{% block title %}
    {{super()}}
    Bienvenido
{% endblock %}

{% block content%}
    {% if userIp %}
        <div class="bienvenida" id="bienvenida">
            <h1>Hola {{ username | capitalize }}, Bienvenido a</h1>
            <img class="logo" src="{{ url_for('static', filename='images/asbama.jpg')
}}" alt="Asbama Logo">
        </div>
        {% if registros %}
            <div class="current__user-info">
                <div class="table__title">
                    Registros De Ingreso
                </div>
                <table class="currents__users-table">
                    <tr class="users__table-master">
                        <th class="users__table-item corner-left">User</th>
                        <th class="users__table-item">Telefono</th>
                        <th class="users__table-item" id="last">Fecha</th>
                        <th class="users__table-item corner-right">Descripción</th>
                        {{ macros.render_registerTableData(registros) }}
                    </tr>
                </table>
            </div>
            <div class="bkg__container">
                <img src="{{ background }}" alt="imagen" class="background__image">
            </div>
        {% endif %}
        {% else %}
            <a href="{{ url_for('index') }}"> Algo ha salido mal, Reload </a>
        {% endif %}
{% endblock %}
```

Fuente: Propia

ANEXO 11. Template base de la aplicación.

```
{% import "macros.html" as macros %}
<!DOCTYPE html>
    <html lang="es">
        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
            <title>
                {% block title %} Asbama |  {% endblock %}
            </title>
            <!-- <link rel="icon" href="https://i.ibb.co/1Qjjj2t/pHcq4C3C.jpg"> -->
            <link rel="stylesheet" href="{{ url_for('static',
filename='styles/styles.css') }}">
            <link href="https://fonts.googleapis.com/css?family=Open+Sans&display=swap"
rel="stylesheet">
            <link rel="icon" href="{{ url_for('static', filename='images/logo_vec.png')
}}">
        </head>
        <body>
            <div class="header__container">
                {% include 'header.html' %}
            </div>
            {% for category, message in get_flashed_messages(with_categories=true) %}
                {{ macros.render_flashes(category, message) }}
            {% endfor %}
            <section class="container">
                {% include 'ribbon.html' %}
                <div class="content">
                    {% block content %}                          content
                    {% endblock %}
                </div>
            </section>
            <div class="footer__container">
                {% include 'footer.html' %}
            </div>
        </body>
        <script type="text/javascript">
            setTimeout(() => {
                CerrarContenedor("alerta");
            }, 5000 );
            setTimeout(() => {
                CerrarContenedor("error");
            }, 10000 );

            setTimeout(() =>{
```

```
                CerrarContenedor("bienvenida");
            }, 15000);

            function CerrarContenedor(id){
                let contenedor = document.getElementById(id);
                contenedor ? contenedor.style.display = "none" : console.log(`Contenedor
con Id: ${id}, no encontrado.`);
            }
        </script>
    </html>
```

Fuente: Propia

ANEXO 12. Implantación para el manejo de códigos de error.

```
    @app.errorhandler(404)
    def notFound(error):
        context = {
            "error": error,
            "number": "404",
            "title": "Not Found",
            "msg": "Lo Siento No Encontramos Lo Que Buscabas."
        }
        return render_template('error.html', **context)

    @app.errorhandler(500)
    def internalError(error):
        context = {
            "error": error,
            "number": "500",
            "title": "Internal Server Error",
            "msg": "Lo Siento Ocurrio un problema en el Servidor, Intenta más Tarde."
        }
        return render_template('error.html', **context
```

Fuente: Propia

ANEXO 13. Template de error.

```
{% extends "base.html"  %}

{% block title %}
    {{super()}}
    {{ title }}
{% endblock %}

{% block content%}
    <div class="error__alert">
        <h1 class="error__alert-code animated bounce">
            {{ number }}
        </h1>
        <p class="error__alert-msg">
            {{ msg }}
        </p>
        <p class="error__alert-ficha-tecnica">{{ error }}</p>
    </div>
{% endblock %}
```

Fuente: Propia

ANEXO 14. Macro de los flashes.

```
{% macro render_flashes(category, message) %}
    {% if category == 'error'  or  category == 'messege' %}
        <div class="alert" id="{{ category }}">
            <button type="button"
                data-dismiss="error"
                class="close" onclick="CerrarNotificacionEmergenteError()">
                X
            </button>
            <!-- window.location.reload() -->
            <div class="mensaje" id="{{ category }}-mensaje">
                {{ message }}
            </div>
        </div>
    {% else %}
        <div class="alert" id="alerta">
            <button type="button"
                data-dismiss="alert"
                class="close" onclick="CerrarNotificacionEmergente()">
```

```
            <!-- window.location.reload() -->
            <div class="mensaje" id="alerta-mensaje">
                {{ message }}
            </div>
        </div>
    {% endif %}
{% endmacro %}
```

Fuente: Propia

ANEXO 15. Macro render_userItemData.

```
{% macro render_userItemData(users) %}
    {% for user in users %}
        <div class="carousel__item">
            <img class="carousel__item-img" src="{{ user.to_dict()['imagen'] }}" alt="{{
user.to_dict()['username'] }}">
            <div class="carousel__item-details">
                    <p class="item__details-username">
                        <a
                            class="vinculo green__text"
                            href="{{ url_for('users.externalUserData',
username=user.to_dict()['username']) }}">
                            {{ user.to_dict()['username'].title() }}
                        </a>
                    </p>
                    <p class="item__details-data" id="last">{{ user.to_dict()['nombre']
}}</p>
                    <p class="item__details-data">{{ user.to_dict()['role'] }}</p>
                    <p class="item__details-data">{{ user.to_dict()['telefono'] }}</p>
            </div>
            <a
                href="{{ url_for('users.externalUserData',
username=user.to_dict()['username']) }}"
                class="carousel__item-anchor">
            </a>
        </div>
    {% endfor %}
{% endmacro %}
```

Fuente: Propia

ANEXO 16. Template del ribbon.

```
{% if current_user.is_authenticated %}
    <div class="ribbon">
        <ul class="ribbon__list">
            {% if current_user.role == 'Administrador' %}
                {% if registroCode == 314 %}
                    <li class="ribbon__list--item">
                        <a href="{{ url_for('registros') }}"
class="linked__ribbon">Ingreso</a>
                    </li>
                {% endif %}
                <li class="ribbon__list--item">
                    <a href="{{ url_for('telefonos') }}"
class="linked__ribbon">Telefonos</a>
                </li>
                {% if update == 1%}
                    <li class="ribbon__list--item">
                        <a href="{{ url_for('auth.updateData') }}"
class="linked__ribbon">Actualizar</a>
                    </li>
                {% endif %}
                {% if update == 2 %}
                    <li class="ribbon__list--item" id="last">
                        <a
                            href="{{ url_for('auth.updateExternalData',
username=user.to_dict()['username']) }}"
                            class="linked__ribbon">
                            Actualizar Rol
                        </a>
                    </li>
                    <li class="ribbon__list--item">
                        <a
                            href="{{ url_for('users.deleteUser',
username=user.to_dict()['username']) }}"
                            class="linked__ribbon">
                            Eliminar
                        </a>
                    </li>
                {% endif %}
                <li class="ribbon__list--item">
```

```
                </li>
                <li class="ribbon__list--item">
                    <a href="{{ url_for('auth.signup') }}"
class="linked__ribbon">Registrar</a>
                </li>
            {% endif %}
            <li class="ribbon__list--item">
                <div class="user__nav">
                    <div class="user__nav--logo">
                            <!-- <img class="icon" src="./images/Logos/icons8-perfil-
de-gato-100.png" alt="Profile"> -->
                            <p class="title"> {{ current_user.username | capitalize }}
</p>
                    </div>
                    <ul class="user__nav--block">
                        <li class="nav__block--item">
                            <a
                                href="{{ url_for('account') }}"
                                class="linked__ribbon--menu">
                                Cuenta
                            </a>
                        </li>
                        <li class="nav__block--item">
                            <a
                                href="{{ url_for('auth.changePassword') }}"
                                class="linked__ribbon--menu">
                                C. Contraseña
                            </a>
                        </li>
                        {% if current_user.role == 'Administrador' %}
                            <li class="nav__block--item">
                                <a
                                    href="{{ url_for( 'auth.configuration' ) }}"
                                    class="linked__ribbon--menu">
                                    Datos de Configuración
                                </a>
                            </li>
                        {% endif %}
                        <li class="nav__block--item">
                            <a
                                href="{{ url_for('auth.logout') }}"
                                class="linked__ribbon--menu">
Log Out
                            </a>
```

```
                </div>
            </li>
        </ul>
    </div>
{% endif %}
```

Fuente: Propia

ANEXO 17. Inicialización de la clase IntruderDetecter.

```
class IntruderDetecter:
    def __init__(self, door=12, moveOne=13, moveTwo=6, lamp=16, setMode="BCM",
setWarnings=False):
        self.door = door
        self.moveOne = moveOne
        self.moveTwo = moveTwo
        self.lamp = lamp
        self.lampState = False
        self.printer = Printer()
        self.areThereIntruders = False
        self._pinModeInit(setMode, setWarnings)


    def _pinModeInit(self, setMode="BCM", setWarnings=False):
        GPIO.setwarnings(setWarnings)
        GPIO.setmode(self._piSetMode(setMode))

        self._piPinSetup(channel=self.door, state="IN" )
        # GPIO.setup(self.door, GPIO.IN,pull_up_down=GPIO.PUD_UP)
        self._piPinSetup(channel=self.moveOne,  state="IN" )
        self._piPinSetup(channel=self.moveTwo,  state="IN")
        self._piPinSetup(channel=self.lamp,  state="OUT")
```

Fuente: Propia

ANEXO 18. Clase encargada de controlar el proceso que envía la información a los usuarios.

```
        from Apps.Hilos.myThread import MyThread
        from Apps.Wiegand.wiegandEngine import WiegandEngine

        class Sim800Thread(MyThread):
            def __init__(self, threadName, modules=[ WiegandEngine() ]):
                super().__init__(threadName=threadName)
                self.service = modules[0]
                self.sim = modules[1]
                self.phones = modules[2]

            def runThread(self):
                for phone in self.phones:
                    if(phone.required):
                        print('Enviando aviso a: {}...'.format(phone.telefono))
                        mensaje = ('Hola {}, ha irrumpido \
                            un intruso en la bananera.'.format(
                                phone.telefono
                            )
                        )
                        self.sim.sms(mensaje, str(phone.telefono))
```

Fuente: Propia

ANEXO 19. Método núcleo del sistema de seguridad.

```
def _run(wiegand, sim, camara, intruderD, service):
    while(True):
        try:
            try:
                recognitionTime = 3.0
                intruderD.enable_interrupt()
                print("Esperando Ingreso...")
                while( True ):
                    if(intruderD.areThereIntruders):
```

```
                        systemState = configutation.to_dict()['estadodelsistema']
                        resolution = configutation.to_dict()['resolucioncamara']
                        waitingTime = configutation.to_dict()['tiempodeespera']
                        print("Data Service: ", systemState, resolution, waitingTime)
                        if( systemState ): # si el sistema está encendido
                            intruderD.disable_interrupt()
                            intruderD.togglesLamp()



                            while(intruderD.areThereIntruders):
                                date = datetime.now()
                                # Hilo para la camara
                                camaraModules = [ camara, resolution, date,waitingTime
]
                                myCamaraThread = CamaraThread("camaraThread",
camaraModules)
                                myCamaraThread.start()
                                #Lectura de Wiegand
                                readCard(wiegand, waitingTime)
                                # Intruders Validation
                                userValidate, user =
service.wiegandIdValidate(str(wiegand.id))
                                if( wiegand.id != 0 and userValidate):
                                    service.createRegister("Ingreso Seguro [OK]",
str(user.id), date)
                                    wiegand.id = 0
                                else:
                                    service.undefinedRegister("Intruso en la Zona
[WARNING]", user, date)
                                    # Hilo para el Sim800L
                                    phones = service.getPhones()
                                    simModules = [ service, sim, phones]
                                    mySimThread = Sim800Thread("simThread",
simModules)
                                    mySimThread.start()
                                    mySimThread.join()

                                myCamaraThread.join()
                                if(myCamaraThread.is_alive()):
```

```
                                print("Validando...")
                                time.sleep(recognitionTime)
                                print("Esperando Ingreso...")
                            except RuntimeError:
                                pass
                            if( intruderD.areThereIntruders ):
                                intruderD.disable_interrupt()
                        intruderD.togglesLamp()
                    else:
                        try:
                            print("El sistema esta apagado, Ingrese \
                                al apartado de configuracion  de la \
                                aplicacion para encenderlo."
                            )
                            intruderD.thereArentIntruders()
                        except RuntimeError:
                           wiegand, sim, camara, intruderD, service =
sistemRestart()
                else:
                    intruderD.thereArentIntruders()
            except google.api_core.exceptions.Unknown:
                wiegand, sim, camara, intruderD, service = sistemRestart()
        except NameError:
            wiegand, sim, camara, IntruderD, service = sistemRestart()
```

Fuente: Propia

ANEXO 20. Comunicación con la DB desde Asbama Security.

Figura 80. Importación de módulos necesarios.

```
1  import firebase_admin
2  from firebase_admin import credentials
3  from firebase_admin import firestore
4  import uuid
5  from datetime import datetime
```

Fuente: Propia

Figura 81. Modelo de usuario asociado a un teléfono.

```
class PhoneModel():
    def __init__(self, user,telefono, role, update, required):
        self.user = user
        self.telefono = telefono
        self.role = role
        self.update = update
        self.required = required

```

Fuente: Propia

Figura 82. Constructor de la clase que maneja el servicio.

```
class Service:
    def __init__(self):
        if (not len(firebase_admin._apps)):
            credential = credentials.ApplicationDefault()
            firebase_admin.initialize_app(credential, {
                'projectId': 'asbama314',
            })
        self.db = firestore.client()

```

Fuente: Propia

Figura 83. Método para obtener la configuración del sistema.

```
def getConfiguration(self):
    return self.db.collection('configuracion').document('configuraciongeneral').get()
```

Fuente: Propia

Figura 84. Validación del identificador obtenido con el Wiegand y los guardados en los usuarios de la base de datos.

```
def wiegandIdValidate(self, rfid):
    users = self.db.collection('users').where('access', '==', str(rfid)).get()
    users = [ user for user in users ]
    validate = True if (len(users) > 0) and (len(users) == 1) else False
    if( validate ):
        for user in users:
            return validate, user
    else:
        return validate, 'undefined'
```

Fuente: Propia

Figura 85. Creación de registro a un usuario específico.

```
def createRegister(self, description, userId, date ):
    registerId = self.getNewId()
    accessRef = self.db.collection('users').document(userId).collection('registers').document(str(registerId))
    information = {
        'descripcion': description,
        'user': userId,
        'fechadecreacion': date
    }
    accessRef.set(information)
    accessGlobalRef = self.db.collection('register').document(str(registerId))
    accessGlobalRef.set(information)
```

Fuente: Propia

Figura 86. Creación de registro para un intruso.

```
def undefinedRegister(self, description, undefinedId, date):
    registerId = self.getNewId()
    accessGlobalRef = self.db.collection('register').document(str(registerId))
    accessGlobalRef.set({
        'descripcion': description,
        'user': undefinedId,
        'fechadecreacion': date
    })
```

Fuente: Propia

Figura 87. Obtener usuario por ID.

```
def getUserById(self, userId):
    return self.db.collection('users').document(userId).get()
```

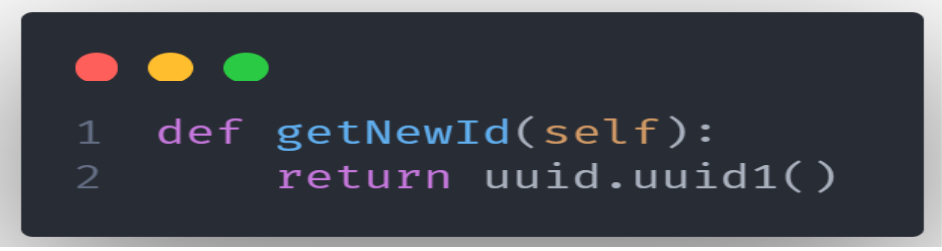
Fuente: Propia

Figura 88. Obtener Teléfonos.

```
def getPhones( self ):
    phones = self.db.collection('telefonos').get()
    phoneTemplateData = list()
    for phone in phones:
        user = self.getUserById(phone.to_dict()['user'])
        model = PhoneModel(user, phone.to_dict()['telefono'], user.to_dict()['role'], phone.to_dict()['fechadeactualizacion'], phone.to_dict()['requerido'])
        phoneTemplateData.append(model)
    return phoneTemplateData
```

Fuente: Propia

Figura 89. Generar un identificador.



```
def getNewId(self):
    return uuid.uuid1()
```

Fuente: Propia