



FACULTAD DE INGENIERIA

Universidad de Buenos Aires

CARRERA DE ESPECIALIZACIÓN EN INTERNET DE LAS COSAS

MEMORIA DEL TRABAJO FINAL

Sistema de gestión de alertas y tareas de procesos de planta - Control de acceso

Autor:
Ing. Lionel Gutierrez

Director:
Ing. Gustavo Ramoscelli (UNS)

Jurados:
Ing. José Alamos (HAW Hamburg)
Mg. Ing. Leandro Lanzieri Rodriguez (UTN FRA/HAW Hamburg)
Esp. Lic. Leopoldo Zimperz (UBA)

*Este trabajo fue realizado en la ciudad de Villa Mercedes,
entre mayo de 2020 y marzo de 2021.*

Resumen

La presente memoria describe el diseño e implementación de un sistema de control de acceso de personal de terceros a una locación industrial. El sistema garantiza que solo aquellas personas que tienen en regla los requisitos legales y médicos solicitados accedan, evitando que la empresa sea responsable ante posibles accidentes o incidentes de dicho personal. El trabajo desarrollado es la primera etapa de un proyecto integral de gestión de alertas y procesos para la empresa Tenaris Metalmecánica, sobre el cual se agregarán a futuro nuevos casos de uso.

Para la elaboración del trabajo se aplicaron conocimientos adquiridos a lo largo de la carrera, principalmente los referidos a gestión de proyectos, desarrollo de aplicaciones web y multiplataforma, protocolos de Internet y seguridad en IoT.

Además, se integraron tecnologías de base de datos relacionales y no relacionales y se aplicaron varias técnicas de testing.

Agradecimientos

Especialmente a mi compañera Belén, que me ha acompañado y apoyado a lo largo del proyecto.

A los profesores de la Carrera de Especialización por compartir sus conocimientos y estar siempre a disposición para ayudar.

Índice general

Resumen	I
1. Introducción general	1
1.1. Estado del arte	1
1.1.1. Tecnología IoT	1
1.1.2. Control de acceso	1
1.2. Motivación	1
1.3. Objetivos y alcance	1
2. Introducción específica	3
2.1. Protocolos de comunicación	3
2.1.1. Tecnología de comunicación Wi-Fi	3
2.1.2. Protocolo HTTP	3
2.2. Componentes de hardware utilizados	3
2.2.1. Módulo ESP32	3
2.2.2. Módulo RFID RC522	3
2.2.3. Cerradura electrónica	3
2.3. Tecnologías de software aplicadas	4
2.3.1. Node.JS	4
2.3.2. Ionic	4
2.3.3. PostgreSQL	4
2.3.4. MongoDB	4
2.3.5. Docker	4
2.3.6. Postman	4
2.4. Software de control de versiones	4
2.4.1. GitFlow	4
2.5. Requerimientos	4
2.5.1. Requerimientos funcionales	4
2.5.2. Requerimientos no funcionales	5
2.5.3. Requerimientos de documentación	5
2.5.4. Requerimientos de validación	5
3. Diseño e implementación	7
3.1. Arquitectura del sistema	7
3.1.1. Módulos del sistema	8
3.1.2. Protocolos de comunicación entre módulos	8
3.1.3. Tecnologías de bases de datos	9
3.1.4. Contenedores docker y escalamiento	9
3.2. Detalle de módulos de hardware	10
3.2.1. Módulo sensor	10
Configuraciones y variables del módulo	10
Comunicación con el backend	11
Leds del sistema	12

3.2.2. Módulo actuador	12
Configuraciones y variables del módulo	14
Comunicación desde el backend	15
Detalle de respuestas ante solicitudes del backend	15
3.3. Detalle de módulos de software	16
3.3.1. Módulo de backend	16
API de autenticación	19
Funcionamiento del módulo ante una solicitud del módulo sensor	19
Funcionamiento del módulo ante una solicitud del módulo de frontend	20
3.3.2. Módulo de frontend	21
Detalle de servicios (“services”) implementados	24
Funcionamiento del módulo ante un inicio de sesión	25
Funcionamiento del módulo ante una solicitud de usuario	26
Pantalla principal de vigilancia	28
3.4. Interfaz con sistema de documentación	28
4. Ensayos y resultados	31
4.1. Detalle de pruebas realizadas	31
4.1.1. Herramientas utilizadas	32
4.1.2. Mocks implementados	33
4.2. Pruebas unitarias	35
4.2.1. Testing del módulo sensor	35
4.2.2. Testing del módulo actuador	37
4.2.3. Testing del módulo de backend	39
Tests de la carpeta Test API	41
Detalle de tests y resultados	43
4.3. Pruebas de sistema	45
4.4. Pruebas de aceptación	47
4.4.1. Descripción y detalles de prueba de ingreso habilitado	47
4.4.2. Descripción y detalles de prueba de ingreso inhabilitado	50
4.5. Comparativa con otras soluciones del mercado	53
5. Conclusiones	55
5.1. Resultados obtenidos	55
5.2. Trabajo futuro	56
Bibliografía	59

Índice de figuras

3.1. Diagrama en bloques del sistema implementado.	7
3.2. Módulo sensor junto a sus componentes.	11
3.3. Módulo actuador junto a sus componentes.	14
3.4. Estructura de directorio del backend desarrollado en Visual Studio Code.	17
3.5. Interacción entre los componentes del módulo ante una solicitud del módulo sensor.	20
3.6. Interacción entre los componentes del módulo ante una solicitud del frontend.	21
3.7. Estructura de directorio del frontend desarrollado en Visual Studio Code.	22
3.8. Diagrama de interacción para el inicio de sesión.	26
3.9. Diagrama de interacción para una solicitud de usuario.	27
4.1. Herramienta Postman junto a su configuración básica.	33
4.2. Configuración del script para testing automático de pruebas unitarias.	34
4.3. Ejecución y resultados de ejecución del script automático de testing.	34
4.4. Procedimiento de prueba para el caso de test 1 con tarjeta sin valor configurado.	36
4.5. Procedimiento de prueba para el caso de test 2 con tarjeta con valor configurado y recibido correctamente por el backend.	36
4.6. Configuración en Postman para el testeo del módulo actuador.	37
4.7. Configuración de sección “Tests” y resultados de la ejecución del test.	38
4.8. Configuración en Postman para el testeo del módulo de backend.	40
4.9. Variables globales definidas en Postman.	41
4.10. Tests de la subcarpeta Con token autenticación – permiso gerente.	42
4.11. Accionamiento de módulo sensor con tarjeta RFID y respuesta del módulo.	48
4.12. Visualización de la alerta recibida en pantalla.	49
4.13. Accionamiento de módulo actuador y respuesta del módulo.	49
4.14. Visualización de la alerta recibida en pantalla.	51
4.15. Accionamiento de módulo actuador y respuesta del módulo.	51
4.16. Email recibido por el usuario con la alerta de intento de ingreso con documentación vencida.	52
4.17. Pantalla de usuario se sector HESA con la tarea de control generada.	52

Índice de tablas

3.1. Protocolos comunicación	8
3.2. Leds módulo sensor	13
3.3. Respuestas backend	16
3.4. Carpetas backend	18
3.5. Carpetas frontend	23
4.1. Tipos de pruebas	32
4.2. Tipos de pruebas sensor	35
4.3. Tipos de pruebas actuador	39
4.4. Tipos de pruebas backend	43
4.5. Tipos de pruebas backend	43
4.6. Tipos de pruebas backend	44
4.7. Tipos de pruebas sistema	45
4.8. Tipos de pruebas sistema	46
4.9. Tipos de pruebas sistema	46
4.10. Tipos de pruebas sistema	47
4.11. Comparación soluciones	53

Capítulo 1

Introducción general

Poner párrafo introductorio.

1.1. Estado del arte

Introducción, propósito y estado del arte de IoT y solución propuesta.

1.1.1. Tecnología IoT

Introducción a las soluciones IoT, posibilidades que brinda para sensado y control de diferentes procesos, ventajas de la tecnología (economía, simplicidad).

1.1.2. Control de acceso

Sistemas de control de acceso que existen en el mercado. Diferencias con el sistema propuesto (valor agregado de la propuesta contra soluciones existentes).

1.2. Motivación

Razones por el cual se desea desarrollar el sistema. Justificaciones. Necesidad del cliente.

1.3. Objetivos y alcance

Objetivos y alcance del trabajo.

Capítulo 2

Introducción específica

Poner párrafo introductorio.

2.1. Protocolos de comunicación

Descripción de los protocolo de comunicación (Wi-Fi/HTTP) utilizados para IoT.

2.1.1. Tecnología de comunicación Wi-Fi

Descripción de tecnología Wi-Fi.

2.1.2. Protocolo HTTP

Descripción de protocolo HTTP.

2.2. Componentes de hardware utilizados

Descripción de los componentes de hardware utilizados: ESP32, lector de tarjetas, cerradura electrónica.

2.2.1. Módulo ESP32

Descripción del módulo.

2.2.2. Módulo RFID RC522

Descripción del módulo.

2.2.3. Cerradura electrónica

Descripción de cerradura.

2.3. Tecnologías de software aplicadas

Descripción de las tecnologías de software utilizadas.

2.3.1. Node.JS

2.3.2. Ionic

2.3.3. PostgreSQL

2.3.4. MongoDB

2.3.5. Docker

2.3.6. Postman

2.4. Software de control de versiones

Descripción del software de control de versiones.

2.4.1. GitFlow

Descripción de la herramienta.

2.5. Requerimientos

Requerimientos del proyecto, tanto funcionales, no funcionales, de documentación y de validación. Enumeración de los mismos.

2.5.1. Requerimientos funcionales

- 1.1 El sistema debe permitir el sensado de datos de distintas fuentes y procesos de planta.
- 1.2 El sistema deberá generar alertas a usuarios finales ante problemas detectados del sensado o situaciones límites/problems potenciales.
- 1.3 El sistema deberá generar tareas de corrección y prevención con un circuito de estados que permita trazar el origen del problema y la solución asociada.
- 1.4 El sistema debe permitir hacer un seguimiento de la cantidad de alarmas diarias y mensuales generadas.
- 1.5 El sistema debe permitir hacer un seguimiento de la cantidad de tareas diarias y mensuales generadas A su vez, se deberá poder ver la cantidad de tareas cerradas, en curso y su antigüedad en días.

- 1.6 El sistema debe permitir gestionar usuarios. La gestión de usuarios incluye dar de alta nuevos usuarios, gestionar la recuperación y cambio de clave de los mismos. Dicho usuario se utilizará para acceder y utilizar el sistema.

2.5.2. Requerimientos no funcionales

- 2.1 El sistema deberá ser escalable, de forma de poder agregar más módulos actuadores y de sensado para los procesos de planta a futuro.
- 2.2 El sistema deberá ser recuperable ante problemas de hardware o software, de forma de asegurar la disponibilidad y no corrupción de la información, cumpliendo con la política de resguardo de datos de la empresa.
- 2.3 El sistema deberá poder operarse aún ante cortes puntuales de energía en algunas áreas, esto es, ante corte que no sean generales de toda la planta. Para ello se deberá contar con una política de suministro alternativo de energía para los servidores donde se ejecute el software.

2.5.3. Requerimientos de documentación

- 3.1 Se debe generar una Memoria Técnica con la documentación de ingeniería detallada.
- 3.2 Se debe generar un documento de casos de prueba.
- 3.3 Se debe generar un documento de la Infraestructura del sistema y de la configuración por ambiente y pasaje entre ambientes.
- 3.4 Se deberá generar la documentación del sistema y del proyecto en el sistema de aprobación y documentación TPA de la empresa.

2.5.4. Requerimientos de validación

- 4.1 Se deberá tener una matriz de trazabilidad entre los casos de uso y los casos de prueba, validando el cumplimiento de cada uno y con la aprobación final del auspiciante.

Capítulo 3

Diseño e implementación

En el presente capítulo se describe la arquitectura del sistema, el diseño y la implementación del hardware y del software y las herramientas de desarrollo utilizadas.

3.1. Arquitectura del sistema

En esta sección se explica la arquitectura propuesta, junto a los módulos implementados y los protocolos de comunicación utilizados para la conexión de los mismos. También se expone como se piensa lograr la escalabilidad del sistema.

En la figura 3.1 se muestra el diagrama en bloques del sistema, junto a los módulos, las tecnologías utilizadas y los protocolos de comunicación que los conectan.

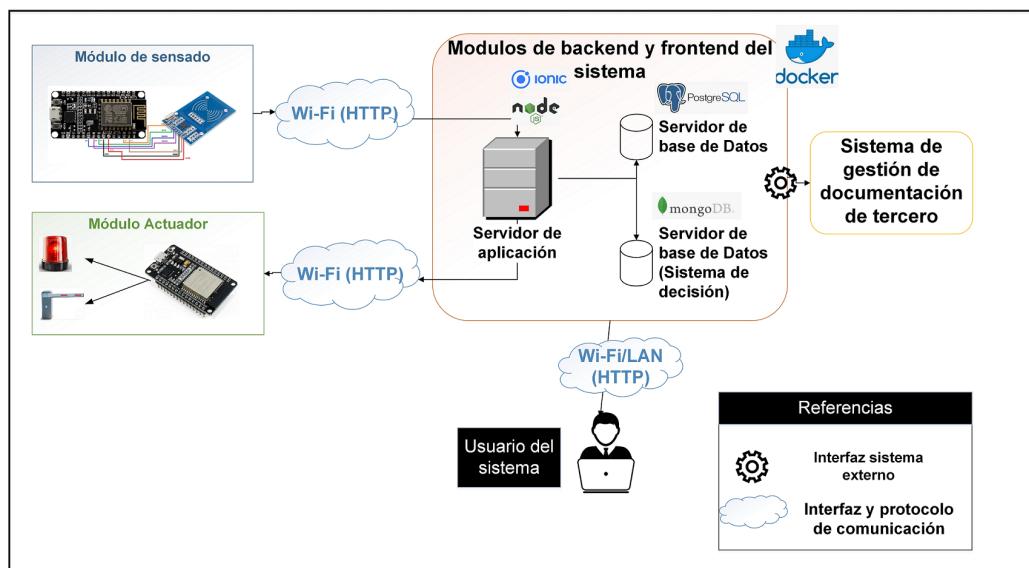


FIGURA 3.1. Diagrama en bloques del sistema implementado.

3.1.1. Módulos del sistema

El trabajo desarrollado se divide en los siguientes módulos:

- Módulo de sensado: es el encargado de leer la tarjeta RFID del personal de tercero que quiere ingresar a la planta y enviar la información al módulo de backend para analizar si la persona cumple con los requisitos para acceder o no.
- Módulo actuador: es el encargado de comandar la cerradura electrónica que permite o evita el ingreso del tercero a la locación industrial. El mismo recibe las órdenes de cómo operar desde el módulo de backend.
- Módulos de backend y frontend: si bien estos módulos están implementados de manera conjunta en un único servidor de aplicación, cumplen funciones diferentes:
 - Módulo de frontend: es el encargado de brindar una interfaz gráfica a los usuarios, para que estos puedan interactuar con la aplicación, recibiendo sus solicitudes y proporcionándoles la información en un formato simple.
 - Módulo de backend: es quien gestiona todas las solicitudes provenientes del módulo de sensado y del módulo de frontend. Es el encargado de analizar las solicitudes y responder a las mismas. Cuando recibe peticiones del frontend responde al mismo. Cuando recibe peticiones del módulo de sensado, actúa enviándole órdenes o comandos al módulo actuador. También es el encargado de comunicarse con el sistema de gestión de documentación de terceros. Para cumplir con sus funciones utiliza información almacenada en los servidores de base de datos.
- Sistema de gestión de documentación de terceros: este sistema es externo y no fue parte del desarrollo. El módulo de backend utiliza el mismo para obtener información de los terceros y en base a ésta tomar las decisiones de habilitar o inhabilitar el acceso y generar alertas y tareas de control.

3.1.2. Protocolos de comunicación entre módulos

Para la comunicación entre los módulos se utilizaron invocaciones HTTP GET y POST. Tomando como referencia el modelo TPC/IP [1], en la tabla 3.1 se muestra el detalle de protocolos empleados en cada capa:

TABLA 3.1. Protocolos de comunicación empleados por el sistema.

Capa del modelo	Protocolo
Aplicación	HTTP (utilizando los verbos GET y POST)
Transporte	TCP
Internet	IP (IPv4)
Acceso al medio	Wi-Fi (802.11n) para la comunicación entre módulos. Wi-Fi o IEEE 802.3 (Ethernet) para la comunicación entre el usuario y el módulo de frontend.

Para la elección de los protocolos, se tomó en cuenta las tecnologías disponibles en la empresa. Además, al utilizar protocolos abiertos, estándares y extendidos mundialmente, se logró un sistema portable y adaptable.

3.1.3. Tecnologías de bases de datos

El sistema en general y el módulo de backend en particular, se soporta en dos bases de datos:

- Una base de datos relacional, implementada en PostgreSQL, que es la que contiene todos los objetos necesarios para la aplicación: usuarios, sensores, actuadores, terceros, eventos del sistema, tareas y sub-tareas de control.
- Una base de datos no relacional, implementada en MongoDB, que es utilizada por el backend para almacenar la relación entre los eventos de entrada y el conjunto de acciones que se deben tomar en función de dichos eventos. La decisión de utilizar una base no relacional se debe a que cada tipo de evento de entrada genera diferentes tipos de acciones de salida. Por ejemplo, en el caso de un evento de ingreso de un tercero con documentación en regla solo se debe realizar una acción de apertura de cerradura para el módulo actuador. Pero para un evento de ingreso con documentación vencida se deben generar acciones para cerrar la cerradura en el módulo actuador, generar tareas de control para diferentes sectores de planta y enviar un mail a las personas definidas por la gerencia de la empresa.

3.1.4. Contenedores docker y escalamiento

A fin de generar una solución escalable y modular, se utilizaron contenedores docker para implementar el módulo de backend, el módulo de frontend y para levantar las instancias de base de datos, tanto PostgreSQL como MongoDB. Esta decisión permitió:

- Simplificar el *deploy* de la aplicación: facilitando la configuración del servidor o servidores donde se ejecuta el sistema.
- Lograr la escalabilidad futura de la solución: al permitir utilizar un orquestador de contenedores como Kubernetes que permite crear o eliminar instancias de cada contenedor dinámicamente en función de diferentes variables, como el consumo de recursos o la cantidad de solicitudes por segundo. Una ventaja adicional es que, si migramos la solución a la nube, al utilizar este esquema de contenedores dinámicos podemos reducir el costo del servicio, dado que estaremos pagando solo por los contenedores que necesitamos en cada instante de tiempo, sin necesidad de tener un número fijo de recursos en todo momento.

3.2. Detalle de módulos de hardware

En esta sección se describe detalladamente la implementación de los dos módulos de hardware desarrollados en el proyecto. El módulo sensor, encargado de la lectura de las tarjetas RFID del personal de tercero, y el módulo actuador, encargado de gestionar la cerradura electrónica para permitir o evitar el ingreso de dicho personal.

3.2.1. Módulo sensor

Es el encargado de leer las tarjetas RFID del tercero y enviar el valor que tiene la misma al módulo de backend.

Cada tarjeta RFID tiene un valor numérico guardado de 4 caracteres de longitud. Las tarjetas permiten definir valores de hasta 16 caracteres, pero dado que la empresa utiliza códigos de 4 caracteres se colocó ese límite para tener uniformidad.

El módulo está compuesto por los siguientes componentes:

- Un lector de tarjetas RFID RC522. La elección del mismo se debió a su bajo costo, alta disponibilidad en el mercado y su capacidad para leer las tarjetas que tiene la empresa, que operan en la frecuencia de 13,56 MHz.
- Un *SoC* (System on a chip) ESP32-WRROM-32. La elección del mismo se debió a su bajo costo, alta disponibilidad en el mercado, facilidad de programación y soporte de redes Wi-Fi (normas 802.11 b/g/n). Esto último simplifica la comunicación del módulo con el backend y evita tener que conectarse a la red LAN de la empresa, lo que hubiera requerido hacer una extensión del cableado de la misma.
- Un conjunto de leds, que permite al usuario conocer el estado del sistema y el estado de sus interacciones con el módulo. Para ello se dispuso un grupo de 3 leds generales de control y otro de 3 leds de respuesta ante las comunicaciones con el backend.

En la figura 3.2 se muestra el módulo sensor junto a sus componentes.

Configuraciones y variables del módulo

Para implementar este módulo, se desarrolló un programa en el entorno Arduino IDE. El mismo lee las tarjetas RFID y se comunica con el backend, enviando los datos requeridos para procesar el intento de ingreso. Dicha comunicación se realiza a través de la red Wi-Fi de la empresa.

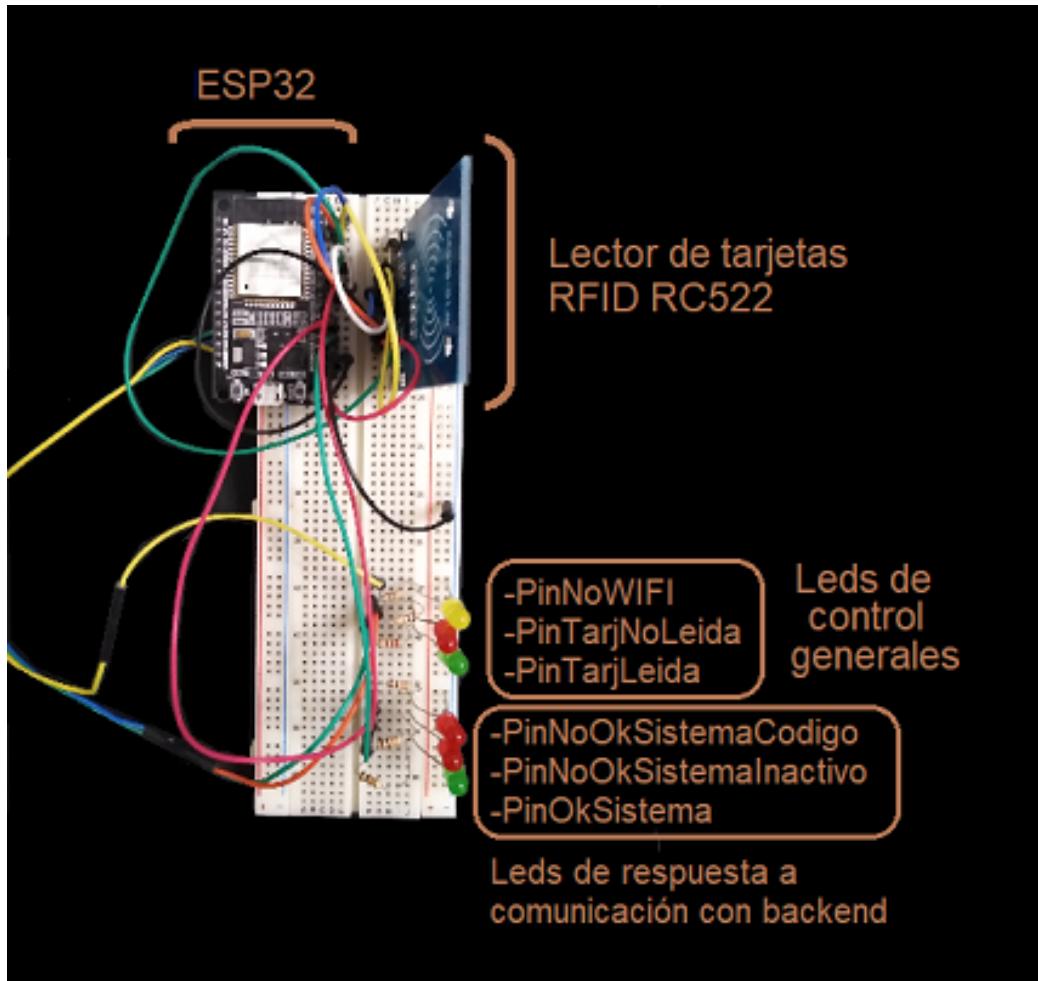


FIGURA 3.2. Módulo sensor junto a sus componentes.

El módulo cuenta con un conjunto de variables a configurar para su correcta operación:

- WIFI_SSID: especifica el SSID de la red Wi-Fi de la empresa.
- WIFI_PASSWORD: especifica el password de la red Wi-Fi de la empresa.
- ID_SENSOR: especifica el ID que tiene el sensor en la base de datos del sistema. Los datos de cada sensor se almacenan en dicha base de datos, la cual incluye su estado, descripción, ubicación y token asociado.
- tokenlocal: especifica el token de 20 caracteres que tiene asociado el sensor en la base de datos. Con este valor se asegura la autenticación del módulo.
- servicioAPISensor: contiene la URL del endpoint que expone el backend para recibir los datos de este módulo.

Comunicación con el backend

El módulo tiene configurada la dirección URL del endpoint que el backend expone para permitir la comunicación entre éstos.

Para enviar los datos solicitados, se realiza un HTTP POST con un objeto JSON que tiene 3 claves:

- id: contiene el valor “ID_SENSOR” del módulo.
- token: contiene el valor de “tokenlocal” del módulo.
- valor: contiene el valor leído de la tarjeta RFID, que representa al id del tercero en el sistema.

Una vez enviado el HTTP POST, el módulo recibe como respuesta un valor que indica si los datos mandados son correctos o si hubo algún error. Con esta respuesta se determina qué leds deben activarse para dar *feedback* al usuario del estado del proceso.

Leds del sistema

El módulo cuenta con un conjunto de leds, que permiten al usuario conocer el estado del mismo y el resultado de sus interacciones con éste.

Cuando el mismo se inicializa hace un chequeo de estos leds, prendiéndolos y apagándolos, uno a uno, durante medio segundo.

En la tabla 3.2 se muestra el detalle de los leds o combinaciones posibles de leds, junto a la información que brindan al usuario cuando se encienden.

3.2.2. Módulo actuador

Es el encargado de comandar la cerradura. El mismo cuenta con un conjunto de leds que brindan información al personal de tercero del estado del módulo y del estado de su ingreso.

Las órdenes de cómo operar las recibe desde el módulo de backend, para lo cual el actuador expone un *endpoint* HTTP, que recibe un JSON con dichas órdenes.

TABLA 3.2. Combinación de leds e información para el usuario cuando se prenden.

Led/combinación de leds	Información para el usuario
PinNoWIFI	Al leer la tarjeta del tercero, si no se cuenta con comunicación Wi-Fi con el backend, el led se prende durante 3 segundos.
PinTarjNoLeida	Falló la lectura de la tarjeta o la misma no tiene valor asignado. Se debe configurar la tarjeta con el valor correspondiente al personal de tercero.
PinTarjLeida	Al acercar la tarjeta al lector, el sistema lee correctamente la misma, junto al valor que tiene almacenado.
PinOkSistema	Una vez leída la tarjeta del personal de tercero, el sistema se comunica correctamente con el backend. Se informa al usuario al prender el led durante 2 segundos.
PinNoOkSistemaCodigo	Una vez leída la tarjeta del personal de tercero, el sistema se comunica correctamente con el backend, pero el valor de "ID_SENSOR" enviado no se corresponde con ningún módulo sensor configurado en el sistema, o el mismo está inactivo. Se informa al usuario al prender el led durante 2 segundos.
PinNoOkSistemaCodigo + PinNoOkSistemaInactivo	Una vez leída la tarjeta del personal de tercero, el sistema se comunica correctamente con el backend, pero éste devuelve un error con un código no especificado. Se informa al usuario al prender el "led PinNoOkSistemaCodigo" durante 1 segundo seguido del led "PinNoOkSistemaInactivo" durante otro segundo.

El módulo está compuesto por los siguientes componentes:

- Un SoC ESP32-WRROM-32. La elección del mismo se debió a su bajo costo, alta disponibilidad en el mercado, facilidad de programación y soporte de redes Wi-Fi (normas 802.11 b/g/n). Esto último simplifica la comunicación del módulo con el backend y evita tener que conectarse a la red LAN de la empresa, lo que hubiera requerido hacer una extensión del cableado de la misma.
- Un regulador de tensión, que brinda los niveles de tensión requeridos para energizar el ESP32 y para la activación del Mosfet IRF520.
- Cerradura electrónica. La misma se acciona y alimenta desde el Mosfet IRF520.
- Conversor de niveles lógicos. Se utiliza para convertir la tensión de salida del ESP32 (3.3 V) a la tensión requerida para accionar el Mosfet IFR520 (5V).

- Mosfet IRF520: permite accionar la cerradura electrónica, brindando el nivel de tensión requerida por la misma (12 V).
- Un conjunto de leds, que permiten conocer si el actuador está encendido y el estado del ingreso del tercero (habilitado/inhabilitado/error).
- Fuente de alimentación de 12 V, que es utilizada para alimentar el Mosfet IRF520 y al regulador de tensión.

En la figura 3.3 se muestra el módulo actuador con sus componentes.

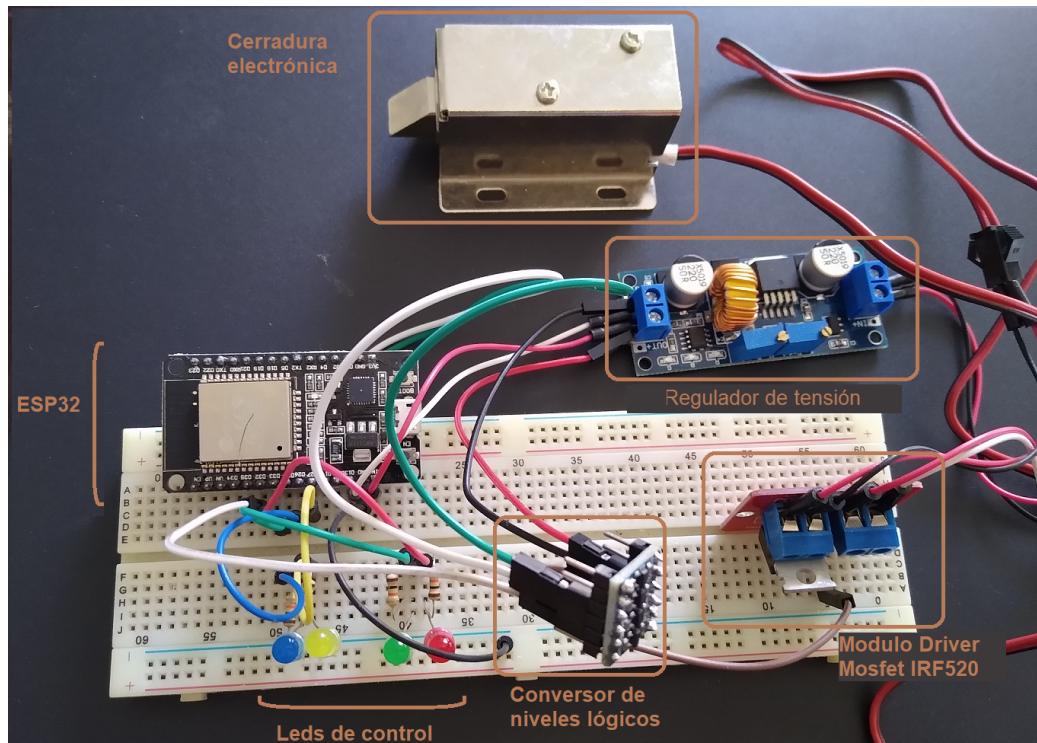


FIGURA 3.3. Módulo actuador junto a sus componentes.

Configuraciones y variables del módulo

Para implementar el actuador, se desarrolló un programa en el entorno Arduino IDE. Este programa expone un *endpoint* HTTP POST, que recibe un JSON con las acciones a realizar. En función de la acción y valor indicados, se acciona la cerradura electrónica y se prenden los leds de control.

En la base de datos del sistema se guarda la información de los actuadores existentes, su ubicación, descripción, estado, dirección IP y token de autenticación.

El actuador cuenta con un conjunto de variables a configurar para su correcta operación:

- **WIFI_SSID**: especifica el SSID de la red Wi-Fi de la empresa.
- **WIFI_PASSWORD**: especifica el password de la red Wi-Fi de la empresa.
- **tokenlocal**: especifica el token de 20 caracteres que tiene asociado el actuador en la base de datos. Con este valor se asegura la autenticación del módulo.

- local_IP: especifica la dirección IP del mismo. Se utiliza una IP fija, para asegurar que el *endpoint* expuesto siempre pueda ser accedido. Si se utilizara una IP dinámica la dirección podría cambiar y quedaría inaccesible dicho *endpoint*.
- Subnet: dirección de sub-red de la red Wi-Fi.

Comunicación desde el backend

El módulo recibe un HTTP POST desde el backend con un objeto JSON que tiene 3 claves:

- token: contiene el token de autenticación.
- acción: contiene la acción a realizar. Es un clasificador de acciones posibles.
- valor: contiene el valor particular para la acción.

Al recibir el objeto se controla si el token coincide con el valor de token que se tiene almacenado localmente, y luego se controla si la acción y valor son válidos. En función de la acción y valor, se abre o cierra la cerradura, y prende el led de ingreso ok, de ingreso no ok o de error. Por último, se responde al backend con un código de error o un ok.

Detalle de respuestas ante solicitudes del backend

El backend realiza solicitudes al actuador como se explica en la sub-sección anterior. En la tabla 3.3 se muestra el detalle las diferentes combinaciones de valores que puede recibir el módulo en las solicitudes y se indica la respuesta brindada al usuario y al backend.

TABLA 3.3. Respuestas posibles del módulo al usuario y al backend ante las solicitudes recibidas.

Valores recibidos	Respuesta al usuario	Respuesta al backend
Acción=“APERTURA” Valor=“ABRIR” Token con valor correcto.	Se prende el led verde de manera intermitente durante 4 segundos. Durante ese tiempo la cerradura electrónica se cierra.	Se envía respuesta HTTP con código 200 y mensaje OK.
Acción=“APERTURA” Valor=“CERRAR” Token con valor correcto. Sin token.	Se prende el led rojo durante 2 segundos. Se prende el led amarillo durante medio segundo y se apaga.	Se envía respuesta HTTP con código 200 y mensaje “Sin token de autenticación.” Se envía respuesta HTTP con código 401 y mensaje “Sin token de autenticación.”
Token con valor incorrecto.	Se prende el led amarillo durante medio segundo y se apaga.	Se envía respuesta HTTP con código 403 y mensaje “Token de autenticación incorrecto.”
Acción no especificada o con valor incorrecto.	Se prende el led amarillo de manera intermitente durante 2 segundos.	Se envía respuesta HTTP con código 400 y mensaje “La acción especificada no es válida.”
Valor no especificado o valor incorrecto.	Se prende el led amarillo de manera intermitente durante 3 segundos.	Se envía respuesta HTTP con código 400 y mensaje “El valor especificado no es válido”.

3.3. Detalle de módulos de software

En esta sección se describe detalladamente la implementación de los dos módulos de software desarrollados en el proyecto: el de backend, encargado tanto de recibir las solicitudes del módulo sensor y de frontend como de enviar comandos al actuador, y el módulo de frontend, encargado de gestionar las solicitudes del usuario mediante una interfaz gráfica.

3.3.1. Módulo de backend

El mismo está implementado como una aplicación web con NodeJS, utiliza las librerías Express y Socket.io y expone:

- Una API Rest para el frontend, la cual responde a sus solicitudes e incluye un WebSocket para mostrar alertas online a la Portería.
- Una API de autenticación, utilizada para la gestión e inicio de sesión de los usuarios.
- Un *endpoint* para recibir las solicitudes de ingreso del módulo sensor.

Para su desarrollo se utilizó el IDE Visual Studio Code. Dentro del mismo se organizaron las carpetas con el código y las configuraciones para el testing automático.

En la figura 3.4 se muestra la estructura en carpetas definidas para el backend.

The screenshot shows the Visual Studio Code interface. On the left is the 'Explorador' (File Explorer) sidebar with a tree view of the project structure. The root folder is 'PROYECTOFINALESPECIALIZACION'. Under it, there's a 'backend' folder containing subfolders like 'apiDatos', 'config', 'controllers', 'DAL', 'database', 'logger', 'logs', 'mainController', 'middleware', 'node_modules', 'routes', 'sistDecision', and 'test'. There are also files: 'docker-compose.yml', 'funcionesEstandar.js', 'index.js', 'package-lock.json', and 'package.json'. A file icon with the number '176' is visible in the sidebar. The main editor area on the right shows the 'index.js' file with the following code:

```
lionelgutierrez, 25 days ago | 1 author (lionelgutierrez)
1 var express = require("express");
2 var bodyParser = require("body-parser");
3 var app = express();
4 var PORT = 4000;
5 var cors = require('cors');
6
7 var corsConfig={
8   |   origin:'*',
9   |   optionsSuccessStatus:200
10  };
11
12 app.use(cors(corsConfig));
13
14 //Configuro conexion de socket
15 var http = require('http');
16 var server = http.createServer(app);
17 var io = require('socket.io')(server, {
18   |   cors: {
19   |     |   origin: '*',
20   |   }
21});
```

FIGURA 3.4. Estructura de directorio del backend desarrollado en Visual Studio Code.

En la tabla 3.4 se expone detalladamente el contenido y función de cada una de las carpetas y archivos del módulo.

TABLA 3.4. Detalle de archivos y carpetas del módulo de backend.

Archivo/ Directorio	Descripción
apiDatos	Contiene cada uno de los <i>endpoints</i> expuestos al frontend. Se implementó utilizando Express, junto a métodos GET y POST, para servir las consultas de información y las altas y actualizaciones en la base de datos.
config	Contiene tanto la configuración del <i>secret</i> necesaria para el submódulo de autenticación, como la configuración de constantes utilizadas en la comunicación con el sistema de documentación de terceros.
controllers	Cuenta con tres controladores: <ul style="list-style-type: none"> ■ generacionAlertas: es el encargado de comunicarse con el módulo actuador y enviarle los comandos para habilitar o prohibir la solicitud de ingreso del tercero a la planta. ■ generacionMensajes: es el encargado de gestionar el envío de emails a los usuarios requeridos. ■ generacionTareas: es el encargado de generar las tareas y sub-tareas, guardando la información en la base de datos.
DAL	La DAL (Data Access Layer), es la encargada de abstraer la comunicación con la base de datos, al brindar un conjunto de métodos para acceder y realizar las altas, bajas y modificaciones, sin necesidad de que los componentes que la usan conozcan la implementación subyacente.
database	Contiene las configuraciones necesarias para conectarse a la base de datos PostgreSQL. Se utiliza un <i>pool</i> de conexiones a fin de mejorar el rendimiento y la escalabilidad del sistema.
logger	Es el encargado de gestionar el <i>logging</i> de eventos.
logs	Almacena los <i>logs</i> del sistema. Se guarda un archivo de <i>log</i> por día para evitar archivos muy extensos y simplificar la búsqueda de información en los mismos.
main Controller	Es el encargado de gestionar las solicitudes de ingreso del módulo sensor. Se comunica con el sistema de documentación de terceros, determina los tipos de acción a realizar y dispara cada una de ellas, invocando a los controladores de la carpeta controllers.
middleware	Contiene los métodos necesarios para la gestión de la autenticación
routes	Contiene cada uno de los <i>endpoints</i> expuestos tanto para la API de autenticación (sub-carpetas “routerAuth” y “routerSensores”) como para la recepción de datos desde el módulo sensor (sub-carpetas “routerSensores”).
sistDecision	Contiene el sub-módulo que se encarga de determinar las acciones a realizar cuando hay una solicitud de ingreso de un tercero, para lo cual utiliza la base de datos implementada en MongoDB.
test	Contiene los archivos necesarios para la ejecución de las pruebas automáticas implementados para la solución. En el capítulo 4 se explican con mayor detalle las pruebas implementadas y los archivos utilizados.
index.js	Contiene la configuración para levantar la aplicación y cada una de las rutas utilizadas por la aplicación. También incluye la configuración de CORS y del WebSocket.

API de autenticación

La API de autenticación se utiliza para segurizar las invocaciones realizadas al backend. Para hacerlo emplea tokens JWT. Además, permite el alta de nuevos usuarios, gestionar el inicio de sesión de los mismos y el cambio y reseteo de passwords.

Para el desarrollo de esta API utilizamos 2 librerías disponibles en node.js: bcryptjs y jsonwebtoken. La primera permite implementar una función de *hash*, que posibilita guardar encriptado el password de los usuarios. La segunda permite generar el token JWT que es entregado al usuario para que pueda acceder a los diferentes *endpoints*, asegurando su autenticidad. Dicho token tiene una duración de 24 horas.

Funcionamiento del módulo ante una solicitud del módulo sensor

Cuando el módulo sensor hace una solicitud al backend invoca al *endpoint* de recepción de sensores. El backend, por su parte, recibe la solicitud y realiza los pasos descriptos a continuación:

1. Envía el pedido al router “routerSensores”.
2. “routerSensores” controla el token y valores recibidos. Si el token no es válido o el id de sensor enviado no es correcto o está inactivo, se envía un mensaje de error al origen y se termina la solicitud.
3. Si los datos son correctos, se envían al “mainController”.
4. El “mainController” realiza estas acciones:
 - a) Se comunica con el sistema de documentación de terceros para determinar si la persona está en condiciones de ingresar.
 - b) Registra el evento de ingreso en la base de datos (no directamente, sino a través de la “DAL”).
 - c) Con los datos obtenidos se comunica con el sub-módulo de decisión (“sistDecision”), el cual le indica las acciones a realizar.
 - d) Para cada una de las acciones indicadas, en función del tipo que sea (de salida, mensaje, tarea), se comunica con los controladores “generacionAlertas”, “generacionMensajes” o “generacionTareas”, para que éstos las procesen y registren.

En la figura 3.5 se muestra la interrelación entre los componentes del módulo y el flujo de datos ante una solicitud desde el módulo sensor.

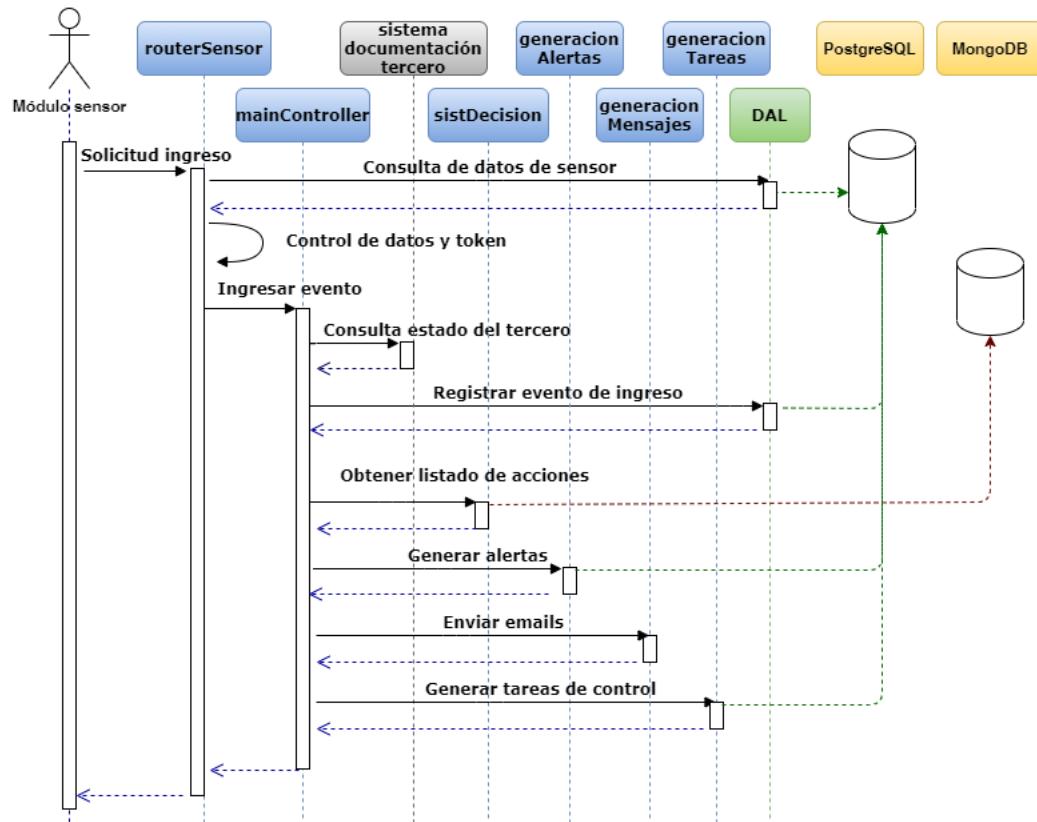


FIGURA 3.5. Interacción entre los componentes del módulo ante una solicitud del módulo sensor.

Funcionamiento del módulo ante una solicitud del módulo de frontend

Cuando el módulo de frontend hace una solicitud al backend invoca algunos de los *endpoints* definidos en “apiDatos”. El backend, por su parte, recibe la solicitud y realiza los pasos descriptos a continuación:

1. Envía el pedido a “apiDatos”.
2. “apiDatos” determina el endpoint solicitado, pasa el control al mismo y éste realiza las siguientes acciones:
 - a) Controla que la solicitud tenga el token de autenticación y lo valida utilizando las funciones del *middleware* de autenticación.
 - b) Si el token no es correcto se rechaza el pedido con un código 403.
 - c) Si el token es correcto, opcionalmente y según la necesidad de cada endpoint, controla el rol de usuario asociado al token utilizando nuevamente el *middleware* de autenticación.
 - d) Si el rol/roles solicitados no son correctos rechaza el pedido con un código 403.
 - e) Si los roles son correctos procede con la solicitud. En general cada solicitud controla los datos de entrada y luego se comunica con la base de datos a través de la “DAL”, ya sea para consultar, agregar o modificar información.

En la figura 3.6 se muestra la interrelación entre los componentes del módulo y el flujo de datos ante una solicitud desde el frontend.

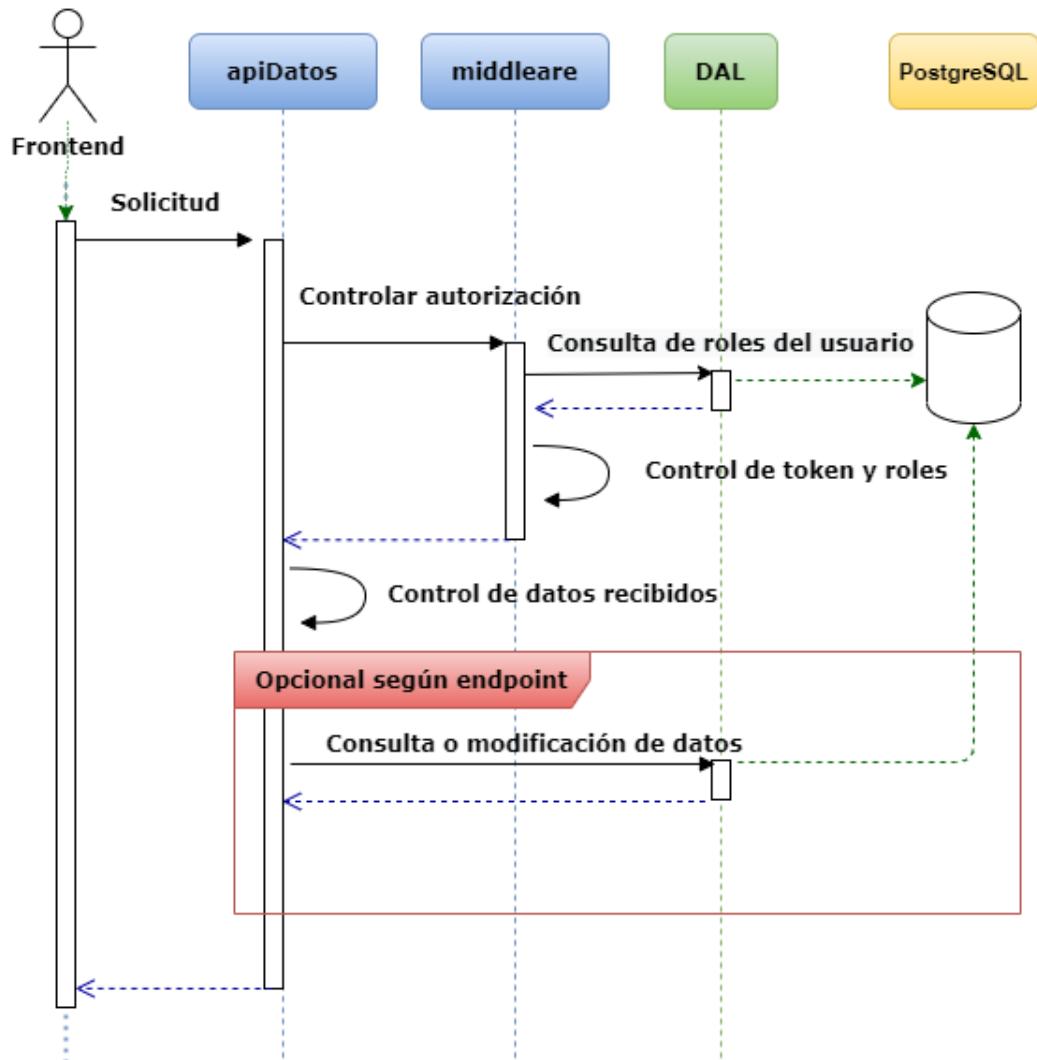
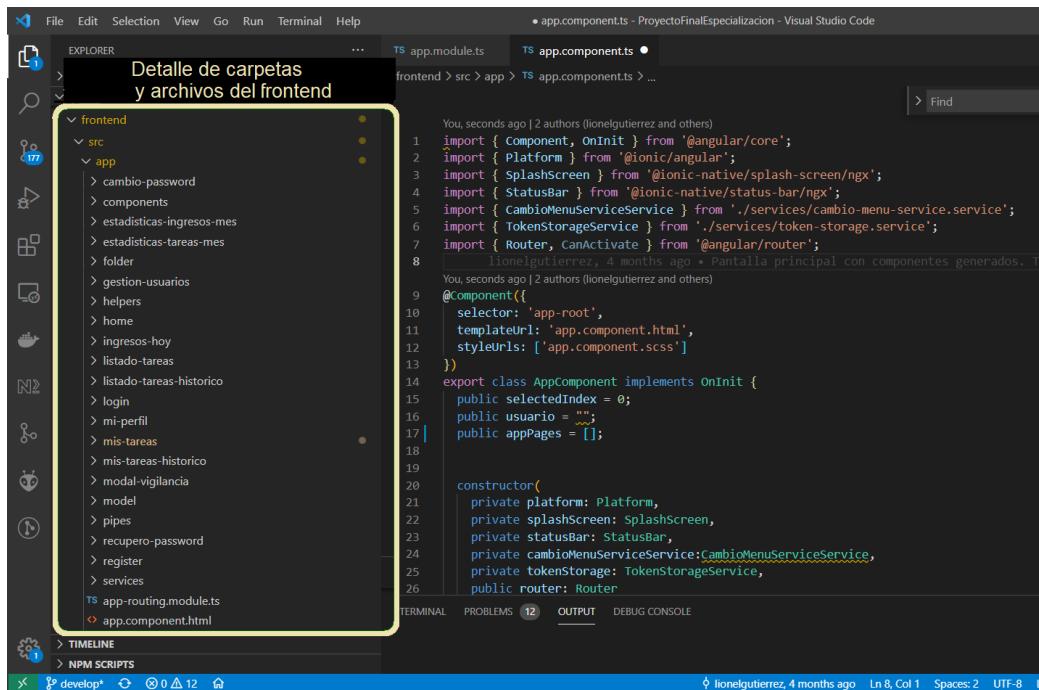


FIGURA 3.6. Interacción entre los componentes del módulo ante una solicitud del frontend.

3.3.2. Módulo de frontend

Este módulo brinda una interfaz gráfica al usuario a través de la cual interactúa con el sistema, ya sea para consultar datos o para registrar acciones. Con el objetivo de cumplir con tales funciones se comunica con el backend a través de una API Rest. Para su desarrollo se empleó Angular y el framework Ionic. Su utilización permitió construir el sistema como una aplicación web responsive con la idea de implementarla a futuro como una *app mobile*. Para la escritura del código fuente apelamos al IDE Visual Studio Code. Dentro del mismo se organizaron las carpetas con el código y cada uno de los diferentes elementos.

En la figura 3.7 se muestra la estructura en carpetas definidas para el frontend.



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "frontend". The "src" folder contains several components like "cambio-password", "components", "estadisticas-ingresos-mes", etc. A yellow box highlights the "src" folder and its contents.
- Code Editor (Right):** Displays the content of the "app.component.ts" file. The code defines an Angular component with properties for platform, splashscreen, statusBar, and services like CambioMenuService and TokenStorage. It also includes a constructor and OnInit methods.
- Bottom Status Bar:** Shows the file path "app.component.ts - ProyectoFinalEspecializacion - Visual Studio Code", the author "lionelgutierrez", the date "4 months ago", and other status indicators.

```

You, seconds ago | 2 authors (lionelgutierrez and others)
1 import { Component, OnInit } from '@angular/core';
2 import { Platform } from '@ionic/angular';
3 import { SplashScreen } from '@ionic-native/splash-screen/ngx';
4 import { StatusBar } from '@ionic-native/status-bar/ngx';
5 import { CambioMenuServiceservice } from './services/cambio-menu-service.service';
6 import { Tokenstorageservice } from './services/token-storage.service';
7 import { Router, CanActivate } from '@angular/router';
8
You, seconds ago | 2 authors (lionelgutierrez and others)
9 @Component({
10   selector: 'app-root',
11   templateUrl: 'app.component.html',
12   styleUrls: ['app.component.scss']
13 })
14 export class AppComponent implements OnInit {
15   public selectedIndex = 0;
16   public usuario = '';
17   public appPages = [ ];
18
19
20   constructor(
21     private platform: Platform,
22     private splashscreen: SplashScreen,
23     private statusBar: StatusBar,
24     private cambioMenuService:CambioMenuServiceService,
25     private tokenStorage: TokenStorageService,
26     public router: Router
27   ) {
28     this.initializeApp();
29   }
30
31   initializeApp() {
32     this.platform.ready().then(() => {
33       this.statusBar.styleDefault();
34       this.splashscreen.hide();
35     });
36   }
37
38   ngOnInit() {
39   }
40 }

```

FIGURA 3.7. Estructura de directorio del frontend desarrollado en Visual Studio Code.

En la tabla 3.5 se expone detalladamente el contenido y función de cada una de las carpetas y archivos del módulo.

TABLA 3.5. Detalle de archivos y carpetas del módulo de frontend.

Archivo/ Direc- torio	Descripción
cambio- password components	<p>Contiene la página que gestiona el cambio de password de los usuarios.</p> <p>Contiene tres sub-carpetas con los componentes desarrollados para la solución. Los componentes implementados son:</p> <ul style="list-style-type: none"> ■ estadística-evento-fecha: muestra la cantidad de ingresos habilitados y rechazados en el día. ■ listar-tareas: muestra una grilla con el listado de tareas y sub-tareas en un rango de fechas y con un estado particular. ■ tareas-usuario: muestra un listado con cada una de las sub-tareas que tiene un usuario.
estadísticas- ingreso-mes	Contiene la página que muestra las estadísticas de cantidad de ingresos por mes.
estadísticas- tareas-mes	Contiene la página que muestra las estadísticas de cantidad de tareas cerradas por año.
gestión- usuarios	Contiene la página que muestra el listado de usuarios del sistema y permite cambiar el estado de los mismos (activo/inactivo) y agregarles o quitarles roles.
helpers	Contiene la clase “authInterceptor” que permite enviar cada solicitud al backend con el token de autenticación. Para esto intercepta el pedido HTTP y le agrega al encabezado dicho token.
home	Contiene la página principal de la aplicación que muestra, según el rol del usuario, las estadísticas de ingreso al sistema o las tareas en curso del mismo.
ingresos-hoy	Contiene la página que muestra el listado de ingresos del día con la fecha de cada ingreso y si el usuario fue habilitado o no.
listado-tareas	Contiene la página que muestra el listado de tareas en curso. Utiliza el componente “listar-tareas”.
listado-tareas -histórico	Contiene la página que muestra el listado de tareas completas. Utiliza el componente “listar-tareas”.
login	Contiene la página de inicio de sesión para los usuarios.
mi-perfil	Contiene la página que muestra el perfil de usuario y sus datos.
mis-tareas	Contiene la página que muestra las tareas en curso asignadas al usuario.
mis-tareas -histórico	Contiene la página que muestra las tareas cerradas del usuario.
model	Contiene las clases que representan a los objetos de negocio del sistema: roles, usuarios, tareas, sub-tareas, sectores.
pipes	Contiene la implementación de un <i>pipe</i> , que define diferentes colores en función del valor de entrada recibido. Sirve para alertar al usuario de la antigüedad de sus tareas.
recupero- password	Contiene la página que permite al usuario recuperar su password.
register	Contiene la página que permite dar de alta nuevos usuarios al sistema.
services	Contiene los servicios que utilizan las diferentes páginas y componentes para gestionar sus datos y consultas al backend.

Detalle de servicios (“services”) implementados

En esta sub-sección se detallan los servicios implementados en Angular. Mientras que los componentes y las páginas están enfocados en brindar una interfaz gráfica simple y fácil de utilizar para los usuarios, los servicios se orientan a las tareas de lógica de negocio, lo que incluye comunicarse con el backend y gestionar la autenticación y los datos del usuario.

Los servicios implementados son los siguientes:

- authService: se comunica con la API de autenticación del backend para gestionar los inicios de sesión, el alta de nuevos usuarios y las funcionalidades de recuperación y cambio de password.
- cambioMenuService: se encarga del armado del menú de aplicaciones del usuario, en función de su rol.
- datosAuxiliaresService: se encarga de comunicarse con el backend para consultar los datos auxiliares del sistema que son de acceso público como, por ejemplo, el listado de sectores de planta para la pantalla de alta de nuevos usuarios.
- ingresosService: se comunica con el backend para obtener la información de ingresos a planta por rango de fechas.
- socketService: gestiona el socket utilizado para que la vigilancia y la gerencia puedan visualizar en tiempo real los ingresos a la planta.
- tareaService: se comunica con el backend para obtener información de las tareas en curso, de las tareas cerradas y para realizar modificaciones en las mismas.
- tokenStorageService: es el encargado de la gestión del token de autenticación que devuelve el backend al iniciar sesión. Dentro de la gestión se incluye su almacenamiento y recuperación.
- usuariosService: se comunica con el backend para realizar cambios en el estado de los usuarios y sus roles. Solo es utilizado por el rol administrador.
- loginGuardService: permite al módulo de ruteo de la aplicación controlar que el usuario cuente con el rol necesario para acceder a una determinada página. Este servicio se utiliza para habilitar los accesos a las páginas solo al rol de usuario normal.
- rolAdminGuardService: permite al módulo de ruteo de la aplicación controlar que el usuario cuente con el rol necesario para acceder a una determinada página. Este servicio se utiliza para habilitar los accesos a las páginas solo al rol de usuario administrador.
- rolAGerenteGuardService: permite al módulo de ruteo de la aplicación controlar que el usuario cuente con el rol necesario para acceder a una determinada página. Este servicio se utiliza para habilitar los accesos a las páginas solo al rol de usuario gerente.

- rolVigilanciaGuardService: permite al módulo de ruteo de la aplicación controlar que el usuario cuente con el rol necesario para acceder a una determinada página. Este servicio se utiliza para habilitar los accesos a las páginas solo al rol de usuario vigilancia.

Funcionamiento del módulo ante un inicio de sesión

En este apartado se explica el inicio de sesión de un usuario, en el que se puede ver la interacción con el backend y el guardado del token de autenticación para futuras consultas. El proceso comienza cuando el usuario ingresa al sistema y visualiza la pantalla de inicio de sesión. Coloca su username y password y hace click en el botón “Loguearse”. Ante el click del usuario, el sistema realiza las siguientes interacciones:

1. El módulo “loginModule” invoca al servicio “authService” con los datos de username y password.
2. “authService” se comunica con el backend mediante un HTTP POST a la API de autenticación y recibe como respuesta el token asociado al usuario y los datos del mismo (username, password, email, sector y roles asociados). El servicio envía los datos recibidos al módulo “loginModule”.
3. El módulo al recibir el token y los datos del usuario utiliza el servicio “tokenStorageService” para almacenar los valores. Luego, invoca al servicio “cambioMenuService” que genera el menú de usuario según sus roles. Por último, invoca al módulo “homeModule” que muestra la página de inicio al usuario.

En la figura 3.8 se muestra el diagrama de interacción para el inicio de sesión.

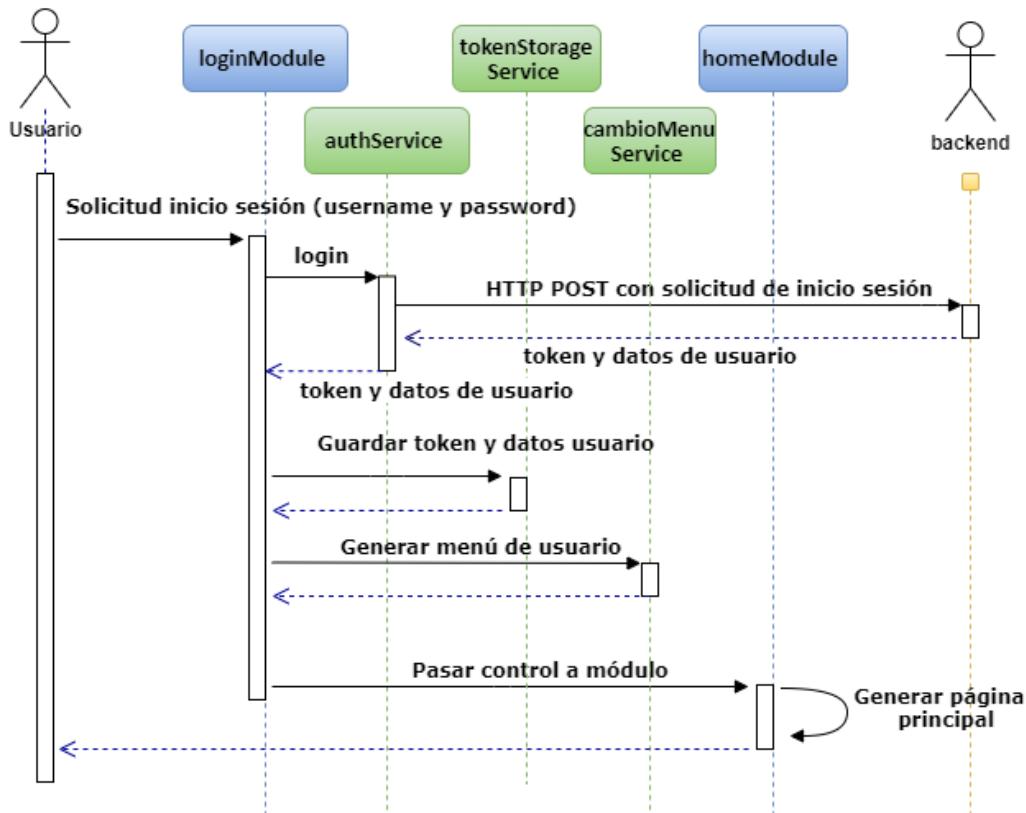


FIGURA 3.8. Diagrama de interacción para el inicio de sesión.

Funcionamiento del módulo ante una solicitud de usuario

En esta sección se explica una interacción típica del usuario en la que se puede ver la relación entre los diferentes elementos del frontend y su comunicación con el backend.

Como pre-requisito, el usuario ya inició sesión en el sistema y desea consultar sus tareas en curso. Para ello hace click en la opción “Mis Tareas en curso” del menú de aplicaciones. Ante el click del usuario el sistema realiza las siguientes interacciones:

1. El módulo de ruteo “app-routing.module” determina quién es el encargado de procesar la solicitud del usuario. En nuestro caso es “mis-tareas-module”. Luego, controla que éste pueda acceder a la página. Para ello consulta con el “loginGuardService”.
2. Si se determina que se puede acceder a la página, se transfiere el control al módulo “mis-tareas-module”. Éste invoca al servicio “tokenStorageService” para adquirir el token de usuario y su id. Con dicho id llama al servicio “tareaService” para obtener las tareas del usuario.
3. El servicio “tareaService” genera la solicitud HTTP GET y la envía al backend.
4. El “authInterceptor” intercepta la solicitud y le agrega un encabezado con el token de autenticación.

5. El backend procesa la solicitud y devuelve el listado de tareas en curso del usuario.
6. El servicio “tareaService” devuelve el listado al módulo “mis-tareas-module”.
7. El módulo arma con el listado la pantalla necesaria para mostrar la información en un formato simple y la envía al usuario.

En la figura 3.9 se muestra el diagrama de interacción para una solicitud de usuario.

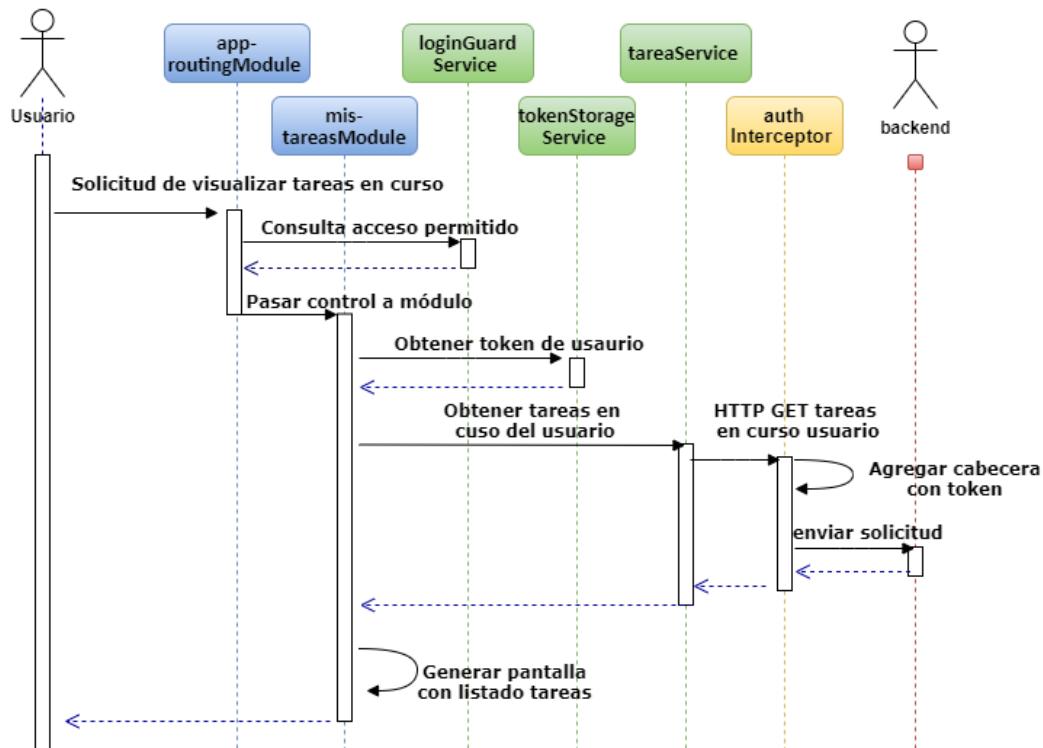


FIGURA 3.9. Diagrama de interacción para una solicitud de usuario.

Pantalla principal de vigilancia

Para el rol de vigilancia se implementó un *WebSocket* que posibilita una comunicación en tiempo real con el backend. Esto permite que ante los intentos de ingreso a planta del personal de tercero, el vigilante pueda tener al instante una alerta, ya sea de un acceso exitoso o de un acceso denegado. Para implementar dicho *WebSocket* se utilizó la librería *Socket.io*, tanto en el backend como en el frontend.

A continuación, se muestran los pasos que realiza el sistema y los componentes involucrados en la generación de una alerta:

1. Cuando el usuario de vigilancia inicia sesión en la aplicación, la misma lo redirige a su página principal (módulo “homeModule”). En ese momento, el frontend utiliza el servicio “socketService” para iniciar el *WebSocket* con el backend y suscribirse a los eventos de tipo “ingreso”.
2. Posteriormente, cuando un usuario de tercero intenta ingresar a planta, el backend recibe la solicitud de ingreso, la procesa y genera las tareas de control y alertas correspondientes, según lo explicado en la subsección 3.3.1. Dentro de dichas acciones, el sistema emite un evento del tipo “ingreso” que contiene el resultado del intento de acceso, los datos del tercero asociado y una descripción con el motivo de la habilitación o denegación.
3. El módulo “homeModule” recibe el evento del backend y genera la alerta al vigilante. La misma se presenta en un *Pop up* que se muestra durante 6 segundos e incluye los detalles del acceso. Para ver el detalle de cada tipo de acceso referirse a la sección 4.4

3.4. Interfaz con sistema de documentación

El sistema de documentación de terceros es un aplicativo que ya está desarrollado en la empresa. El acceso al mismo es a través de un *endpoint* HTTP GET, al cual se le indica el id del usuario de tercero que se quiere consultar, y devuelve un objeto JSON con la información del nombre y apellido de la persona, si el acceso se debe permitir o no (variable con valor OK o NO) y el motivo por el cual se habilita o no a la mismo. Para configurar el acceso a dicho sistema dentro de nuestro desarrollo, bastó con contar con la dirección del *endpoint*.

Dado que el desarrollo y las pruebas se realizaron en un entorno desconectado de la empresa, se implementó un *mock* para simular el acceso al sistema. Dicho *mock* se desarrolló como una aplicación web con Node.JS y la librería Express, lo que permitió exponer un *endpoint* HTTP GET del mismo modo que lo hace el sistema original. Para el desarrollo se utilizó el IDE Visual Studio Code y se tomaron 4 casos típicos como respuesta:

1. Usuario activo con documentación en regla.
2. Usuario activo con documentación vencida.
3. Usuario dado de baja (fin de contratación).
4. Usuario no existente (id de usuario nunca dado de alta).

Con estos 4 casos definidos se pudieron probar todas las alternativas y asegurar la respuesta correcta de nuestra aplicación.

Capítulo 4

Ensayos y resultados

En el presente capítulo se describen las pruebas realizadas sobre el sistema desarrollado y se muestran los resultados obtenidos.

4.1. Detalle de pruebas realizadas

Para planificar y gestionar todo el proceso de pruebas se desarrolló un *Master Test Plan* [2], donde se especificaron los objetivos de las pruebas, los responsables, la estrategia general y la estrategia por niveles de prueba.

Los objetivos de las pruebas realizadas sobre el software y hardware fueron:

- Determinar si el sistema cumple con los requerimientos especificados en la sección 2.5.
- Reportar las diferencias entre lo observado y el comportamiento deseado.
- Dejar evidencias y documentación para probar las siguientes versiones del software y solucionar cualquier *bug* detectado.

Para dar cuenta del proceso, en la tabla 4.1, se muestra como se organizaron los tipos de prueba realizadas.

Para lograr una trazabilidad entre los requerimientos y los casos de test se implementó, por un lado, una matriz de trazabilidad entre los requerimientos y los casos de uso definidos para el sistema, y por el otro, una matriz de trazabilidad entre los casos de uso y los casos de test. Esto quedó registrado en el documento de Casos de Uso y Casos de Test [4].

TABLA 4.1. Tipos de pruebas realizadas sobre el sistema.

Tipo de Prueba	Objetivo	Descripción
Pruebas unitarias	Este tipo de pruebas permitió verificar de forma separada cada uno de los módulos de hardware y software del sistema.	Para aquellos módulos que tenían interfaces con otros se utilizaron <i>mocks</i> [3], de forma de simular el comportamiento de los mismos sin necesidad de tenerlos implementados. Se emplearon las herramientas Postman/- Newman.
Pruebas de sistema	Este tipo de prueba permitió verificar el sistema de manera integral, asegurando la correcta comunicación e interrelación de los módulos.	Pruebas en ambiente de desarrollo, con los módulos implementados.
Pruebas de aceptación	Este tipo de pruebas permitió validar el desarrollo de manera integral junto al cliente. De esta forma se aseguró no solo que el sistema se comporte según lo especificado, sino que el usuario final corrobore que el sistema actúa según sus necesidades y requisitos.	Pruebas en ambiente de desarrollo, con los módulos implementados.

4.1.1. Herramientas utilizadas

Para la realización de las pruebas unitarias se utilizó Postman, la cual permitió centralizar y gestionar todas las pruebas desde una única herramienta. Además, se utilizó Newman para automatizar la ejecución de las mismas.

En la figura 4.1 se muestra la herramienta Postman, junto a la colección de pruebas unitarias definidas, su organización y la configuración del ambiente de trabajo.

Por su parte, Newman permite importar los conjuntos de tests definidos en Postman, y mediante un script, ejecutar los mismos y ver los resultados obtenidos. De esta manera podemos correr todos los tests automáticamente, de forma rápida y simple, sin necesidad de hacerlo manualmente.

Para llevar adelante dicho procedimiento desde Postman se exportaron, tanto la colección de tests, como el ambiente de trabajo (*environment* [5]). Esto generó dos archivos que se utilizaron para configurar la ejecución automática desde Newman. Luego, se desarrolló un script en el backend del trabajo para poder correr la colección con la herramienta de *npm*.

En la figura 4.2 se muestra el script generado y la configuración del mismo en el archivo `package.json` del módulo de backend.

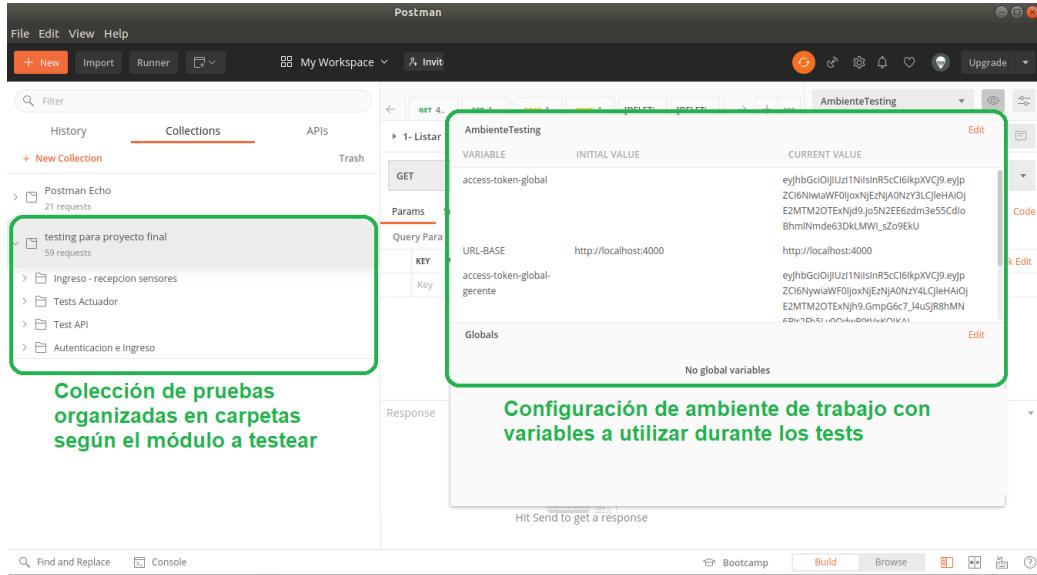


FIGURA 4.1. Herramienta Postman junto a su configuración básica.

Por otra parte, en la figura 4.3 se muestra la ejecución del script desde la consola de Visual Studio Code y los resultados obtenidos.

4.1.2. Mocks implementados

Para aquellos módulos que tenían dependencias (interfaces) con otros se utilizaron *mocks*, lo que posibilitó testearlos simulando el comportamiento de los dependientes, sin necesidad de tenerlos implementados.

Se desarrollaron dos *mocks* en el trabajo:

1. *mockActuador*: permitió simular la operatoria del módulo actuador. El mismo se implementó como una aplicación web en Node.JS que respondía a las peticiones realizadas desde el backend.
2. *mockSistemaTerceros*: permitió simular la interfaz entre el sistema desarrollado y el sistema de documentación de terceros. El mismo se implementó como una aplicación web en Node.JS,

```

backend > {} package.json > ...
  2   "name": "backend",
  3   "version": "1.0.0",
  4   "description": "backend del proyecto final CeIoT de la FIUBA",
  5   "main": "index.js",
  6   ▶ Debug
  7   "scripts": {
  8     "test": "newman run
  9       ./test/listadoTests.postman_collection.json -e
 10      ./test/AmbienteTesting.postman_environment.json
 11      --delay-request 1000"
 12   },
 13   "keywords": [
 14     "bakend", Script generado para
 15     "nodejs", testing automático
 16     "control",
 17     "terceros"
 18   ],

```

FIGURA 4.2. Configuración del script para testing automático de pruebas unitarias.

```

DEBUG CONSOLE PROBLEMS OUTPUT TERMINAL
(base) lionel@lionel-Z87X-0C:~/Dropbox/especializacion_IOT/trabajo_final/ProyectoFinalEspecializacion$ cd backend/
(base) lionel@lionel-Z87X-0C:~/Dropbox/especializacion_IOT/trabajo_final/ProyectoFinalEspecializacion/backend$ npm test
...
↳ 3- SignIn con password invalido
  POST http://localhost:4000/auth/signin [401 Unauthorized, 380B, 88ms]
    ✓ Testeo status
    ✓ Testeo respuesta
↳ 4- SignIn con password valido
  POST http://localhost:4000/auth/signin [200 OK, 597B, 91ms]
    ✓ Testeo status
    ✓ Testeo respuesta
} } Ejecución
                                          automática de
                                          cada uno de los
                                          tests
                                          Resultado de la
                                          ejecución

```

	executed	failed
iterations	1	0
requests	59	0
test-scripts	75	0
prerequest-scripts	16	0
assertions	118	0
total run duration:	1m 34.9s	
total data received:	47.3KB (approx)	
average response time:	24ms [min: 4ms, max: 132ms, s.d.: 32ms]	

FIGURA 4.3. Ejecución y resultados de ejecución del script automático de testing.

4.2. Pruebas unitarias

En esta sección se detalla el conjunto de pruebas unitarias realizadas sobre cada uno de los módulos del sistema.

4.2.1. Testing del módulo sensor

Al momento de testear el módulo sensor, ya se contaba con el módulo del backend desarrollado, con lo cual no fue necesario implementar un *mock*.

Para proceder con las pruebas se configuraron algunas tarjetas RFID con diferentes valores que simularon cada uno de los escenarios posibles. Se utilizó con tal propósito un programa escrito en Arduino IDE que permitió asignar el valor a la tarjeta a través del monitor serie. Luego se alimentó el módulo y se procedió a testear cada caso, acercando una a una las tarjetas ya configuradas para observar los resultados obtenidos.

En la tabla 4.2 se detalla el conjunto de casos de tests más relevantes implementados para probar el módulo sensor y sus resultados.¹

TABLA 4.2. Casos de test del módulo sensor.

Caso test	Escenario a testear	Resultados
1	Tarjeta RFID sin valor configurado.	El led “PinTarjNoLeida” se prende durante dos segundos y se apaga.
2	Valor de tarjeta RFID recibido correctamente por el backend.	El led “PinOkSistema” se prende durante dos segundos y se apaga.
3	El módulo sensor no registrado en el sistema.	El led “PinNoOkSistemaCodigo” se prende durante dos segundos y se apaga.
4	El módulo sensor no está activo en el sistema.	El led “PinNoOkSistemaCodigo” se prende durante dos segundos y se apaga.

En la figura 4.4 se muestra el caso de test 1 donde la tarjeta no tiene valor configurado. En la primera imagen se ve el momento en que se acerca la tarjeta y en la segunda imagen se ve el momento en que el módulo responde al usuario.

En la figura 4.5 se muestra el caso de test 2 donde la tarjeta es válida, y el sistema se comunica correctamente con el módulo del backend. En la primera imagen se ve el momento en que se acerca la tarjeta y en la segunda imagen se ve el momento en que el módulo responde al usuario.

¹Para tener un detalle completo de los test remitirse al documento de Casos de Uso y Casos de Test [4].

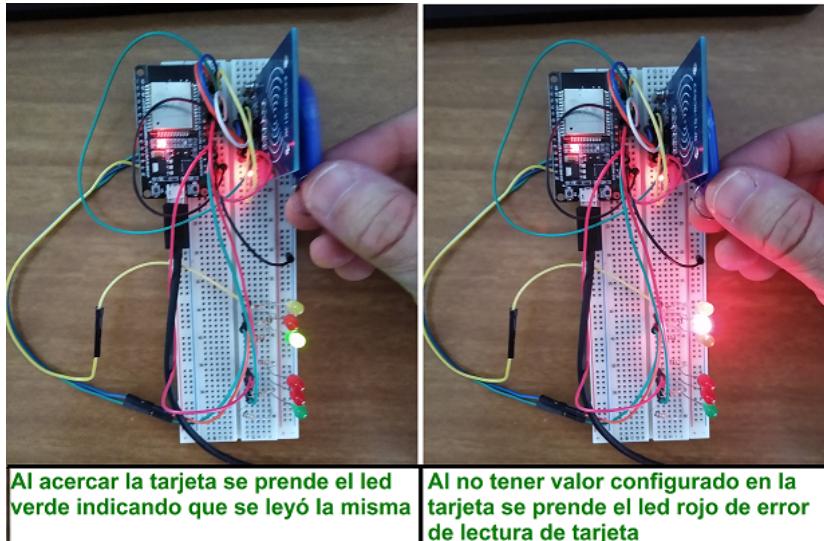


FIGURA 4.4. Procedimiento de prueba para el caso de test 1 con tarjeta sin valor configurado.

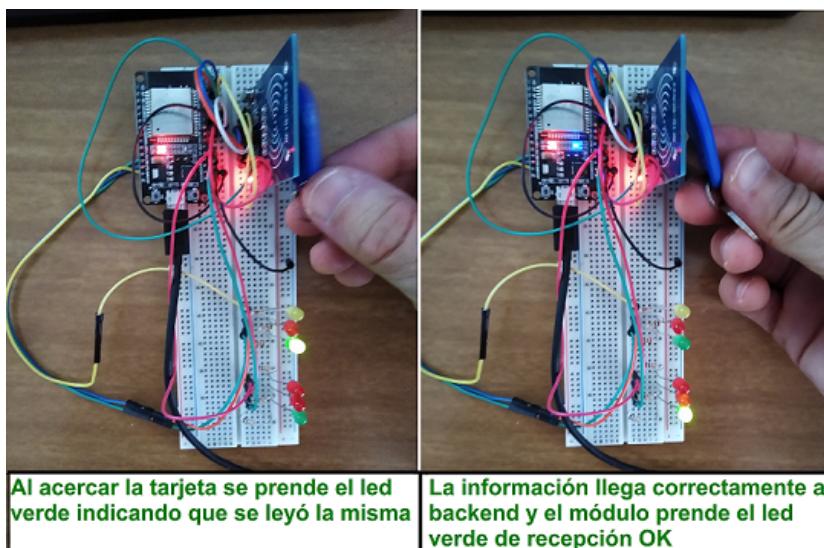


FIGURA 4.5. Procedimiento de prueba para el caso de test 2 con tarjeta con valor configurado y recibido correctamente por el backend.

4.2.2. Testing del módulo actuador

Para testear el módulo actuador se utilizó Postman, dado que el módulo expone una conexión HTTP POST. Dentro de Postman generamos una carpeta llamada Test Actuador, en la cual almacenamos todos los casos de test. En el *body* se envían los valores como un objeto JSON.

En la figura 4.6 se muestra la configuración de Postman para el testeo del módulo actuador.

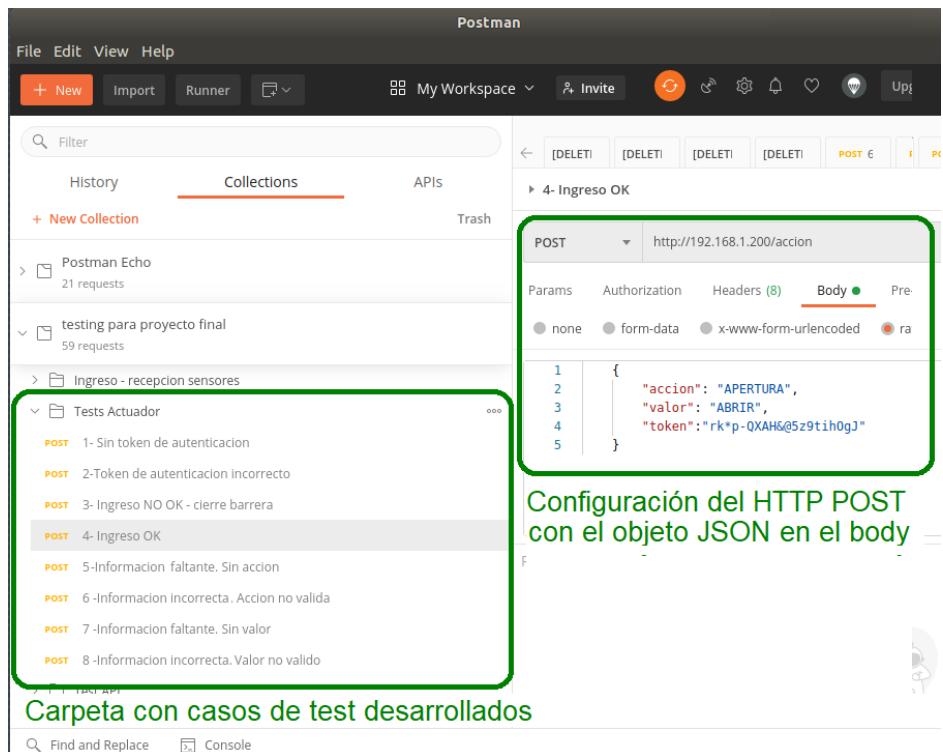


FIGURA 4.6. Configuración en Postman para el testeo del módulo actuador.

La prueba implicó testear cada uno de los casos generados, ejecutándolos desde Postman. A fin de automatizar los tests, y no depender de un control manual de la respuesta obtenida luego de cada ejecución, se configuraron para cada test las condiciones esperadas o aserciones, utilizando la sección “Tests” de la herramienta. Algunas de las aserciones definidas fueron que el código de estado HTTP sea el esperado y/o que la respuesta contenga cierto dato o cierto objeto JSON con determinados valores. Al ejecutar el test, si los resultados eran los previstos, Postman indicaba en el apartado “Test Results” que los mismos eran correctos.

En la figura 4.7 se muestra un ejemplo de configuración de la sección “Tests” y la ejecución del caso de test.

En la tabla 4.3 se detalla el conjunto de casos de tests más relevantes implementados para probar el módulo actuador y sus resultados. Dentro del escenario a testear entre paréntesis se hace referencia al nombre del test en Postman.²

²Para tener un detalle completo de los test remitirse al documento de Casos de Uso y Casos de Test [4].

The screenshot shows the Postman application interface. At the top, there's a dark header bar with the word "Postman" and a navigation menu with options like "File", "Edit", "View", and "Help". Below the header is a toolbar with buttons for "New", "Import", "Runner", "My Workspace", and "Invite". The main area is a request editor for a POST method to the URL "{{URL-BASE}}/auth/signup". The "Tests" tab is highlighted with a red box. Inside the "Tests" tab, there is a code block containing two test cases:

```
1 pm.test("Testeo status", function () {  
2     pm.expect(pm.response.code).to.equal(400);  
3 });  
4  
5 pm.test("Testeo respuesta", function () {  
6     const responseJson = pm.response.json();  
7     pm.expect(responseJson.message).to.be.equal("El username ya existe en el sistema");  
8 });
```

Below the request editor, the text "Aserciones definidas en la sección Tests" is displayed in green. The bottom section shows the "Test Results (2/2)" tab selected, with a red box around it. It displays two test results: "Testeo status" and "Testeo respuesta", both marked as "PASS". To the right of this section, the text "Resultado de la ejecución del test" is displayed in green.

FIGURA 4.7. Configuración de sección “Tests” y resultados de la ejecución del test.

En la sección 4.4 se muestra el detalle de los casos de ingreso habilitado e inhabilitado junto a una figura del módulo actuador. Remitirse a dicha sección para más detalles.

TABLA 4.3. Casos de test del módulo actuador.

Caso test	Escenario a testear	Resultados
1	Habilitar ingreso a la planta (Ingreso OK).	El led “IngresoOk” parpadea durante 4 segundos y se activa la cerradura.
2	Inhabilitar ingreso a la planta (Ingreso NO OK – cierre barrera).	El led “IngresoNOOk” se prende durante dos segundos y se apaga.
3	El pedido HTTP POST no cuenta con el campo de acción a realizar (Información faltante – Acción no válida).	El led Error parpadea dos veces.
4	El pedido HTTP POST no cuenta con token de autenticación (Sin token de autenticación).	El led Error parpadea una vez.

4.2.3. Testing del módulo de backend

Para testear el módulo de backend se utilizó Postman, dado que el módulo expone una API Rest al frontend, una API de autenticación y un *endpoint* [6] para recibir los datos de los módulos sensores. Dentro de Postman generamos una carpeta llamada “Test API” para testear la API expuesta al frontend, una carpeta llamada “Autenticación e ingreso” para testear la API de autenticación y una carpeta llamada “Ingreso – recepción sensores” para testear los ingresos generados desde el módulo sensor.

Dentro de la carpeta de “Test API” generamos cuatro subcarpetas para testear los casos sin autenticación y con autenticación para los diferentes perfiles de usuario. Dentro de la carpeta de “Autenticación e ingreso”, se colocaron los tests para el alta de usuario y el ingreso al sistema. Y dentro de la capeta “Ingreso – recepción sensores”, los tests para los ingresos generados desde el módulo sensor.

En la figura 4.8 se muestra las carpetas configuradas en Postman para el testeо de los casos de test de backend.

Para testear cada caso, se procedió a configurar en Postman la URL de cada *endpoint*, junto a los parámetros requeridos. Para los pedidos HTTP GET se configuró directamente la URL con los parámetros, y para los HTTP POST se completó el *body* con el objeto JSON, junto a los valores requeridos según cada caso.

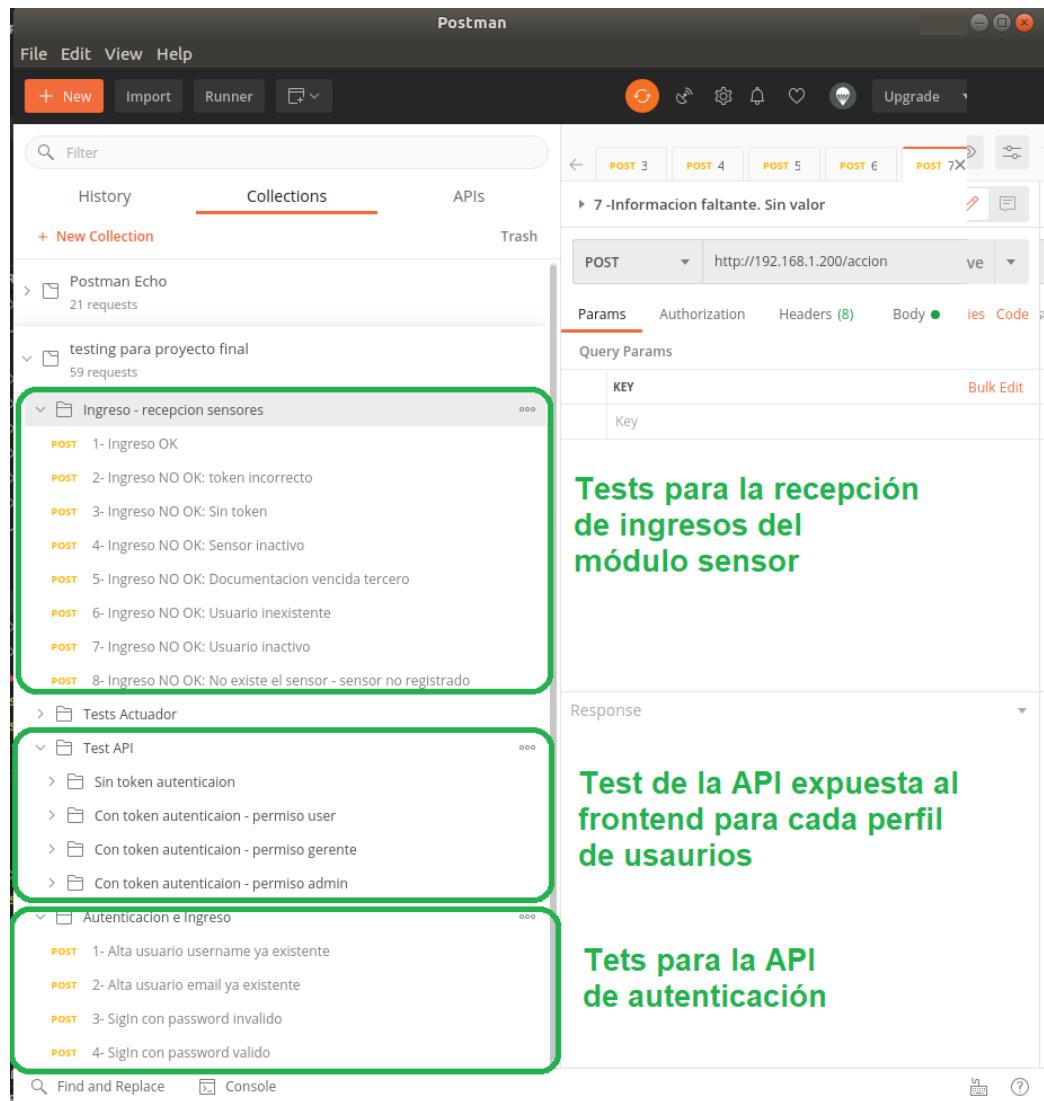


FIGURA 4.8. Configuración en Postman para el testeo del módulo de backend.

Para los casos de test con autenticación definidos dentro de la carpeta “Test API”, se procedió a utilizar los *environments* y las variables globales de Postman. Un *environment* permite definir ambientes de testing dentro de los cuales podemos definir variables globales que pueden utilizarse en varios tests. Un test puede obtener un valor de la respuesta HTTP y guardarlo en una variable global para luego ser utilizado por otro. De este modo, definimos un *environment* llamado “Ambiente-Testing” y dentro del mismo configuramos las siguientes variables globales:

- URL-BASE: contiene la URL base del backend a partir del cual se define cada *endpoint*. Con la misma configuramos cada HTTP POST o GET, de modo que si cambia la URL de nuestro backend, con solo modificar esta variable quedan ajustados todos los *endpoints*.
- Access-token-global: utilizado para los *endpoints* que son accesibles para cualquier usuario del sistema. Permite guardar el token de acceso del rol usuario.
- Access-token-global-gerente: utilizado para los *endpoints* que son accesibles

solo por el perfil de gerente de la aplicación. Permite guardar el token de acceso del rol gerente.

- Access-token-global-admin: utilizado para los *endpoints* que son accesibles solo por el perfil de administrador de la aplicación. Permite guardar el token de acceso para el rol de administrador.

Los tokens de acceso se configuran dentro de la sección header del test, con clave x-access-token y con el valor del token.

En la figura 4.9 se muestran las variables globales definidas en Postman.

The screenshot shows the 'MANAGE ENVIRONMENTS' interface in Postman. At the top, it says 'Environment Name' and has a field containing 'AmbienteTesting'. Below this is a table of global variables:

	VARIABLE	INITIAL VALUE ⓘ	CURRENT VALUE ⓘ	...	Persist All	Reset All
<input checked="" type="checkbox"/>	access-token-global		eyJhbGciOiJIUzI1NlslslnR5cCl6IkpxVCJ9.eyJpZCI6Ni...			
<input checked="" type="checkbox"/>	URL-BASE	http://localhost:4000	http://localhost:4000			
<input checked="" type="checkbox"/>	access-token-global-gerente		eyJhbGciOiJIUzI1NlslslnR5cCl6IkpxVCJ9.eyJpZCI6N...			
<input checked="" type="checkbox"/>	access-token-global-admin		eyJhbGciOiJIUzI1NlslslnR5cCl6IkpxVCJ9.eyJpZCI6M...			
Add a new variable						

At the bottom, there is a note: 'Use variables to reuse values in different places. Work with the current value of a variable to prevent sharing sensitive values with your team. [Learn more about variable values](#)' with a close button. Below the table are 'Cancel' and 'Update' buttons.

FIGURA 4.9. Variables globales definidas en Postman.

Tests de la carpeta Test API

Dentro de la carpeta Test API tenemos cuatro subcarpetas:

- Sin token autenticación: testea los *endpoints* sin un token de autenticación.
- Con token autenticación – permiso user: testea los *endpoints* que son accesibles para cualquier usuario del sistema. En esta carpeta se define un test inicial llamado "SignIn OK", que se utilizó para simular la autenticación de un usuario normal y completar la variable global access-token-global que fue utilizada por el resto de los tests de la carpeta.
- Con token autenticación – permiso gerente: testea los *endpoints* que son accesibles para el usuario con rol gerente. En esta carpeta se define un test inicial llamado "SignIn OK", que se utilizó para simular la autenticación de un usuario gerente y completar la variable global access-token-global-gerente que fue utilizada por el resto de los tests de la carpeta.

- Con token autenticación – permiso admin: testea los *endpoints* que son accesibles para el usuario con rol administrador. En esta carpeta se define un test inicial llamado “SigIn OK”, que se utilizó para simular la autenticación de un usuario administrador y completar la variable global access-token-global-admin que fue utilizada por el resto de los tests de la carpeta.

En la figura 4.10 se muestran los tests de la subcarpeta “Con token autenticación – permiso gerente”, junto al test inicial “SignIn OK” y al resto de los tests.

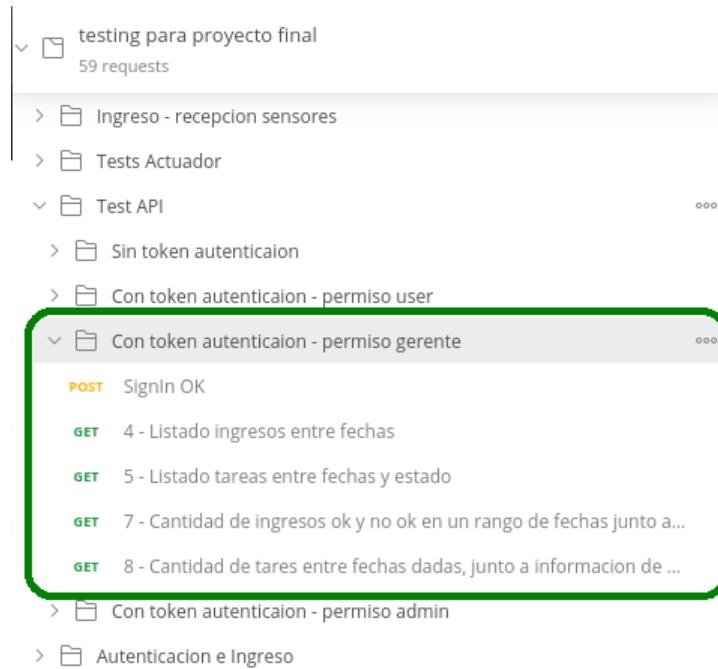


FIGURA 4.10. Tests de la subcarpeta Con token autenticación – permiso gerente.

Detalle de tests y resultados

En la tabla 4.4 se detalla el conjunto de casos de tests más relevantes implementados para probar la recepción de los datos de los módulos sensores y sus resultados. En la columna escenario a testear se hace referencia, entre paréntesis, al número del test en Postman.³

TABLA 4.4. Casos de test del módulo backend. Testing de recepción de los datos de los módulos sensores.

Subcarpeta man	Post-	Escenario a testear	Resultados
Ingreso – recepción sensores.		Ingreso con valor de tarjeta registrado en el sistema y módulo sensor activo (1). El módulo sensor no está registrado en el sistema (8). El módulo sensor no está activo en el sistema (4).	Se confirma recepción con HTTP código 200. Se rechaza el ingreso con mensaje de error de sensor inexistente. Se rechaza el ingreso con mensaje de error de sensor inactivo.

En la tabla 4.5 se detalla el conjunto de casos de tests más relevantes implementados para probar la API de autenticación y sus resultados. En la columna escenario a testear se hace referencia, entre paréntesis, al número del test en Postman.

TABLA 4.5. Casos de test del módulo backend. Testing de la API de autenticación.

Subcarpeta man	Post-	Escenario a testear	Resultados
Autenticación e ingreso.		Alta de usuario con username, email y password correcto (5). Alta de usuario con username o email repetido (1 y2). Inicio de sesión con username o password incorrecto (3 y 4).	Se confirma el alta del usuario con HTTP código 200. Se rechaza el alta. Respuesta HTTP con código 400 y mensaje de error. Se rechaza el inicio de sesión. Respuesta HTTP con código 401 y mensaje de error.

En la tabla 4.6 se detalla el conjunto de casos de tests más relevantes implementados para probar la API Rest expuesta al frontend y sus resultados. En la columna

³Para tener un detalle completo de los test remitirse al documento de Casos de Uso y Casos de Test [4].

escenario a testear se hace referencia, entre paréntesis, al número del test en Postman.⁴

TABLA 4.6. Casos de test del módulo backend. Testing de la API Rest expuesta al frontend.

Subcarpeta Post-man	Escenario a testear	Resultados
Test API/Sin token autenticación.	Listar sectores de la empresa (1).	Se acepta el pedido y se devuelve el listado de sectores de la empresa.
	Resto de los <i>end-points</i> .	Se rechaza el pedido por falta de token de autenticación con un HTTP 403.
Test API/Con token autenticación – permiso user.	Listado de tareas en curso del usuario (2).	Se acepta el pedido y se devuelve el listado de tareas en curso del usuario.
	Listado de tareas completas del usuario (3).	Se acepta el pedido y se devuelve el listado de tareas completas del usuario.
	Cantidad ingresos a la empresa por día (7).	Se rechaza el pedido por no tener rol gerente o administrador con un HTTP 403.
Test API/Con token autenticación – permiso gerente.	Cantidad ingresos a la empresa por día (7).	Se acepta el pedido y se devuelve el listado de ingresos OK y no OK para las fechas indicadas.
	Listado detalle de tareas en curso entre rango de fechas (5).	Se acepta el pedido y se devuelve el listado de tareas y sub-tareas para las fechas indicadas.
Test API/Con token autenticación – permiso admin.	Listado de usuarios y roles (9).	Se acepta el pedido y se devuelve el listado de usuarios y roles de cada usuario.
	Cambiar estado de rol de usuario (12).	Se acepta el pedido y ajusta el estado del rol para el usuario indicado.

⁴Para tener un detalle completo de los test remitirse al documento de Casos de Uso y Casos de Test [4].

4.3. Pruebas de sistema

En esta sección se detalla el conjunto de pruebas integrales realizadas sobre el sistema con todos sus módulos. Este test fue ejecutado por el desarrollador en el ambiente de pruebas, sin el cliente.

Una vez implementados todos los módulos, se realizaron las siguientes tareas:

1. Se crearon usuarios con cada uno de los roles en la base de datos.
2. Se alimentó el módulo sensor y el módulo actuador.
3. Se corrió el módulo de backend, el *mock* del sistema de terceros y el módulo de frontend. Con el propósito de automatizar este paso, se desarrolló un script para ejecutar y levantar los módulos antes mencionados, mediante el uso de contenedores *Docker* y de la herramienta *docker-compose*.

Con los módulos en ejecución, se procedió a *loguearse* en la aplicación. En primer lugar, con un rol de usuario normal, luego con el rol de usuario gerente, posteriormente con el rol vigilante y por último con el rol administrador.

En la tabla 4.7 se detalla el conjunto de casos de tests más relevantes implementados para probar el rol de usuario normal (rol user) y los resultados obtenidos.⁵

TABLA 4.7. Listado de pruebas realizadas para el rol de usuario normal.

Pantalla del sistema testeada	Acción	Resultados
Mis tareas en curso	Visualizar tareas.	Se muestra el listado de tareas en curso del usuario o se indica que no tiene tareas en curso.
Mis tareas en curso	Cerrar tarea.	Se marca como completa la tarea del usuario y se muestra una alerta indicando que se cerró la tarea correctamente.
Mis tareas en curso	Ajustar observación en tarea.	Se ajusta la observación en la tarea y se muestra una alerta indicando que se actualizó la observación correctamente.
Mi historial tareas	Visualizar historial tareas de usuario.	Se muestra el listado de tareas cerradas del usuario o se indica que no tiene tareas cerradas.
Mi perfil	Visualizar perfil.	Se muestra la información del usuario, incluyendo su <i>username</i> , email, sector y roles asignados.

En la tabla 4.8 se detalla el conjunto de casos de tests más relevantes implementados para probar el rol de usuario administrador y los resultados obtenidos.

En la tabla 4.9 se detalla el conjunto de casos de tests más relevantes implementados para probar el rol de usuario gerente y los resultados obtenidos.

⁵Para tener un detalle completo de los test remitirse al documento de Casos de Uso y Casos de Test [4].

TABLA 4.8. Listado de pruebas realizadas para el rol de usuario administrador.

Pantalla del sistema testeada	Acción	Resultados
Gestión usuarios	Modificar estado de rol de usuario.	Se modifica el estado del rol del usuario en la base de datos y se muestra una alerta indicando que el cambio se realizó correctamente.

TABLA 4.9. Listado de pruebas realizadas para el rol de usuario gerente.

Pantalla del sistema testeada	Acción	Resultados
Estadísticas ingresos	Visualizar ingresos del mes.	Se muestra un gráfico de barras con la cantidad de ingresos de cada día del mes especificado (ingresos ok y no ok) y un gráfico de torta con el total de ingresos ok y no ok del mes.
Estadísticas tareas	Visualizar estadísticas de tareas.	Se muestra un gráfico de torta con la cantidad de tareas en curso, un gráfico de barras con la cantidad de tareas cerradas por cada mes del año y un gráfico de torta con el total de tareas completas del año.
Tareas en curso	Visualizar tareas en curso.	Se muestra el listado de tareas en curso, junto a información de antigüedad, estado, fecha de inicio y detalle de las sub-tarea de cada sector.
Ingresos del día	Visualizar ingresos del día.	Se muestra el listado de los ingresos, junto a información de horario, si el ingreso fue permitido o no y el detalle de la habilitación o rechazo junto al nombre del tercero.

En la tabla 4.10 se detalla el conjunto de casos de tests más relevantes implementados para probar el rol de usuario vigilante y los resultados obtenidos.

TABLA 4.10. Listado de pruebas realizadas para el rol de usuario vigilante.

Pantalla del sistema testeada	Acción	Resultados
Pantalla principal	Visualizar ingreso en portería.	Cuando un tercero pretende ingresar en planta, se muestra en la pantalla principal una alerta, indicando el intento de acceso y si el mismo se permitió o se rechazó, junto a un color particular para diferenciar ambos casos. La alerta se muestra por 6 segundos.

4.4. Pruebas de aceptación

En esta sección se detallan dos de las pruebas de aceptación realizadas en conjunto con el cliente. Estos tests fueron llevados a cabo por el desarrollador, en el ambiente de pruebas, para verificar el funcionamiento correcto del sistema implementado.

Para preparar dicho ambiente se procedió de igual modo que el descripto en la sección anterior para las pruebas de sistema.

4.4.1. Descripción y detalles de prueba de ingreso habilitado

En esta sección se detalla la prueba de ingreso de un tercero a la empresa que cuenta con la documentación requerida en regla.

Para realizar la prueba se preparó el sistema y se lo dejó operativo. Se utilizó la tarjeta RFID de un tercero que estaba habilitado y un usuario con rol de vigilancia generado previamente.

A continuación, se detallan los pasos realizados para el caso descripto:

1. Se ingresó a la aplicación con el usuario Portería y se accedió a la página principal.
2. Se tomó la tarjeta RFID del personal de tercero habilitado y se la acercó al módulo sensor. Como respuesta a esta interacción, sucedieron las siguientes acciones:
 - a) El módulo sensor prendió el led verde de lectura de la tarjeta y luego prendió el led verde de comunicación correcta con el backend.
 - b) En la pantalla del usuario de portería se visualizó la alerta de ingreso del tercero durante 6 segundos. A continuación, se corroboró que el contador de ingresos correctos se había incrementado en la pantalla.
 - c) El módulo actuador prendió de forma intermitente el led verde de ingreso habilitado durante 4 segundos y se cerró el pestillo de la cerradura electrónica. Pasado ese tiempo el pestillo se abrió nuevamente.

En la figura 4.11 se muestra el paso 2 de la prueba con el acercamiento de la tarjeta RFID al módulo sensor y la respuesta de dicho módulo.

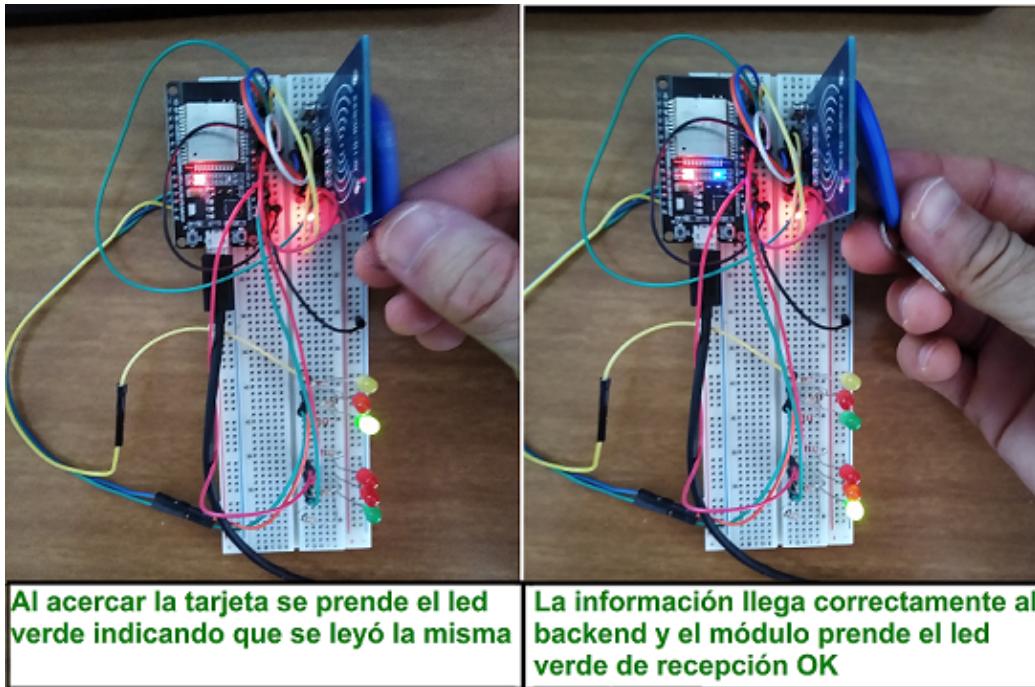


FIGURA 4.11. Accionamiento de módulo sensor con tarjeta RFID y respuesta del módulo.

En la figura 4.12 se muestra el paso 2 b) de prueba con la información visualizada por la vigilancia.

En la figura 4.13 se muestra el paso 2 c) de la prueba, donde se ve el módulo actuador con el led prendido y el pestillo de la cerradura cerrada.

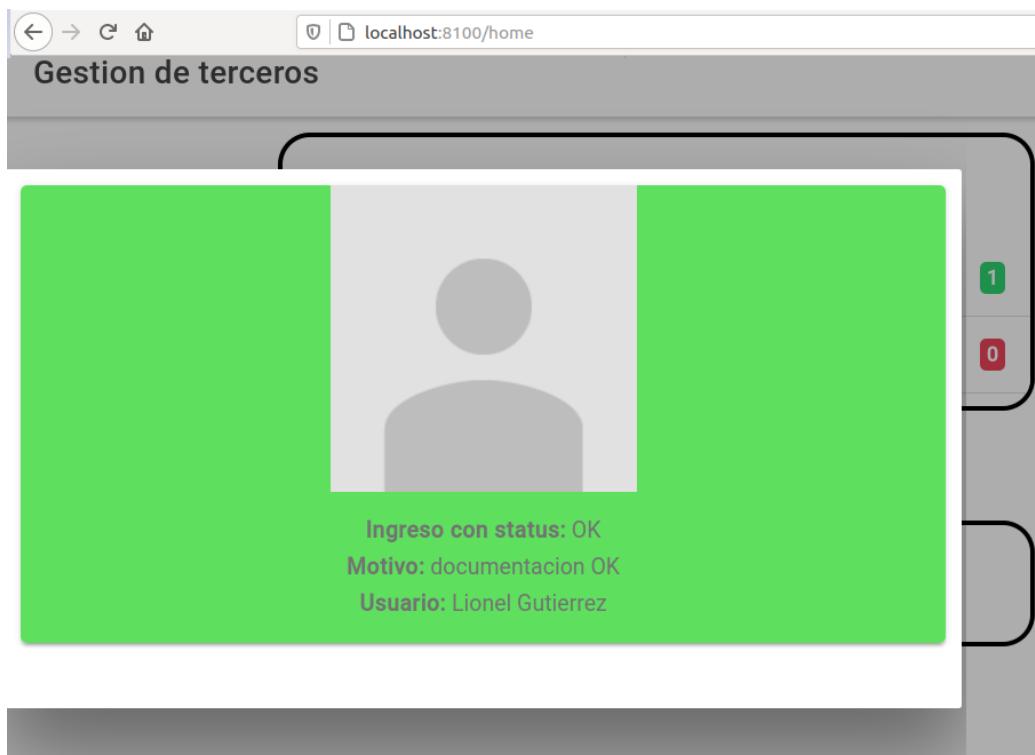


FIGURA 4.12. Visualización de la alerta recibida en pantalla.

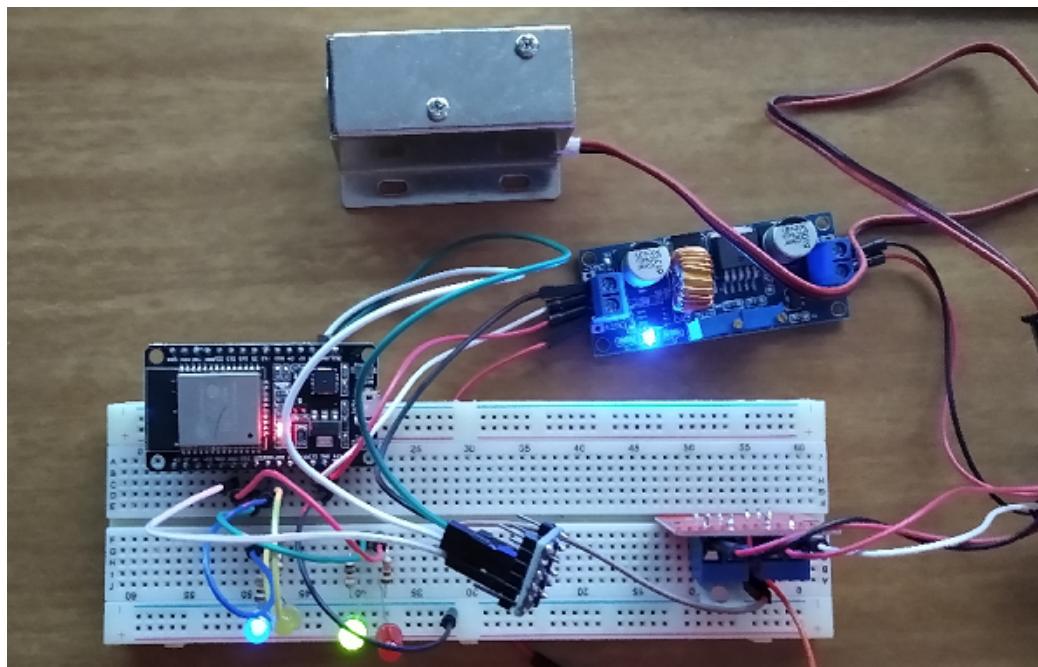


FIGURA 4.13. Accionamiento de módulo actuador y respuesta del módulo.

4.4.2. Descripción y detalles de prueba de ingreso inhabilitado

En esta sección se detalla la prueba de ingreso de un tercero a la empresa que no cuenta con la documentación requerida en regla.

Para realizar la prueba se preparó el sistema y se lo dejó operativo. Se utilizó la tarjeta RFID de un tercero que tenía problemas con su documentación y un usuario con rol de vigilancia generado previamente.

A continuación, se detallan los pasos realizados para el caso descripto:

1. Se ingresó a la aplicación con el usuario Portería y se accedió a la página principal.
2. Se tomó la tarjeta RFID del personal de tercero inhabilitado y se acercó la misma al módulo sensor. Como respuesta a esta interacción, sucedieron las siguientes acciones:
 - a) El módulo sensor prendió el led verde de lectura de la tarjeta y luego prendió el led verde de comunicación correcta con el backend.
 - b) En la pantalla del usuario de portería se visualizó la alerta de ingreso inhabilitado del tercero durante 6 segundos. A continuación, se corroboró que el contador de intento de ingresos incorrectos se había incrementado en la pantalla.
 - c) El módulo actuador prendió durante 2 segundos el led rojo de ingreso inhabilitado. El pestillo de la cerradura electrónica se mantuvo abierto todo el tiempo.
 - d) Se envió un email a los usuarios de los sectores encargados de controlar la documentación del tercero, con el detalle del intento de ingreso.
 - e) Se generó una tarea de control de documentación para cada uno de los sectores de la empresa responsables del proceso.
3. Se ingresó a la aplicación con un usuario de sector HESA (Seguridad y Medio Ambiente) para corroborar la generación de la tarea de control. Se accedió a la aplicación “Mis tareas en curso” y se visualizó la nueva tarea asignada al usuario.

En la figura 4.14 se muestra el paso 2 b) de la prueba con la información visualizada por la vigilancia.

En la figura 4.15 se muestra el paso 2 c) de la prueba, donde se ve el módulo actuador con el led rojo prendido y el pestillo de la cerradura abierta.

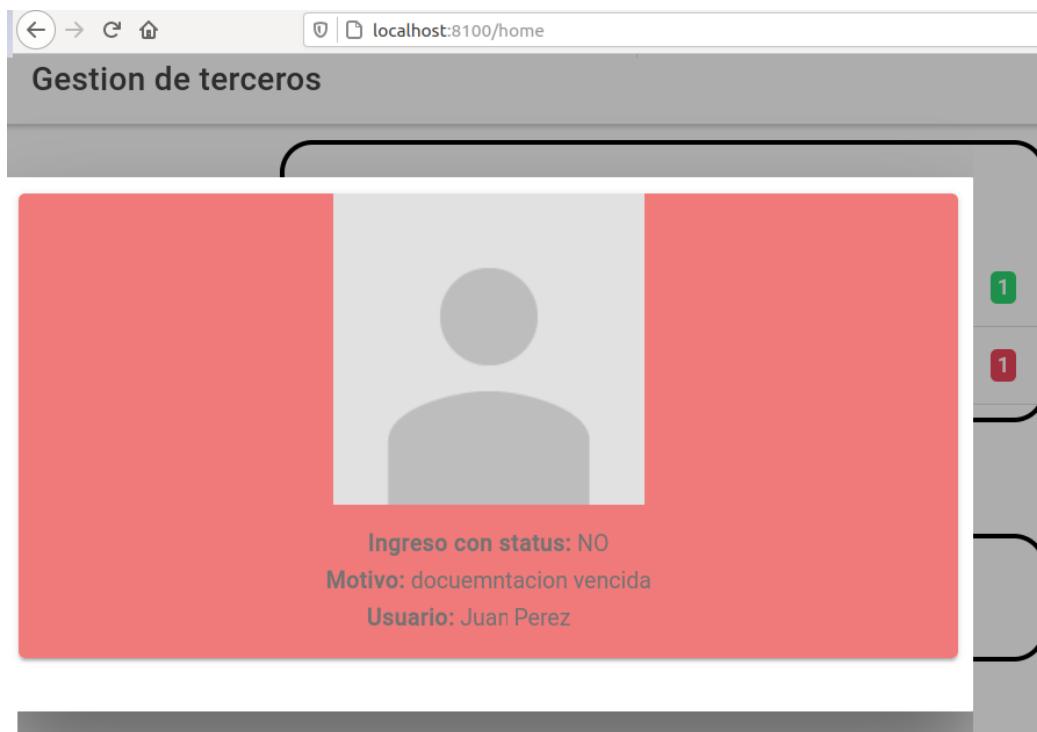


FIGURA 4.14. Visualización de la alerta recibida en pantalla.

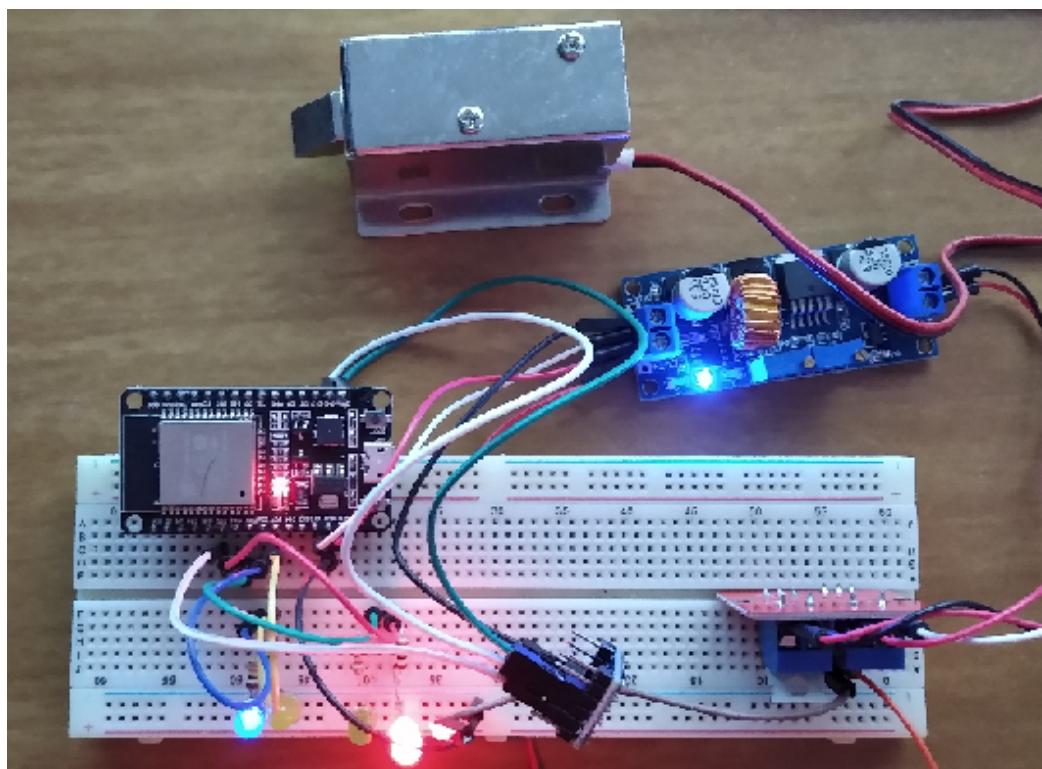


FIGURA 4.15. Accionamiento de módulo actuador y respuesta del módulo.

En la figura 4.16 se ve el paso 2 d) de la prueba, con el email recibido por el usuario con la alerta del intento de ingreso con documentación vencida.

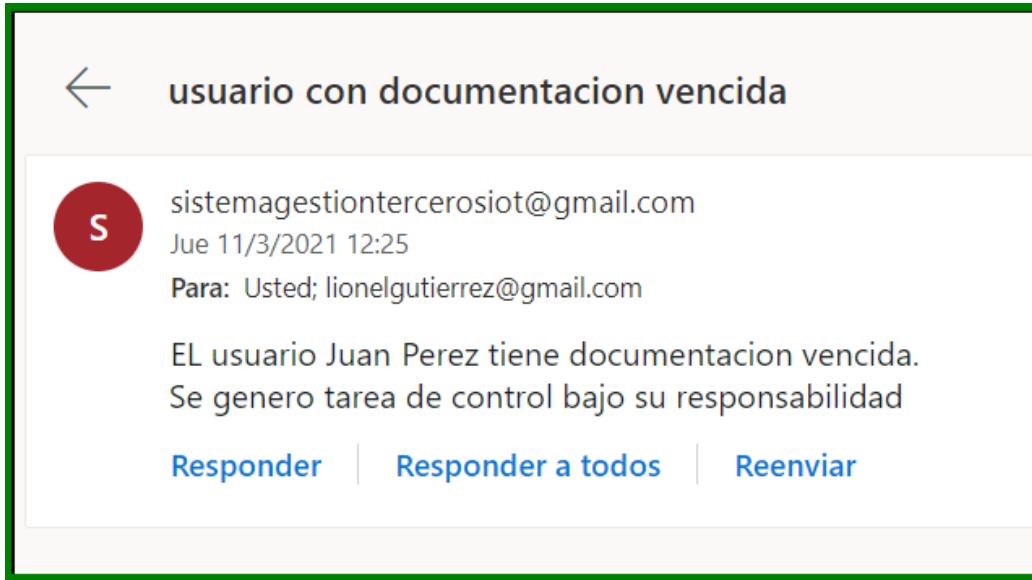


FIGURA 4.16. Email recibido por el usuario con la alerta de intento de ingreso con documentación vencida.

En la figura 4.17 se ve el paso 3 de la prueba, donde el usuario ve la tarea generada por parte del usuario.

The image is a screenshot of a web application titled "Sistema Gestión Terceros". The left sidebar has links for "Principal", "Mis Tareas en curso" (which is highlighted in blue), "Mi Historial Tareas", and "Mi Perfil". At the bottom of the sidebar is a red "CERRAR SESION" button. The main content area is titled "MisTareas Pendientes". It displays a single task card with the following details: "Tarea: Controlar documentacion vencida usuario Juan Perez", "Subtarea: Ajustar documentacion", "Días atraso: 0", and an "Observacion:" field containing a placeholder. There are two buttons at the bottom of the card: "AJUSTAR OBS." and a blue "CERRAR TAREA" button. The URL in the browser bar is "localhost:8100/mis-tareas".

FIGURA 4.17. Pantalla de usuario se sector HESA con la tarea de control generada.

4.5. Comparativa con otras soluciones del mercado

A modo de conclusión de los ensayos realizados, se hace una comparativa contra soluciones similares que existen en el mercado para el control de ingreso. De acuerdo con el análisis comparativo realizado, se destaca el diferencial de este sistema al permitir comunicarse con otros sistemas de terceros y lograr una gestión integral de accesos, con capacidad de generar alertas y tareas de control sobre el proceso.

En la tabla 4.11 se muestra la comparación entre el sistema implementado y soluciones similares del mercado, en la que se puede apreciar el diferencial del sistema desarrollado.

TABLA 4.11. Comparación contra otras soluciones similares del mercado.

Característica	Sistema	Pronext KY800	Samsung H505
Log de ingresos	Si	Si	No
Interfaz a sistemas externos	Si	No	No
Gestión integral de accesos	Si	No	No
Máximo usuarios	Indefinido	500	30
Conectividad/Protocolos	Wi-Fi	Bluetooth	No
Doble factor autenticación	No	No	Si
Tarjetas RFID	Si	Si	Si
Alerta de acceso	Si	Si	Si
Acceso con huella	No	Si	No

Capítulo 5

Conclusiones

5.1. Resultados obtenidos

Se logró cumplir el alcance y objetivo del proyecto. En primer lugar, se implementó el sistema de control solicitado que incluyó la puesta en funcionamiento de un módulo sensor, un módulo actuador y una aplicación web para la gestión de tareas de control y de alertas. En segundo lugar, se sentaron las bases para el agregado de futuros casos de uso, para lo cual se hizo un diseño modular. Esto permitirá al sistema el sensado de datos de nuevos procesos de planta, según el requerimiento 1.1, especificado en la sección 2.5.

Adicionalmente, se incorporó al sistema un módulo para gestionar la autenticación y autorización de los usuarios y el acceso a la API Rest del backend del sistema de modo seguro. Esto permitió cumplir con el requerimiento 1.6, que fue agregado al trabajo durante su desarrollo. Este requerimiento permite independizar al sistema desarrollado del sistema de autenticación de la empresa y lograr una mayor portabilidad, lo que le da la posibilidad a futuro de expandir el mismo a otras empresas.

Finalmente, el trabajo cumplió con el requerimiento de lograr una gestión efectiva de los terceros, al implementar un proceso de control estricto de los requisitos de ingreso. Si bien existen otros sistemas de control de ingreso en el mercado, se logró agregar valor mediante la comunicación del sistema en cuestión con el sistema de control de documentación de terceros sumado a los procesos de alerta y gestión implementados. El trabajo realizado habilita una gestión proactiva, rápida y ordenada de los terceros, de forma de actuar inmediatamente ante problemas de ingresos. Las ventajas obtenidas incluyen:

- Reducir tiempo para la gestión.
- Evitar atrasos en ingresos por falta de ajustes en la documentación.
- Evitar problemas legales ante incidentes del personal externo.
- Evitar el uso de papel y herramientas des-centralizadas para el control y gestión de los terceros por parte de cada sector de la empresa.

Si bien todavía el sistema no se implementó en operativo, con las pruebas realizadas y analizando los ingresos de personal en los últimos dos años, se prevé evitar un 5 % de ingresos incorrectos o con problemas, y reducir los tiempos ante inconvenientes con la documentación en unas 10 horas hombre/mes.

Cabe destacar la importancia de los conocimientos obtenidos a lo largo de la carrera. En primer lugar, fueron muy importantes los aportes de la asignatura de gestión de proyectos. Una buena gestión de proyectos fue fundamental, tanto para lograr una planificación clara que actúe como guía a lo largo de todo el desarrollo, como para minimizar riesgos. En lo particular del trabajo, luego del comienzo del desarrollo se solicitó agregar un nuevo requerimiento al mismo. Dado que se contaba con un diagrama de Gantt [7] detallado y el avance real al momento de la solicitud, se pudo determinar que el nuevo requisito podía cumplirse en tiempo y forma, sin penalizar la fecha de finalización del proyecto. Sin una gestión de proyectos clara, es muy probable que este nuevo requerimiento haya sido rechazado.

Adicionalmente, los conocimientos en desarrollo de aplicaciones multiplataforma permitieron plantear una solución que sea *web responsive* y que a futuro se puede implementar en un entorno mobile con mínimo esfuerzo. También se abre la posibilidad de implementar el sistema en la nube.

5.2. Trabajo futuro

Para la continuidad y mejora de este trabajo se plantean dos líneas de acción.

Como primera línea de acción, referida al trabajo desarrollado, se incluyen:

- Realizar mejoras en seguridad:
 - Agregar comunicación HTTPS entre los diferentes módulos del sistema. De este modo, se asegura que la información viaje encriptada y se evitan ataques del tipo *man in the middle*. Esto permitirá migrar el sistema a la nube, donde las comunicaciones viajan a través de Internet y no son seguras.
 - Agregar un segundo factor de autenticación al sistema. Con el objetivo de evitar que ante pérdidas o robos de la tarjeta RFID de ingreso o duplicación de la misma un atacante pueda ingresar a planta, se plantea la posibilidad de agregar un teclado matricial de forma de requerir además de la tarjeta una clave numérica durante el proceso de ingreso.
- Automatizar tareas de configuración: se analiza implementar una aplicación para poder configurar las acciones del sistema ante las diferentes variantes de entradas (ingreso correcto, ingreso de usuario inactivo, ingreso con documentación vencida). Actualmente esta información se guarda y administra en una base de datos no relacional, por parte del personal de sistemas, que permite definir para cada tipo y valor de entrada un conjunto de acciones de salidas (tareas de control, alertas, emails). Se evalúa desarrollar una aplicación para que el usuario pueda configurar estas salidas y generar diferentes tipos de acción, independizándose del área de sistemas.
- Realizar una prueba de implementación en la nube: utilizar la nube de Azure para probar y asegurar el escalamiento de la solución. Esto permitirá incluir nuevas locaciones o plantas industriales al trabajo, ya sea dentro de la empresa actual o para ser implementado en nuevas empresas.

Como segunda línea de acción, en el marco del proyecto integral de gestión de alertas y procesos, el objetivo es incorporar nuevos procesos y casos de uso al sistema. De hecho, ya fue solicitado un primer caso de uso por parte del laboratorio de metrología de la empresa. El mismo implica el control de temperatura y humedad de dicho laboratorio, para asegurar que ambas variables se encuentren dentro de los límites requeridos y generar alertas en caso de desvíos para poder actuar en consecuencia, manual o automáticamente.

Bibliografía

- [1] Oracle. *Modelo de arquitectura del protocolo TCP/IP.*
<https://docs.oracle.com/cd/E19957-01/820-2981/ipvov-10/>. Ene. de 2010.
(Visitado 12-03-2021).
- [2] Lionel Gutiérrez. *Master Test Plan TP final CeIoT Lionel Gutiérrez.*
<https://docs.google.com/document/d/1-KEsjv7V-AbL9H0XLFSUNzzbdWEjcfD8p6gnV1Ut7A/edit?usp=sharing>. Oct. de 2020. (Visitado 12-03-2021).
- [3] C.B. Jurado. *Diseño Ágil con TDD.*
<https://docs.google.com/document/d/1-KEsjv7V-AbL9H0XLFSUNzzbdWEjcfD8p6gnV1Ut7A/edit?usp=sharing>. Ene. de 2020. (Visitado 12-03-2021).
- [4] Lionel Gutiérrez. *Casos de uso y de test del TP final CeIoT Lionel Gutiérrez.*
https://docs.google.com/document/d/1RDIIGW6ZQfH5MMIm_lq8_HcNIlauIL8euAkz1Hp3-zlKI/edit?usp=sharing. Feb. de 2021. (Visitado 12-03-2021).
- [5] Postman Learning Center. *Managing environments in Postman.*
<https://learning.postman.com/docs/sending-requests/managing-environments/>. Ene. de 2021. (Visitado 12-03-2021).
- [6] Smartbear. *What is an API Endpoint?*
<https://smartbear.com/learn/performance-monitoring/api-endpoints>. Nov. de 2018. (Visitado 12-03-2021).
- [7] Wikipedia. *Diagrama de Gantt.*
https://es.wikipedia.org/wiki/Diagrama_de_Gantt. Oct. de 2020.
(Visitado 07-03-2021).