Universitat Autònoma de Barcelona

Dipòsit digital de documents de la UAB

This is the **published version** of the bachelor thesis:

De León Sosa, Christian Junior; Montón Macian, Màrius, dir. Hodor: Control de accesos motorizados. 2023. (Enginyeria Informàtica)

This version is available at https://ddd.uab.cat/record/280699

under the terms of the BY-NC-ND license

Hodor: Control de accesos motorizados

De León Sosa,J.C

Resumen– Este Trabajo de fin de grado presenta el desarrollo de un sistema de control de accesos motorizados no invasivo, que completa y no anula los sistemas de controles existentes. La innovación de Hodor respecto a los dos sistemas parecidos más vendidos actualmente es la filosofía modular. El sistema está compuesto por una aplicación web construida con *Angular*, un *Back-end* en *Node.js* para interactuar con una base de datos MongoBD y una serie de dispositivos *Raspberry Pi* que funcionan en conjunto. Se comunican mediante el protocolo *MQTT*y realizan funciones como la apertura de puertas, recolección de datos de sensores, apertura mediante análisis facial y detección de instrusos con cámara de seguridad. El sistema tiene aplicaciones reales en el mercado, ya que ofrece una solución eficiente y segura para el control de accesos en diversos entornos, tanto empresariales como particulares.

Palabras clave– Angular, Node.js, MongoDB, MQTT, Raspberry Pi, sensores, acceso motorizado,Python, nodos modulares, seguridad, reconocimiento facial, IoT.

Abstract– This Bachelor's thesis presents the development of a non-invasive motorized access control system that complements existing control systems rather than replacing them. With regards to the two currently best-selling motorized access systems, Hodor's innovation stems from its modular philosophy. The system consists of a web application built with Angular, a Node.js backend to interact with a MongoDB database, and a set of Raspberry Pi devices that work together. They communicate via the MQTT protocol and perform functions such as door opening, sensor data collection, facial recognition-based access, and intruder detection with security camera. The system has practical applications in the market as it offers an efficient and secure solution for access control in a variety of environments, both for corporate and private users.

Keywords– Angular, Node.js, MongoDB, MQTT, Raspberry Pi, sensors, motorized access, Python, modular nodes, security, facial recognition, IoT.

1 Introducción - Contexto

El proyecto de fin de grado titulado *Hodor: Control de accesos motorizado* presenta el desarrollo de un sistema de control de accesos motorizados no invasivo que complementa y mejora los sistemas de control existentes en diversos entornos, tanto empresariales como particulares. El sistema pone a disposición una aplicación web que proporciona una interfaz de usuario cómoda, un backend con gestión de la base de datos y una serie de dispositivos *Raspberry Pi*, que trabajan de manera conjunta en un mismo sistema. Estos nodos no solo permiten realizar aperturas y cierres, sino que también incorporan características adicionales, como desbloqueo facial, monitorización mediante sensores, y control de videos cortos al detectar intrusos. El mayor atributo del proyecto es su potencial de aplicaciones reales en el mercado, debido a la solución eficiente y segura que ofrece. Una de las ventajas destacadas del sistema es su fácil instalación, lo que permite a los usuarios implementarlo de manera sencilla. La filosofía principal del proyecto se basa en tener un sistema totalmente modularizado que trabaje de forma integrada, optimizando así su funcionamiento y rendimiento.

2 Objetivos

De la premisa anterior se podrían extraer gran multitud de objetivos. Por este motivo, es muy importante centrar el proyecto y hacer un análisis previo de cada una de las partes, con el fin de definir las prioridades. La primera diferenciación que hay que realizar es: objetivos académicos (qué quiero aprender) y objetivos de desarrollo (finalización del proyecto).

• E-mail de contacte: 1492721@uab.cat
• Mención realitzada: Ingeniería de computación
• Treball tutoritzat per: Màrius Montón Macian (Microelectrónica y Sistemas Electrónicos)
• Curs 2022/23

El proyecto consta de 5 campos bien diferenciados: *Front-end*, *Back-end* , base de datos, comunicación y programación *Python* de las diferentes *Raspberry Pi*. A raíz de analizar estas partes se han definido:

- Objetivos generales:
  - Definición de manera clara y precisa de los requisitos y necesidades del proyecto.
  - Establecer y definir la metodología y plan de trabajo para alcanzar los objetivos..
  - Diseñar e implementar una interfaz de usuario intuitiva y funcional.
  - Desarrollo de un *Back-end* eficiente y escalable para la correcta gestión y operación del servidor.
  - Implementar un sistema de comunicación entre las distintas partes del sistema.
  - Programación modular *Python* para cada nodo.
  - Diseño y realización de test de funcionamiento.
  - Tener ordenado y almacenado correctamente él código desarrollado
- Objetivos académicos:
  - Aprender a utilizar *Angular* para el desarrollo del *Front-end*.
  - Adquirir conocimientos en *Node.js* para el desarrollo del *Back-end*.
  - Familiarizarme con *MongoDB*para el manejo de la base de datos.
  - Dominar el uso de *MQTT* para la implementación de comunicaciones.
  - Aprender a programar *Raspberry Pi*s utilizando *Python*.
- Objetivos de desarrollo:
  - Desarrollo del *Front-end* utilizando *Angular*.
  - Creación del *Back-end* utilizando *Node.js*.
  - Configuración de una base de datos *MongoDB* para el almacenamiento de datos.
  - Implementación de comunicaciones utilizando MQTT.
  - Programación de *Raspberry Pi*s utilizando *Python*.

Siendo la finalización del proyecto la meta más importante, la planificación del trabajo se ha centrado en el cumplimiento de los objetivos de desarrollo.

3 Estado del arte

Con el creciente interés en la domotización de nuestro entorno, ha surgido una gran variedad de productos comerciales que buscan cubrir dicha necesidad. Este interés no solo existe en entornos particulares, sino que también se extiende al ámbito empresarial. En el caso que nos concierne, los usuarios buscan soluciones que les permitan controlar e interactuar con sus accesos motorizados de manera cómoda y segura, prescindiendo del uso de llaves físicas, teniendo la posibilidad de conocer en todo momento el estado del acceso e interactuando con él a distancia. En el mercado actual, dos empresas destacan en el control de accesos: *Baintex*[1] y *Parkingdoor*[2].

Baintex es una empresa española con sede en Valencia especializada en el desarrollo de sistemas de control de accesos inteligentes. Han desarrollado una cerradura electrónica basada en el protocolo de seguridad TLS y encriptación AES. Su enfoque principal se centra en el uso particular, donde los usuarios pueden interactuar con la puerta a través de una aplicación móvil y una conexión Bluetooth.

Por otro lado, *Parkingdoor* es una empresa dedicada al desarrollo de hardware y software, especializada en el control de accesos automáticos de puertas y barreras en garajes, tanto comunitarios como corporativos. Nuevamente, los usuarios pueden interactuar con el acceso a través de una aplicación móvil con conectividad Bluetooth.

Estos dos sistemas son similares al propuesto, pero tienen filosofías distintas. *Baintex* requiere la instalación de una cerradura electrónica, lo cual puede complicar su puesta en marcha. En cambio, en *Parkingdoor* esto no es necesario, ya que su instalación es similar a la del sistema propuesto. Sin embargo, la diferencia clave que aporta *Hodor* es la utilización de nodos interconectados para la monitorización, así como la posibilidad de utilizar desbloqueo facial en lugar de depender exclusivamente de Bluetooth para realizar aperturas a distancia. También es importante diferenciar que estas dos empresas no buscan la escalabilidad del sistema.

4 Requisitos

Para el correcto funcionamento del sistema es imprecindible establecer los distintos requisitos funcionales y no funcionales extraidos de los objetivos. A continuación, se enumeran dichos requisitos.

4.1 Funcionales

- El sistema debe permitir el sing-in y el login de usuarios.
- Los usuarios deben poder registrar el dispositivo a utilizar (id unico).
- El sistema debe visualizar los dispositivos registrados por el usuario.
- Con el dispositivo registrado, el usuario debe poder realizar apertura/cierre pulsando un botón.
- El usuario debe poder eliminar dispositivos registrados.
- El usuario debe de poder descargar los logs de cada acceso.
- El sistema muestra los datos de los sensores.
- Las *Raspberry Pi*s recogen datos de los sensores, realizan la apertura mediante el desbloqueo facial y gravan videos de detector de intrusos.

4.2 No funcionales

- El sistema debe ser fácil de instalar.
- La interfaz de usuario debe de ser intuitiva y fácil de usar.
- Las comunicaciones se efectúan mediante MQTT.
- El *Back-end* recoge las peticiones para interactuar con la base de datos.
- El sistema muestra los datos de los sensores a tiempo real.
- El sistema debe de ser escalable.
- El sistema debe de cumplir con los estándares de seguridad y protección.

5 METODOLOGÍA

Para el desarrollo del proyecto, se ha seguido una metodología ágil basada en programación extensiva. En colaboración con el tutor, se definieron iteraciones de dos semanas de media, en las cuales se establecieron objetivos y tareas específicas para continuar con el desarrollo. Al finalizar cada iteración, se llevaron a cabo reuniones de seguimiento con el tutor para revisar el progreso y asegurar el cumplimiento de los requisitos, objetivos del proyecto y alternativas en caso de encallamientos. En cada iteración, se definió un conjunto de tareas y se establecieron plazos para su documentación y finalización. La prioridad de las tareas se asignó según los conocimientos previos y la importancia en el desarrollo del dispositivo. Durante las reuniones de seguimiento con el tutor, se presentó el progreso en relación con las tareas y objetivos definidos para la iteración actual. El tutor brindó feedback y realizó las correcciones necesarias para asegurar el cumplimiento adecuado de los objetivos y requisitos del proyecto. Además, se discutieron los siguientes pasos a seguir y se planificaron las próximas iteraciones, teniendo en cuenta los ajustes y mejoras sugeridos. Un aspecto fundamental en la metodología utilizada fue el control de versiones a través de GitHub. Se creó un repositorio para el proyecto, donde se gestionó el código fuente y se efectuaron las fusiones y actualizaciones correspondientes. El mayor beneficio que aportó fue mantener un registro de los cambios y asegurar la integridad del código. La metodología ágil basada en programación extensiva permitió una adaptación flexible a los cambios surgidos, ya sea por problemas o mejores alternativas, y requisitos del proyecto, facilitando la entrega continua de funcionalidades y la detección temprana de posibles problemas. A continuación se proporcionará el marco teórico necesario para comprender los conceptos fundamentales y las bases teóricas relacionadas con el desarrollo del dispositivo. El repositirio del proyecto es el siguiente [3].

5.1 Marco Teórico

5.1.1 Angular

Angular[4] es un framework destinado a la creación de aplicaciones web, desarrollado por Google basado en *TypeScript*. En este momento la última versión estable de *Angular* es *Angular* 15, que se lanzó el 14 del 2022 de diciembre. El pilar principal de *Angular* es basarse en el concepto de componentes reutilizables, los cuales son elementos independientes entre sí, encargados tanto de la interfaz de usuario como de la lógica específica de la aplicación. Estos componentes permiten una arquitectura modular y escalable en el desarrollo de aplicaciones web. Otra ventaja importante de Angular es su capacidad de optimizar el rendimiento de las aplicaciones web. El framework utiliza técnicas como la detección de cambios y la compilación just-in-time (JIT) para garantizar un rendimiento eficiente y una carga rápida de la aplicación.

5.1.2 Node.js

Node.js[5] es un entorno de desarrollo open source para *JavaScript* construido con V8, el cual es el motor de *JavaScript* de Chrome. Está especialmente diseñado para el desarrollo de textitBack-ends, gracias a que permite a los desarrolladores crear aplicaciones fácilmente escalables con una gran velocidad de respuesta. Su principal punto fuerte es el enfoque de tener un solo subproceso con bucle de eventos, lo que lo hace eficiente en el manejo de múltiples solicitudes concurrentes. Otra ventaja de Node.js es su capacidad para escalar horizontalmente. Esto significa que se puede distribuir la carga de trabajo entre múltiples instancias de Node.js, lo que permite manejar grandes volúmenes de tráfico y proporcionar una alta disponibilidad en entornos de producción.

5.1.3 MongoDB

MongoDB[6] es una base de datos no relacional open source, que se ha convertido rápidamente en todo un referente para el almacenamiento y procesamiento de datos no estructurados. Destaca por su gran flexibilidad, ya que utiliza un modelo de datos basados en documentos de tipo JSON en lugar de tablas. Esta misma flexibilidad dota de escalabilidad horizontal, distribuyendo los datos en múltiples servidores para poder manejar un gran volumen de datos con altas cargas de trabajo. MongoDB también cuenta con una amplia comunidad de desarrolladores y una gran cantidad de recursos disponibles. Esto incluye documentación detallada, tutoriales y ejemplos de código que facilitan el aprendizaje y el uso del sistema. Además, MongoDB se integra bien con otros lenguajes y frameworks populares, lo que lo hace compatible con una amplia variedad de aplicaciones y entornos de desarrollo.

5.1.4 MQTT

MQTT [7] es un protocolo de mensajería máquina (M2M) diseñado para enviar mensajes entre dispositivos IoT a un servidor centralizado. Es un protocolo ligero y eficiente, lo que lo hace que necesite recursos mínimos para funcionar, provocando que pueda implementarse en casi cualquier microcontrolador, por simple que sea. Funciona mediante eventos, así que se mantiene un volumen de transmisión mínimo, organizádose en publicador-subscriptor, donde los publicadores, publican mensajes y los subcriptores lo reciben. MQTT ha ganado popularidad en el ámbito de la IoT debido a su simplicidad, eficiencia

y capacidad para funcionar en entornos con recursos limitados. Además, cuenta con una amplia gama de implementaciones y bibliotecas disponibles en diversos lenguajes de programación, lo que facilita su adopción y uso en diferentes plataformas y sistemas.

5.1.5 Python

Python[8] es un lenguaje de alto nivel interpretado y ampliamente usado en gran variedad de campos. Cuenta con grana documentación y una gran comunidad de desarrolladores. Sigue la filosofía de hacer hincapié en la legibilidad de su código, para la programación de cualquier tipo de aplicación. A nivel profesional, un gran número de empresas lo utiliza, tales como Instagram, Netflix, etc.

6 Diseño del sistema

En esta sección, se presentará el diseño integral del sistema, el cual consta de dos componentes fundamentales: el software, encargado del procesamiento de datos y la lógica del sistema, y el hardware, que proporciona la infraestructura necesaria para ejecutar el sistema. A continuación, se mostrará una descripción detallada de cada una de estas partes. En la figura 1 se visualiza un esquema de sistema.

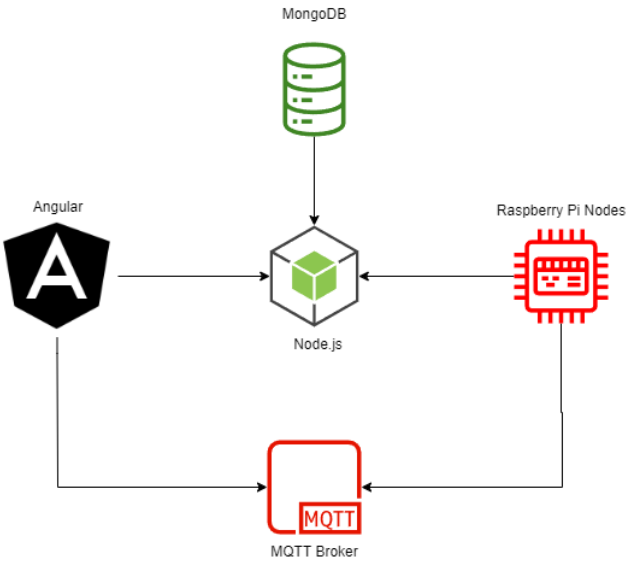


Fig. 1: Esquema del Sistema

6.1 Arquitectura Hardware

El hardware del sistema proporciona la infraestructura necesaria para ejecutar y soportar las funcionalidades de los nodos del sistema, haciéndose uso de componentes electrónicos específicos y realizándose los diseños adecuados para cada nodo. La selección de los componentes hardware se ha basado en los requisitos del sistema y prestando especial atención a la capacidad de respuesta, la escalabilidad y la confiabilidad del dispositivo. La selección de los componentes se visualiza en la tabla mostrada a continuación.

Los esquemas electronicos son muy simples, ya que las distintas placas se conecnctan unicamente con las camaras

TAULA 1: Tabla de componentes

Nombre	Descripción	Datasheet
Orange Pi 5	Placa de desarrollo	[9]
RaspPi 3 B+	Placa de desarrollo	[10]
RaspPi Zero W	Placa de desarrollo	[11]
Camara AZDelivery	Cámara	[12]
ADXL345	Acelerómetro	[13]
IM Gebildet	IM	[14]
KY-019	Rele	[15]

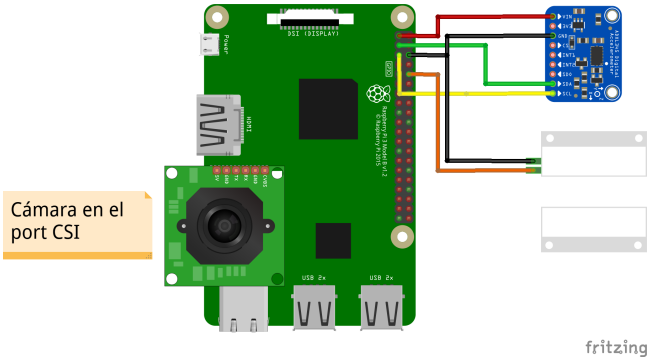


Fig. 2: Esquema Raspberry 3B+

por los conectores CSI y DSI, los sensores mediate *GPIO* e *I2C* y los actuadures por *GPIO*. En la figura 2 se observa el esquema de la Raspberry Pi 3B+, que es algo más complejo, con los distintos sesores. Los esquemáticos de las demás placas están situados en el anexo [A.1].

6.2 Arquitectura Software

El software del sistema es el encargado de manejar el procesamiento de datos, lógica de control y la interfaz de usuario. Para desarrollar todas las partes se ha utilizado el *IDE VSCode*. Los componentes software son:

- *Front-end*: Desarrollado en *Angular*.
- *Back-end*: Desarrollado con *Node.js*, encargado de los End Points para interactuar con la base de datos.
- Base de datos: Desarrollado con MongoDB, encargada de almacenar los datos de usuario y de los dispositivos.
- Comunicación: *MQTT* es el protocolo de comunicación utilizado entre los nodos y el *Front-end*.
- Procesamiento: Realizado en *Python*. Dependiendo del nodo realizará una tarea o otra.
  - *RaspPi Zero W*: Encargada de interactuar con el panel de maniobras del acceso.
  - *Orange Pi 5*: Encargada de realizar el reconocimiento facial.
  - RaspPi 3 B+: Encargado recoger los datos de los sensores de la puerta y de implementar el sistema “Tesla”, grabar pequeños videos en caso de detección de intrusos.

6.3 Integración

La integración es un aspecto fundamental del sistema, ya que se compone de cinco grandes bloques: *Front-end*, textitBack-end, Base de datos, *MQTT* y *Raspberry Pi*s. La comunicación efectiva entre estos componentes es esencial para garantizar un funcionamiento coordinado y sin errores.

En primer lugar, todas las comunicaciones que involucran al *Back-end* se realizan a través de request HTTP. Esto permite la transferencia de datos y la interacción entre el cliente y el servidor de manera eficiente y estandarizada.

En cuanto a la comunicación entre el *Back-end* y la base de datos, se utiliza la tecnología de Moongose [16]. Moongose proporciona una interfaz sencilla y conveniente para interactuar con la base de datos MongoDB, permitiendo el almacenamiento y recuperación de datos de manera eficiente y segura.

Por último, el *Front-end* y los nodos se comunican entre sí mediante el protocolo *MQTT*. Este protocolo ligero y eficiente es ideal para la comunicación entre dispositivos IoT, ya que requiere recursos mínimos y permite una fácil implementación en microcontroladores. La comunicación basada en *MQTT* permite una transmisión de datos eficiente y una mayor escalabilidad.

En la Figura 3 se presenta un diagrama que ilustra la visión global de la integración del sistema, mostrando la interconexión entre los diferentes bloques y cómo se comunican entre sí.

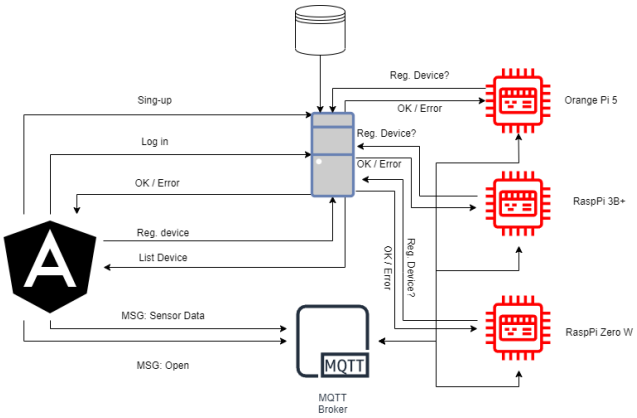


Fig. 3: Comunicaciones del sistema

7 Implementación

A continuación se muestra un resumen de lo que ha consistido cada bloque del proyecto.

7.1 Base de Datos - MongoDB

La implementación del sistema incluye el diseño y la configuración de la base de datos utilizando *MongoDB*. Además, se ha utilizado la herramienta *Mongo-express*[17] para visualizar y administrar la base de datos de forma gráfica. La base de datos *MongoDB* ha sido dockerizada para facilitar su despliegue y gestión. El contenedor de *MongoDB* se encuentra alojado en el puerto 8081, mientras que *Mongo-express* se ha alojado en el puerto 27017. Esta configuración permite acceder a la base de datos y realizar operaciones de administración a través de una interfaz gráfica, facilitando así su visualización. Se ha implementado la persistencia de datos para evitar la pérdida de contenido en las colecciones. Esto garantiza que la información almacenada en la base de datos se mantenga incluso después de reiniciar el sistema o detener el contenedor de *MongoDB*. En el apéndice adjunto se encuentra el archivo ”docker-compose.yml”, el cual contiene la configuración necesaria para desplegar los contenedores de *MongoDB* y *Mongo-express*, junto con sus respectivas dependencias y configuraciones[A.2]. Las colecciones son: Usuarios, Dispositivos y Logs.

7.2 Front-end - Angular

Se ha utilizado el framework *Angular* para implementar el desarrollo del *Front-end* de la aplicación web. Para asegurar una estructura organizada del código, se ha seguido el patrón de diseño MVC (modelo, vista y controlador), además de utilizar un directorio adicional para los servicios, lo que contribuye a un código más ordenado y mantenible. Se han implementado las vistas de ”sing-up” y ”login” para la identificación de usuarios. Para validar los correos electrónicos, se ha utilizado la herramienta MailTrap[18], que permite simular el envío y recepción de correos electrónicos en entornos de desarrollo. A continuación, se muestra un panel de control (”dashboard”) que presenta los dispositivos registrados. Se incluye una tabla desplegable que permite seleccionar con cuál dispositivo se desea interactuar y muestra los datos en tiempo real. Adicionalmente, si se ha detectado una intrusión, aparecen botones de descarga del video grabado. Además, se ha agregado una vista adicional en el menú lateral para registrar nuevos dispositivos. Se muestran todos los dispositivos registrados hasta el momento, y se proporciona un botón para descargar el registro de actividades y eliminar dispositivos registrados. Todas las interacciones con la base de datos se realizan a través de solicitudes HTTP a los End-Points correspondientes. Una vez que un dispositivo se ha registrado, se habilita la opción de interactuar con él mediante el broker MQTT. En cuanto al aspecto gráfico de la aplicación, se ha utilizado las librerias Bootstrap[19] y *Angular* Material[20], que ofrecen una amplia gama de estilos para garantizar un diseño visualmente atractivo y responsivo.

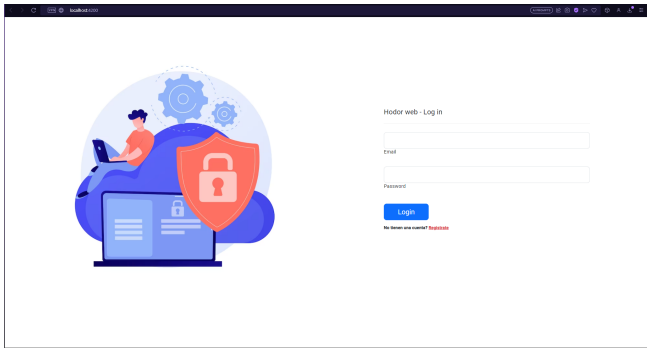


Fig. 4: Vista Login Angular

En la figura 4, se visualiza la vista del Login efectuada con Boostrap.

7.3 Back-end - Node.js

Se ha diseñado un servidor de *Back-end* en *Node.js* siguiendo el patrón de diseño MVC. En este servidor, se han implementado los endpoints necesarios para el sign-in y log-in, así como para los datos del dispositivo y los registros de actividad (logs). Para interactuar con las colecciones de los dispositivos y logs, se ha implementado un middleware que cumple con la funcion de realizar la comprobación de si el usuario está autenticado. Además, se han implementado todos los mensajes de bad requets, como los códigos 200 (éxito), 500 (error interno del servidor), entre otros, para manejar adecuadamente las solicitudes y respuestas. Para establecer la comunicación entre el *Back-end* y el *Front-end*, se ha utilizado el módulo Cors[21]. Este módulo se encarga de habilitar y gestionar automáticamente las solicitudes de recursos cruzados (Cross-Origin Resource Sharing, CORS), permitiendo así que el *Front-end* pueda interactuar con el *Back-end* de forma automática sin ninguna especificación extra. Para el almacenamiento de los videos e imágenes, se han habilitado dos midelware adicionales con *Multer* [22], el cual es un paquete de npm, que permite guardar archivos en el back-end. Las imágenes se almacenan en un subdirectorio con el id del usuario. Esta se utiliza para el desbloqueo fácil y únicamente abra una guardada. El funcionamiento del almacenaje de los videos es similar, con la peculiaridad de que únicamente como máximo se almacenaran 5, eliminando elmás antiguo y guardando el nuevo.

Las rutas que se han habilitado han sido las siguientes:

- */account*: Encargada de la colección de usuarios.
  - *POST: /signup*: Registro de usuario
  - *GET: /confirm/:token*: Comprobación de confirmación de usuario.
  - *POST: /login*: Login usuario.
  - *POST: /current_user*: Información del usuario (ruta protegida con un midelware para saber si está logeado el usuario).
  - *Put: /:id*: Subir imagen del usuario (ruta protegida con un midelware para saber si está logeado el usuario).
- */device*: Encargada de la colección de dispositivos(todas las rutas protegida con un midelware para saber si está logeado el usuario)..
  - *POST: /regdvc*: Registro de dispositivo.
  - *GET: /*: Return la información de todos los dispositivos.
  - *Get: /:id*: Return la información de un dispositivo.
  - *PUT: /:id*: Actualizar información del dispositivo.
  - *DELETE: /:id*: Eliminar dispositivo.
- */idDvc*: Ruta no protegida para las Raspberry Pis.
  - *GET: /*: Return del id del dispositivo.
- */videos*: Ruta de gestión de videos
  - *POST: /uploadVideo*: Guardar videos en el Back-End.
  - *GET: /infoVideos*: Return información videos del usuario (ruta protegida con un midelware para saber si está logeado el usuario).
  - *GET: /dowloadVideos*: Ruta de descarga de un video (ruta protegida con un midelware para saber si está logeado el usuario).
- */logs*: Encargada de la colección de Logs.
  - *POST: /savelog*: Guardar logs del usuario.
  - *GET: /returnlog*: Descargar logs de usuario.

.

7.4 Raspberry Pi - Python

Se ha desarrollado un código *Python* único que es compatible con diferentes placas de desarrollo. Mediante un enfoque modular, cada placa realiza su función asignada cargando los módulos correspondientes. Esto permite una flexibilidad y reutilización del código en diferentes dispositivos. Antes de que un nodo realice cualquier acción, se verifica si ha sido registrado en una base de datos por un usuario. Para facilitar este proceso, se ha habilitado un punto final (end-point) que no requiere verificación de usuario logueado. Esto permite una fácil integración y registro de los nodos en el sistema. Para interactuar con las cámaras, se ha requerido habilitar la interfaz en cada una de las placas que las utilizan. Además, se ha tenido que habilitar la interfaz *I2C* para facilitar la comunicación con los diferentes sensores y actuadores. Para lograr esto, se han utilizado bibliotecas específicas desarrolladas tanto por la comunidad como por la empresa fabricante de estos componentes. Estas bibliotecas proporcionan una capa de abstracción que faclita la interacción con los dispositivos y permite un control sin errores de los mismos. En resumen, el enfoque modular del código *Python* permite su reutilización en diferentes placas de desarrollo, garantizando un diseño flexible y escalable. La verificación de registro en la base de datos y el uso de puntos finales sin autenticación facilitan la integración de nuevos nodos.

Para la detección facial se utilizará la librería CV2 [23], la cual es una biblioteca de programación de visión por computador de código abierto, desarrollado por Intel. CV2 utiliza un algoritmo de detención de objetos basados en características, como el clasificador en cascada Haar, para reconocer las caras en imágenes. Este clasificador es un modelo de aprendizaje automático que se entrena con un gran numero de rostros y no rostros para detectar patrones comunes de características fáciles. El proceso comienza con la conversión de la imagen a escala de grises y la segmentación de la imagen en múltiples regiones de interés (ROI). A continuación, se aplican filtros de características para cada ROI para detectar la presencia o ausencia de patrones faciales, como los bordes de los ojos, nariz y boca. Una vez que se detectan las ROI que contienen caras, se pueden aplicar técnicas adicionales de reconocimiento fácil para identificar a la persona en la imagen, como la extracción de características faciales y el emparejamiento con una base de datos de rostros conocidos.

7.5 MQTT - Mosquitto

Se ha llevado a cabo la implementación de *MQTT*(Message Queuing Telemetry Transport) utilizando una imagen dockerizada del *Eclipse Mosquitto*[24], un bróker de código abierto ampliamente utilizado en el ámbito de la mensajería IoT, sindo de esta manera una implementacion muy sencilla. La configuración del bróker se ha simplificado al habilitar el protocolo WebSocket (ws) para permitir conexiones web sin requerir la validación de credenciales de usuario. Esto proporciona una mayor flexibilidad y facilidad de acceso para las aplicaciones web que se conectarán al bróker. Para establecer la conexión entre mi aplicación web y el bróker *MQTT*, he utilizado la biblioteca *ngx-mqtt*[25] en el lado del cliente. Esta librería pone a disposición una interfaz sencilla y eficiente para interactuar con el bróker *MQTT* desde una aplicación web, permitiendo la suscripción a temas específicos y el envío de mensajes. Además, para desarrollar un cliente *MQTT* en *Python*, he empleado la biblioteca "paho-mqtt" [26], una opción popular y bien documentada que brinda las funcionalidades necesarias para crear un cliente *MQTT* robusto y confiable. En el apéndice adjunto se encuentra el archivo "docker-compose.yml", el cual contiene la configuración necesaria para crear y ejecutar el entorno *Docker* que aloja el bróker *MQTT* [A.3].

8 PRUEBAS Y VALIDACIÓN

Durante el proceso de desarrollo, se han llevado a cabo diversas pruebas y pruebas de validación para asegurar el correcto funcionamiento del sistema. Estas pruebas se han diseñado para someter al sistema a una amplia variedad de situaciones, incluyendo casos normales y anomalías.

8.1 Test de Conexiones

Durante el proceso de pruebas y testeo, se ha tenido en cuenta un escenario en el cual el sistema pueda enfrentar condiciones de red inestable o de baja potencia. Para simular estas condiciones, se ha ubicado el conjunto de nodos a una distancia considerable de una fuente de señal WiFi confiable, de manera que se puedan experimentar situaciones donde la conectividad sea deficiente. Dado que el sistema se basa en la tecnología *MQTT*, se ha evaluado específicamente cómo se comporta en estas condiciones adversas. En tales circunstancias, puede ocurrir que algunos mensajes no se publiquen o se lean correctamente debido a la falta de una conexión estable. Sin embargo, es importante destacar que estos incidentes no representarían un riesgo para la seguridad, como aperturas o cierres involuntarios. A pesar de la posibilidad de que algunos mensajes no se entreguen o se pierdan en un entorno de red inestable o de baja potencia, *MQTT* cuenta con mecanismos de tolerancia a fallos. Esto implica que el propio protocolo pueda manejar de manera adecuada tales situaciones, evitando que se produzcan consecuencias graves o errores críticos en el funcionamiento general. En conclusión, dado que puede haber casos donde algunos mensajes no se publiquen o no se lean correctamente debido a la falta de conexión estable, se aconseja instalar siempre que sea posible Hodor lo más cercano posible a una red WiFi.

8.2 Test nodo faltantes

Se ha tenido en cuenta la posibilidad de fallos o averías en uno o varios nodos. Estas situaciones pueden ocurrir debido a un mal funcionamiento, rotura o vandalización de los nodos. Para garantizar el correcto funcionamiento del sistema, se identificó el nodo crítico responsable de las operaciones de apertura y cierre, el cual es el nodo conectado al panel de maniobras (*Orange Pi 5*). En caso de que se produzca la pérdida del nodo crítico, el sistema no podrá realizar las operaciones de apertura y cierre del acceso. Esto implica que, sin estos nodos funcionando correctamente, el acceso quedará inaccesible y no se podrán realizar las acciones correspondientes. Sin embargo, si se pierde uno de los otros nodos restantes, ya sea el nodo de sensores, el nodo encargado de grabar en caso de detección de intrusos o el del desbloqueo facial, el sistema seguirá estando disponible y funcionando. Aunque se perderá la funcionalidad específica relacionada con ese nodo particular, el resto de las características y operaciones del sistema seguirán operativas. En resumen, durante las pruebas del sistema se ha evaluado la posibilidad de fallos o averías en los nodos. Se ha identificado que la Orange Pi 5 es el nodo crítico para las operaciones de apertura y cierre, y se ha considerado el impacto de la pérdida de este nodos en el funcionamiento general del sistema.

8.3 Test desbloqueo facial

En las pruebas del sistema de desbloqueo facial se ha evaluado el grado de precisión para identificar correctamente las caras. Para testear esta caso, se han utilizado fotos tomadas desde diferentes ángulos y con diferentes niveles de iluminación. Este desafío es fácilmente solucionable aplicando un procesamiento de imágenes en el momento de capturar la foto.

Además, se ha examinado el grado de precisión para evitar confusiones entre diferentes caras. Si bien se ha logrado un alto nivel de precisión, es importante destacar que no se ha alcanzado una perfección absoluta. Esto se debe principalmente a que para obtener un grado de precisión perfecto, sería necesario utilizar modelos de inteligencia artificial demasiado pesados para ejecutarse en una *Raspberry Pi* y obtener resultados en tiempo real. Para superar esta limitación, se ha empleado la API de Video Intelligence [27] de Google Cloud. Al aprovechar esta API, se pueden realizar tareas de análisis de video y procesamiento de imágenes en la nube, lo que permite realizar cálculos y reconocimiento facial más complejos y precisos. Esto ha permitido mejorar el grado de precisión del sistema sin comprometer el rendimiento de la *Raspberry Pi*.

8.4 Test sistema completo

Por último, se llevó a cabo la prueba de instalar el sistema en un acceso motorizado real con el objetivo de poner a prueba todas sus funcionalidades en un entorno físico. Esta prueba se realizó con el objetivo de comprobar el rendimiento en el mundo real y validar su funcionamiento en condiciones de uso. Durante este test, se pusieron a prueba la totalidad de las funcionalidades del sistema, incluyendo el desbloqueo facial, la detección de intrusos, el registro de eventos y

cualquier otra característica relevante. Se revisó que todas las operaciones se ejecutaban correctamente. El resultado de esta prueba fue satisfactorio, lo que indica que el sistema cumplió con las expectativas y logró funcionar de manera efectiva en la puerta real. Se validó que todas las funcionalidades se compontaron de acuerdo a lo esperado y que el sistema fue capaz de proporcionar la seguridad y el control de acceso deseado. En resumen, la prueba de instalar el sistema en una puerta real resultó satisfactoria, demostrando que el sistema es capaz de funcionar de manera efectiva y cumplir con los requisitos establecidos. Esto brinda confianza en la viabilidad y eficacia del sistema en situaciones reales de implementación.

9 RESULTADOS

El resultado obtenido después del desarrollo del sistema de Hodor ha sido muy satisfactorio. Se ha logrado implementar un sistema completamente funcional y altamente modularizado, lo que ha permitido una mayor flexibilidad y facilidad de mantenimiento. El sistema se ha diseñado de manera eficaz, con una arquitectura escalable y de alto rendimiento.

Una de las fortalezas destacadas es la completa implementación de un *Front-end* interactivo, que proporciona una experiencia fluida al interactuar con los nodos de un dispositivo específico. Además, se ha integrado un sistema de autenticación de usuarios propio, para garantizar la seguridad de acceso a la plataforma. La interfaz ha sido diseñada con la idea principal para ser intuitiva, poniendo un enfoque especial en la presentación clara de la información, mejorando la usabilidad y la comprensión de los datos, buscando así maximizar la experiencia de usuario.

En cuanto al *Back-end* , se ha realizado una implementación eficiente para procesar todas las peticiones HTTP necesarias y permitir la interacción con la base de datos. Este componente ha sido diseñado de manera versátil, lo que lo hace independiente de la plataforma desde la cual se realizan las peticiones. Además, se ha dedicado especial atención en el diseño para su optimización, lo que garantiza un buen rendimiento rendimiento incluso bajo cargas de trabajo intensivas. Se han realizado pruebas de estrés de peticiones, para obtener metricas de velocidad de respuesta bajo condiciones desfavorables.

La elección de MongoDB como base de datos ha sido un acierto, ya que ha brindado una gran flexibilidad en el manejo de grandes volúmenes de datos. La estructura flexible de documentos de MongoDB ha facilitado el almacenamiento y la gestión eficiente de los datos necesarios para el sistema. Aunque se ha demostrado que la base de datos es altamente escalable, es importante mencionar que, en comparación con MySQL, algunas consultas pueden presentar una ligera diferencia en el tiempo de ejecución. Sin embargo, estas diferencias se ven compensadas por la mayor flexibilidad y capacidad de adaptación que ofrece MongoDB.

En términos de rendimiento, el broker Mosquitto ha demostrado una capacidad excepcional para gestionar un gran tráfico de mensajes. La plataforma ha sido probada en diferentes escenarios y ha mantenido una excelente estabilidad y rendimiento incluso en situaciones de alta demanda. Las métricas de rendimiento, como la tasa de transferencia de mensajes y la latencia, han sido constantemente monitoreadas y se han cumplido los estándares establecidos.

Por último, se ha logrado un código modularizado y organizado. Esta estructura facilita la mantenibilidad y la escalabilidad del sistema, permitiendo que futuras modificaciones y la implementación de nuevas funcionalidades sean más sencillas y rápidas. Con el objetivo de mejorar la legibilidad del sistema, se ha seguido la buena práctica de poner comentarios claros.

En resumen, el resultado obtenido del sistema de Hodor es un logro notable, con una implementación completa, una arquitectura escalable y un diseño atractivo. Se han logrado los requisitos definidos.

9.1 Implementación Real

Para poner a prueba los resultados obtenidos y evaluar la facilidad de uso del sistema, se proporcionaron instrucciones mínimas de instalación y uso a un usuario que no estuvo involucrado en el desarrollo. El sistema se mantuvo instalado durante un fin de semana completo.

El feedback recibido fue muy positivo, ya que encontró la instalación y la interacción con el sistema muy sencillas. Se adaptó fácilmente al uso del sistema. Como aspecto negativo, expresó la necesidad de desarrollar una aplicación móvil para facilitar la comunicación de apertura y cierre. Por último, manifestó que, si fuera un producto comercial, utilizaría el sistema para la gestión de los accesos de su casa y taller.

10 CONCLUSIONES

Este proyecto a nacido del puro interes en aprender y tabajar con las herramieentas de *Angular*, *Node.js*, *Python* y *MQTT*. Estas son ampliamente utilizadas en el ámbito laboral, lo que me motivó a aprender y comprender su funcionamento, para mi desarrollo como ingeniero.

Durante el desarrollo del proyecto, he comprendido la importancia de seguir patrones de diseño y escribir código modularizado, especialmente al trabajar en proyectos de gran escala. Una buena estructura y organización del código resultan fundamentales para facilitar futuros cambios y añadir nuevas funcionalidades de manera eficiente. La facilidad de desplegar y gestionar un broker *MQTT* me ha sorprendido gratamente, lo que me ha permitido comprender por qué es tan utilizado en entornos *IoT*. Esta experiencia me ha brindado un conocimiento práctico invaluable sobre las ventajas y el funcionamiento de este protocolo de comunicación.

Si bien mi experiencia previa se limitaba principalmente a *Python*, me he visto en la necesidad de adquirir nuevos conocimientos en varios aspectos. No solo me he familiarizado con los diferentes campos involucrados en el proyecto, sino que también he aprendido a utilizar herramientas externas, como *Insomnia*[28] para la validación de End points y *MQTT Explorer*[29] Explorer para verificar el correcto funcionamiento del broker. Además, he considerado esencial dockerizar cada una de las partes del proyecto con el fin de facilitar el despliegue en la nube. La utilización de contenedores ha demostrado ser una

práctica eficiente y escalable para administrar y distribuir las diferentes componentes del sistema. Originalmente, tenía previsto utilizar placas de desarrollo *Raspberry Pi Zero W* debido a su bajo costo y tamaño compacto. Sin embargo, debido a las condiciones del mercado, me he enfrentado a dificultades para adquirirlas, lo que me ha llevado a buscar alternativas viables para implementar el proyecto.

En el transcurso de este proyecto, se marcaron una serie de objetivos, generales, académicos y de desarrollo.

- Definición de requisitos: Se realizó un análisis detallado con ayuda del tutor, lo cual permitió comprender que necedades iban a surgir en el desarrollo.
- Metodología: Se implementó una metodología ágil (programación extensiva) para realizar reuniones periódicas después de cada esprint. Este objetivo ha sido uno de los más difíciles, debido a la dificultad de estimar el tiempo de desarrollo de las tareas.
- Interfaz: Se ha implementado correctamente una interfaz de usuario con *Angular*.
- *Back-End*: Se ha implementado correctamente un *Back-End* con *Node.js*.
- Comunicación: Se ha implementado correctamente un broker *MQTT*, el cual realiza todas las comunicaciones entre las *Rasperry Pis* y el *Front-End*.
- Plan de test: Cumplimento con todos los test de robustez e integridad del sistema.
- Código ordenado: debido a la gran cantidad de código desarrollado y distintos lenguajes se ha implementado patrones de diseño como MVC y se ha almacenado en un repositorio de *GitHub*.

En conclusión, he invertido numerosas horas en revisar y mejorar el código desarrollado, lo que me ha llevado a la creencia de que este sistema podría ser comercializable en un futuro. La dedicación en la revisión y mejora continua del código ha permitido obtener un producto robusto y de calidad que podría ser de interés para potenciales clientes o usuarios finales.

AGRADECIMIENTOS

Quiero expresar mis agradecimientos a las personas que me han sido fundamentales en la realización de este proyecto.

A mi Tutor, Màrius Montón Macian, por su mentoraje como experto en sistemas embebidos. Su conocimiento y sugerencias han sido de ayuda para plantear, desencallar y guiar el desarrollo del TFG.

A mi familia, especialmente mi madre, Denisse Sosa Rodríguez, por su apoyo incondicional a lo largo de mis estudios académicos. Su positividad y consejos han sido fundamentales para superar este desafío.

A mis amigos, especialmente a Luis Cobo, quien ha sido el artífice de la idea principal del proyecto, así como aportando feedback.

Por último y no menos importante, mi pareja, Paulina Olender, lingüista, la cual no solo me ha dado ánimos, sino que también, ha participado de forma activa en el proyecto, haciéndome correcciones de estilo en los informes entregados.

REFERENCES

[1] https://www.baintex.com/es/kit-easy-parking-automatizar-puerta-garaje

[2] https://parkingdoor.com

[3] https://github.com/subjectE25/Hodor-TFG

[4] https://angular.io

[5] https://nodejs.org/es

[6] https://www.mongodb.com

[7] https://mqtt.org

[8] https://www.python.org

[9] https://tinyurl.com/2tw7r3fw

[10] https://docs.rs-online.com/a608/A700000007750677.pdf

[11] https://cdn-learn.adafruit.com/downloads/pdf/introducing-the-raspberry-pi-zero.pdf

[12] https://tinyurl.com/4zberan9

[13] https://www.analog.com/media/en/technical-documentation/data-sheets/adxl345.pdf

[14] https://www.verical.com/datasheet/adafruit-proximity-375-5038403.pdf

[15] https://datasheetspdf.com/pdf-file/1402030/Joy-IT/KY-019/1

[16] https://mongoosejs.com

[17] https://tinyurl.com/mwzrxkax

[18] https://mailtrap.io

[19] https://getbootstrap.com

[20] https://material.angular.io

[21] https://www.npmjs.com/package/cors

[22] https://www.npmjs.com/package/multer

[23] https://pypi.org/project/opencv-python/

[24] https://mosquitto.org

[25] https://www.npmjs.com/package/ngx-mqtt

[26] https://pypi.org/project/paho-mqtt/

[27] https://cloud.google.com/video-intelligence?hl=es

[28] https://insomnia.rest

[29] http://mqtt-explorer.com

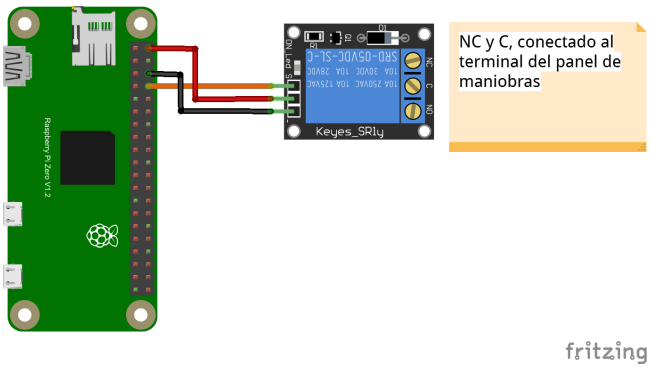


Fig. 5: Esquematico Raspberry Pi Zero W



Fig. 6: Esquematico Orange Pi 5

APENDICE

A.1 Esquematicos

A.2 Docker-compose.yml - MongoDB / Mongo-Express

```
version: '3.1'

services:
  mongo:
    image: mongo
    container_name: mongo
    restart: always
    ports:
      - "27017:27017"
    volumes:
      - ~\Desktop\Hodor-TFG\BBDD\data:/data/db
    networks:
      - mongo-network

  mongo-express:
    image: mongo-express
    container_name: mongo-express
    ports:
      - 8081:8081
    depends_on:
      - mongo
    environment:
      ME_CONFIG_BASICAUTH_USERNAME: root
      ME_CONFIG_BASICAUTH_PASSWORD: root
      ME_CONFIG_MONGODB_PORT: 27017
      ME_CONFIG_MONGODB_ADMINUSERNAME: root
      ME_CONFIG_MONGODB_ADMINPASSWORD: root
    links:
      - mongo
    networks:
      - mongo-network

networks:
  mongo-network:
    driver: bridge
```

A.3 Docker-compose.yml - Mosquitto

```
version: "3"
services:
  mosquitto:
    image: eclipse-mosquitto:latest
    container_name: mosquitto
    environment:
      - TZ=Europe/Madrid
    volumes:
      - ~\Desktop\MQTT\config:/mosquitto/config
      - ~\Desktop\MQTT\data:/mosquitto/data
      - ~\Desktop\MQTT\log:/mosquitto/log
    ports:
      - 1883:1883
      - 9001:9001
```