Automatización del Seguimiento Docente
Systematizing Attendance Tracking

Trabajo de Fin de Grado
Curso 2023–2024

Autores

Helena Antón Navarro
Javier Galdo Martínez
Jaime González Fábregas

Directores

José Ignacio Gómez Pérez
Iván Martínez Ortiz

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Automatización del Seguimiento Docente
Systematizing Attendance Tracking

Trabajo de Fin de Grado en Ingeniería Informática

Autores
Helena Antón Navarro
Javier Galdo Martínez
Jaime González Fábregas

Directores
José Ignacio Gómez Pérez
Iván Martínez Ortiz

Convocatoria: Junio 2024

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

26 de Mayo de 2024

Dedicatoria

A Laura

A Marián

A Juan

Agradecimientos

A Enrique, por el diseño y producción del encapsulado para los nodos IOT. A Iván y Nacho, por toda la ayuda que nos han dado durante el proyecto, y lo mucho que nos han enseñado.

Resumen

Automatización del Seguimiento Docente

El sistema actual de seguimiento docente es manual, tedioso y muy propenso a errores. El objetivo de este TFG es proponer una solución que ayude a sistematizar este proceso utilizando dispositivos IoT y una aplicación web de gestión. El objetivo final de este trabajo es que estos terminales se desplieguen en todas las aulas con capacidad Wi-Fi y Bluetooth para ofrecer diversas formas de autentificación a los profesores. La gestión de todos estos terminales la llevará a cabo a través de un hub IoT, que a su vez terminará de llevar la información recogida a la aplicación de gestión.

La aplicación de gestión se encargará de procesar la información obtenida de los dispositivos IoT y ayudará a generar los informes de asistencia agregando la información obtenida por las diferentes formas de registro. Este servidor también deberá exponer una aplicación web para que los usuarios puedan informar de situaciones excepcionales relativas a su horario, y para que los gestores encargados de comprobar las asistencias puedan ver y confirmar de manera sencilla el correcto estado de una jornada laboral en cuestión de asistencia entre otras funcionalidades.

Palabras clave

NodeJS, ESP32, Thingsboard, registro de asistencia, seguimiento docente, aplicación web, NFCs, Bluetooth

Abstract

Systematizing Attendance Tracking

The current attendance tracking system is manual, tedious and prone to errors. This project aims to solve this problem, offering an automatic system that takes care of attendance tracking.

Our objective in this project is to propose an automatized solution for this process with the use of Internet of Things (IoT) and NodeJS. Our main idea is to deploy devices with both Wi-Fi and Bluetooth features in every teaching related space, so as to offer diverse identification methods to the professors. The management of these devices will be taken care of by an IoT hub, which will also be tasked with the carrying of information to a main server.

The main server shall process all information obtained from the deployed devices, and generate attendance logs with any information received in any manner of identification. This server must also expose a web application for the users to inform of exceptional situations relative to their schedules, and for those tasked with attendance tracking to easily verify the correct state of a workday in the aspect related to attendance, among other features.

Keywords

NodeJS, ESP32, Thingsboard, identification, attendance tracking, web application, NFCs, Bluetooth

Índice

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

El sistema de seguimiento docente actual en la Facultad de Informática de la Universidad Complutense de Madrid, recopila la información de asistencia a través de hojas de firmas. Esto requiere que todos los días estas hojas sean distribuidas y recogidas de todas las aulas de la facultad. Este sistema ha demostrado ser propenso a errores y vulnerable a descuidos.

Con este Trabajo de Fin de Grado pretendemos ofrecer una solución que ayude a sistematizar este proceso de gestión, facilitando el trabajo, acelerando la obtención de los datos, reduciendo el gasto de papel y aumentando las garantías.

1.2. Objetivos

Para abordar este problema nos proponemos los siguientes objetivos:

- Desarrollo de un dispositivo hardware que permita varias formas de registro de la identidad del docente.
- Desarrollo de un sistema capaz de detectar anomalías en la asistencia.
- Desarrollo de una aplicación de gestión capaz de recopilar el registro de asistencias y comprobar su corrección.
- Desarrollo de un sistema interactivo que sirva los datos necesarios a los usuarios de manera comprensible y permita tomar acción sobre estos datos.

1.3. Plan de trabajo

El trabajo se ha realizado en dos líneas de desarrollo paralelas. Por un lado el aspecto relacionado con el desarrollo de la aplicación web de gestión para ofrecer acceso desde cualquier parte a los usuarios y por otro lado el ensamblaje y desarrollo del firmware de los nodos IoT que se colocarán en cada espacio (e.g. aula, laboratorio, etc.). En la tabla 1.1 mostramos el trabajo realizado dividido en iteraciones de cada rama de desarrollo.

Iteración	Aplicación Web	Firmware
2 Oct - 9 Oct	Planificación de las reuniones	
9 Oct - 23 Oct	Investigar sobre HTML, Bootstrap y Javascript. Formular el modelo de datos.	Toma de contacto con el framework de Espressif. Adaptación del sistema OTA. Prototipo de comunicaciones HTTPS. Prototipo del uso de la cámara y la pantalla. Creación del docker de Thingsboard. Montaje de la red de desarrollo.
23 Oct - 6 Nov	Realizar historias de usuario. Realizar diseños en papel de las distintas pantallas de la aplicación. Realizar un esqueleto de aplicación web. Prototipos HTML.	Definición de la arquitectura software y reparto de responsabilidades entre los componentes. Prototipo de renderización de texto. Comienzos del modulo MQTT.
6 Nov - 20 Nov	Usar plantillas para las páginas HTML. Realizar diseños en papel de las distintas pantallas de la aplicación y prototipos HTML. Investigar sobre QRs.	Integración de los botones en la interfaz. Prototipo de modulo de boot. Integración de NVS. Creación de app web para aprovisionar los terminales.
20 Nov - 4 Dic	Montar la base de datos e interacciones de esta con la web. Realizar diseños en papel de las distintas pantallas de la aplicación y prototipos HTML. Realizar un QR del espacio que se pida.	Carga de iconos para la pantalla. Primera comunicación hasta el servidor. Generación de códigos TOTP.
4 Dic - 18 Dic	Realizar el formulario avanzado. Realizar una API para poder comunicarse con el firmware. Documentar la API en un YAML. Investigar sobre TOTP.	Refactor de código repetido con macros de compilador. Refactor del módulo MQTT.
18 Dic - 1 Feb	Exámenes	

	Reordenar el proyecto usando *Boilerplates*. Mover la comunicación entre la aplicación web y la base de datos exclusivamente a la API. Establecer comunicación entre Aplicación y API. Implementar TOTP. Investigar sobre sesiones y *cookies*. Mover la generación del QR al cliente para generarlo dinámicamente.	
1 Feb - 15 Feb	Mover la comunicación entre la aplicación web y la base de datos exclusivamente a la API. Crear migraciones y *seeders* para la base de datos. Usar *select2* en HTML. Codificar las contraseñas de los usuarios. Añadir `dotenv` para declarar las variables de entorno.	Primera integración del motor Bluetooth con Bluedroid.
15 Feb - 29 Feb	Implementar lo necesario para realizar el formulario por BLE. Desplazar las funciones de las distintas rutas de la Aplicación y la API a controladores. Implementar sesiones. Realizar nuevas páginas en EJS. Realizar un conversor de CSV a actividades de la base de datos.	Cambio de motor BLE a NimBle y programación del modulo exterior de RFID. Primera comunicación al endpoint de seguimiento.
29 Feb - 21 Mar	Realizar nuevas páginas en EJS. Realizar minimización de los archivos que vayan al cliente. Realizar archivos configurables para el proyecto. Crear *middleware* para determinar recurrencias, manejar *cookies* y escapar peticiones. Realizar código para determinar las clases no asistidas. Implementar lo necesario para realizar el formulario con NFCs	Refactor global y primeras pruebas transversales en entorno de desarrollo.
21 Mar - 4 Abr	Funcionalidad de informes sobre las asistencias, 'firmar'. Adaptar el *middleware* de escapado para aceptar caracteres españoles. Cambio en el modelo de Excepción.	Adaptación para el entorno de producción y primeras pruebas en él.

4 Abr - 18 Abr	Adaptar la base de datos a UTC. Añadir pino para hacer *logs*.	Cambio de los iconos y rediseño de la interfaz de cara al usuario.
18 Abr - 2 May	Mejora de páginas web. Crear el registro de MACs y NFCs. Añadir Morgan. Cambios en las llamadas a los distintos *middleware*.	
2 May - 16 May	Crear informes de las asistencias registradas durante el día. Crear imágenes de QRs para los espacios desde la API. Implementación de funcionalidades de administración y decanato. Añadir trustproxy.	

Tabla 1.1: Diagrama temporal de trabajo

Capítulo 2

Estado de la Cuestión

El proceso establecido en la Facultad de Informática de la UCM, utiliza hojas de firma en cada clase para recoger la información necesaria. Estas firmas después se cotejan y se preparan los informes en la sección de personal para transmitir las incidencias a Decanato. Creemos que este sistema deja mucho margen para la mejora. Con una solución informática se podría reducir el gasto de papel y el trabajo manual.

Existen muchas alternativas establecidas para informatizar este proceso. Desde soluciones totalmente virtuales como es el caso de Moodle hasta soluciones basadas en biometría y tarjetas inteligentes. Nuestra propuesta intenta coger lo mejor de todos los mundos agregando información de asistencia de muchas fuentes alternativas.

A la hora de la gestión de rebaños de nodos IoT nos encontramos con una rica variedad de opciones. Las opciones más profesionales ofrecen mejores garantías de funcionamiento a costa de su precio y estar a la merced de una gran tecnológica. En nuestro caso hemos preferido una opción de software libre que nos da mas libertad y resulta más económica.

Por ultimo debemos considerar la empresa que nos va a proveer del hardware de los nodos. En este aspecto todas las opciones son bastante buenas, pero encontrar la más adecuada requiere una comparación de estas alternativas.

2.1. Sistemas de Fichaje

El ámbito docente no es el único que necesita control horario. Con la puesta en vigor del Real Decreto-ley 8/2019, es necesario controlar las horas que trabajan los empleados de distintas organizaciones. Antes de decidir qué hacer contemplamos distintas opciones actuales.

2.1.1. Fichaje en Oficinas

Comercialmente se ofrecen distintos servicios de control horario para trabajadores en oficinas, como es un servicio de fichaje que temporiza la duración de una jornada laboral, contando los descansos (TimeTac); o los servicios dedicados a la asignación de tareas y el registro del tiempo que se pasa en ellas (Clockify). Algunos de estos servicios se pueden adaptar a nuestro caso de uso docente pero vienen con el inconveniente del coste, habitualmente por empleado, a tener en cuenta. Los casos más asequibles sufren de falta de funcionalidades, como control horario en diferentes aulas, generación de informes o falta de retro-compatibilidad con el sistema establecido. Aquellos que podrían llegar a utilizarse, que sean capaces de localizar a un empleado con GPS, como Connecteam, podrían llegar a confundir espacios docentes (el Aula 3 con el Aula 8, está encima), a lo cual también habría que encontrar alternativa.

2.1.2. Moodle

Moodle ofrece una posible solución a este problema con un plugin llamado *Attendance*. Este plugin lleva disponible desde 2010, y ofrece un sistema de asistencia en el que el docente puede registrar la asistencia de alumnos asociados a la asignatura en la que están matriculados. Se podría adaptar esta solución a nuestro problema configurando el entrono del campus. Moodle es un entorno conocido para los docentes, lo que facilitaría la adopción. Este sistema sufre la misma vulnerabilidad a descuidos que el sistema actual sin añadir ninguna garantía de que los datos recogidos sean reales. No todos los profesores utilizan el Campus Virtual durante sus clases, por lo que para ellos esta opción supondría más esfuerzo que la solución actual.

2.1.3. Fichaje a Papel

Este es el mecanismo actual de registro de asistencias. Es simple para la parte de los docentes, pero puede resultar muy tedioso para el personal administrativo encargado del reparto, la recogida y la digitalización de las hojas. Es importante, durante el desarrollo de nuestra solución, no perder de vista las ventajas que proporciona este sistema e intentar integrarlo en nuestra solución final. De esta forma no será necesario que los profesores cambien sus hábitos repentinamente y facilitaremos la transición entre las dos soluciones.

2.1.4. Fichaje con RFID

Uno de los mecanismos actuales más usados para el fichaje es el uso de lectura de dispositivos RFIDs asociados a trabajadores. Esto sería aplicable al caso de un aula, permite sustituir la hoja de firmas actuales y podríamos hacer uso de la Tarjeta Universitaria Inteligente (TUI) que todos los miembros de la comunidad educativa

complutense ya tienen. Este sistema es más robusto que las hojas de firmas por tener la capacidad de registrar no solo la identidad del docente sino también la hora del registro. Con este dato extra se puede verificar la integridad de la información y producir informes más fiables. En particular, este es el sistema de fichaje utilizado por el Personal Técnico de Gestión y Administración de Servicios (PTGAS) de la UCM.

2.1.5. Fichaje Biométrico

La opción más segura y certera de registrar la asistencia sería usando las huellas dactilares de los docente. Es una opción que haría imposible las actividades fraudulentas que este proyecto quiere mitigar. Sin embargo, también requiere de un sistema con suficientes garantías de seguridad para poder albergar datos biométricos. Un fallo de seguridad en un sistema tan crítico podría exponer a los docentes a la suplantación de identidad.

En conclusión, podemos descartar las opciones de fichaje en oficinas debido a su coste, o a su falta de funcionalidades, la cual podríamos solucionar nosotros mismos. También podemos descartar la opción de utilizar Moodle, pues comparado con la manera actual de hacerlo, dejaría descontentos a los docentes que tienen que ir directos al Campus Virtual para fichar, y rompería la estructura de asignaturas del campus. El resto de medidas sin embargo, podrían llegar a utilizarse. Ofrecer distintas maneras de registrar la asistencia permitiría contentar a un gran número de docentes, incluyendo aquellos a los que les gusta la manera actual de hacerlo.

2.2. Hubs IoT

Debido a la naturaleza de nuestra solución es necesario incorporar al sistema un hub IoT[1] que gestione el rebaño de nodos. Con esto queremos decir que necesitamos un servicio de software al que podamos delegar las tareas que impliquen gestión masiva de nodos, como la actualización, autenticación o configuración. Esto nos permite centrarnos en la funcionalidad principal de la solución. A continuación se listan algunas alternativas que hemos considerado. Todos las opciones que se mencionan a continuación destacan por su escalabilidad y su gama de protocolos admitidos.

2.2.1. AWS IoT Core

Ofrece una amplia gama de servicios vinculados con AWS, incluyendo almacenamiento, análisis, aprendidaje máquina y seguridad. Aparte de HTTP y MQTT ofrece soporte para MQTT sobre WebSocket[2]. El despliegue está externalizado en la

[1] `https://en.wikipedia.org/wiki/Internet_of_things#Architecture`
[2] `https://docs.aws.amazon.com/iot/latest/developerguide/protocols.html`

nube de AWS. El precio depende de la región del mundo desde la que se conecten los nodos, el tiempo de conexión, el tamaño de los mensajes y otras características de la comunicación. Esto aumentaría la complejidad del sistema añadiendo un factor económico a las comunicaciones y desviando la atención lejos del problema real que queremos resolver.

2.2.2. Google Cloud IoT

Ofrece integración con servicios de Google Cloud como BigQuery, Dataflow y Pub/Sub. Soporta protocolos populares como MQTT y HTTP. Un inconveniente de esta alternativa es que la gestión del sistema está externalizada en la nube de Google. Ofrecen capacidad *edge* para con los centros de computo que Google tiene por todo el mundo. Esto permite que muchas peticiones no tengan que llegar al centro de computo central donde está la base de datos y la lógica de negocio sino que pueden quedarse en uno de sus servidores *edge* distribuidos. Aunque esto tiene ventajas de velocidad no nos permite poseer nuestros datos directamente. Idealmente el hub IoT debe poder alojarse desde dentro de la red de la facultad para maximizar el tiempo de servicio y minimizar los costes.

2.2.3. Azure IoT Hub

Tiene buena escalabilidad horizontal, capaz de manejar millones de dispositivos. Puede integrarse fácilmente con servicios de Azure como Azure Functions, Azure Stream Analytics y Azure Cosmos DB. Soporta una amplia gama de dispositivos y protocolos. Ofrecen la opción de un alojamiento gratuito limitado en sus servidores pero es posible desplegar el tuyo propio localmente. Esta solución sufre de no ser código libre. Elegir esta solución nos dejaría a la merced de los designios de Microsoft.

2.2.4. DeviceHive

Opción escalable, adecuada para aplicaciones de tamaño medio. Ofrece una serie de servicios básicos para la gestión de dispositivos y datos fuera de ninguna nube comercial. Tiene integración con Grafana[3], una herramienta de diseño de paneles de visualización de datos muy extendida. También es compatible con una variedad de dispositivos y protocolos, pero menos extenso que los proveedores en la nube. Proporciona características básicas de seguridad, pero puede requerir configuración adicional para entornos más críticos. Sus creadores ofrecen alojamiento gratuito limitado en sus servidores de su herramienta. También es posible desplegar el tuyo propio localmente. Es un proyecto de código abierto.

[3] `https://grafana.com/`

2.2.5. ThingsBoard

Igual que las otras alternativas Thingsboard es escalable, adecuado para implementaciones de pequeña a mediana escala. Ofrece servicios básicos para la gestión de dispositivos y datos, con capacidades de personalización. Desde Thingsboard se pueden diseñar paneles de control, flujos de datos y protocolos de actuación personalizados. Es compatible con una amplia variedad de dispositivos y protocolos, con un enfoque en la flexibilidad y la personalización. Proporciona características básicas de seguridad, con opciones de personalización para requisitos específicos. Ofrecen la opción de un alojamiento gratuito limitado en sus servidores pero es posible desplegar el tuyo propio localmente. A parte de la versión *Community* existe una versión profesional con más utilidades pero de pago. La versión gratis es un proyecto de código abierto.

Tomando en consideración la comparativa entre todas ellas el producto que mejor se adapta a nuestras necesidades es sin duda Thingsboard. Ser de código abierto y gratuito está por delante de los productos comerciales y está mucho más establecido en la industria que DeviceHive.

2.3. Nodo IoT

Para el éxito de la implementación es esencial encontrar un nodo capaz de soportar toda la funcionalidad que deseamos pero al mismo tiempo no suponga un problema económico que comprometa la escalabilidad el sistema.

2.3.1. Texas Instruments

Los microcontroladores y sensores de Texas Instruments (TI) pueden ser más caros en comparación con otras opciones disponibles en el mercado. En cambio TI proporciona una amplia gama de recursos educativos, incluyendo tutoriales, guías de inicio rápido y material de capacitación. Por esta faceta educativa la universidad ya tiene experiencia trabajando con ellos. También ofrecen un entorno de desarrollo integrado (Code Composer Studio) específico para el desarrollo de firmware para sus productos.

2.3.2. STMicroelectronics

ST ofrece una variedad de microcontroladores y kits de desarrollo a precios competitivos. ST proporciona una gran cantidad de recursos educativos en forma de documentación, tutoriales y videos. Su entorno de desarrollo integrado, STM32CubeIDE, es relativamente fácil de usar y está diseñado para simplificar el proceso de desarrollo.

2.3.3. NXP Semiconductors

Los productos de NXP tienden a estar en un rango de precio medio a alto debido a su calidad. Son una empresa muy establecida en el ámbito profesional. NXP ofrece una amplia gama de recursos educativos, incluyendo guías de inicio rápido, tutoriales en línea y material de capacitación. También tienen su propio entorno de desarrollo MCUXpresso.

2.3.4. Raspberry

Aunque esta empresa se centra en sistemas tan capaces como un ordenador dentro de su catálogo ofrecen la Raspberry pico. Este producto es un microcontrolador mucho menos capaz que sus hermanos mayores. En lo que al precio se refiere esta es una opción muy asequible. A nivel de desarrollo se puede programar con MicroPython y con C.

2.3.5. Espressif

Los chips de Espressif, como el ESP8266 y el ESP32, son conocidos por ser muy asequibles. Hay muchas placas de desarrollo basadas en estos chips disponibles a precios bajos. Esto permite comprar placas con los accesorios deseados y eliminar el trabajo de montaje. Espressif proporciona una documentación detallada y una amplia gama de recursos en línea, incluyendo tutoriales y ejemplos de código. Su entorno de desarrollo, el ESP-IDF, puede ser un poco más avanzado en comparación con otros, pero permite mucho control de bajo nivel sobre las características del hardware. También es posible programar sus placas con el entorno de Arduino, facilitando el desarrollo de proyectos pequeños pero obstaculizando el crecimiento de un proyecto como el nuestro.

Al final decidimos implementar nuestro sistema en hardware de Espressif. No solo aporta una buena relación calidad-precio, sino que nos permite usar su entorno ESP-IDF para implementar las funcionalidades más complejas y usar kits de desarrollo con periféricos avanzados más fácilmente que si usáramos otra alternativa y tuviéramos que diseñar nosotros la placa.

Capítulo 3

Visión de alto nivel

El problema de automatizar el seguimiento docente tiene muchos componentes que deben colaborar para que funcione el sistema completo. En este capítulo vamos a tratar una visión general de la arquitectura que hemos diseñado para materializar la solución junto a las decisiones de diseño que nos han llevado hasta ella.

Todo el código de esta solución se puede encontrar en este[1] y este[2] repositorio.

3.1. Estudio funcional previo

Con el fin de documentar los objetivos de la solución ideada, hemos hecho uso de la técnica de personas introducida en Cooper (2004). Se adjuntan en el apéndice A las personas utilizadas.

3.1.1. Roles

Una vez definidas las personas debemos distinguir los diferentes roles a los que pueden pertenecer los usuarios.

- **Administración de Personal**: En este rol se encuentran aquellos encargados de comprobar la corrección de una jornada docente, teniendo en cuenta las asistencias del día y generando informes de estas.
- **Decanato**: En este rol se encuentran aquellos encargados de comprobar la corrección del entorno docente, incluyendo el seguimiento de problemas que puedan surgir, e identificando posibles docentes que eludan su labor.
- **Docente**: En este rol se encuentran los docentes encargados de impartir clase.

[1] `https://github.com/jaimegonzalezfabregas/TFG-ASD-WebServer`
[2] `https://github.com/jaimegonzalezfabregas/TFG-ASD-Devices`

3.1.2. Historias de usuario

Hecho este proceso, creamos las historias de usuario que representan el comportamiento deseado del sistema.

- Como parte de la administración de personal, quiero reducir el tiempo que le dedico a verificar que se han impartido todas las tareas docentes para un día concreto.
- Como parte de la administración de personal, quiero poder generar avisos mediante correos electrónicos a los profesores que no han impartido sus clases, para que se notifique la razón por la que no han asistido a sus clases, y que la respuesta quede registrada para que sea simple su verificación.
- Como parte de la administración de personal, quiero que a la hora de cotejar las hojas de firmas sólo se muestren aquellas actividades que no han sido registradas por otro mecanismo, de modo que el proceso se aplique en el número mínimo de casos.
- Como parte de la administración de personal, quiero poder registrar como asistidas aquellas actividades que, aunque no estén registradas por otro mecanismo, sí lo estén en las hojas de firmas.
- Como parte de decanato, quiero poder saber si hay algún profesor que no imparte regularmente sus clases.
- Como docente, quiero que se registre mi asistencia mediante el login en el puesto del profesor (Puesto de Laboratorio en Aula, o PLA) para no tener que firmar en la hoja de firmas.
- Como docente, quiero que se registre mi asistencia utilizando la TUI para no tener que firmar con la hoja de firmas.
- Como docente, quiero que se registre mi asistencia escaneando un simple código QR, para no tener que firmar en la hoja de firmas.
- Como docente, quiero que se registre mi asistencia utilizando un dispositivo inteligente que lleve encima, para no tener que firmar en la hoja de firmas.
- Como docente, quiero poder seguir utilizando la hoja de firmas, porque no quiero llevar otro dispositivo o herramienta a mis clases y no quiero utilizar el equipo del profesor.
- Como docente, quiero poder registrar las anulaciones y / o cambios de clase con otros profesores de manera previa, para no tener problemas con personal o decanato.
- Como docente, quiero poder registrar de manera ágil las causas más frecuentes a la hora de no haber impartido una clase:

- Olvidado firmar / login / etc.
- Indisposición médica del interesado.
- Cuidado de descendiente o ascendiente del interesado.

A partir de estas podemos distinguir una gran variedad de casos a tener en cuenta. Por un lado, algunos profesores quieren omitir cualquier interacción explícita para registrar su asistencia, como Rosa e Ignacio. Por otro, otros tantos profesores si quieren tener esa interacción, pero de una manera que les resulte cómoda. Para adaptarnos a las necesidades de cada profesor, vamos a necesitar un sistema que, desde distintas entradas, tomen el mismo resultado, registrar una asistencia. A su vez, tenemos que ofrecer una manera de ver e interactuar con la base de datos para Marta y Alejandro, e incluso para los docentes cuando necesiten cambiar o cancelar una clase, o registrar el motivo de una falta.

3.2. Arquitectura

De las historias de usuario podemos obtener los requerimientos de nuestra solución.

Empezando por las necesidades docentes, hemos de recoger todos las maneras distintas de registrar una asistencia (por Bluetooth, por TUI, ...). Para esto tenemos que imponernos las siguientes metas:

- Con el fin de permitir un registro de asistencia a través de un código QR, es necesario su generación. Además, para no hacer de este QR uno estático, deberá cambiar con el tiempo.
- Con el fin de recoger asistencia a través de un login en el PLA, hará falta un mecanismo capaz de interpretar los datos del login para pasarlos a una asistencia.
- Con el fin de recoger asistencia a través de un dispositivo inteligente, es necesaria la capacidad de detectar este tipo de dispositivos mediante BLE.
- Con el fin de recoger asistencia a través de una TUI, hemos de ofrecer un lector capaz de obtener el UID del NFC de dicha tarjeta.

Por otra parte también necesitamos un servicio web para facilitar a administración de personal y decanato el acceso a la información sobre asistencias relevante a sus roles, y que así puedan interpretar esta para sus fines. También hay que proporcionar una manera de interactuar con esta información para el registro de firmas, la gestión de avisos y el registro de situaciones excepcionales en los horarios de los docentes.

A la hora de manejar los dispositivos para el registro de la asistencia, es necesario comunicarlos con la base de datos. Sin embargo, también es necesario codificar y decodificar la información de los dispositivos, y manejarlos en conjunto. Debe haber un encargado del mantenimiento y la configuración de estos dispositivos para asegurar el correcto funcionamiento del sistema.

Por último, necesitamos comunicación entre los dispositivos y la base de datos, y entre el servicio web y la base de datos. También es necesario un encargado de manejar la información que entra y sale de la base de datos, para entregarla a donde sea necesario.

A partir de lo anterior, hemos ideado la solución representada en la figura 3.1. Esta solución se divide en varios componentes.

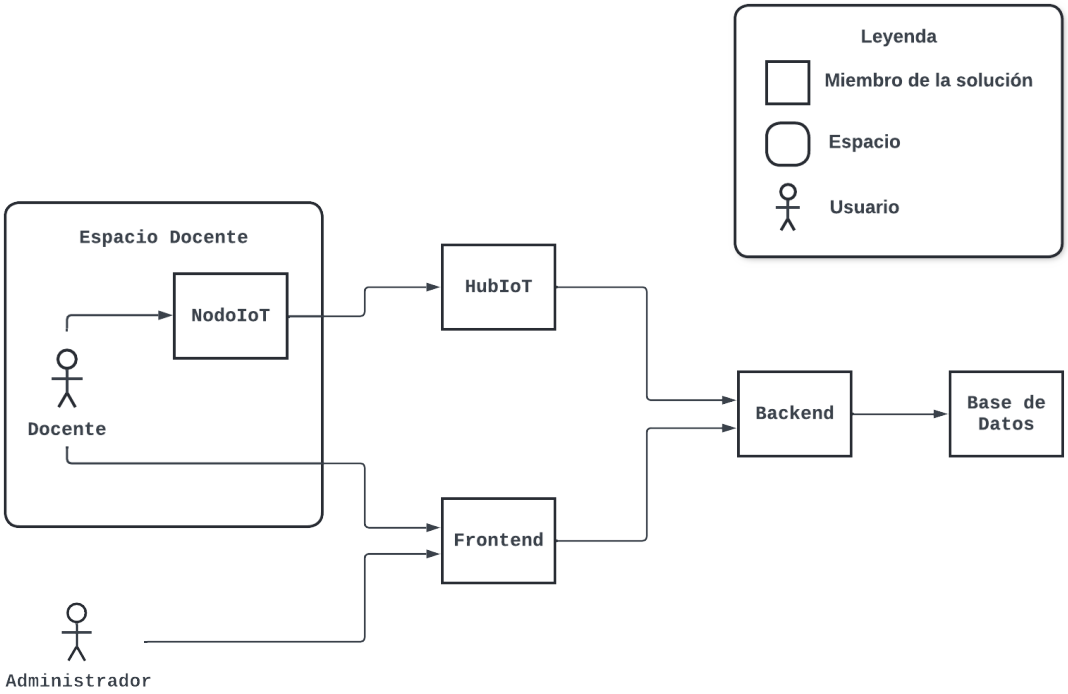


Figura 3.1: Arquitectura del funcionamiento del sistema completo.

- **Nodo IoT**: Este dispositivo será multifuncional para permitir varias formas de verificación de presencia. Es necesario que el dispositivo tenga pantalla, WiFi y Bluetooth para abarcar la funcionalidad que trataremos más adelante. Servirá para verificar que el profesor se encuentra en el aula durante el proceso de registro a través del móvil. Estos terminales deberán comunicarse con el hub IoT.

- **Frontend**: Algunas formas de verificación de asistencia pueden llevarse a cabo desde un dispositivo del profesor con cierta participación de un nodo IoT. Además este portal también será el portal de gestión para comunicar la cancelación o reprogramación de una clase y consultar los informes.

- **Backend**: Este componente ofrece una comunicación con la base de datos al hub IoT y al Frontend para recibir y procesar la información que recoge el sistema.
- **Hub IoT**: Durante la operación del sistema este módulo se responsabiliza de traducir los mensajes de los nodo IoT para enviarlos al Backend y en gestionar las incorporaciones de nuevos terminales a la red. También debe ofrecer una interfaz administrativa que de una visión general del estado de los nodos.
- **Base de Datos**: Este componente alberga los datos y las relaciones entre ellos. Es consultado por el Backend al ejecutar trabajos periódicos y a la hora de responder los mensajes del Frontend o del hub IoT.

Con estos componentes podemos cubrir las necesidades de todos los usuarios. Teniendo un hub IoT para el manejo de nodos IoT podemos separar una parte más relacionada con la gestión y el funcionamiento de los dispositivos. Al tener que acceder a los datos en la web, es indispensable que el Backend esté expuesto a esta, tal y como el Frontend debe estarlo para que los usuarios interactúen con la solución. La Base de Datos (BD), sin embargo, solo debe estar expuesta al Backend.

Capítulo 4

Aplicación de gestión

En un principio, las responsabilidades del Frontend y el Backend se ejecutaban en el mismo proceso. Sin embargo, tras llegar a cierto punto del desarrollo vimos adecuado separar estas responsabilidades en dos procesos debido a la dificultad creciente del mantenimiento del código. Esto divide la aplicación en dos. Por una parte, el Frontend actúa como el sistema que se comunica con los usuarios, y por otra, el Backend actúa como el responsable de la gestión de la base de datos. Esta separación sigue algunos de los principios del patrón *Backend for Frontend*.

El Frontend comprende una serie de servicios expuestos al público que usan datos de la base de datos para que los usuarios puedan interactuar con ellos. El Backend consiste en una serie de servicios de consulta, creación, eliminación y modificación de datos de la base de datos. El acceso al Backend está restringido únicamente a agentes de confianza. Este se ha implementado como una API REST. Para ajustarnos al patrón elegido solamente el Backend tiene acceso a la base de datos, como se puede ver en la figura 3.1.

El código completo de la aplicación de gestión web se puede encontrar en el siguiente repositorio de GitHub[1].

4.1. Tecnologías utilizadas

Durante el desarrollo de la aplicación web hemos tenido que considerar distintas tecnologías a utilizar para cada una de las distintas partes dependiendo de la funcionalidad deseada.

La tarea de almacenar los datos recae sobre una base de datos. Esta debe almacenar todo el contenido de manera lo suficientemente simple como para mantener, pero lo suficientemente completa como para que efectúe lo que se pide del proyecto. La primera consideración es elegir el tipo de base de datos a utilizar. Consideramos

[1]El enlace es `https://github.com/jaimegonzalezfabregas/TFG-ASD-WebServer`

que al ser un proyecto relativo al tiempo tanto la consistencia como la accesibilidad a la base de datos son las dos características clave a mantener. Estas dos características son por las que las bases de datos relacionales destacan, por tanto usaremos estas.

El proyecto comenzó usando una base de datos de SQLite, elegida por nuestra experiencia anterior con ella. Sin embargo, debido a la escasez de recursos para este tipo de base de datos tomamos la decisión de cambiar a una base de datos más seria. Elegimos como alternativa MySQL y hemos continuado con ella durante el resto del desarrollo.

Al principio del desarrollo de la aplicación web, se contrastó la posibilidad de utilizar Spring o NodeJS junto a ExpressJS como entornos para su programación. Tras ver lo que ofrecía cada alternativa se eligió NodeJS junto a ExpressJS. Sentíamos que la curva de aprendizaje de Spring iba a ser mucho más complicada de abordar dentro del ámbito del TFG en comparación con su alternativa.

Esto contrasta con la sencillez de NodeJS. Si además consideramos las ventajas de usar el mismo lenguaje (Javascript) en el cliente y el servidor sin coste adicional para el escalado, NodeJS se convierte en la mejor opción.

Los lenguajes HTML y CSS son los cimientos de todas las aplicaciones web. Para facilitar su uso de HTML y CSS hemos elegido utilizar Bootstrap con plantillas ya hechas siguiendo la recomendación de nuestros tutores. Decidimos utilizar la plantilla SB Admin[2]. La plantilla original se ha modificado personalizando el HTML y el CSS para ajustar la aplicación a los colores y tipografía corporativa de la UCM[3]. El código CSS creado por nosotros se ha separado del de Bootstrap. Debido a la limitada funcionalidad que ofrecían las tablas de la plantilla hemos utilizado las tablas de la librería DataTables[4].

Para la visualización de fechas en la página web, se han utilizado *input fields* de tipo `date`. Para la visualización de múltiples opciones se ha hecho uso de la librería *select2*. Estos dos elementos ayudan al usuario a ver e interactuar con estos formato.

4.2. Base de Datos

Tras haber decidido usar una base de datos relacional, empezamos a pensar el modelado de los datos. Para ello ideamos un diagrama entidad-relación. En este, las tres entidades principales son Docente, Espacio, y Actividad. La entidad Docente tiene los datos de los docentes que impartirán actividades. La entidad Espacio tiene datos sobre un espacio en el que se puede dar una actividad. La entidad Actividad es un evento en el tiempo dado en un espacio en específico, asociado a una o más clases. La asociación entre múltiples clases se considera por casos específicos como la unión

[2]Disponible en `https://startbootstrap.com/template/sb-admin`
[3]Esto se especifica en `https://ssii.ucm.es/colores-y-tipografia`
[4]`https://datatables.net/`

de grupos A y E durante nuestra promoción para los laboratorios de Tecnología de la Programación (TP), o el caso de Aplicaciones Web (AW) y Sistemas Web (SW), que se imparten en la misma hora y mismo aula, y así ahorrarse actividades.

Estas clases se representan en la entidad Clase, la cual hemos considerado representarla como una agrupación de dos entidades, Grupo y Asignatura. Se han separado así para considerar los distintos horarios que se elaboran para los distintos grupos, incluidos los cambios en asignaturas por ser de distintos grados. Para mantener con propiedad las asignaturas, consideramos a su vez las entidades de Plan y Titulación. Una titulación (el Grado en Ingeniería Informática, por ejemplo) ofrece un plan de estudios al alumno, el cual es el que tiene información sobre las asignaturas que se impartirán.

Para el manejo de las recurrencias de las distintas actividades se introdujeron las entidades Recurrencia y Excepción, basadas en el modelo de datos expuesto en (Kher, 2016). Recurrencia y Actividad siguen una base similar al mencionado en dicho artículo, pero Excepción se ha extendido para poder registrar la actividad que ha sido cancelada o reprogramada. También se han incorporado en la entidad Docente dos atributos multivaluados, MAC y NFC, para poder hacer los registros de asistencias con Bluetooth y NFC.

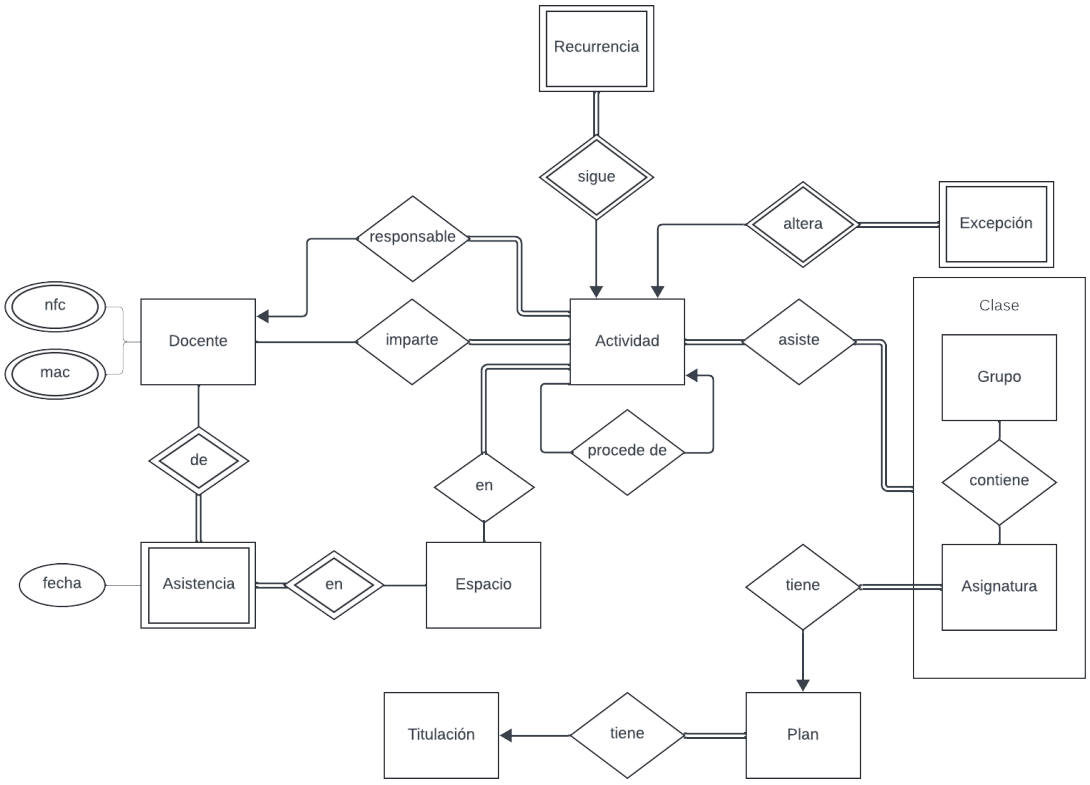


Figura 4.1: Diagrama del modelo de datos

La arquitectura de datos descrita se ve ilustrada en la figura 4.1. Esta representa un modelo entidad-relación, en el que solo representamos las entidades, sus

relaciones, y los atributos que hemos considerado más relevantes.

A la hora de pasar el modelo entidad-relación a la base de datos, los atributos multivaluados se transformaron en tablas, y también se crearon tablas para las distintas relaciones N:M. A su vez, las entidades débiles deben incluir la clave foránea de su relación al insertarse. También se ha usado de `timestamps` para saber cuando se ha creo y cuando se actualizo por última vez una fila. Todas las tablas tienen un campo id que identifica cada una de sus filas. En el caso de la tabla asignatura, en vez de utilizar un id hemos aprovechado el código único proporcionado por la Gestión Académica de la universidad (GeA).

4.3. Frontend

El Frontend es la parte de la aplicación con la que van a interactuar los usuarios. Este módulo está implementado como un servidor ExpressJS que actúa de puente entre los clientes y el Backend.

4.3.1. Bocetos

Al principio del desarrollo se realizaron bocetos para cada funcionalidad, que luego se pasaron a prototipo HTML. Posteriormente se retocaron los prototipos para dejar el HTML final, y se pasaron a EJS para utilizar como plantillas, con campos obtenidos de datos de la aplicación.

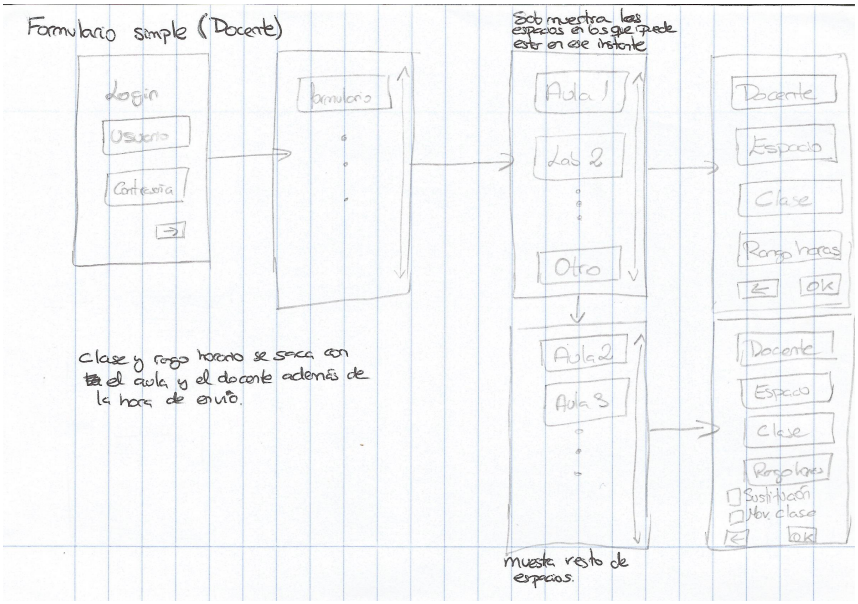


Figura 4.2: Boceto para Registro de Asistencias

Para poder registrar la asistencia desde el móvil, ideamos un formulario sencillo (figura 4.2). El docente solo hace tiene que decir en qué aula está y confirmar su

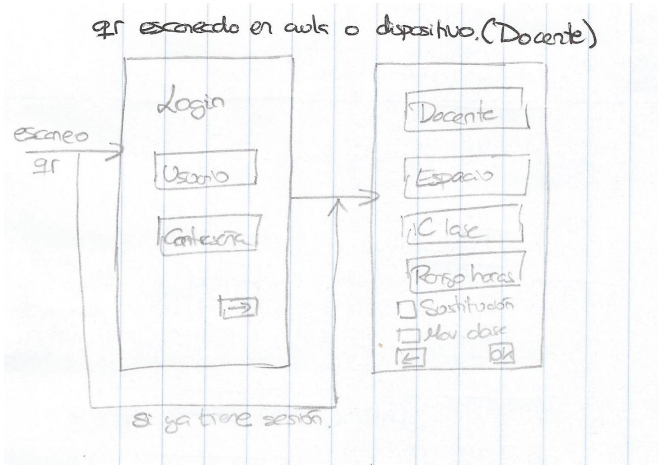


Figura 4.3: Boceto para Acceso desde QR

asistencia.

Para agilizar este proceso también consideramos utilizar un QR. Escaneando este QR el campo aula se completaría automáticamente como se puede ver en el boceto de la figura 4.3.

Sin embargo, no incluimos bocetos para registrar por Bluetooth o por TUI. Tras considerar el uso de un nodo IoT concluimos que no era necesario hacer una interfaz web para estos casos.

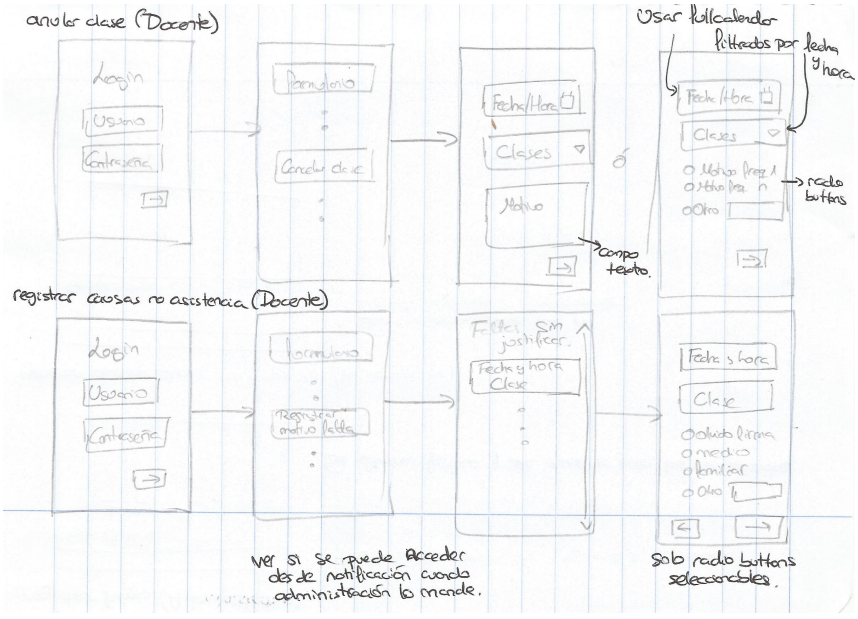


Figura 4.4: Bocetos para Cancelar clase y Registrar motivo de falta

Para proporcionar a los docentes una manera de comunicar incompatibilidades con sus horarios, se idearon los bocetos de la figura 4.4. En el primero, damos a los docentes a elegir una clase a cancelar, y obligamos a proporcionar un motivo para la cancelación. Esto sirve para comunicar clases a las que no pueda ir de antemano.

Si el docente se olvidase de registrar su asistencia, se haría uso de la interfaz que se muestra en el segundo boceto, donde dada una actividad no asistida permitimos al docente registrar un motivo para esta.

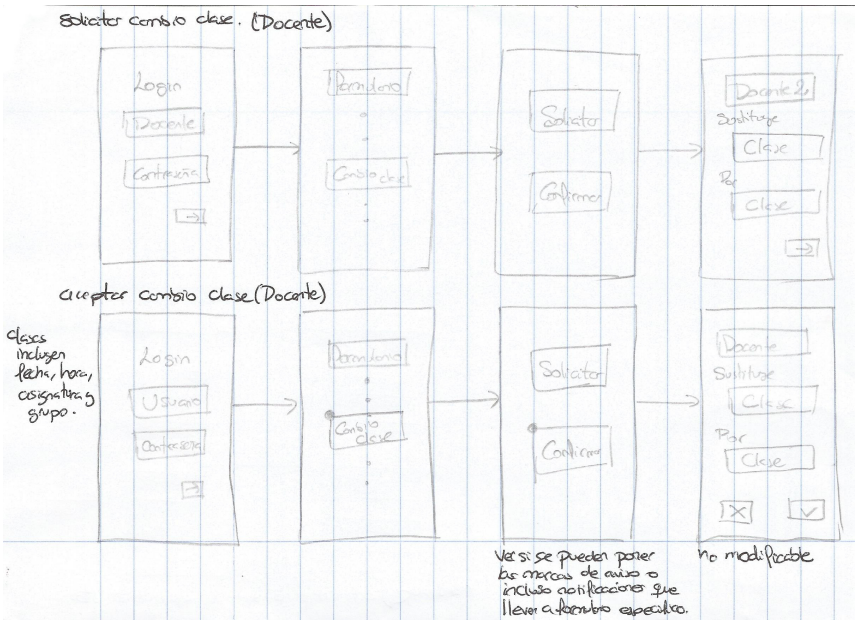


Figura 4.5: Boceto para Cambio de clase

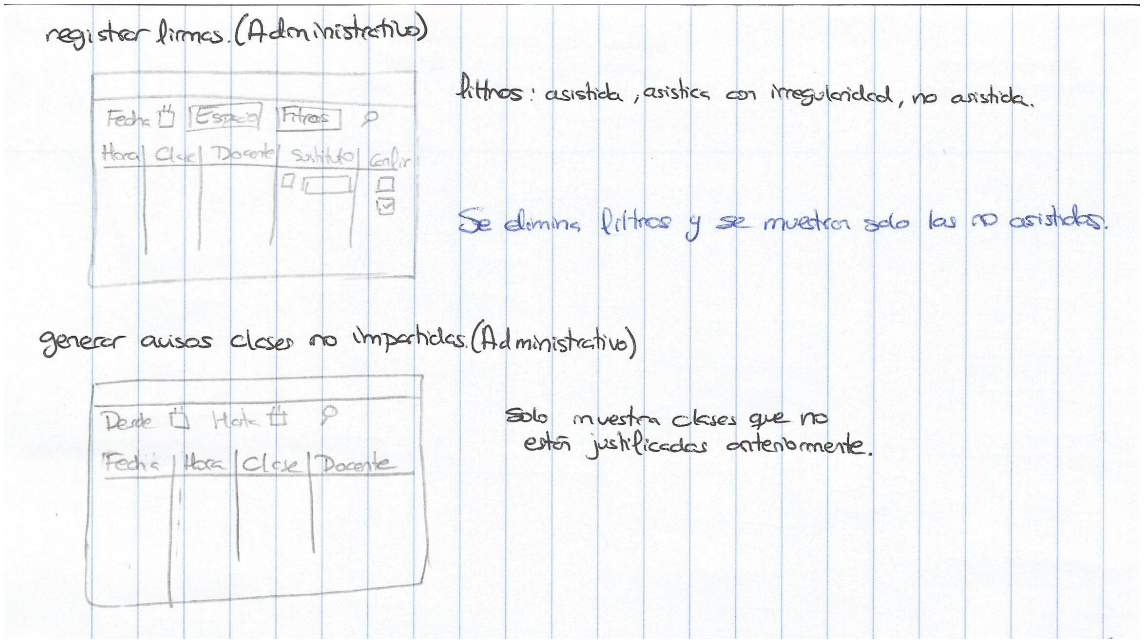


Figura 4.6: Bocetos para Registro de Firmas y Generar Avisos

Dentro de la funcionalidad de la cancelación de clases hemos considerado la opción de configurar un cambio entre profesores. Esto tiene como efecto que se programa la clase de cada uno de los profesores en el hueco de la otra. Ofrecer a los docentes una manera de intercambiar sus clases permite que se organicen sin tener

que pasar por administración. Su interfaz se representa en el boceto de la figura 4.5. Al pedir un cambio de clase con otro docente se le envía un mensaje para que acepte o rechace la solicitud.

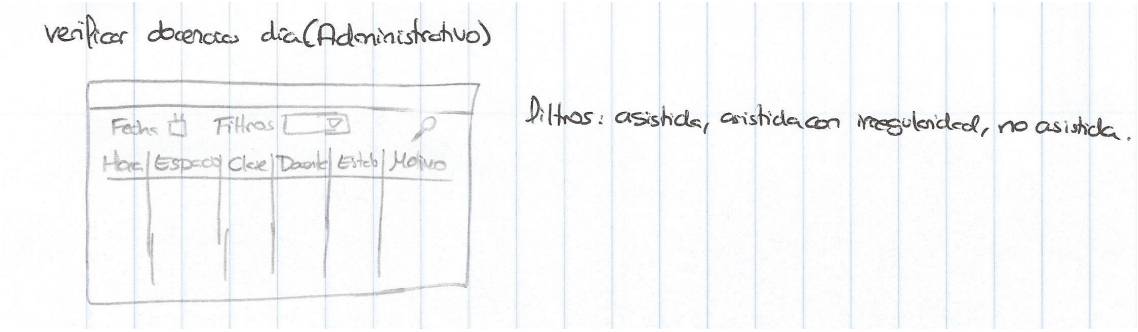


Figura 4.7: Bocetos para Verificar Docencias

Para la parte de administración es necesario proporcionar una manera de registrar firmas, ver docencias y generar avisos. Para hacer la tarea sencilla, ideamos los bocetos de las figuras 4.6 y 4.7. La parte del registro de firmas consiste en una tabla que muestre la información relevante al usuario, y permite confirmar con un *click* qué asistencia está registrada. La parte de generar avisos permite al encargado de las asistencias enviar correos informativos a los docentes que no hayan asistido a las actividades que se muestren en la tabla. Y por último se usa la tabla de verificar docencias como manera de comprobar visualmente que las asistencias se registren bien. Esta tabla permite ver todas las asistencias de un día y filtrarlas por sus respectivos tipos.

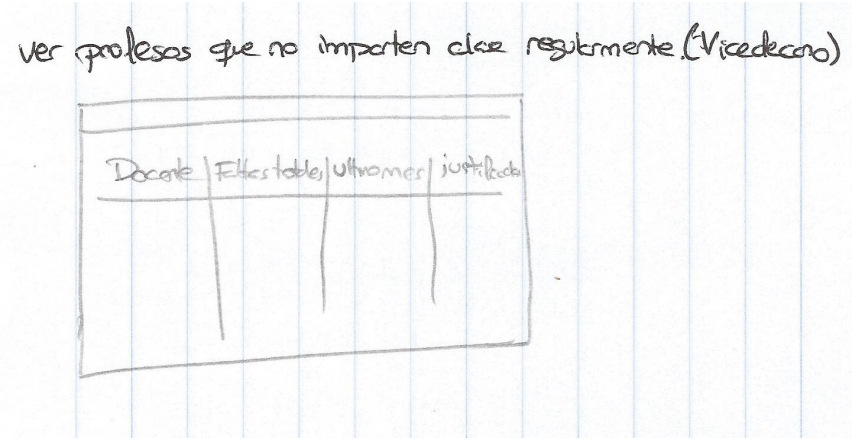


Figura 4.8: Boceto para Ver profesores que no imparten clase regularmente

Para decanato se necesita poder ver los docentes que no han impartido clase regularmente. El boceto para esta tarea (figura 4.8) consiste en una tabla que muestra el nombre del docente y distintos agrupamientos de faltas, como aquellas del último mes o aquellas sin justificar. Esta información ofrece al usuario más detalle que si solo fuese el número de faltas.

Como estamos tratando con una aplicación web hay que tener en cuenta la Experiencia de Usuario (User Experience, o UX). Para garantizar una buena UX, nuestra página web cumple lo siguiente:

- En menos de tres *clicks*, un usuario puede llegar a cualquier parte de la aplicación.
- Desactivamos botones con los que no pueda a interactuar en determinados estados.
- No volvemos a pedir información que ya nos hayan dado.
- No quitamos nunca el control al usuario.
- Damos retroalimentación cuando el usuario realiza una acción.
- Para los casos en los que un usuario se pierda, siempre es posible volver a la página principal pulsando el título de la aplicación.
- Usamos texto e iconos para facilitar la comprensión de las distintas funcionalidades.
- Seguimos siempre un mismo formato de página para mantener la consistencia interna y que el contenido sea uniforme.

Estos aspectos se han tenido en cuenta al hacer los bocetos vistos anteriormente, y al pasar estos a HTML.

4.3.2. Detalles de la implementación

Tras la etapa de diseño se empezó con la implementación.

Al ser una aplicación web donde hace falta autenticarse, se hace uso de sesiones para reconocer a los usuarios. En nuestro caso, consideramos que la mejor práctica es guardar las sesiones en el servidor. Esto permite aumentar el tamaño de los atributos de sesión y mantener en secreto los datos relacionados con cada cliente. Esto protege la confidencialidad de los datos y dificulta las actividades fraudulentas. Para guardar las sesiones en el servidor se hace uso de una *session store*.

Para acelerar la recuperación de datos se cachean algunos resultados de consulta en la sesión del usuario. Hemos escrito un *middleware* de ExpressJS que periódicamente limpia está caché para evitar sobrecargar de información redundante el servidor.

Al ser una aplicación web tenemos que desconfiar de los mensajes que recibimos. Para evitar ataques por inyección SQL o por XSS hemos hecho uso de un *middleware* que se encarga de realizar el escapado de los caracteres de un mensaje enviado por un usuario. Hemos encontrado información sobre este tipo de ataques en Ionel Jacob (2020) y Alsaffar et al. (2022).

4.4. Backend

Este componente se responsabiliza de la gestión de la base de datos de cara al frontend y de responder las peticiones que llegan desde el sistema IoT. Para esto hemos construido una API REST paralela al Frontend que se encarga de las responsabilidades ya mencionadas. En esta nueva arquitectura existe una parte que se encarga de la página web mientras que la otra parte se encarga de manejar los datos que entran y salen de la base de datos. Esta segunda parte es la que se ha realizado como una API REST y es el único elemento con acceso a la base de datos. Esta API no está diseñada para interactuar con el usuario, y solo considera comunicación con el Frontend y el cliente de Thingsboard (con el que se comunican los nodos Esp32) para el manejo de datos.

Para solo aceptar peticiones del Frontend y de Thingsboard se hace uso de las cabeceras de las peticiones. Al hacer las peticiones se debe añadir una cabecera X-Token con un identificador de cliente y un secreto. Esto permite descartar aquellas peticiones fraudulentas que no queremos que tengan acceso a los datos.

Para definir que servicios ofrece el Backend y cómo los ofrece se ha documentado su implementación con OpenAPI. Hemos usado la herramienta de edición de Swagger[5] para confecionar dicha documentación. Nos hemos decidido a utilizar OpenAPI y Swagger tras leer Casas et al. (2021).

Debido a la complejidad del modelo de datos, consideramos necesario estructurar la API de tal manera que sea posible obtener datos relacionados con otros, o incluso datos a partir de una tupla única de la entidad. Para ello, la gran mayoría de las rutas que se encarguen de siguen una de las siguientes convenciones:

- `GET /[Modelo]`: Devuelve el id de todas las entidades del modelo.
- `GET /[Modelo]/[ModeloID]`: Devuelve los datos del modelo identificado.
- `GET /[ModeloHijo]/[ModeloPadre]/[ModeloPadreId]`: Devuelve los ids de los modelos hijos relacionados con el modelo padre identificado por su id.
- `POST /[Modelo]/compose`: Permite enviar los atributos identificativos a través de el cuerpo de la petición. Esto significa que se pueden recuperar datos por campos distintos al id.

Las excepciones a estas convenciones son las rutas de seguimiento, rutas que proporcionen la capacidad de filtrado dentro de la petición a la hora de obtener datos, y rutas especiales que tienen en cuenta una funcionalidad única, como puede ser la petición de MACs esperadas en un espacio y una hora específicas.

Durante todo el proceso de implementación del Backend, se ha utilizado la librería de Sequelize[6] para manejar la comunicación del código con la base de datos.

[5]Disponible en `https://editor.swagger.io/`
[6]Disponible en `https://www.npmjs.com/package/sequelize`

Sequelize es una librería que aplica un modelo Object Relational Mapping (ORM). Esto nos ofrece una capa de abstracción entre el código y la base de datos, y permite por lo tanto no tener que preocuparnos por el SQL que hay debajo de las consultas que realizamos (Abba (2022)). A su vez, Sequelize nos ha resultado muy útil al hacer el cambio de SQLite a MySQL en la base de datos, pues se pudo reutilizar la gran mayoría de código.

Sequelize también nos ha ayudado a establecer un orden a los archivos para las migraciones y *seeders* del proyecto. Por una parte, las migraciones sirven para representar cualquier tipo de cambio en el esquema de los datos, mientras que por otro, los *seeders* ayudan a ejecutar grandes inserciones predeterminadas. Esto es muy útil a la hora de proporcionar datos iniciales a la base de datos, o a la hora de preparar la base de datos para un caso de pruebas.

Para mantener las contraseñas de forma seguro en la base de datos, se ha utilizado el algoritmo de *bcrypt*, el cual aplica ralentizado. A su vez, se añade a cada contraseña una cadena de caracteres aleatorios como sufijo, generada antes de insertarla en la base de datos denominada sal, y otra cadena de caracteres aleatorios elegida entre diez opciones ubicadas únicamente en el código del programa, denominada pimienta. Tras "especiar" la contraseña, se realiza el *hashing* con *bcrypt* y se inserta en la base de datos.

4.5. Consideraciones Generales

Durante el desarrollo de la aplicación, a pesar de su división, existen aspectos comunes en la totalidad de la aplicación. Estos se tratan en esta sección.

Para mantener la aplicación configurable, se ha utilizado la librería de `dotenv`. La configuración esta distribuida en una serie de archivos para dividirla en partes relacionadas por su uso. También existen valores por defecto para cada parámetro configurable.

La organización de la lógica interna se ha hecho en diferentes controladores. Para el Backend se ha dividido la funcionalidad considerando los caminos de esta, mientras que para el Frontend se ha separado por la historia de usuario en la que participan.

Para llevar un registro de las interacciones con la aplicación, se realizan *logs* informativos. Estos los se guardan en distintos archivos dependiendo de su naturaleza. Registramos tanto mensajes simples como mensajes más complejos que reflejen el estado la conexión entre usuario y aplicación. Durante el desarrollo esto nos ha permitido identificar problemas como la carga lenta de datos o peticiones erróneas.

Diariamente comprobamos que se hayan cubierto todas las asistencias previstas. En caso contrario se generan las ausencias correspondientes, usando un proceso independiente.

Nuestro sistema necesita ser alimentado a principio de curso con los horarios planificados. Para facilitar esta tarea hemos incluido un programa que carga estos datos de un archivo CSV.

Durante los procesos internos para calcular las instancias de las actividades es necesario convertir las ideas abstractas de actividad y recurrencia en fechas concretas en el mundo real. La hora en la actividad se guarda explícitamente como texto. El problema surge al intentar sintetizar las fechas reales de la instancia. En este proceso debemos tener en cuenta el Daylight Saving Time (DST) para que las fechas se muestren correctamente a los usuarios. El DST es la práctica de adelantar la hora durante primavera y de atrasarla en invierno para ajustar el horario con la luz del sol. A nivel técnico esto se codifica como un cambio de *offset* horario. Esto implica considerar el punto en el que se realiza ese cambio de hora para mantener las recurrencias de los eventos bien, o en caso contrario se esperarán actividades docentes una hora antes o después.

Cambiar el tipo de las horas de texto a `datetime` no solucionaría este problema, ya que tener el *offset* guardado no aporta información adicional. Para calcular la fecha real sigue haciendo falta algo que determine el offset actual, no el guardado en la base de datos.

La solución a la que hemos llegado es mantener el mismo *offset* para todo las columnas de tipo `datetime`, ya que pueden ser influenciadas por el DST. Dejando las horas como texto podemos asumir un *offset* común e introducir la zona horaria durante el manejo posterior.

Para implementar esta solución, hemos utilizado Momentjs[7], el cual la convierte las horas y fechas al *offset* actual de la máquina. Aplicando esta solución el proceso de obtener el tiempo de una actividad recurrente se traduce en:

1. Hallar la fecha en la que sucederá la instancia de la actividad, siguiendo las reglas de su recurrencia.
2. Obtener la hora local de esa instancia de actividad.
3. Juntar la fecha con la hora para obtener el momento en el tiempo en el que se dará la asignatura, en un *offset* específico.

El *offset* empleado es UTC (+00:00). Nos aseguramos de que todas las fechas enviadas entre el Frontend y Backend tomen este *offset*. Guardamos los tiempos en UTC en la base de datos, exceptuando las horas locales en las que suceden los eventos. Internamente los datos se codifican como está definido en el ISO 8601[8].

Esto sin embargo añade una restricción al código que maneje este tiempo. Momentjs transforma una fecha automáticamente al *offset* local. Tenemos que asegurarnos de que el código que realice esto se ejecute en el mismo *offset* en el que vaya a

[7]Disponible en `https://www.npmjs.com/package/moment`
[8]Para más información `https://www.iso.org/iso-8601-date-and-time-format.html`

suceder el evento para obtener la hora correcta. En nuestro caso los *offset* usados son los definidos por la zona horaria de *Europe/Madrid* según la Autoridad de Números Asignados de Internet (Internet Assigned Numbers Authority, o IANA), +01:00 en invierno, +02:00 en verano.

A pesar de utilizar Momentjs para el manejo de tiempos en el servidor no es posible hacer lo mismo en el navegador del cliente debido al tamaño de la librería. Se podría intentar optimizar su tamaño con algoritmos de tree-shaking. Estos algoritmos reducen el tamaño del código eliminando las funciones a las que no se llama nunca. Sin embargo, Momentjs ha demostrado inestabilidad con este tipo de algoritmos. Debido a estas consideraciones, hemos descartado la opción de calcular el *offset* del cliente desde su navegador. La alternativa es calcularlo en el servidor antes de enviarlo al cliente.

Capítulo 5

Hub IoT

Como ya introdujimos anteriormente el hub IoT que hemos elegido usar es Thingsboard. Thingsboard puede ocuparse de tareas como la actualización de firmware remota (OTA update[1]), la comunicación individualizada con los nodos, la autenticación de dispositivos, la generación de paneles de control o el aprovisionamiento. El objetivo de este capítulo es detallar los servicios que ofrece Thingsboard y de qué formas hemos personalizado algunos de ellos para ajustar su funcionamiento a nuestro proyecto.

5.1. Sobre ThingsBoard

ThingsBoard es una plataforma de código abierto[2] para la gestión de dispositivos IoT, la recopilación de datos y su análisis. Cuenta con una extensa documentación cubriendo sus funcionalidades. Proporciona una solución escalable y flexible para la implementación y gestión de aplicaciones IoT, integrando una amplia gama de funcionalidades clave:

- **Gestión de Dispositivos**: Registro, monitorización y control de dispositivos IoT de manera centralizada a través de un sistema de gemelos digitales.
- **Gestión de Usuarios**: Para sistemas IoT que requieran de operarios o de cara al cliente Thingsboard ofrece un sistema de usuarios y permisos para facilitar el acceso a la información a usuarios no administradores.
- **Recolección y Procesamiento de Datos**: Adquisición de datos de dispositivos en tiempo real y procesamiento mediante flujos configurables (*RuleChains*).

[1] Por sus siglas en inglés: *Over The Air update*
[2] El código puede encontrarse en `https://github.com/thingsboard/thingsboard`

- **Visualización de Datos**: Creación de paneles de control interactivos que permiten la visualización de datos en tiempo real.
- **Alertas y Notificaciones**: Configuración de reglas para generar alertas y notificaciones basadas en eventos o umbrales específicos.
- **Actualización de Firmware (OTA)**: Gestión remota de actualizaciones de *firmware* para dispositivos.
- **Seguridad y Autenticación**: Implementación de mecanismos de seguridad robustos, incluyendo TLS/SSL para la comunicación segura y autenticación basada en tokens.
- **Escalabilidad**: Soporte para arquitecturas escalables que permiten manejar grandes volúmenes de datos y dispositivos.
- **Integraciones**: Conexiones con sistemas externos mediante APIs RESTful y otros protocolos estándar como MQTT, CoAP y HTTP.

Todas estas funcionalidades se encuentran detalladas en la documentación The Thingsboard Authors (2024).

5.2. Gemelos digitales

Los gemelos digitales son representaciones virtuales de dispositivos físicos. En Thingsboard los gemelos digitales albergan los atributos que definen el estado del dispositivos físico. En nuestro caso Thingsboard gestiona los datos relacionados con la versión del *firmware* (necesario para el funcionamiento del OTA) y atributos de configuración como el tiempo entre *pings*. Para cada nodo Thingsboard crea una nueva instancia de gemelo digital configurable independientemente de las otras.

Cada gemelo digital pertenece a una familia de dispositivos (*device profiles*). Este concepto permite a Thingsboard gestionar correctamente nodos con diferente funcionalidad. Todos los dispositivos de una familia comparten propiedades a nivel administrativo, como qué usuarios de Thingsboard tienen permiso para verlos, en qué paneles de control aparecen sus datos, qué *RuleChain* gestiona sus mensajes, etc... Para nuestro proyecto solo es necesaria una familia de nodos ya que la funcionalidad de cara a Thingsboard de todos los nodos va a ser idéntica.

5.3. Autenticación

Thingsboard está encargado de la autenticación de los nodos. Antes de crear una instancia de gemelo digital es necesario que el dispositivo físico se autentique frente a Thingsboard. Para ello se crea una conexión MQTT sin autenticar y se intercambian

las claves. El nodo transmite las claves que identifican a su familia y Thingsboard responde con las claves que identifican a su gemelo digital recién creado.

Implantar este proceso en el sistema simplifica la tarea de añadir nuevos nodos y agiliza el escalado.

5.4. Protocolos de comunicación

Dentro de la amplia gama de protocolos que ofrece Thingsboard hemos elegido usar MQTT[3] para la comunicación con los nodos por ser ligero y no introducir complejidad innecesaria (Maria Salagean (2020)). MQTT se basa en un modelo de publicación/suscripción, donde los dispositivos pueden actuar tanto de publicadores de mensajes como de suscriptores de mensajes. En este estándar de comunicación los mensajes que emiten los dispositivos están siempre dirigidos a un *topic*. Para recibir mensajes los dispositivos deben haberse previamente subscrito al tema en el que se envía. Thingsboard organiza la comunicación a través de MQTT con *topics* de entrada a los que los nodos deben dirigir sus mensajes y *topics* de salida a los que los nodos deben estar suscritos para recibir las comunicaciones de vuelta.

En una red MQTT este intercambio ocurre a través de un broker MQTT central encargado de recibir y reenviar los mensajes a los dispositivos suscritos. En nuestro caso Thingsboard cubre esta necesidad, actuando como *broker* del resto de dispositivos en la red.

Adicionalmente este protocolo soporta autenticación por TLS/SSL. Con este mecanismo se autentica Thingsboard frente a los clientes. En la otra dirección se utilizan credenciales generadas durante el proceso de aprovisionamiento para autenticar el nodo frente a Thingsboard.

MQTT también ofrece niveles de calidad de servicio. Esta funcionalidad permite reducir aun más el tamaño de la transmisión a costa de la fiabilidad. En nuestro caso hemos optado por la calidad más alta para evitar problemas de comunicación ya que no esperamos que haya problemas relacionados con la conectividad.

La comunicación de Thingsboard al Backend se llevará a cabo usando HTTP. El Backend es una API REST. Mediante las *RuleChains* que comentaremos a continuación transformaremos algunas peticiones de los nodos en peticiones a la api REST y las respuestas del Backend de vuelta a mensajes MQTT. De esta forma el Backend solo debe verificar la identidad de Thingsboard, que se lleva a cabo con un secreto compartido que se envía en la cabecera de las peticiones. Thingsboard verifica la identidad del servidor por TLS/SLL con HTTPS.

Estas medidas de seguridad crean un canal seguro desde el nodo hasta el Backend, evitando así posibles ataques de suplantación y la recogida de datos fraudulentos.

[3]De sus siglas en ingles Message Queues Telemetry Transport, definido en el RFC 9431 (M'Raihi, et al (2011))

5.5. RuleChains

Thingsboard también puede actuar como un dispositivo más dentro de la red MQTT que intercambia mensajes e interactúa con el resto. Su funcionamiento se controla a través de las *RuleChains*. Cada perfil de dispositivo puede tener su propia *RuleChain* asignada que gestiona todos los mensajes de sus instancias. En el caso de este proyecto hemos creado 2 *RuleChains* personalizadas. La primera es la *RuleChain* principal de los dispositivos de perfil "SeguimientoDocente"que se puede ver en la figura 5.1.

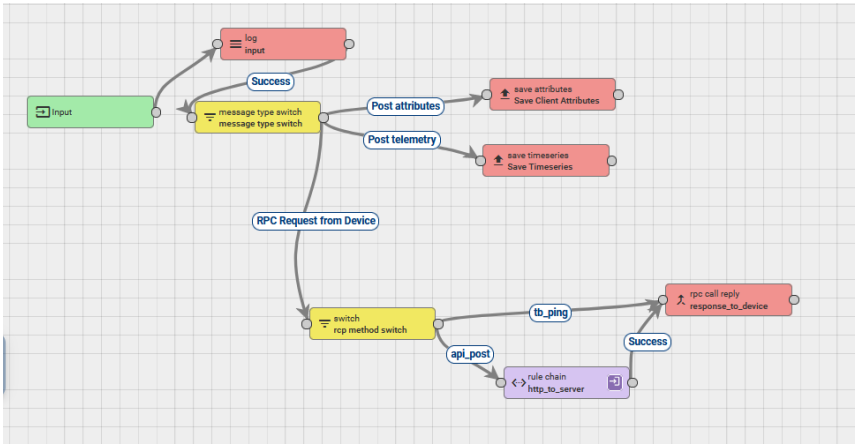


Figura 5.1: *RuleChain* raíz para el perfil SeguimientoDocente

Esta *rulechain* recibe los mensajes y decide si son un *ping* a Thingboard o una comunicación con el servidor del Backend. Si son un *ping* inmediatamente se envía una respuesta. Cuando el mensaje va dirigido al Backend se desempaqueta y se envía a una segunda *RuleChain* (figura 5.2).

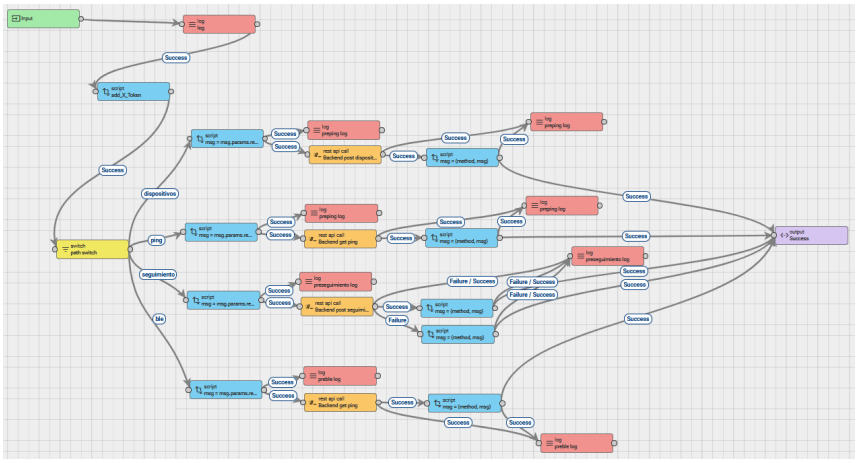


Figura 5.2: *RuleChain* para la gestión HTTP

Esta *RuleChain* gestiona únicamente las comunicaciones con el Backend. Decide según el contenido del mensaje a que ruta de la API REST debe mandarlo, genera la URL del destino, añade las credenciales necesarias y lo envía.

Una vez la petición al Backend tiene respuesta se encapsula de forma que el nodo pueda procesarla y se envía de vuelta por MQTT.

Estas comunicaciones ocurren a través de los *topics* `v1/devices/me/rpc/request/X` y `v1/devices/me/rpc/response/X` (sustituyendo X por el id del mensaje).

Capítulo 6

Nodo IoT

Una vez elegida a Espresif como fabricante empezamos en desarrollo con la placa ESP32-devkitc-v4 (figura 6.1). Esta placa lleva un módulo ESP32-WROOM-32E equipado con Wi-Fi, Bluetooth, 520KB de SRAM, 16KB extra de SRAM RTC, 448KB de ROM, 4MB de flash[1]. Todo el código del TFG en relación con el dispositivo se encuentra en GitHub[2].

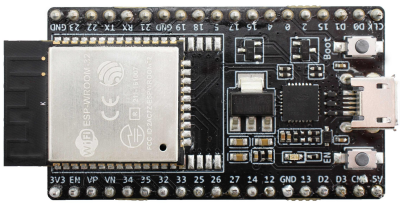


Figura 6.1: esp32-devkitc-v4

A lo largo del desarrollo se fueron introduciendo más requisitos y aumentando la huella de memoria del programa. Cuando la memoria flash de la placa devkitc-v4 se quedó pequeña dimos el salto a otra placa de desarrollo: ESP32-s3-eye (figura 6.2). Esta plataforma de desarrollo incorpora una pantalla oled, una cámara y 4 botones más, pero no expone pines para conexiones GPIO. El chip principal es un ESP32-S3-WROOM-1. Está equipada con más 384KB de ROM, 512KB de SRAM, 16KB de SRAM RTC, 8MB de Flash y 2MB de PSRAM. También mantiene las capacidades Bluetooth y WiFi del chip ESP32-WROOM-32E. La principal diferencia son los chips de flash y PSRAM que vamos a usar para permitirnos tener una ROM más grande y descargar la memoria principal de las responsabilidades WiFi y Bluetooth[3].

[1]Más información sobre este system-on-chip se puede encontrar en `https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf`

[2]`https://github.com/jaimegonzalezfabregas/TFG-ASD-Devices`

[3]Más información sobre este system-on-chip se puede encontrar en `https://www.espressif.com/sites/default/files/documentation/esp32-s3-wroom-1_wroom-1u_datasheet_en.pdf`

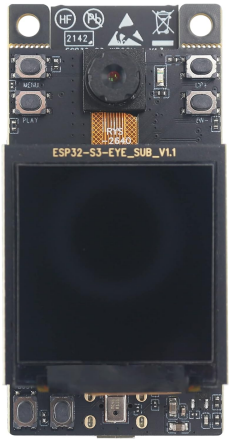


Figura 6.2: Imagen del microcontrolador esp32-s3-eye

En esta nueva plataforma se continuó el desarrollo hasta que se procedió a la integración del lector RFID. La falta de GPIO nos empujó a volver a integrar el devkitc como micro auxiliar para poder conectarnos a los pines de la antena.

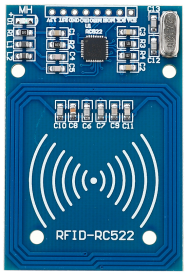
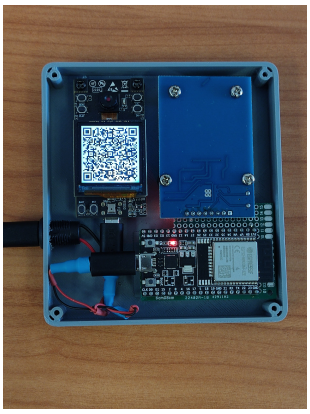


Figura 6.3: chip RFID-RC522

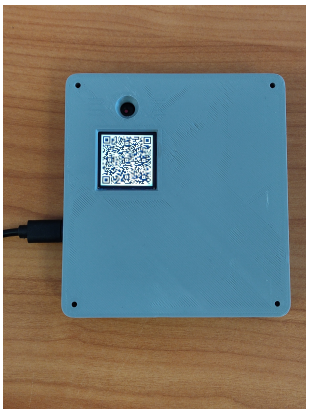
La configuración final del nodo está integrada por los dos microprocesadores comunicados a través de mensajes Bluetooth y el solenoide RFID conectados al GPIO del primero. Como se puede observar en la figura 6.4a, distribuimos la energía con un bifurcador de usb-micro-b.

6.1. Capacidades de Identificación de profesores

El objetivo de instalar un nodo IoT en las aulas es proporcionar un terminal físico con el que los profesores puedan interactuar para identificarse. Debido a la versatilidad del sistema elegido hemos investigado muchas alternativas para llevar a cabo esta tarea. En esta sección las trataremos con detalle y expondremos nuestras conclusiones a cerca de su viabilidad.



(a) Disposición del nodo IoT



(b) Diseño final del nodo IoT

6.1.1. Identificación por TOTP

El estandar TOTP[4] está definido en el RFC 6238 ((M'Raihi, et al, 2011)). El objetivo de esta tecnología es verificar que el otro extremo de la comunicación conoce el secreto común. Para verificar que no esta siendo víctima de un ataque de *replay*[5] el algoritmo integra en el código TOTP el tiempo del sistema, invalidando los códigos antiguos.

El uso más común es como doble factor de autenticación. Nosotros le vamos a dar el papel de testigo para probar la localización, usándolo para demostrar que el usuario se encuentra en presencia del dispositivo con la clave. No somos los primeros en considerar está idea (Onur C¸ iftci y Serif (2022)).

En este caso el registro de asistencia se hace a través del portal web en el móvil del docente. En el nodo se muestra un QR con la URL al formulario de registro. Esta URL llevará información extra que auto-completará en qué aula está el docente y el campo TOTP del formulario. Después de verificar la identidad del docente, si los datos son correctos, se registra la asistencia.

Hemos prestado mucha atención a la experiencia de usuario, ya que competimos contra una firma en un papel. El proceso de rellenado del formulario y autenticación serán mayoritariamente automáticos, asistidos por la información de sesiones pasadas guardada en el móvil del docente y por la información que el nodo inyecta en la URL. Esto reduce el proceso a una lectura de QR, identificación y pulsar en aceptar.

Al contrario de otros sistemas que comentaremos a continuación, este mecanismo no depende de que funcione la comunicación entre el dispositivo y el servidor durante la autenticación, ya que el registro de efectúa a través de la web.

[4]De sus siglas en inglés Time-based one-time password
[5]`https://es.wikipedia.org/wiki/Ataque_de_REPLAY`

Generación del QR

El QR que se ve en la pantalla del nodo se genera teniendo en cuenta la hora del sistema y la configuración guardada en la partición de la flash dedicada a NVS[6]. En NVS se encuentra el secreto, el código de aula, el código de dispositivo y la URL base sin argumentos GET; a partir de todos esos datos, se construye la URL final. Por otro lado, la implementación del algoritmo TOTP es una adaptación del código de David M. Syzdek[7].

6.1.2. Identificación por TUI

La UCM proporciona a sus alumnos y docentes una TUI (Tarjeta Universitaria Inteligente) para autenticación automática en los diferentes servicios que ofrece. Sería deseable integrar este formato de autenticación en nuestra propuesta porque no requeriría la configuración previa que otras de las funcionalidades sí. A continuación, se presentan los resultados de los intentos de integración.

Código de barras

En el reverso de las tarjetas se encuentra un código de barras identificativo. La primera aproximación a este problema consistía en intentar leer el código de barras con la cámara del esp32-s3-eye. La baja calidad de la imagen producida unida a los límites de tamaño de las imágenes impuestos por el tamaño de la memoria hacen que esto sea imposible. La lectura de códigos de barras está limitada a hardware especializado o a procesadores más potentes. Tras considerar todo esto determinamos que esta forma de identificar la TUI no era viable dentro de los confines de este proyecto.

TUI virtual

La UCM distribuye junto a la tarjeta física una aplicación para móvil que permite mostrar un QR identificativo. Este QR no contiene directamente la información de la tarjeta sino un código de un solo uso generado dinámicamente en sus servidores. Para hacer uso de este método de autenticación necesitaríamos acceso a la API del servidor de autenticación, lo que podría resultar en un problema de confidencialidad al sistema ya establecido.

Otro problema que tiene este método tiene que ver con el diseño de la aplicación. El QR que se muestra tiene mas de 300 bytes de información, produciendo una imagen con detalle muy fino. Esto añadido a que el QR deja un margen a su alrededor

[6]De sus siglas en inglés, *Non Volatile Storage*
[7]`https://gist.github.com/syzdek/eba233ca33e1b5a45a99`

muy grande produce una imagen demasiado pequeña y detallada para que se pueda enfocar con la cámara de lente fija del microcontrolador.

Estos 2 obstáculos nos obligaron a abandonar la idea y considerar una tercera alternativa, el chip RFID.

RFID

Las tarjetas físicas contienen un chip RFID que proporciona cierta inteligencia a la tarjeta. Estos chips pueden utilizarse para leer y escribir de la pequeña memoria interna de la tarjeta. En otros ámbitos de uso los chips pueden llegar a guardar imágenes, certificados público-privados[8] o información bancaria[9]. Nosotros no estamos interesados en la información que contiene la tarjeta, sino solamente en identificarla. Para eso la UCM utiliza en sus otros sistemas establecidos el número de serie de las tarjetas. El estándar RFID no especifica que este número deba ser único, pero consultando a los técnicos que han desarrollado los sistemas existentes nos confirmaron que nunca se han encontrado duplicados.

Para la lectura de chips RFID se ha utilizado el chip RFID-RC522 (figura 6.3).

El primer problema que debemos abordar es que el microcontrolador principal no tiene pines GPIO expuestos que podamos dedicar a la comunicación con la antena. La forma de salvar este obstáculo es usando otro micro de un modelo que si tenga GPIO y comunicarlos por Bluetooth. Como ya mencionamos al principio del capítulo el micro elegido fue el Esp32-devkitc-v4. La conexión entre el s3-eye y el devkitc ocurre a través de los mensajes Bluetooth del segundo. El micro auxiliar envía en sus mensajes de descubrimiento GAP el estado de la ultima tarjeta escaneada. El datagrama de este protocolo tiene reservados 24 bytes de información específica del fabricante que podemos utilizar para este objetivo.

Este microcontrolador auxiliar se podría sustituir con cualquier otro microcontrolador con capacidad Bluetooth y pines GPIO suficientes.

Este nuevo microcontrolador se conecta al chip RFID-RC522 usando la librería escrita por Alija Bobija[10] al efecto. Cuando se detecta una tarjeta RFID se extrae su número de serie (que ocupa 8 bytes) y se emite en los mencionados 24 bytes de información del fabricante.

Cuando desde el esp32-s3-eye se detecta el cambio en este campo se envía con un mensaje MQTT a Thingsboard y se queda a la espera de la respuesta del Backend. En la respuesta está codificado el resultado de la operación, que puede ser uno de los siguientes.

[8] `https://www.dnielectronico.es/PortalDNIe/PRF1_Cons02.action?pag=REF_240`

[9] `https://usa.visa.com/pay-with-visa/contactless-payments/contactless-payments.html`

[10] El código fuente original se puede encontrar en `https://github.com/abobija/esp-idf-rc522`

- **Ok**: Se ha reconocido el usuario y se ha registrado la asistencia.
- **Aviso**: Se ha reconocido el usuario pero no tiene que estar en este aula.
- **Error**: No se ha reconocido la tarjeta.

Este resultado se muestra al usuario con un icono en la pantalla durante unos segundos.

6.1.3. Identificación por Bluetooth Low Energy

El microcontrolador está constantemente monitorizando el ambiente en búsqueda de paquetes BLE. El objetivo de la monitorización es encontrar dispositivos activos por Bluetooth, como pulseras de actividad, relojes o móviles. Más tarde podremos cotejar los dispositivos encontrados con los dispositivos de los docentes que esperamos que se registren en el aula y deducir la asistencia de esa forma.

Existen varios perfiles de comunicación en el estándar BLE. Nosotros estamos interesados en el perfil GAP[11]. Este perfil se especializa en el envío de paquetes *broadcast* y se usa en la anunciación del dispositivo.

El nodo registra la MAC de todos los paquetes GAP. Cada pocos minutos los nodos piden al servidor central una lista de las MACs que se espera encontrar en el lugar donde estén, se cotejan con la historia de dispositivos y se mandan las coincidencias de vuelta.

Este sistema de identificación tiene varios puntos débiles. Para empezar, es complicado de configurar. Los docentes tienen que encontrar un rato para buscar e introducir en el sistema las MACs que quieren que estén vinculadas con ellos. Además la divulgación de identidad solo está activa en aparatos que estén buscando una conexión. Este es el caso de los relojes y las pulseras de actividad mientras no estén emparejadas. Si el docente ha sincronizado previamente sus dispositivos con su móvil no emitirán paquetes.

Además de los relojes y las pulseras de actividad también es posible hacer que el móvil o el ordenador anuncien su presencia mediante mensajes GAP. Sin embargo, en Android, Windows 11, Linux y posiblemente otros sistemas, el dispositivo solo anuncia su presencia mientras el usuario escanea dispositivos cercanos. Esto limita este método a un rol secundario dentro de las opciones que ofrece nuestro nodo. A pesar de esto, consideramos importante introducirlo para poder mejorar sobre ello en el futuro y como método redundante para los casos en los que el docente no se acuerde de hacer la gestión.

[11]Por sus siglas en inglés Generic Access Profile

6.1.4. Identificación biométrica

Vale la pena mencionar que se investigó la posibilidad de usar un lector de huellas dactilares para identificar a los docentes. Esta forma de autenticación desde el principio se clasificó como de baja prioridad, ya que en primera instancia conlleva un aumento fuerte en los requisitos de seguridad del sistema. Aunque actualmente estamos tomando todas las medidas razonables para garantizar la confidencialidad de los datos añadir al conjunto datos biométricos aumentaría la exigencia y complicaría mucho la gestión de identidad en todos los aspectos. Para implementar lectura de huellas debemos transmitir y recibir el vector identificativo desde la base de datos del Backend hasta el microcontrolador auxiliar, obligándonos a cambiar la estrategia de comunicación entre las placas de desarrollo ESP32 a una basada en GATT para incrementar en mucho la tasa de transferencia de datos.

Además de todo esto no hemos encontrado un sitio en la planificación para desarrollo, por lo que la tarea se quedó en la fase de investigación teórica.

6.2. Administración del Nodo

Además del funcionamiento normal el nodo IoT debe estar preparado para las tareas de puesta en marcha y mantenimiento. En esta sección detallaremos en que funcionalidades ofrece en este aspecto.

6.2.1. OTA

Tras la etapa de despliegue de este sistema se tendrán que gestionar docenas de nodos IoT alejados. Una actualización de *firmware* OTA[12] ocurre de forma remota, sin necesidad de interacción física con los nodos. Sin esta funcionalidad las actualizaciones del *firmware* conllevarían mucho trabajo manual administrativo.

Una vez disponemos de la nueva imagen con la que queremos actualiza los nodos IoT, el proceso OTA comienza desde la interfaz administrativa de Thingsboard. El primer paso es informar a Thingsboard de la existencia de un nuevo paquete del tipo *firmware*. El administrador después tiene que asignar esa versión al gemelo digital de un dispositivo existente o a una familia de dispositivos completa para que comience el proceso automático.

Inmediatamente después de la asignación del *firmware* se recalculan y modifican automáticamente los atributos relacionados con el OTA para cada gemelo digital afectado:

1. `fw_checksum`

[12]De sus siglas en inglés Over The Air

2. `fw_checksum_algorithm`
3. `fw_size`
4. `fw_tag`
5. `fw_title`
6. `fw_version`

El cambio de estos atributos dispara un mensaje MQTT al dispositivo en cuestión. El dispositivo, tras hacer las comprobaciones necesarias, accede por HTTPS a la nueva imagen (fichero .bin enviado anteriormente a Thingsboard). La imagen binaria se encuentra en /api/v1/[access_token]/firmware/?title=[titulo del fw]&version=[versión del fw].

El proceso de descarga puede ser lento. Para evitar que el usuario interfiera se paralizan el resto de tareas que no son esenciales para el OTA, como la recogida de fotogramas de la pantalla o la generación de códigos TOTP entre otros.

A partir de este punto la librería de Espressif encargada del OTA hace la mayor parte del trabajo: descarga el *firmware* en la partición que corresponda y reinicia el dispositivo[13]. Para hacer este proceso compatible con el sistema de OTA de Thingsboard el proceso normal de OTA que recomienda Espressif está intercalado de envios MQTT con mensajes comunicando el progreso de la actualización a Thingsboard. Este *feedback* automático tiene la doble función de mostrar al administrador en su interfaz el estado del proceso y ayudar en caso de fallo a encontrar la causa del problema.

No es posible escribir en la partición de *boot* mientras el dispositivo está encendido sobre ella sin consecuencias imprevisibles, por lo que un sistema que soporte OTA tiene al menos 2 particiones dentro del dispositivo de almacenamiento Flash. Estas particiones se turnan el rol de partición de boot para que durante la descarga se guarde en una partición que no esté en uso. En nuestro caso tenemos 3 particiones donde puede encontrarse el firmware: `Factory`, `OTA1` y `OTA2`. `Factory` contiene la versión del *firmware* original y se conserva para poder recuperarse de posibles corrupciones durante la transmisión de la imagen. Las otras dos particiones albergan las imágenes nuevas fruto de una actualización OTA.

Durante el primer arranque de una versión nueva se deben hacer las comprobaciones necesarias para garantizar la integridad y marcar la versión como válida. Si no se marca como válida en el siguiente arranque que ocurra se efectuará un *rollback* a la última partición validada. Esta medida de seguridad mitiga la posibilidad de que un sistema se quede en un estado inconsistente.

Si todas las comprobaciones de encendido terminan satisfactoriamente además de marcar la imagen como válida se envía el último mensaje de estatus a Thingsboard informando que la actualización a finalizado con éxito.

[13]La documentación de este modulo de Espressif se encuentra en `https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/ota.html`

6.2.2. Aprovisionamiento

La escalabilidad de proyecto ha sido una prioridad desde el principio. Esto significa que no nos podemos permitir compilar ROMs exclusivas para cada dispositivo con el fin de configurarlos individualmente ya que esta técnica de configuración individual conllevaría un coste muy grande al escalar. El proceso de aprovisionamiento debe ocurrir durante tiempo de ejecución en el primer encendido.

El provisionamiento tiene 3 fases, correspondientes con la configuración inicial, la obtención de credenciales para Thingsboard y la obtención de credenciales para el Backend.

Aprovisionamiento Físico

Hemos optado usar la cámara del esp32-s3-eye para introducir, por medio de una secuencia de códigos QR, la información de aprovisionamiento. Esta fase se encarga de configurar los siguientes parámetros:

- `wifi_ssid`: Nombre de la red WiFi. Los nodos en este proyecto han usado la red `UCMOT`.
- `wifi_psw`: Contraseña de la red WiFI.
- `thingsboard_url`: URL donde encontrar el servicio HTTP de Thingsboard.
- `mqtt_broker_url`: URL donde encontrar el servicio MQTT de Thingsboard.
- `device_name`: Nombre identificativo del dispositivo para humanos.
- `space_id`: Identificación del espacio donde se encuentra (número de aula/lab).
- `provisioning_device_key`: Credenciales de Thingsboard.
- `provisioning_device_secret`: Credenciales de Thingsboard.
- `totp_form_base_url`: URL base para generar el QR con el TOTP.

Para producir la secuencia de QR se ha desarrollado una pequeña utilidad web publicada en GitHub[14]. Esta utilidad te permite reconfigurar el dispositivo desde cero o sobrescribir parte de la configuración existente, pudiendo aportar solamente algunos de los campos para acelerar la transmisión de datos.

En este paso también es posible invalidar las credenciales obtenidas por el Backend o por Thingboard en caso de que sea necesario por cambios imprevistos. Esto producirá que durante el proceso de recuperación de la conexión se vuelva a pedir una identidad nueva a los diferentes servicios.

[14] `https://htmlpreview.github.io/?https://github.com/jaimegonzalezfabregas/TFG-ASD-Devices/blob/main/provisioning_app/dist/index.html`

La transmisión está divida en muchos códigos QR para facilitar la lectura de cada código individual. Intentar transmitir toda la información en un único código no es viable. La cantidad de información puede variar dependiendo del tamaño de los datos, por lo que es necesario un código QR de cabecera que configura la recepción de datos, seguido de los datos en sí.

La lectura de los códigos QR no siempre toma el mismo tiempo, por lo que es posible que un QR del medio de la secuencia no se transmita. Por eso todos los QR tienen su número de segmento integrado y la transmisión comienza con un QR cabezera con metadatos. La aplicación de aprovisionamiento continua repitiendo la secuencia de códigos QR hasta que la transmisión termina.

Aprovisionamiento de Thingsboard

Si los datos del aprovisionamiento físico están completos se puede proceder a registar este dispositivo en Thingsboard. Para eso se sigue el protocolo detallado en la documentación de Thingsboard[15]. Para iniciar la conexión de aprovisionamiento el dispositivo no puede usar credenciales propias, ya que no tiene todavía asignadas. Por esa razón la conexión de MQTT de aprovisionamiento se establece autenticado con el usuario `"provision"` y sin contraseña. El dispositivo envía a Thingboard su nombre y las claves de perfil que ha obtenido del aprovisionamiento físico. Thingsboard reconoce las claves como relacionadas con un perfil de dispositivo y crea una instancia de dispositivo de ese perfil con el nombre proporcionado. Además devuelve la clave de acceso al dispositivo virtual para que el dispositivo físico pueda vincularse más adelante. Esta comunicación ocurre a través de los topics `/provision/request` y `/provision/response`.

Aprovisionamiento de Backend

Cuando un dispositivo nuevo accede al Backend es su responsabilidad presentarse para que el Backend le genere su secreto TOTP y le tenga en consideración de ese momento en adelante. Este aprovisionamiento ocurre con una llamada al endpoint POST `/api/v1/dispositivos` del Backend que lleva el nombre y la id del espacio. No es necesario comprobar de nuevo que es un dispositivo de confianza ya que al Backend le llega el mensaje desde Thingsboard por una vía segura y el dispositivo ya se autenticó ante Thingsboard.

El Backend devuelve los parámetros `secret` y techt0, parámetros necesarios para el funcionamiento del TOTP. Una vez hecho este trámite el dispositivo será capaz de generar los códigos TOTP aún cuando se pierda la conexión.

[15] `https://thingsboard.io/docs/paas/user-guide/device-provisioning/`

Capítulo 7

Firmware del Nodo IoT

Este capítulo da una visión más específica de la implementación del nodo IoT. Ponemos el foco en la estructura interna del código, los patrones utilizados y las diferentes soluciones algorítmicas para cubrir los requisitos que nos hemos propuesto en los capítulos anteriores.

7.1. ESP-IDF

Para el desarrollo de este proyecto hemos elegido el framework que proporciona la empresa Espressif (Espressif IoT Development Framework).

ESP-IDF se distingue por su enfoque en microcontroladores altamente integrados y su amplio soporte para protocolos de conectividad como Wi-Fi y Bluetooth. Mientras que Arduino es conocido por su simplicidad y facilidad de uso, Espressif ofrece una mayor potencia de procesamiento y capacidad de integración, lo que lo hace más adecuado para proyectos IoT que requieren funcionalidades de más bajo nivel como el nuestro.

Espressif también pone a la disposición de sus usuarios un extenso manual con documentación Espressif Systems (Shanghai) Co., Ltd. (2016 - 2024).

7.2. Arquitectura

El software que debe correr en los microcontroladores tiene requisitos funcionales muy altos. Esto resulta un problema en una plataforma tan simple. Para suplir la falta de un framework escalable hemos desarrollado una arquitectura basada en los hilos de freeRTOS y colas de mensajes que permite repartir responsabilidades a diferentes módulos desacoplados.

Además del sistema de paso de mensajes apareció la necesidad de gestionar un estado menos espurio. Para ello todos los módulos tienen acceso a la dependencia común `sys_mode`, que usando `xSemaphore` de freeRTOS gestiona el acceso de cada hilo a una zona de memoria común en la que se guarda estado del sistema.

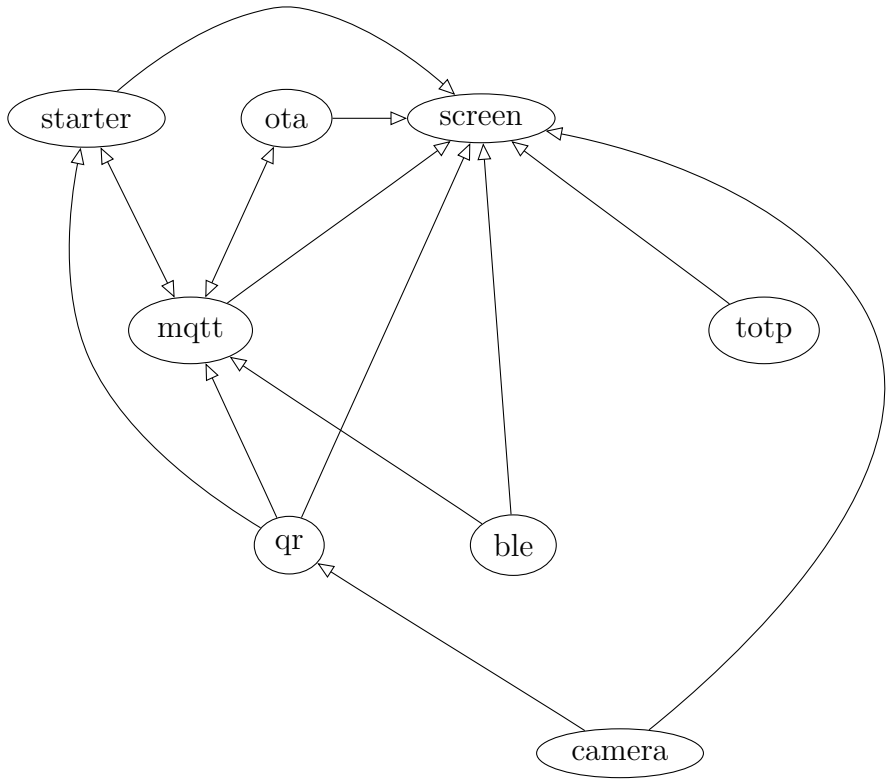


Figura 7.1: Arquitectura de los módulos dentro del Nodo IoT

Las lineas continuas representan los canales de comunicación entre módulos y la dirección de los mensajes.

Resumen de módulos ilustrados en la figura 7.1 y sus responsabilidades:

- **Starter**: Orquestra el resto de módulos. Su carga de trabajo más grande ocurre al encender el dispositivo, pero también es el encargado de recuperarse de los fallos.
- **OTA**: Este módulo contiene la lógica para llevar a cabo las actualizaciones Over The Air.
- **Screen**: Este módulo se responsabiliza del contenido de la pantalla.
- **MQTT**: Gestiona la entrada y salida de mensajes MQTT.
- **TOTP**: Calcula las cadenas de Time-based One Time Password para mostrarlas en la pantalla.
- **QR**: Recibe las imágenes de la cámara y busca en ellas códigos QR.

- **BLE**: Haciendo uso de la librería NimBLE escanea en búsqueda de paquetes GAP para determinar la MAC Bluetooth de dispositivos cercanos.
- **Camera**: Recoge y envía las imágenes de la cámara. También es el encargado de gestionar la memoria que consumen los fotogramas de la cámara.

7.3. Boot y Recuperación de errores

Como se muestra en la figura 3.1 entre el dispositivo y el servidor central existen varios sistemas de conexión y todos ellos son susceptibles a sufrir tiempo de inactividad. La recuperación de errores se ha integrado en el sistema de encendido, ya que el proceso de conexión es el mismo tras un fallo de red que durante el primer encendido.

Además Thingsboard y el Backend necesitan identificar a cada dispositivo individualmente, por lo que es necesario un proceso de una sola vez para el primer aprovisionamiento que también debe ocurrir durante el encendido.

Para cubrir todos estos requisitos las acciones del sistema de encendido es guían por la máquina de estados de la figura 7.2. Todos los estados están definidos los siguientes parámetros.

- **Condición de éxito**: Que debe cumplirse para pasar al estado de éxito.
- **Estado de éxito**: Siguiente estado.
- **Rutina de intento**: Función que intenta cambiar el estado del sistema para que se cumpla la condición de éxito.
- **Política de reintentos**: Esto define la cantidad de intentos y como espaciarlos en el tiempo. Algunas implementaciones de esta propiedad son el espaciado constante, el espaciado lineal o el *exponential backoff*.
- **Estado de recuperación**: Si la política de intentos determina que es imposible satisfacer la condición de éxito se baja a algún estado anterior para intentar reparar el problema.

A continuación hay una explicación general de cada estado y sus transiciones.

- `NoQRConfig`: Este es el estado en el que el sistema permanece cuando no se encuentra la configuración inicial. Para salir de este estado es necesario completar el proceso de aprovisionamiento mencionado anteriormente en la sección 6.2.2. Si cuando se llega a este estado ya están los datos necesarios en la memoria NVS se salta a `NoWifi` inmediatamente. La rutina de intento de este estado está vacía, ya que el sistema de lectura de códigos QR funciona de forma independiente.

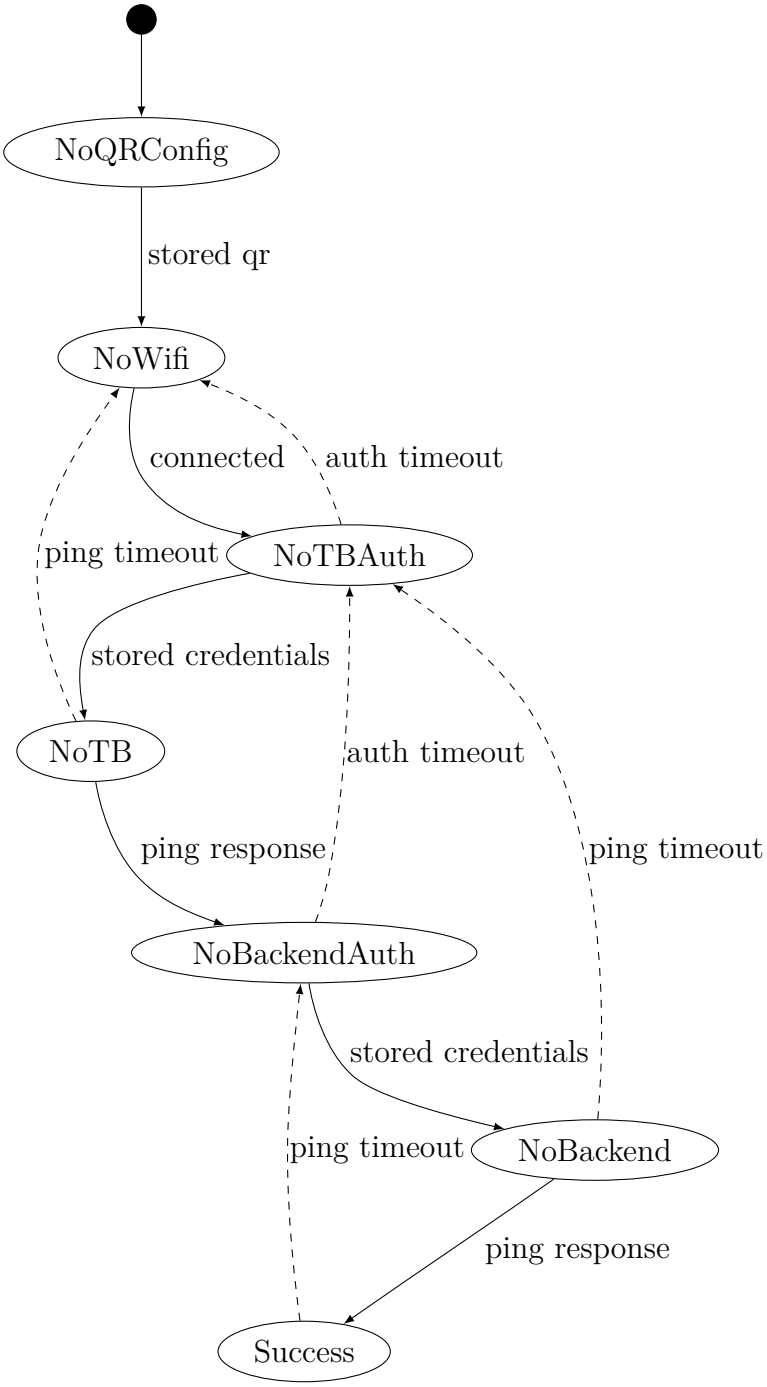


Figura 7.2: Maquina de estados del sistema de encendido

Las lineas continuas representan las transiciones cuando se completa la condición de éxito, las lineas discontinuas representan las transiciones de recuperación

- `NoWifi`: La conexión con el AP configurado se establece en este estado. Cuando este estado agota todos los intentos vuelve a si mismo. Cuando la conexión se termina de establecer se pasa a `NoTBAuth`.
- `NoTBAuth`: Este estado, al igual que `NoQRConfig`, espera a que en la memoria no volátil se encuentren las credenciales de Thingsboard. Si las credenciales

ya están ahí se continua al estado `NoTB` inmediatamente. La rutina de intento establece una conexión MQTT ejecuta los pasos detallados anteriormente en la sección 6.2.2.

- `NoTB`: Este estado reinicia la conexión MQTT con la autenticación conseguida en el estado anterior y las subscripciones necesarias para el funcionamiento normal. El objetivo de este estado es asegurarse de que la conexión con Thingsboard funciona antes de comenzar a enviar mensajes a través de él. Esto se hace mediante un ping. Este estado es redundante si acabamos de pasar por `NoTBAuth`, pero en la mayoría de casos ese estado se saltará ya que el aparato recordará sus credenciales pasadas.

- `NoBackendAuth`: Una vez la distribución de mensajes a través de Thingsboard está operativa se comprueba si tenemos credenciales para el Backend. En caso negativo se llevan a cabo los pasos detallados anteriormente en la sección 6.2.2.

- `NoBackend`: Igual que el estado `NoTB` este estado es importante para los encendidos en los que saltamos `NoBackendAuth`. Mediante un *ping* determinamos si tenemos conexión con el Backend y sincronizamos los relojes. La sincronización del tiempo es necesario para el funcionamiento del TOTP y se lleva a cabo en cada *ping* que se lleva a cabo. El reloj interno de los dispositivos se desvía un segundo cada 6 horas aproximadamente. Este resultado está basado en la fiabilidad de los dispositivos usados durante el prototipado.

- `Success`: Cuando el dispositivo está en este estado se desbloquea envío general de mensajes MQTT para otros módulos y se envían *pings* periódicos para mantener en hora el RTC[1]. Si se pierde la conexión se vuelve a `NoBackendAuth`.

7.4. Gráficos

La gestión gráfica de la pantalla la lleva a cabo el módulo `screen`. Este modulo traduce las necesidades del resto de módulos en instrucciones para la librería LVGL[2].

Esta librería gráfica está enfocada para sistemas muy poco potentes, lo que la hace perfecta para nuestro caso de uso. La propia librería esta preparada para integrarse con los productos de Espressif[3].

En particular, en nuestra implementación esta librería se encarga del renderizado de texto, imágenes y la generación del código QR a partir de sus contenidos.

[1] De sus siglas en inglés Real Time Clock
[2] `https://www.lvgl.io/`
[3] `https://docs.lvgl.io/7.11/get-started/espressif.html`

7.5. Particiones

En un microcontrolador como los que estamos usando la memoria flash está dividida en particiones. Cada partición es un segmento de memoria dedicado a una tarea en específico. ESP-IDF genera automáticamente la tabla de particiones óptima conociendo el dispositivo y basándose en la configuración. En nuestro proyecto la tabla de particiones esta especificada manualmente como se muestra en al figura 7.3.

```
# ESP-IDF Partition Table
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x4000,
otadata,  data, ota,     0xd000,  0x2000,
phy_init, data, phy,     0xf000,  0x1000,
factory,  app,  factory, 0x10000,  2500K,
ota_0,    app,  ota_0,   0x290000, 2500K,
ota_1,    app,  ota_1,   0x510000, 2500K,
```

Figura 7.3: Tabla de particiones

- La partición **`nvs`** es utilizada para el almacenamiento no volátil, donde se guardan de manera persistente pares clave-valor.
- La partición **`otadata`** está dedicada a los datos OTA, que incluyen información sobre la versión del *firmware*, la ranura en uso y la ranura siguiente.
- La partición **`phy_init`** almacena los datos de inicialización PHY, esenciales para la configuración y calibración de la capa física de la comunicación inalámbrica.
- La partición **`factory`** contiene el firmware de fábrica, que es el software inicial del dispositivo. Proporciona un punto de partida y una versión a la que volver en el caso de que las otras dos ranuras se corrompan.
- Las particiones **`ota_0`** y **`ota_1`** son usadas para el *firmware* OTA. Permiten actualizar el *firmware* de manera remota sin interrumpir las operaciones normales del dispositivo. Durante el funcionamiento normal la partición que no se está usando como partición de encendido contiene la versión anterior del código. Si durante el primer encendido de una imagen nueva se detecta un

problema se descarta y revierte la actualización. Este proceso utiliza los datos guardados en la partición `otadata` para tomar la decisión final.

Capítulo 8

Pruebas

Con motivo de comprobar que nuestra aplicación se adapta a las necesidades de los posibles usuarios, y para comprobar la viabilidad de esta tras una gran serie de iteraciones, hemos hecho una serie de pruebas. Para estas pruebas, hemos ideado un flujo de tareas con datos de prueba, y hemos proporcionado al usuario un guión a seguir, intentando emular el comportamiento que seguiría si usase la aplicación para cumplir sus objetivos.

Las pruebas se han realizado presencialmente el día 17 de mayo de 2024, llevando primero a la persona usuaria al dominio de la aplicación, y tras esto, dejando que navegue por la aplicación por si misma, para así ver si es lo suficientemente intuitiva frente a un nuevo usuario. Como se ha dicho antes, se ha proporcionado al usuario un guión a seguir, hablando con el usuario únicamente si necesitaba aclaraciones sobre las funcionalidades o el uso de alguna pantalla.

Estas pruebas se han realizado para los roles de administración y docencia.

Tras el análisis de la retroalimentación de futuros usuarios administrativos hemos detectado lo siguiente:

- Es necesario poder ver el departamento de un docente en las pantallas de administración.
- Los correos de las faltas de asistencia también se deben mandar al jefe del departamento al que pertenece el docente que ha faltado.
- A la hora de registrar firmas, la vista es confusa, sobre todo la selección de sustitutos y el uso del botón del *tick*.
- La interfaz no es lo suficientemente específica y es fácil confundir en qué página se está en el momento.
- Además de enviar los correos en el momento, hace falta mandar recordatorios cada cierto tiempo si no se ha proporcionado motivo de la falta de asistencia.

Aun así, el usuario ha conseguido completar el guión, y ha expresado tener una buena impresión de la aplicación. Consideramos que tras pulir esta parte de la aplicación e implementar los casos restantes para que la administración pueda cumplir sus objetivos, los futuros usuarios podrán hacer uso de la aplicación y así ahorrar tiempo en comparación con el proceso actual de registro de asistencias.

Por otra parte, la realización de las pruebas con futuros usuarios docentes no ha proporcionado retroalimentación significativa. La persona usuaria ha conseguido de manera rápida completar sus múltiples objetivos, lo cual indica que la aplicación cumple su funcionamiento de manera efectiva.

Hemos retocado la aplicación levemente para hacer pequeñas mejoras que solucionen algún problema recibido en la retroalimentación, como ha sido introducir *breadcrumbs* a modo de guía para que así el usuario no se confunda al navegar por la aplicación.

Capítulo 9

Conclusiones y Trabajo Futuro

9.1. Conclusiones

A modo de conclusión, a lo largo del desarrollo de este proyecto hemos integrado tecnologías de alto y bajo nivel para construir un sistema completo y funcional. Tras estudiar las alternativas que existen, las tecnologías elegidas para llevar a cabo del proyecto fueron NodeJS, ExpressJS y MySQL para la construcción del portal web, Thingsboard como hub IoT y hardware de Espressif con ESP-IDF para el despliegue de los nodos IoT.

A continuación analizamos el grado de cumplimiento de los objetivos del proyecto en el estado actual de este.

El primero de los objetivos era el desarrollo de un dispositivo hardware que permitiera varias formas de registro de la identidad del docente. Durante el proyecto este objetivo se ha concretado en la capacidad de identificarse por Bluetooth, usar el RFID de la TUI, el QR de la aplicación de TUI virtual de la UCM, el código de barras, la huella digital o el formulario a través del código QR. Se tuvieron que descartar el registro por TUI virtual y el uso del código de barras debido a las limitaciones del hardware utilizado. También, con la introducción de nueva normativa pertinente al guardado de datos biométricos, no hemos sido capaces de realizar registro por huella digital. A pesar de todo esto, hemos completado las funcionalidades de identificarse por Bluetooth, RFID y formulario web, ofreciendo una variedad de métodos por los que un docente pueda registrarse, cumpliendo así este objetivo.

En cuanto al desarrollo de un sistema capaz de detectar anomalías en la asistencia, hemos implementado una solución flexible que se adapta a las necesidades de los docentes. Somos capaces de codificar los cambios usando la entidad Excepción. El sistema integra dinámicamente las excepciones dentro de la generación del horario para ofrecer la mejor experiencia posible. Por tanto este objetivo está cumplido.

El objetivo de desarrollar una aplicación de gestión capaz de recopilar el registro

de asistencias y comprobar su corrección ha consistido en su mayoría en el desarrollo de formularios en la aplicación de gestión, y caminos capaces de identificar distintos casos dentro de los formularios en la API REST. En el estado actual del proyecto, se recogen todos los tipos de asistencias implementados, y se verifica si estos deben suceder o no durante la jornada. Consideramos así que este objetivo ha sido cumplido.

Finalmente, para desarrollar un sistema interactivo que presente los datos necesarios a los usuarios de manera comprensible y les permita tomar acción sobre ellos, se ha implementado una interfaz de usuario en la aplicación de gestión. Esta interfaz facilita la visualización de los datos en diversas páginas, adaptadas a diferentes tipos de usuarios, y en la mayoría de los casos realiza operaciones sobre la base de datos para registrar asistencias, justificar faltas o cancelar clases. Consideramos que esta interfaz de usuario, como portal de la aplicación de gestión, cumple con este objetivo de manera parcial. Aún hay funcionalidades que no se han implementado en la aplicación y que serían necesarias para alcanzar completamente este objetivo, como la reprogramación de clases.

9.2. Trabajo Futuro

En cuanto al trabajo futuro, hay varias áreas que podríamos abordar para extender las posibilidades de identificación y de otras áreas que mejorarían las funcionalidades del sistema. A continuación, se presentan algunas sugerencias a considerar:

Integración con la aplicación de TUI virtual: Es crucial investigar y explorar la posibilidad de obtener acceso a la API del servidor de autenticación para implementar el método de autenticación basado en el QR de forma segura y eficiente. Esto puede requerir un análisis más detallado de los protocolos de seguridad y la integración con el sistema existente para garantizar la confidencialidad y la integridad de los datos.

Re-evaluación de la viabilidad de la lectura de códigos de barras: En las condiciones actuales leer códigos de barras no es posible por la calidad de la cámara y el tamaño de la imagen resultante. Una posible solución a los problemas actuales sería integrar en el sistema un componente de hardware especializado en lectura de códigos de barras.

Re-evaluación de la viabilidad de la autenticación biométrica: Una de las capacidades que mejorarían mucho la usabilidad del sistema sería la identificación biométrica de los docentes. Para hacer posible esta mejora es importante reforzar la seguridad de la base de datos. Para la integración del sensor sería necesario también una mejora de la comunicación entre el microprocesador central y el auxiliar, ya que el flujo de datos incrementaría mucho en comparación al intercambio de datos actual.

Implementación de Deep Sleep para los nodos durante la noche: Los

nodos desplegados están conectados a la red eléctrica de la universidad en vez de a baterías, lo que pone en segundo plano las preocupaciones sobre eficiencia energética. Aún así, no deberíamos pasar por alto que hay largos periodos de inactividad. Aunque el consumo de un solo nodo sea muy pequeño, el consumo agregado del sistema totalmente desplegado puede ser significativo. Es por eso que en el futuro deberíamos considerar implementar periodos de Deep Sleep durante la noche, fines de semana y periodos de vacaciones para minimizar el consumo.

Separación de API REST para seguir más fielmente el patrón Backend for Frontend: Durante el desarrollo nos hemos inspirado en este patrón para delegar lógica en el Backend. Nuestra implementación sigue los principios más fundamentales pero no hemos dado la misma importancia a algunas cualidades de este patrón. Para ajustarnos totalmente al patrón habría que separar la API actual en dos, una dedicada únicamente a la aplicación de gestión, y otra dedicada únicamente a los servicios IoT.

Mejorar la forma de registro de los UIDs de los NFC: La forma actual pide directamente ese UID al usuario. Los UID son prácticamente imposibles de obtener para un usuario normal sin un lector. Idealmente los servicios informáticos de la UCM nos darían acceso a la relación de profesores y UID de sus TUI, pero entendemos que esto puede tener implicaciones en la seguridad. Como alternativa proponemos el uso del propio lector del nodo IoT. El nodo IoT empleado puede ser el de un aula, o uno específico externo preparado especialmente para este objetivo.

Integración de autenticación por logins en los puestos del docente: Se ha preparado un camino en el Backend con este fin. Este acepta un formato específico de logs y registra sus asistencias, pero no se ha podido probar con datos reales. Este tipo de registro añadiría una nueva alternativa a las ya existentes.

Implementación de un mecanismo de cambios e intercambios de clase: Creemos que sería especialmente útil que los docentes puedan comunicar situaciones excepcionales de su horario cambiando clases a un momento distinto. Un sistema de intercambio de clases podría ser interesante en algunos casos en los que a dos docentes les convenga un cambio. Implementarlo en la aplicación simplificaría el trabajo tanto de los docentes como del personal de administración.

Extensión del modelo de datos para la inclusión de departamentos: Las personas usuarias con rol de administración tienen la tarea de informar no solo a la persona que haya faltado de su falta, si no también al director o la directora del departamento al que pertenezcan. Para esto es necesario una extensión del modelo de la base de datos, concretamente la adición de una nueva entidad que represente a un departamento.

Ampliación del modelo de datos para incluir de varios responsables: Existe la posibilidad de que varios docentes sean responsables de una misma asignatura, pudiendo estos alternar sus asistencias, y no considerándose falta con tal de que un docente asista. Este caso no se ha tenido en cuenta en la solución inicial, pero bastaría con cambiar la cardinalidad de una relación y adaptar el código para

completar este objetivo.

Implementación de un mecanismo de recordatorios para las faltas de asistencia: La administración de la facultad envía recordatorios cada cierto tiempo de las faltas de asistencia a las cuales no se ha proporcionado motivo. En el futuro esta tarea también podría ser automatizada.

Integración de un sistema de aviso de sustituto: Un sistema que permita avisar con antelación de un sustituto beneficiaría mucho la gestión de faltas. Actualmente se reportaría una falta del docente titular y una sustitución aparte. Este cambio permitiría justificar la falta antes de que ocurriera y evitaría el trabajo de gestión.

Unificación de las funcionalidades de registro de firmas y generación de avisos: Durante el desarrollo de las historias de usuario pertenecientes a la administración de personal hemos identificado similitudes entre ellas. Creemos que permitir generar avisos directamente mientras se registran las firmas ayudaría a la labor desempeñada por los usuarios a cargo de la administración de personal.

Creación de informes de asistencias en archivos PDF descargables: Las asistencias se registran en la base de datos y todos los intentos de registro se guardan en un *log*. Con el fin de poder mantener almacenada esta información fuera del servidor y de poder acceder a ella sin utilizar la aplicación, creemos que dar la posibilidad de descargar un PDF con las asistencias ayudaría mucho a su mantenimiento y rápida revisión.

Por último debemos anotar que Thingsboard es capaz de gestionar muchos rebaños con diferentes funciones. Para futuros despliegues se puede usar la misma plataforma que hemos inaugurado y agrupar todo lo relacionado con el IoT de la Facultad de Informática en un mismo portal.

9.3. Conclusiones personales

Este proyecto ha sido para todos nosotros el más grande en el que hemos participado. No solo ha sido una tarea demandante por la cantidad de tiempo dedicada, sino también por todas las tecnologías y conocimientos distintos que se han entretejido para cumplir los objetivos propuestos. Entrando en el proyecto ninguno de nosotros tenía experiencia con las tecnologías que ha aplicado. Entre todos hemos conseguido aplicar nuestros aprendizajes para construir un sistema funcional del que estamos muy orgullosos. Trabajar en este proyecto ha supuesto una evolución personal para todos. Lo aprendido en este trabajo definirá los profesionales que seremos en el futuro.

Chapter 10

Introduction

10.1. Motivation

The current teaching attendance tracking system at the Faculty of Computer Science of the Complutense University of Madrid collects attendance information through sign-in sheets. This requires daily distribution and collection of these sheets from all the classrooms in the faculty. This system has proven to be error-prone and susceptible to oversight.

With this Undergraduate Thesis Project, we aim to offer a solution that helps systematize this management process, facilitating the work, speeding up data collection, reducing paper waste, and increasing reliability.

10.2. Objectives

To address this problem, we propose the following objectives:

- Development of a hardware device that allows multiple forms of identity registration for the professsor.
- Development of a system capable of detecting anomalies in attendance.
- Development of a management application capable of collecting attendance records and verifying their accuracy.
- Development of an interactive system that provides necessary data to users in an understandable way and allows them to take action based on this data.

10.3. Work Plan

The work has been carried out in two parallel development lines. On one hand, the management web application aspect to provide access from anywhere to the users, and on the other hand, the firmware of the IoT nodes to be deployed in each education-ready space (e.g. classrooms, laboratories, etc.). The table 10.1 reflects the work realized, divided in sprints for each development line.

Sprint	Web Application	Firmware
Oct 2 - Oct 9	Meeting Planning	
Oct 9 - Oct 23	Research on HTML, Bootstrap, and JavaScript. Formulation the data model.	Contact with the Espressif framework. OTA system adaptation. HTTPS communication prototype. Camera and screen usage prototype. Creation of the Thingsboard docker. Development network assembly.
Oct 23 - Nov 6	User stories creation. Paper designs of different application screens. Web application skeleton. HTML prototypes.	Software architecture definition and responsibilities distribution among components. Text rendering prototype. MQTT module beginnings.
Nov 6 - Nov 20	Use templates for HTML pages. Paper designs of different application screens and HTML prototypes. Research on QRs.	Integration of buttons into the interface. Boot module prototype. NVS integration. Web app creation to provision terminals.
Nov 20 - Dec 4	Set up of the database and its interactions with the web. Paper designs of different application screens and HTML prototypes. QR creation for requested space.	Screen icons loading. First communication to the server. TOTP code generation.
Dec 4 - Dec 18	Advanced form creation. API creation for firmware communication. API documentation in YAML. TOTP research.	Refactoring of repeated code with compiler macros. MQTT module refactoring.
Dec 18 - Feb 1	Exams	

	Project rearrangement using Boilerplates. Move communication between web application and database exclusively to the API. Establish communication between Application and API. Implement TOTP. Research on sessions and cookies. Move QR generation to the client to generate it dynamically.	
Feb 1 - Feb 15	Move communication between web application and database exclusively to the API. Create migrations and seeders for the database. Use select2 in HTML. Encode user passwords. Add dotenv to declare environment variables.	First integration of Bluetooth engine with NimBLE.
Feb 15 - Feb 29	Implementation of everything necessary to perform the form by BLE. Move functions from different routes of the Application and API to controllers. Implementation of sessions. New pages creation in EJS. Create a CSV to database activities converter.	Change from BLE engine to Bluedroid and programming of the RFID external module. First communication to the tracking endpoint.
Feb 29 - Mar 21	New pages creation in EJS. Minimize files sent to the client. Create configurable files for the project. Create middleware to determine recurrences, handle cookies, and escape requests. Create code to determine unattended classes. Implement all things necessary to perform the form with NFCs.	Global refactor and initial cross-development tests.
Mar 21 - Apr 4	Attendance reports functionality, 'sign'. Adapt escape middleware to accept Spanish characters. Exception model change.	Adaptation for production environment and initial tests in it.
Apr 4 - Apr 18	Adapt database to UTC. Add pino for logging.	Icon change and interface redesign for user-facing aspect.

18 Apr - 2 May	Webpage improvement. Creation of the registration forms for MACs and NFCs. Morgan log addition. Middleware call format change.	
2 May - 16 May	Reports of the attendance recorded during the day. QR image creation in API. Administration and deanship feature implementation. Trustproxy addition.	

Table 10.1: Work Timeline Diagram

Chapter 11

Conclusions and Future Work

11.1. Conclusions

In conclusion, throughout the development of this project, we have integrated high and low-level technologies to build a complete and functional system. After studying the existing alternatives, the chosen technologies for carrying out the project were NodeJS, ExpressJS, and MySQL for the construction of the web portal, Thingsboard as an IoT hub, and Espressif hardware with ESP-IDF for the deployment of IoT nodes.

Below, we analyze the degree of fulfillment of the project objectives in its current state.

The first objective was the development of a hardware device that allowed multiple forms of professor identity registration. During the project, this objective was materialized in the ability to identify professors using Bluetooth, the RFID of the UCM virtual TUI, the QR code of the UCM virtual TUI application, barcodes, fingerprints, or the attendance form through the QR code. Registration via virtual TUI and the use of barcodes had to be discarded due to the limitations of the hardware used. Also, with the introduction of new relevant regulations on biometric data storage, we have not been able to carry out fingerprint registration. Despite all this, we have completed the functionalities of identification via Bluetooth, RFID, and web form, offering a variety of methods for a professor to register their attendance, thus fulfilling this objective.

Regarding the development of a system capable of detecting attendance anomalies, we have implemented a flexible solution that adapts to the needs of professors. We can encode changes using the Exception entity. The system dynamically integrates exceptions within the schedule generation to offer the best possible experience. Therefore, this objective is fulfilled.

The objective of developing a management application capable of collecting at-

tendance registries and verifying their correction has mostly consisted of developing forms in the management application and paths capable of identifying different cases within the forms in the REST API. In the current state of the project, all implemented attendance types are collected, and it is verified whether these should occur or not during a workday. Thus, we consider this objective fulfilled.

Lastly, to develop an interactive system that presents the necessary data to users in a comprehensible manner and allows them to take action on said data, a user interface has been implemented in the management application. This interface facilitates data visualization on various pages, adapted to different user types, and in most cases performs operations on the database to register attendance, justify absences, or cancel classes. We consider that this user interface, as the management application portal, partially fulfills this objective. There are still functionalities that have not been implemented in the application and would be necessary to fully achieve this objective, such as class rescheduling.

11.2. Future Work

Regarding future work, there are several areas we could address to extend identification possibilities and other areas that would improve system functionalities. Below are some suggestions to consider:

Integration with the virtual TUI app: It is crucial to research and explore the possibility of obtaining access to the API of the authentication server to implement the QR-based authentication method securely and efficiently. This may require a more detailed analysis of security protocols and integration with the existing system to ensure data confidentiality and integrity.

Re-evaluation of barcode reading feasibility: In the current conditions, reading barcodes is not possible due to the camera quality and the resulting image size. One possible solution to the current issues would be to integrate a hardware component specialized in barcode reading into the system.

Re-evaluation of biometric authentication feasibility: One capability that would greatly enhance the usability of the system would be biometric identification of professors. To make this enhancement possible, it is important to strengthen the database security. Integrating the sensor would also require improved communication between the central microprocessor and the auxiliary, as the data flow would increase significantly compared to the current data exchange.

Implementation of Deep Sleep for nodes during the night: The deployed nodes are connected to the electrical network instead of batteries, which puts energy efficiency concerns in the background. However, we should not overlook the long periods of inactivity. Although the consumption of a single node is very small, the aggregated consumption of the fully deployed system can be significant. Therefore, in the future, we should consider implementing Deep Sleep periods during nighttime,

weekends, and vacation periods to minimize consumption.

Separation of REST API to more closely follow the Backend for Frontend pattern: During development, we were inspired by this pattern to delegate logic to the Backend. Our implementation follows the most fundamental principles, but we did not give the same importance to some qualities of this pattern. To fully conform to the pattern, we would need to separate the current API into two, one dedicated solely to the management application and another dedicated solely to IoT services.

Improvement in NFC UID registration method: The current method directly requests this UID from the user. UIDs are practically impossible to obtain for a normal user without a reader. Ideally, the IT services of the UCM would give us access to the relationship between professors and their TUI UIDs, but we understand this could have security implications. As an alternative, we propose using the reader of IoT nodes. The IoT node tasked with this could be any already available one or an external one specially prepared for this purpose.

Integration of login-based authentication at professor stand: A path in the Backend has already been prepared for this. This path accepts a specific log format and registers their attendances, but it has not been tested with real data. This type of registration would add a new alternative to the existing ones.

Implementation of class change and exchange mechanism: We believe it would be especially useful for professors to communicate exceptional schedule situations by changing class to a different time. A class exchange system could be interesting in cases where two professors benefit from a change. Implementing this mechanism in the application would alleviate the work amount given to both professors and administration personnel.

Extension of data model for department inclusion: Users with administrative roles have the task of informing not only the person who missed their class but also the head of the department to which they belong. This requires an extension of the database model, specifically the addition of a new entity representing a department.

Expansion of data model to include multiple responsible professors: There is a possibility that several professors are responsible for the same subject, being able to alternate their attendances, and not considering it an absence as long as one professor attends. This possibility was not considered during the creation of the data model, but it could be easily implemented by changing the cardinality of a relationship and adapting the related code to this change.

Implementation of absence reminder mechanism: The faculty administration sends reminders every certain amount of time for absences without registered motive. In the future, this task could also be automated.

Integration of substitute notification system: A system that allows notification of a substitute beforehand would greatly benefit the management of absences.

Currently, the absence and replacement of a substitute professor are reported separately. This change would allow justifying the absence before it occurs and would avoid management workload.

Unification of signature registration and notification generation functionalities: During the development of the user stories belonging to personnel administration, we identified similarities between them. We believe that allowing notifications to be generated directly while registering signatures would help the work performed by users in charge of personnel administration.

Creation of attendance reports in downloadable PDF files: Attendances are recorded in the database, and all registration attempts are saved in a log. To keep this information stored outside the server and access it without using the application, we believe that providing the option to download a PDF with the attendance records would greatly help its maintenance and quick review.

Lastly, we should note that Thingsboard is capable of managing many herds with different functions. For future deployments, the same platform we inaugurated can be used to group everything related to IoT of the Faculty of Computer Science in a single portal.

11.3. Personal Conclusions

This project has been the largest any of us has participated in. Not only has it been a demanding task due to the amount of time dedicated, but also because of all the technologies and different knowledge interwoven to achieve the proposed objectives. Entering the project, none of us had experience with the technologies we applied. Together, we managed to apply our newly gained knowledge to build a functional system that we are very proud of. Working on this project has been a personal evolution for all of us. What we learned in this work will define the professionals we will be in the future.

Capítulo 12

Contribuciones Personales

A continuación listamos, para cada uno de nosotros, nuestras contribuciones al proyecto.

Helena Antón Navarro

Mi aportación a este trabajo ha consistido en lo tratado en el capitulo 4, junto con mi compañero Javier, la aplicación web. La aplicación web al estar formada del Frontend y el Backend tienen muchas funcionalidades y tareas distintas en las que he trabajado.

Historias de Usuario

Participé en la creación de historias de usuario junto con Javier. Partimos de unas personas y algunas historias de usuario que nos proporcionó Iván, uno de los directores de este TFG. A esto le añadimos alguna persona más y más historias de usuario, con el objetivo de cubrir las distintas tareas que se podrían llegar a realizar con la aplicación.

Bocetos de Páginas Web

Los bocetos de las pantallas fueron hechos en conjunto con Javier. Cada boceto que se hizo se basaba en una historia de usuario y mostraba el flujo completo para dicha historia de usuario.

Frontend

Esta ha sido la parte en la que más he trabajado del proyecto, desde la creación de los bocetos como el paso a páginas en HTML y EJS, a la vez que haciendo el funcionamiento en sí de cada historia de usuario que tiene una interacción con la web, y de pantallas adicionales de información o inicio de sesión.

HTML

Por una parte, esto ha implicado la realización de HTML, cambios en este, su pasada a plantilla EJS, y la introducción de parámetros que recibe la plantilla. Por otra parte, también ha incluido investigar sobre lo que ofrece HTML, así como su funcionamiento. Esto ha incluido cosas como el uso de distintos tipos de input en los formularios, como puede ser los campos tipo `datetime`, y en especial el uso de *select2* y *datatables*. Sin embargo, el código que se usa en las páginas para manejar el cambio de datos o algunos envíos se ha realizado en conjunto a Javier.

CSS

Trabajé con CSS modificando el que venía dado por Bootstrap y añadiendo nuevas reglas para modificar tanto los colores como la tipografía a la usada por la UCM en sus páginas web. También modifiqué como se ve la pantalla principal de la aplicación dependiendo del tamaño de la pantalla.

NodeJS/Javascript

La mayoría de trabajo que he realizado del Frontend se encuentra en el código, como la lógica encargada de realizar cada función para los distintos métodos de cada camino, o la encargada de interaccionar con la API para conseguir los datos necesarios en cada caso y prepararlos para mostrarlos al cliente, así como la recepción de los datos del cliente para enviarlos a la API. También incluye la revisión de dicha lógica, especialmente tras haber pasado de los casos sencillos de solo considerar una actividad, a tener que considerar tanto actividades con recurrencias como excepciones.

Con la ayuda del *middleware* proporcionado por Javier, la tarea de realizar escapado de las peticiones, el control de las sesiones y sus atributos, o las comprobaciones de las recurrencias de las distintas actividades ha sido realizable.

Backend

En esta parte del proyecto he trabajado en el momento de la división de procesos, para ayudar a cambiar el código que accedía a la base de datos en la aplicación

al nuevo proceso, en el planteamiento de la API REST y ocasionalmente en la documentación de esta. También me he encargado en esta parte de realizar las migraciones y los seeders de la base de datos.

Base de Datos

En esta parte he realizado el modelado de los datos con Javier, que consistía en el modelo entidad-relación mostrado en la sección 4.1.

Al plantear el modelo, se han tenido que tener en cuenta casos especiales de clases en las que se juntan dos grupos o titulaciones distintas, como pueden ser las asignaturas de primer curso o Aplicaciones Web con Sistemas Web. Ya que afectan a la representación de la asistencia, cuya simplicidad nos penaliza cuando queremos acceder a sus relaciones o su actividad. Y estos casos especiales se deben tratar con más cuidado. Como beneficio de la forma de representación está la facilidad de escritura de estas, que es la operación que más se va a realizar en esta tabla.

Configuración de la Aplicación

Junto con Javier, he conseguido dejar la aplicación web como algo fácilmente configurable desde un archivo, permitiendo simplicidad a la hora de tener que cambiar la configuración. También tiene parámetros con valor por defecto, para ayudar a probar la aplicación.

Estoy muy orgullosa del trabajo que he hecho. He aprendido sobre todo un mundo del cual no sabía casi nada antes, y he conseguido aplicar todo este conocimiento en un proyecto real. Haber superado este reto es algo que me servirá como profesional y no dudo que me ayudará en experiencias futuras.

Javier Galdo Martínez

Durante mi trabajo, me he hecho cargo del desarrollo de la aplicación web (capitulo 4) junto con mi compañera, Helena. En este, se ha implementado de manera completa un portal web que permite el registro y revisión de asistencias, la cancelación de clases, y el registro de MACs y UIDs de NFC a un usuario.

A su vez, también he realizado gran parte del desarrollo del Backend y su interacción con la base de datos. Esto abarca el ámbito del estructurado de la API, las consultas realizadas a la base de datos, el estado de la base de datos, los retornos de los distintos caminos y demás.

A lo largo del proyecto también he ido haciendo *middleware* utilizado en distintas partes de la aplicación, lo que nos ha servido mucho para aumentar la legibilidad del código.

A continuación listo en detalle las distintas tareas que he realizado.

Historias de Usuario

A la hora de hacer historias de usuario, nuestro director, Iván, nos proporcionó una base de casos que él consideraba necesarios, de la que Helena y yo continuamos proponiendo distintos casos en los que se utilizaría la aplicación. Creemos haber conseguido abarcar la gran mayoría de posibilidades que pueden llegar a utilizarse por distintos docentes, haciendo así de nuestra aplicación algo que intenta contentar a todos los docentes con las opciones, pues la idea es que les ayude.

Bocetos de Páginas Web

Esto se ha realizado en conjunto con Helena. Ha tratado de plantear distintos flujos que cumpliesen lo que pedían las distintas historias de usuario, y reflejarlos como una serie de pantallas a seguir para cumplir cada historia. También, tras la realización de estos bocetos y decidir utilizar la plantilla de SB admin, ha tratado de pasar los bocetos a papel a código HTML, pasando de cada boceto a un esqueleto de código de lo que posteriormente sería cada página.

Frontend

En la parte del Frontend me he encargado más de código en Javascript que de la parte gráfica. En su mayoría, he ayudado a Helena a que la interacción de una página web tenga el comportamiento deseado. Esto incluye parte del código Javascript enviado al cliente, parte del código del servidor, y el manejo de sesiones, de sus atributos, y escapado de mensajes del cliente.

Backend

He estado especialmente a cargo de esta parte. En concreto, a partir de una base mínima de documentación asentada por nuestro director de proyecto, Iván, y tras hablar con Helena sobre la división de código para así hacer un Backend separado del Frontend, he desarrollado una API REST, completamente documentada, que permite una extracción de datos de la base de datos para aquellos clientes que se identifiquen mediante el uso de una cabecera X-Token. Los datos abarcan desde simples elementos de tablas hasta elementos concretos que estén en relación con otros de distintas tablas de la base de datos. Algunos de los caminos también incluyen filtros por fecha, pues era lo que requería el Frontend para hacer uso de estos, y otros se encargan de la creación de nuevos datos necesarios, como la base de este proyecto, las asistencias. Helena me ayudo en esta tarea, especialmente a la hora de hacer la versión básica de la API, y posteriormente seguí yo con su desarrollo. También cabe mencionar que se ha hecho uso de *bcrypt* para codificar las contraseñas, con un sistema de sal y pimienta.

Base de Datos

Junto con mi compañera Helena, hemos realizado un modelo entidad relación de la totalidad de la base de datos. Este ha sido la base de la posterior implementación de la Base de Datos con el uso de Sequelize, permitiendo que el modelado de ORM nos librase de tener que hacer mayores cambios al cambiar de SQLite a MySQL.

Middleware

Durante el desarrollo del proyecto, el código se ha ido haciendo cada vez más y más difícil de mantener. Para mejorar el mantenimiento del código he ido dividiendo distintas tareas en *middleware*, como por ejemplo, un *middleware* que permite purgar los atributos de sesión que no se necesiten en distintos caminos, o un *middleware* que escapa el contenido de una petición por parte de un cliente, para así librarnos de inyección SQL o ataques por XSS, sea persistente o no.

Como añadido a este apartado, el *middleware* que más hemos utilizado ha sido el ideado para las recurrencias de la base de datos. El sistema de recurrencias en la base de datos permite una gran variedad de posibilidades, y es de vital importancia que estas se calculen bien. Para ello, con el uso de moment, he desarrollado una serie de funciones que son capaces de dada una recurrencia con un periodo y las horas en las que sucede, determinar la primera actividad posible, y a partir de ahí, el resto de actividades.

Configuración de la Aplicación

Una parte importante de la aplicación depende de una serie de simples parámetros, y un cambio en estos provocaría un cambio en parte del funcionamiento de la aplicación. Para permitir un fácil cambio de estos parámetros, Helena y yo hemos ido recolectando las partes importantes de la configuración, las hemos abstraído a ficheros con terminación identificativa, para así reconocerlos de manera más sencilla, y posteriormente utilizado dotenv para poner todos estos parámetros en un mismo lugar. Esto hace que, para cambiar el comportamiento de la aplicación, solo sea necesario cambiar unas pocas líneas en un mismo archivo, y ejecutar la aplicación. A su vez, para considerar el caso de una prueba simple de la aplicación, hemos establecido valores por defecto a lo largo de la asignación de estos parámetros, lo cual permite que los parámetros de configuración sean omitibles en muchos casos.

Todo esto ha supuesto un desafío para mí, tanto por mi completo desconocimiento de ciertas áreas de desarrollo, como por el tiempo que ha requerido el proyecto. Estoy orgulloso de tanto el aprendizaje como el trabajo que he hecho en estos últimos meses, y sé que utilizaré la experiencia ganada en este para perseverar en el futuro.

Jaime González Fábregas

Yo he estado encargado del desarrollo de la parte hardware. Mi trabajo ha consistido en la implementación de las diferentes funcionalidades mencionadas anteriormente con relación a los nodos IoT. También he sido el encargado de la configuración del hub IoT para adaptarlo a nuestro caso de uso.

Hub IoT

Para preparar a Thingsboard para el paso de mensajes he configurado diferentes *ChainRules*. Estas reglas han sido cruciales para gestionar las comunicaciones entre los nodos IoT y el Backend, permitiendo delegar la mayoría de la gestión individualizada a Thingsboard para que el Backend tenga una implementación de más alto nivel. En la parte de Thingsboard también ha sido importante la configuración de los perfiles de dispositivo y el manejo de las herramientas necesarias para permitir actualizaciones OTA de los dispositivos.

Nodo IoT

Soy el autor de toda la parte de este TFG en relación con el nodo IoT. He estado encargado del desarrollo y las pruebas de los dispositivos físicos que componen la red de terminales de cara al conjunto de personal docente en las aulas.

Métodos de identificación

Implementé la funcionalidad TOTP para la autenticación de docentes, integrando la implementación del estándar RFC 6238 (M'Raihi, et al (2011)), incluyendo la generación de códigos QR y la correspondiente comunicación con el Backend necesaria para hacer esto posible. Trabajé en la integración del chip RFID para la identificación de docentes, incluyendo la lectura del número de serie, la comunicación con el microcontrolador auxiliar y el envío de datos al Backend. Además desarrollé el sistema de escaneo de paquetes Bluetooth para encontrar dispositivos cercanos y identificar de esa forma los docentes que se encuentren en el aula. También invertí parte de mi tiempo en la investigación de factibilidad para más métodos alternativos, como la lectura del código de barras de la TUI, lectura de QR para la configuración y autenticación biométrica por huella dactilar.

Otras funcionalidades

Un componente central de la funcionalidad del nodo es su capacidad de recibir actualizaciones OTA, incluyendo la descarga de la imagen del *firmware*, la repro-

gramación del dispositivo y el proceso de reinicio. También diseñé y implementé el proceso de aprovisionamiento, incluyendo una herramienta externa para generar los códigos QR, la lectura de los parámetros desde la cámara del Esp32-s3-eye, la autenticación de los diferentes servicios.

Arquitectura de la implementación

Como he detallado anteriormente ha sido necesario organizar la funcionalidad en módulos para compartimentalizar las áreas de responsabilidad. Diseñé e implementé una arquitectura de software basada en hilos de FreeRTOS y colas de mensajes para poder distribuir responsabilidades entre módulos desacoplados y mejorar la modularidad y la escalabilidad del sistema. Esta arquitectura está cimentada sobre el paso de mensajes entre procesos y en *multithreading* cooperativo.

Sistema de encendido y recuperación de errores

También soy el autor del sistema de encendido. Este módulo se encarga orquestar el encendido de los diferentes sistemas para que las dependencias entre sistemas no causen un error durante el encendido.

Durante el encendido del sistema los diferentes pasos que se toman siguen el orden marcado por una maquina de estados de mi invención. Esto se asegura de que todas las condiciones para comenzar el siguiente proceso se estén cumpliendo antes de intentar proceder con el siguiente paso. Mi objetivo era crear un protocolo de encendido fácilmente configurable que permitiera expandir las tareas según creciera el proyecto.

Implementé la recuperación de errores dentro del sistema de encendido expandiendo la máquina de estados para permitir transiciones hacia estados anteriores en caso de fallo. De esta forma permito que el proceso de conexión se reinicie tras un fallo de red, mejorando la disponibilidad y la robustez del sistema.

Implementación del proceso de aprovisionamiento

Diseñé e implementé un proceso de aprovisionamiento para que Thingsboard y el Backend puedan identificar a cada dispositivo individualmente. Este proceso se ejecuta en paralelo con el encendido del dispositivo, optimizando el tiempo de arranque.

Me siento orgulloso de haber contribuido al desarrollo del nodo IoT. He adquirido valiosos conocimientos y experiencia en el desarrollo de sistemas embebidos, comunicación inalámbrica y seguridad informática. Durante el proyecto he tenido que llevar a cabo una labor de investigación y experimentación que ha moldeado mi manera de enfrentarme a nuevos desafíos.

Considero que mi trabajo en este proyecto ha sido una experiencia de aprendizaje valiosa que me ha permitido desarrollar mis habilidades técnicas y de resolución de problemas. Estoy seguro de que estas habilidades me serán útiles en futuros proyectos.

Bibliografía

Abba, I. V. What is an orm – the meaning of object relational mapping database tools. Disponible en https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools.

Alsaffar, M., Aljaloud, S., Mohammed, B. A., Al-Mekhlafi, Z. G., Almurayziq, T. S., Alshammari, G. y Alshammari, A. Detection of web cross-site scripting (xss) attacks. *Electronics*, vol. 11(14), 2022.

Casas, S., Cruz, D., Vidal, G. y Constanzo, M. Uses and applications of the openapi/swagger specification: a systematic mapping of the literature. En *2021 40th International Conference of the Chilean Computer Science Society (SCCC)*, páginas 1–8. 2021.

Cooper, A. *The Inmates Are Running the Asylum*. Sams - Pearson Education, 2004.

Espressif Systems (Shanghai) Co., Ltd. Esp-idf programming guide. Disponible en https://docs.espressif.com/projects/esp-idf/en/latest/esp32/index.html.

Ionel Jacob, M. P. Sql injection attacks and vulnerabilities. 2020.

Kher, S. Again and again! managing recurring events in a data model. Disponible en https://vertabelo.com/blog/again-and-again-managing-recurring-events-in-a-data-model/.

Maria Salagean, D. Z. *IoT Applications based on MQTT Protocol*. 2020.

M'Raihi, et al. Totp: Time-based one-time password algorithm. Disponible en https://www.rfc-editor.org/rfc/rfc6238.

Onur C¸ iftci, Z. E. B., Elif Nurdan Pektas¸ y Serif, T. Location proving with a single witness: A hybrid prover-witness and totp-based solution. 2022.

The Thingsboard Authors. Thingsboard community edition documentation. Disponible en https://thingsboard.io/docs/.

Apéndice A

Personas

Con la intención de abordar el diseño de la solución, hemos creado las personalidades de las personas hipotéticas que usaremos para crear las historias de usuario.

- **Marta**: Marta es una persona de 48 años que disfruta de viajar y leer. Para leer y comparar ofertas de viajes hace uso de su portátil, aunque suele tener que pedirle ayuda a sus hijos para utilizarlo. También hace uso de este para trabajar. Justo como parte de este trabajo, cada día Marta tiene que controlar la asistencia del profesorado de la Facultad de Informática, tanto en aulas como en laboratorios, y luego generar un informe diario de asistencias. Si no encuentra a un docente en la hoja de firmas o en el registro del docente no aparece una asignatura que se debería haber dado, contacta con este para determinar el motivo de la falta y cuando se recuperará la clase perdida.
- **Alejandro**: Alejandro, con 40 años, es una persona con interés en el kendo y la cultura japonesa, y mucha pasión por la informática. Habitualmente organiza su agenda combinando múltiples herramientas y automatizando las tareas cotidianas con ellas. Como parte de su trabajo, Alejandro tiene que supervisar las clases impartidas en la Facultad de Informática y realizar seguimiento de cualquier problema que surja, para identificar a los docentes que se salten su labor.
- **Marcelo**: Marcelo es una persona de 27 años. Posee una mente analítica, y disfruta mucho con puzzles y problemas matemáticos. Vive pegado a su teléfono móvil, en el cual participa en múltiples partidas de ajedrez concurrentemente. Quiere maximizar y usar de manera efectiva el tiempo de clase, minimizando las distracciones. Tras dejar sus pertenencias en la mesa del profesor, utiliza tiza y pizarra para dar la clase, con la intención de que sus alumnos sigan las explicaciones que da, y hagan apuntes de estas. Para minimizar las distracciones, le gustaría poder registrar su asistencia desde el móvil, y así no tener que llevar un bolígrafo, o pedirlo si se le olvida.
- **Ignacio**: Ignacio tiene 46 años. Además de tocar el saxofón en la banda de

la UCM, es un apasionado del cine francés de los años 70. Es una persona extrovertida, y un tanto despistada. Suele llegar a clase justo a tiempo, por lo que comienza la misma, y suele terminar olvidándose de firmar o hacer login en el puesto de laboratorio antes de irse. Habitualmente suele darse cuenta de este despiste poco después de salir de clase, por lo que tiene que bajar a recursos humanos para notificar dicho despiste. Suele acceder al puesto de laboratorio para obtener su material docente, por lo que le gustaría que acceder al puesto de laboratorio registrase su asistencia. También le gustaría poder justificar sus faltas desde el móvil, para ahorrarse el viaje a recursos humanos.

- **Rosa**: Rosa es una persona de 43 años. Es muy meticulosa y precisa, en especial, en la preparación de sus clases y la elaboración de retroalimentación de actividades de sus alumnos. Debido a una condición médica, tiene que tomar una medicación cada 6h, y hace uso de su smartwatch, el cual siempre lleva, para poner alarmas en esos tiempos. Esta condición médica también la obliga a realizar revisiones médicas periódicamente, teniendo que cambiar alguna clase de vez en cuando. Dedica toda su atención a los estudiantes, así que siempre está alternando entre el puesto del profesor, dando explicaciones, y las mesas o puestos de los estudiantes, resolviendo dudas. Le gustaría no tener que perder el tiempo en temas burocráticos, y que su smartwatch sirviese para validar su asistencia. También le gustaría poder cambiar o cancelar sus clases de antemano, para evitar problemas con temas burocráticos.
- **Pedro**: Pedro, con 53 años, es un amante del fútbol, y disfruta de pasar tiempo con sus hijos. Compagina su trabajo como técnico en los servicios informáticos de la UCM, e impartiendo clases en la Facultad de Informática por diversión, y para captar talento que se quede en la UCM. Está acostumbrado a fichar con la TUI de la UCM para fichar o al cambiar de edificio. Le gustaría poder utilizar este mismo sistema para registrar su asistencia a clase.
- **María Ángeles**: María Ángeles, o Ángeles para los amigos, tiene 61 años. Disfruta de largos paseos por el campo, y le encantan los viajes al extranjero. Aprovecha su trabajo en la UCM para ir de congresos a distintas partes del mundo, además de impartir clase a la vieja escuela, con tiza y pizarra. No suele utilizar mucho dispositivos electrónicos, y muchas veces se deja el móvil en el despacho. Le gustaría poder utilizar la hoja de firmas para registrar su asistencia, y no tener que llevar un dispositivo electrónico encima.

Tras esto, podemos distinguir los roles que efectúan las distintas personas. Por un lado, Marcelo, Ignacio, Rosa, Pedro y María Ángeles tienen en cuenta distintos objetivos, pero cumplen la misma tarea, dar clase. Hemos considerado este rol como el rol docente. Por otro lado, Marta y Alejandro no tienen la misma tarea como objetivo, y hay que distinguirlos de los docentes. Marta se hace cargo de administrar todo el mecanismo de la asistencia, es decir, que los docentes hayan asistido a las clases que se debían impartir, lo que consideramos como el rol de administración de personal. Alejandro, aunque no se encargue exactamente de administración, también tiene necesidades distintas a los docentes, especialmente su supervisión. Llamamos a este rol decanato.