



## CARRERA DE ESPECIALIZACIÓN EN INTERNET DE LAS COSAS

MEMORIA DEL TRABAJO FINAL

### Sistema de seguimiento de procesos en rectificadoras de motopartes

**Autor:**  
**Lic. Pablo Arancibia**

Director:  
Esp. Ing. Diego Fernandez (FIUBA)

Codirector:  
Esp. Ing. Miguel del Valle Camino (FIUBA)

Jurados:  
Mg. Ing. Gustavo Zocco (FIUBA)  
Esp. Ing. Pedro Rosito (FIUBA)  
Esp. Ing. Lionel Gutiérrez (FIUBA)

*Este trabajo fue realizado en la ciudad de Resistencia,  
entre enero de 2022 y agosto de 2022.*



## CARRERA DE ESPECIALIZACIÓN EN INTERNET DE LAS COSAS

MEMORIA DEL TRABAJO FINAL

### Sistema de seguimiento de procesos en rectificadoras de motopartes

**Autor:**  
**Lic. Pablo Arancibia**

Director:  
Esp. Ing. Diego Fernandez (FIUBA)

Codirector:  
Esp. Ing. Miguel del Valle Camino (FIUBA)

Jurados:  
Mg. Ing. Gustavo Zocco (FIUBA)  
Esp. Ing. Pedro Rosito (FIUBA)  
Esp. Ing. Lionel Gutiérrez (FIUBA)

*Este trabajo fue realizado en la ciudad de Resistencia,  
entre enero de 2022 y junio de 2023.*

III

### *Agradecimientos*

III

### *Agradecimientos*

A todo el personal docente y administrativo de la Facultad de Ingeniería de la UBA. A mi director, codirector y a los compañeros de la Especialización en Internet de las Cosas que me acompañaron en esta formación profesional.

# Índice general

<b>Resumen</b>	1
<b>1. Introducción general</b>	1
1.1. Descripción del sistema . . . . .	1
1.2. Motivación . . . . .	2
1.3. Estado del arte . . . . .	2
1.4. Objetivos y alcance . . . . .	3
<b>2. Introducción específica</b>	5
2.1. Internet de las cosas . . . . .	5
2.2. Protocolos de comunicación . . . . .	5
2.2.1. Protocolo HTTP . . . . .	5
2.2.2. Protocolo MQTT . . . . .	6
2.2.3. Broker Eclipse Mosquitto . . . . .	7
2.3. Bases de datos . . . . .	7
2.3.1. Bases de datos relacionales . . . . .	7
2.3.2. Sistema de gestión de base de datos MySQL . . . . .	7
2.4. Tecnologías backend . . . . .	8
2.4.1. API RESTful . . . . .	8
2.4.2. Node.js . . . . .	9
2.4.3. Express . . . . .	9
2.4.4. Sequelize . . . . .	9
2.5. Tecnologías frontend . . . . .	10
2.5.1. Ionic framework . . . . .	10
2.6. Tecnologías de servidor . . . . .	11
2.6.1. Docker y Docker Compose . . . . .	11
2.6.2. Servidor web Nginx . . . . .	11
2.7. Hardware utilizado . . . . .	11
2.7.1. Raspberry Pi . . . . .	11
2.7.2. NodeMCU Esp32 . . . . .	12
2.7.3. Identificación por radiofrecuencia . . . . .	13
2.7.4. Módulo RC522 . . . . .	13
2.7.5. Buzzer sonoro . . . . .	13
<b>3. Diseño e implementación</b>	15
3.1. Arquitectura general del sistema . . . . .	15
<b>4. Ensayos y resultados</b>	17
4.1. Pruebas funcionales del hardware . . . . .	17
<b>5. Conclusiones</b>	19
5.1. Conclusiones generales . . . . .	19
5.2. Próximos pasos . . . . .	19

# Índice general

<b>Resumen</b>	1
<b>1. Introducción general</b>	1
1.1. Descripción del sistema . . . . .	1
1.2. Motivación . . . . .	2
1.3. Estado del arte . . . . .	2
1.4. Objetivos y alcance . . . . .	3
<b>2. Introducción específica</b>	5
2.1. Internet de las cosas . . . . .	5
2.2. Protocolos de comunicación . . . . .	5
2.2.1. Protocolo HTTP . . . . .	5
2.2.2. Protocolo MQTT . . . . .	6
2.2.3. Broker Eclipse Mosquitto . . . . .	7
2.3. Bases de datos . . . . .	7
2.3.1. Bases de datos relacionales . . . . .	7
2.3.2. Sistema de gestión de base de datos MySQL . . . . .	7
2.4. Tecnologías backend . . . . .	8
2.4.1. API RESTful . . . . .	8
2.4.2. Node.js . . . . .	9
2.4.3. Express . . . . .	9
2.4.4. Sequelize . . . . .	9
2.4.5. API messenger . . . . .	10
2.4.6. Portainer . . . . .	11
2.5. Tecnologías frontend . . . . .	12
2.5.1. Ionic framework . . . . .	12
2.6. Tecnologías de servidor . . . . .	13
2.6.1. Docker y Docker Compose . . . . .	13
2.6.2. Servidor web Nginx . . . . .	13
2.7. Hardware utilizado . . . . .	14
2.7.1. Raspberry Pi . . . . .	14
2.7.2. NodeMCU Esp32 . . . . .	14
2.7.3. Identificación por radiofrecuencia . . . . .	15
2.7.4. Módulo RC522 . . . . .	15
2.7.5. Buzzer sonoro . . . . .	16
<b>3. Diseño e implementación</b>	17
3.1. Arquitectura general del sistema . . . . .	17
3.1.1. Funcionamiento . . . . .	17
3.1.2. Diagrama de bloques . . . . .	17
3.2. Flujo general del sistema . . . . .	18
3.2.1. Ingreso de repuesto . . . . .	18
3.2.2. Cambio de estado . . . . .	19

**Bibliografía**

21

3.2.3.	Retiro de repuesto . . . . .	20
3.3.	Arquitectura de datos . . . . .	20
3.3.1.	Diagrama de base de datos . . . . .	20
3.3.2.	Estructura de archivos para ORM . . . . .	21
3.3.3.	Desarrollo de modelos y tablas . . . . .	22
3.3.4.	Desarrollo de migraciones . . . . .	24
3.3.5.	Interacción con la base de datos . . . . .	25
3.4.	Desarrollo API REST . . . . .	27
3.4.1.	Patrón de desarrollo . . . . .	27
3.4.2.	Ruteo de la API . . . . .	28
3.4.3.	Controladores de la API . . . . .	29
3.4.4.	Endpoints HTTP . . . . .	30
3.5.	Comunicación MQTT . . . . .	31
3.5.1.	Diagrama MQTT . . . . .	31
3.5.2.	Topics MQTT . . . . .	31
3.5.3.	Broker MQTT . . . . .	32
3.5.4.	MQTT en nodos . . . . .	33
3.5.5.	MQTT en API REST . . . . .	34
3.6.	API para mensajería . . . . .	36
3.6.1.	Implementación . . . . .	37
3.7.	Desarrollo frontend . . . . .	38
3.7.1.	Patrón de desarrollo . . . . .	38
3.7.2.	Interfaces de usuario . . . . .	39
	Nueva orden de trabajo . . . . .	39
	Listar órdenes de trabajo . . . . .	43
	Retirar orden de trabajo . . . . .	46
	Autenticación a API de mensajería . . . . .	47
3.8.	Arquitectura de servidor . . . . .	48
3.8.1.	Implementación de contenedores . . . . .	48
3.9.	Implementación de Hardware . . . . .	50
4.	<b>Ensayos y resultados</b> . . . . .	53
4.1.	Banco de pruebas . . . . .	53
4.2.	Ensayos sobre la API . . . . .	54
4.3.	Ensayos sobre el sistema . . . . .	55
4.4.	Resultados . . . . .	56
5.	<b>Conclusiones</b> . . . . .	57
5.1.	Conclusiones generales . . . . .	57
5.2.	Próximos pasos . . . . .	57
	<b>Bibliografía</b> . . . . .	59

## Índice de figuras

2.1. Arquitectura MQTT publish/subscribe. . . . .	6
2.2. Diagrama base de datos relacional. . . . .	8
2.3. Arquitectura API Rest. . . . .	9

## Índice de figuras

2.1. Arquitectura MQTT publish/subscribe. . . . .	6
2.2. Diagrama base de datos relacional. . . . .	8
2.3. Arquitectura API Rest. . . . .	9
2.4. Arquitectura API Rest. . . . .	11
2.5. Arquitectura API Rest. . . . .	12
3.1. Diagrama de funciones generales. . . . .	17
3.2. Diagrama de bloques y tecnologías del sistema. . . . .	18
3.3. Flujo en el ingreso de una nueva orden de trabajo. . . . .	19
3.4. Flujo en el cambio de estado de una orden de trabajo. . . . .	19
3.5. Flujo en el retiro de una orden de trabajo. . . . .	20
3.6. Diagrama de base de datos. . . . .	21
3.7. Estructura de carpetas y archivos para ORM Sequelize. . . . .	21
3.8. Estructura de carpetas en patrón de desarrollo modular. . . . .	27
3.9. Estructura de archivos en carpeta de rutas. . . . .	28
3.10. Estructura de archivos en la carpeta controllers. . . . .	29
3.11. Tabla descriptiva de endpoints o rutas de la API. . . . .	30
3.12. Diagrama de funciones MQTT. . . . .	31
3.13. Tabla de topics MQTT. . . . .	32
3.14. Estructura de carpetas del servicio Eclipse Mosquitto en Docker. . . . .	33
3.15. Tabla de detalles de sonidos de nodos. . . . .	34
3.16. Flujo de código fuente para MQTT en API REST. . . . .	36
3.17. Flujo en MVC. . . . .	38
3.18. Interfaz de usuario para nueva orden de trabajo. . . . .	40
3.19. Interfaz de usuario de tipo modal para seleccionar tipo de trabajo. . . . .	40
3.20. Interfaz de usuario de tipo modal para seleccionar o buscar cliente. . . . .	41
3.21. Interfaz de usuario de tipo modal con el resultado de búsqueda de clientes. . . . .	41
3.22. Interfaz de usuario de tipo modal para seleccionar o buscar motocicleta. . . . .	42
3.23. Interfaz de usuario de tipo modal con el resultado de búsqueda de motocicletas. . . . .	42
3.24. Interfaz de usuario para nueva orden de trabajo. . . . .	43
3.25. Interfaz de usuario para listar las órdenes de trabajo. . . . .	44
3.26. Confirmar envío de mensaje de texto al cliente. . . . .	45
3.27. Confirmar reenvío de mensaje de texto al cliente. . . . .	46
3.28. Formulario para retirar orden de trabajo. . . . .	47
3.29. Confirmar retiro de orden de trabajo. . . . .	47
3.30. Interfaz para autenticación en API de mensajería. . . . .	48
3.31. Contenedores de Docker. . . . .	50
3.32. Diagrama de red del sistema. . . . .	51
3.33. Build en PlatformIO. . . . .	51
3.34. Upload en PlatformIO. . . . .	52

## VIII

3.35. Upload en PlatformIO. . . . .	52
4.1. Entornos de desarrollo y pruebas del sistema. . . . .	53
4.2. Estructura de carpetas y archivos en Insomnia. . . . .	54
4.3. Estructura de carpetas y archivos en Insomnia. . . . .	55
4.4. Respuesta de una petición en Insomnia. . . . .	55

## Índice de tablas

1.1. <span style="color:red;">caption corto . . . . .</span>	3
--	---

## Índice de tablas

1.1. <span style="color:red;">servicios estado del arte . . . . .</span>	3
--	---

## Capítulo 1

# Introducción general

El presente capítulo aborda cuestiones relativas a las etapas en los procesos de rectificación de motopartes en la empresa Arancibia Rectificaciones y las problemáticas de administración que motivaron la implementación del sistema.

### 1.1. Descripción del sistema

En el taller de rectificaciones de motopartes se realizan diferentes tipos de trabajos relacionados a la tornería [1] de piezas pertenecientes a los motores de motocicletas. Estos trabajos pasan por distintas etapas o estados en los cuales se realizan procesos específicos como encamisado de cilindro [2], cambio de biela [3], balanceo de cigüeñal [4], rectificación de cilindro [5], rectificación de tapa de cilindro [6], entre otros.

Las etapas en general que atraviesa un repuesto desde que ingresa hasta que es retirado de la empresa son:

1. Ingreso de la pieza o repuesto a la empresa:

Un cliente de la empresa se presenta con una pieza para ser reparada, el personal de atención le informa el precio del servicio, fecha de entrega, entre otros datos.

2. Registro de datos del cliente y generación de orden de trabajo:

Luego de aceptadas las condiciones por el cliente, se registran sus datos y se genera una orden de trabajo.

3. Puesta en espera del repuesto:

Se ubica la pieza en el sector de trabajos en espera.

4. Trabajo de mano de obra correspondiente:

Una vez que un empleado de taller de la empresa está libre toma el repuesto para efectuar la mano de obra necesaria.

5. Finalización del trabajo de mano de obra:

Se coloca el repuesto en el sector de finalizados a la espera de ser retirado por el cliente.

6. Entrega del repuesto al cliente:

Cuando el cliente pasa a retirar su pieza, se registran los datos correspondientes y se hace la entrega finalizando así todas las etapas del servicio.

*Dedicado a mis padres.*

## 1.2. Motivación

Este trabajo surgió de la necesidad de desarrollar un sistema que permita visualizar en qué etapa se encuentra un repuesto en particular en la empresa, esto permite conocer el estado general de los trabajos, informar a los clientes, tomar decisiones administrativas o técnicas, realizar reportes, etc.

Cuando un cliente se comunica con la empresa para saber si puede pasar a retirar la pieza, el personal de atención tiene que consultar a los empleados de taller el estado en el que se encuentra el trabajo, estos deben dejar de hacer sus tareas por un momento para buscar y responder la consulta, lo cual interrumpe el proceso, genera demoras y consume tiempo. Además, mientras esto sucede, el cliente debe esperar varios minutos.

Por otro lado, resulta complicado cuando el personal de la empresa desea obtener información como: cantidad de trabajos en cada sector, tiempos promedio de proceso, trabajos para ser retirados, cantidad de servicios efectuados en un lapso de tiempo determinado, etc., ya que la manera de obtener estos datos es realizando conteos manuales lo cual resulta improductivo y demanda demasiado tiempo por lo que nunca se realizan estos informes.

Ante este escenario es evidente la necesidad de contar con un sistema informático que posibilite registrar las etapas del proceso y generar la información necesaria para cuando esta sea requerida.

## 1.3. Estado del arte

En el mercado argentino actualmente se ofrecen diferentes soluciones para resolver problemas relacionados al control de productos ya sea de stock, logística, trazabilidad, transporte, distribución, entre otros. Estas soluciones están basadas en su mayoría en tecnología de lectura de código de barras o de ingresos manuales de datos mediante teclado. Además, existen algunas soluciones de empresas extranjeras, más orientadas al sector industrial, basadas en tecnología RFID [7], en su mayoría por banda UHF [8].

No se encontraron soluciones en el mercado para las necesidades específicas que se plantean en este trabajo. Una de las problemáticas que plantea el escenario para el cual se desarrolla este sistema, es el contexto en el que se realizan los servicios. Las piezas que se reparan están sometidas constantemente a aceites, residuos grasos, polvo, etc. Este escenario hace que se descarte el uso de la tecnología de lectura de código de barras, ya que cualquier lectura a un código sería dificultada por lo mencionado, quedando como mejor opción la utilización de tecnología RFID.

Las soluciones RFID encontradas están planteadas para otro tipo de rubros o industrias, usan generalmente banda UHF y son demasiado costosas para una empresa chica o mediana.

Únicamente se encontró una empresa en Argentina que ofrece servicios algo similares a los que se plantean en este trabajo, Teletrónica S.A. [9]. A continuación se detallan algunas características.

#### 1.4. Objetivos y alcance

3

TABLA 1.1. servicios ofrecidos por Telelectrónica.

Característica	Telelectrónica
Tecnología RFID	Sí
Rubro motores	No
Costo accesible a empresa pequeña	No
Hardware económico	No

La principal característica que imposibilita acceder a este tipo de servicios con empresas argentinas o extranjeras es el alto costo de desarrollo e implementación, debido a que están enfocadas en industrias o empresas grandes que pueden afrontar inversiones de gran escala.

#### 1.4. Objetivos y alcance

El objetivo de este trabajo fue desarrollar un sistema que permita registrar los estados por los que va pasando un repuesto en el taller de tornería de la empresa y poder visualizar esos estados en una plataforma web o móvil.

En primer lugar, se realizó el abordaje de requerimientos de la empresa y se comenzó con la planificación del proyecto. Se continuó con el diseño de la arquitectura tecnológica que se emplearía para el sistema, tanto a nivel de herramientas de desarrollo de software como el hardware a utilizar.

Además, se tuvo en cuenta que los trabajadores de la empresa no debían detener sus tareas para realizar ingresos en teclados ya que esto generaría una interrupción en el flujo de trabajo y el registro de datos en el sistema sería incomodo. Fue por esta razón, principalmente, que se pensó en una tecnología que permita enviar datos a un servidor sin necesidad de manipulación de teclados o dispositivos similares. La tecnología que cumple con este requerimiento es la RFID, la que abordaremos en el siguiente capítulo.

Una vez determinado el diseño y la planificación se comenzaron las investigaciones necesarias, las cuales requirieron una parte importante del tiempo total del trabajo.

El alcance del trabajo se acotó a lo siguiente:

- Desarrollo frontend: aplicación web compatible con móvil.
- Desarrollo backend: API Rest.
- Desarrollo de base de datos.
- Desarrollo e implementación en dispositivos de hardware IoT.
- Desarrollo e implementación de la infraestructura total del sistema, servidor basado en contenedores para servicio web, API Rest, bróker MQTT y base de datos.
- Implementaciones particulares como gabinetes, soportes para tags RFID, entre otros.

1

## Capítulo 1

### Introducción general

El presente capítulo aborda cuestiones relativas a las etapas en los procesos de rectificación de motopartes en la empresa Arancibia Rectificaciones y las problemáticas de administración que motivaron la implementación del sistema.

#### 1.1. Descripción del sistema

En el taller de rectificaciones de motopartes se realizan diferentes tipos de trabajos relacionados a la tornería [1] de piezas pertenecientes a los motores de motocicletas. Estos trabajos pasan por distintas etapas o estados en los cuales se realizan procesos específicos como encamisado de cilindro [2], cambio de biela [3], balanceo de cigüeñal [4], rectificación de cilindro [5], rectificación de tapa de cilindro [6], entre otros.

Las etapas en general que atraviesa un repuesto desde que ingresa hasta que es retirado de la empresa son:

1. Ingreso de la pieza o repuesto a la empresa:  
Un cliente de la empresa se presenta con una pieza para ser reparada, el personal de atención le informa el precio del servicio, fecha de entrega, entre otros datos.
2. Registro de datos del cliente y generación de orden de trabajo:  
Luego de aceptadas las condiciones por el cliente, se registran sus datos y se genera una orden de trabajo.
3. Puesta en espera del repuesto:  
Se ubica la pieza en el sector de trabajos en espera.
4. Trabajo de mano de obra correspondiente:  
Una vez que un empleado de taller de la empresa está libre toma el repuesto para efectuar la mano de obra necesaria.
5. Finalización del trabajo de mano de obra:  
Se coloca el repuesto en el sector de finalizados a la espera de ser retirado por el cliente.
6. Entrega del repuesto al cliente:  
Cuando el cliente pasa a retirar su pieza, se registran los datos correspondientes y se hace la entrega finalizando así todas las etapas del servicio.

## 1.2. Motivación

Este trabajo surgió de la necesidad de desarrollar un sistema que permita visualizar en qué etapa se encuentra un repuesto en particular en la empresa, esto permite conocer el estado general de los trabajos, informar a los clientes, tomar decisiones administrativas o técnicas, realizar reportes, etc.

Cuando un cliente se comunica con la empresa para saber si puede pasar a retirar la pieza, el personal de atención tiene que consultar a los empleados de taller el estado en el que se encuentra el trabajo, estos deben dejar de hacer sus tareas por un momento para buscar y responder la consulta, lo cual interrumpe el proceso, genera demoras y consume tiempo. Además, mientras esto sucede, el cliente debe esperar varios minutos.

Por otro lado, resulta complicado cuando el personal de la empresa desea obtener información como: cantidad de trabajos en cada sector, tiempos promedio de proceso, trabajos para ser retirados, cantidad de servicios efectuados en un lapso de tiempo determinado, etc., ya que la manera de obtener estos datos es realizando conteos manuales lo cual resulta improductivo y demanda demasiado tiempo por lo que nunca se realizan estos informes.

Ante este escenario es evidente la necesidad de contar con un sistema informático que posibilite registrar las etapas del proceso y generar la información necesaria para cuando esta sea requerida.

## 1.3. Estado del arte

En el mercado argentino actualmente se ofrecen diferentes soluciones para resolver problemas relacionados al control de productos ya sea de stock, logística, trazabilidad, transporte, distribución, entre otros. Estas soluciones están basadas en su mayoría en tecnología de lectura de código de barras o de ingresos manuales de datos mediante teclado. Además, existen algunas soluciones de empresas extranjeras, más orientadas al sector industrial, basadas en tecnología RFID [7], en su mayoría por banda UHF [8].

No se encontraron soluciones en el mercado para las necesidades específicas que se plantean en este trabajo. Una de las problemáticas que plantea el escenario para el cual se desarrolla este sistema, es el contexto en el que se realizan los servicios. Las piezas que se reparan están sometidas constantemente a aceites, residuos grasos, polvo, etc. Este escenario hace que se descarte el uso de la tecnología de lectura de código de barras, ya que cualquier lectura a un código sería dificultada por lo mencionado, quedando como mejor opción la utilización de tecnología RFID.

Las soluciones RFID encontradas están planteadas para otro tipo de rubros o industrias, usan generalmente banda UHF y son demasiado costosas para una empresa chica o mediana.

Únicamente se encontró una empresa en Argentina que ofrece servicios algo similares a los que se plantean en este trabajo, Telectrónica S.A. [9]. A continuación se detallan algunas características.

## Capítulo 2

### Introducción específica

En este capítulo se describen las herramientas, tecnologías y hardware que se utilizó para el desarrollo del sistema.

#### 2.1. Internet de las cosas

IoT [33] (*Internet of Things*) o Internet de las cosas en español, es un concepto tecnológico que hace referencia a la interconexión digital de objetos cotidianos a través de internet. Se trata de una red de dispositivos, sensores y otros elementos físicos que se comunican entre sí y con sistemas de información, lo que permite recopilar y procesar datos de manera remota.

El IoT abarca desde dispositivos simples como sensores de temperatura hasta dispositivos más complejos como automóviles autónomos, todos conectados a internet y capaces de intercambiar información. Esta tecnología tiene el potencial de cambiar la forma en la interacción con el mundo físico, mejorando la eficiencia, seguridad y calidad de vida en muchos ámbitos, como la industria, el hogar, la salud, la agricultura, el transporte y más.

La conexión en red de los objetos permite controlarlos de manera remota, obtener información en tiempo real y tomar decisiones basadas en los datos recolectados. Esto permite la creación de sistemas inteligentes que pueden automatizar tareas, reducir costos y mejorar la eficiencia. Además, la interconexión de dispositivos puede generar nuevos modelos de negocio y oportunidades de innovación en diferentes sectores.

#### 2.2. Protocolos de comunicación

##### 2.2.1. Protocolo HTTP

HTTP [10], por sus siglas en inglés: *Hypertext Transfer Protocol*, es un protocolo de tipo cliente-servidor [11], mediante el cual se establece una comunicación enviando peticiones y obteniendo respuestas.

Las características principales de este protocolo son:

- Basado en arquitectura cliente-servidor.
- Además de hipertexto (HTML [12]) se puede utilizar para transmitir otro tipo de documentos como imágenes o videos.
- Es un protocolo de capa de aplicación del modelo OSI [13].

#### 1.4. Objetivos y alcance

TABLA 1.1. servicios ofrecidos por Telectrónica.

Característica	Telectrónica
Tecnología RFID	Sí
Rubro motores	No
Costo accesible a empresa pequeña	No
Hardware económico	No

La principal característica que imposibilita acceder a este tipo de servicios con empresas argentinas o extranjeras es el alto costo de desarrollo e implementación, debido a que están enfocadas en industrias o empresas grandes que pueden afrontar inversiones de gran escala.

#### 1.4. Objetivos y alcance

El objetivo de este trabajo fue desarrollar un sistema que permita registrar los estados por los que va pasando un repuesto en el taller de tornería de la empresa y poder visualizar esos estados en una plataforma web o móvil.

En primer lugar, se realizó el abordaje de requerimientos de la empresa y se comenzó con la planificación del proyecto. Se continuó con el diseño de la arquitectura tecnológica que se emplearía para el sistema, tanto a nivel de herramientas de desarrollo de software como el hardware a utilizar.

Además, se tuvo en cuenta que los trabajadores de la empresa no debían detener sus tareas para realizar ingresos en teclados ya que esto generaría una interrupción en el flujo de trabajo y el registro de datos en el sistema sería incomodo. Fue por esta razón, principalmente, que se pensó en una tecnología que permita enviar datos a un servidor sin necesidad de manipulación de teclados o dispositivos similares. La tecnología que cumple con este requerimiento es la RFID, la que abordaremos en el siguiente capítulo.

Una vez determinado el diseño y la planificación se comenzaron las investigaciones necesarias, las cuales requirieron una parte importante del tiempo total del trabajo.

El alcance del trabajo se acotó a lo siguiente:

- Desarrollo frontend: aplicación web compatible con móvil.
- Desarrollo backend: API Rest.
- Desarrollo de base de datos.
- Desarrollo e implementación en dispositivos de hardware IoT.
- Desarrollo e implementación de la infraestructura total del sistema, servidor basado en contenedores para servicio web, API Rest, bróker MQTT y base de datos.
- Implementaciones particulares como gabinetes, soportes para tags RFID, entre otros.

- Se transmite principalmente sobre el protocolo TCP [14].

HTTP define un conjunto de métodos de petición, cada uno indica una acción a ejecutar en el servidor. Los más utilizados son:

- GET: se utiliza para recuperar datos.
- POST: sirve principalmente para cargar nuevos datos.
- PATCH: este método aplica modificaciones parciales a los datos existentes.
- PUT: permite reemplazar completamente un registro.
- DELETE: elimina datos específicos.

### 2.2.2. Protocolo MQTT

MQTT [15] son las siglas de *Message Queuing Telemetry Transport*. Se trata de un protocolo de mensajería liviano para usar en casos donde existen recursos limitados de ancho de banda.

Se transmite sobre protocolo TCP en la arquitectura *publish/subscribe* [16].

Los roles que intervienen en un protocolo MQTT son los siguientes:

- Publicadores: son los que envían los datos.
- Suscriptores: son los que consumen los datos.
- Broker: transmite los mensajes publicados a los suscriptores.

Un cliente puede ser publicador, suscriptor o ambos. El broker es el punto central de la comunicación ya que sin este los mensajes nunca llegarían a destino.

En la figura 2.1 se puede apreciar un ejemplo de comunicación en la arquitectura MQTT.

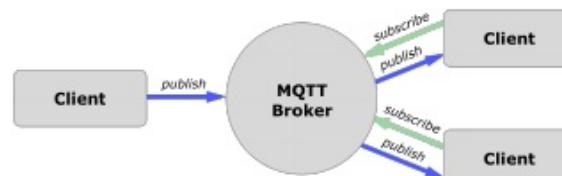


FIGURA 2.1. Arquitectura MQTT publish/subscribe.

La estructura de mensajes en este protocolo se divide en dos: *topics* los cuales son de tipo jerárquicos, utilizando la barra (/) como separador, y *payload* en dónde se incluye el mensaje que se quiere transmitir. Por ejemplo: topic: "nodos/procesos/guardar", payload: "mensaje de ejemplo". Siguiendo este ejemplo un cliente podría suscribirse a ese topic o a una jerarquía más alta y recibir todos los mensajes de los topics que comiencen con nodo/procesos.

### 2.2.3. Broker Eclipse Mosquitto

Eclipse Mosquitto [17] es un broker MQTT *Open Source* liviano y adecuado para utilizar en todo tipo de dispositivos sobre todo aquellos que cuenten con baja potencia como microcontroladores.

El objetivo de Mosquitto es proporcionar una implementación ligera y de bajo consumo de recursos para permitir la comunicación entre dispositivos IoT en redes con ancho de banda limitado y recursos de memoria.

Mosquitto es compatible con una amplia gama de lenguajes de programación, lo que lo hace fácilmente integrable con diferentes aplicaciones. Además, cuenta con una arquitectura flexible y escalable que permite su implementación en dispositivos con diferentes capacidades de procesamiento y memoria.

Otra característica importante de este broker es su capacidad para manejar conexiones seguras a través del uso de protocolos de seguridad como SSL/TLS y SASL. Esto permite la comunicación segura y cifrada entre dispositivos de IoT en diferentes entornos de red.

## 2.3. Bases de datos

Las bases de datos son una parte esencial de cualquier aplicación IoT, ya que se utilizan para almacenar y gestionar los datos recopilados por los dispositivos IoT. En esta sección, se describen las tecnologías utilizadas en bases de datos.

### 2.3.1. Bases de datos relacionales

Las bases de datos relacionales [18] son un tipo de sistema de gestión de bases de datos (SGBD) que se basa en el modelo de datos relacional. Este modelo se utiliza para organizar y almacenar datos en tablas, donde cada tabla representa una entidad o concepto del mundo real y cada fila representa una instancia de esa entidad.

Las tablas se relacionan entre sí mediante claves primarias y claves externas, lo que permite establecer relaciones entre las entidades y realizar consultas complejas que combinan datos de varias tablas. Además, las bases de datos relacionales utilizan el lenguaje de consulta estructurado (SQL o *Structured Query Language*) para interactuar con los datos almacenados en la base de datos.

En la figura 2.2 se puede apreciar un ejemplo de un diagrama de base de datos relacional con sus tablas, filas y relaciones.

### 2.3.2. Sistema de gestión de base de datos MySQL

MySQL [19] es un sistema de gestión de bases de datos relacional y de código abierto, que utiliza el lenguaje SQL para interactuar con los datos almacenados en la base de datos.

Ofrece una amplia gama de características avanzadas, como soporte para transacciones ACID, índices avanzados, clústeres de alta disponibilidad y replicación. Además, admite múltiples lenguajes de programación, incluyendo C, C++, Python, Java y Ruby, lo que lo hace extremadamente flexible y escalable.

## Capítulo 2

# Introducción específica

En este capítulo se describen las herramientas, tecnologías y hardware que se utilizó para el desarrollo del sistema.

### 2.1. Internet de las cosas

IoT [10] (*Internet of Things*) o Internet de las cosas en español, es un concepto tecnológico que hace referencia a la interconexión digital de objetos cotidianos a través de internet. Se trata de una red de dispositivos, sensores y otros elementos físicos que se comunican entre sí y con sistemas de información, lo que permite recopilar y procesar datos de manera remota.

El IoT abarca desde dispositivos simples como sensores de temperatura hasta dispositivos más complejos como automóviles autónomos, todos conectados a internet y capaces de intercambiar información. Esta tecnología tiene el potencial de cambiar la forma en la interacción con el mundo físico, mejorando la eficiencia, seguridad y calidad de vida en muchos ámbitos, como la industria, el hogar, la salud, la agricultura, el transporte y más.

La conexión en red de los objetos permite controlarlos de manera remota, obtener información en tiempo real y tomar decisiones basadas en los datos recolectados. Esto permite la creación de sistemas inteligentes que pueden automatizar tareas, reducir costos y mejorar la eficiencia. Además, la interconexión de dispositivos puede generar nuevos modelos de negocio y oportunidades de innovación en diferentes sectores.

### 2.2. Protocolos de comunicación

#### 2.2.1. Protocolo HTTP

HTTP [11], por sus siglas en inglés: *Hypertext Transfer Protocol*, es un protocolo de tipo cliente-servidor [12], mediante el cual se establece una comunicación enviando peticiones y obteniendo respuestas.

Las características principales de este protocolo son:

- Basado en arquitectura cliente-servidor.
- Además de hipertexto (HTML [13]) se puede utilizar para transmitir otro tipo de documentos como imágenes o videos.
- Es un protocolo de capa de aplicación del modelo OSI [14].

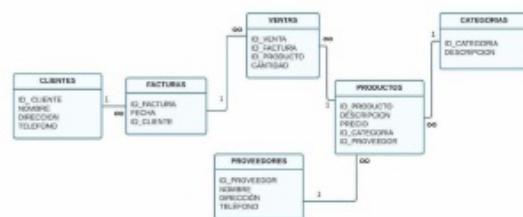


FIGURA 2.2. Diagrama base de datos relacional.

Tiene múltiples capas de seguridad integradas en el sistema, incluyendo autenticación y control de acceso basado en roles.

Debido a su combinación de características avanzadas, flexibilidad y seguridad, MySQL se utiliza ampliamente en aplicaciones de misión crítica y de alta disponibilidad, incluyendo aplicaciones IoT.

## 2.4. Tecnologías backend

Este tipo de tecnologías se utilizan para desarrollar la lógica de la aplicación y gestionar la comunicación entre el servidor y los dispositivos IoT. En esta sección, se describen las tecnologías backend utilizadas.

### 2.4.1. API RESTful

Las APIs RESTful [20] (*Representational State Transfer*) son una arquitectura de diseño de aplicaciones web que utiliza el protocolo HTTP para transferir datos. Las API RESTful están diseñadas para ser escalables, flexibles y fáciles de entender para los desarrolladores y los clientes que las consumen.

Están basadas en el concepto de recursos, que son objetos o conjuntos de datos que se pueden acceder a través de una URI (Identificador de recurso uniforme, por sus siglas en inglés). Cada recurso tiene un conjunto de operaciones que se pueden realizar sobre él, como GET (para obtener los datos del recurso), POST (para crear un nuevo recurso), PUT (para actualizar un recurso existente) y DELETE (para eliminar un recurso).

En la figura 2.3 se puede apreciar un ejemplo de una arquitectura API rest, incluyendo la petición o *request* del usuario en formato JSON, el método HTTP y la respuesta o *response* del servidor.

- Se transmite principalmente sobre el protocolo TCP [15].

HTTP define un conjunto de métodos de petición, cada uno indica una acción a ejecutar en el servidor. Los más utilizados son:

- GET: se utiliza para recuperar datos.
- POST: sirve principalmente para cargar nuevos datos.
- PATCH: este método aplica modificaciones parciales a los datos existentes.
- PUT: permite reemplazar completamente un registro.
- DELETE: elimina datos específicos.

### 2.2.2. Protocolo MQTT

MQTT [16] son las siglas de *Message Queuing Telemetry Transport*. Se trata de un protocolo de mensajería liviano para usar en casos donde existen recursos limitados de ancho de banda.

Se transmite sobre protocolo TCP en la arquitectura *publish/subscribe* [17].

Los roles que intervienen en un protocolo MQTT son los siguientes:

- Publicadores: son los que envían los datos.
- Suscriptores: son los que consumen los datos.
- Broker: transmite los mensajes publicados a los suscriptores.

Un cliente puede ser publicador, suscriptor o ambos. El broker es el punto central de la comunicación ya que sin este los mensajes nunca llegarían a destino.

En la figura 2.1 se puede apreciar un ejemplo de comunicación en la arquitectura MQTT.

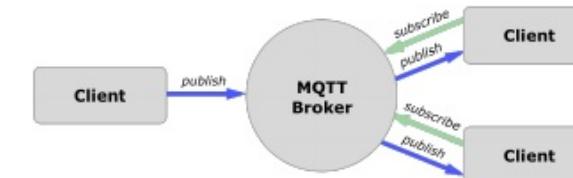


FIGURA 2.1. Arquitectura MQTT publish/subscribe.

La estructura de mensajes en este protocolo se divide en dos: *topics* los cuales son de tipo jerárquicos, utilizando la barra (/) como separador, y *payload* en dónde se incluye el mensaje que se quiere transmitir. Por ejemplo: topic: "nodos/procesos/guardar", payload: "mensaje de ejemplo". Siguiendo este ejemplo un cliente podría suscribirse a ese topic o a una jerarquía más alta y recibir todos los mensajes de los topics que comienzan con nodo/procesos.

## 2.4. Tecnologías backend

9

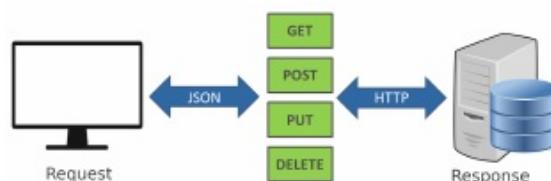


FIGURA 2.3. Arquitectura API Rest.

### 2.4.2. Node.js

Node.js [21] es un entorno de tiempo de ejecución de JavaScript de código abierto que se ejecuta en el servidor. Fue creado en 2009 con el objetivo de poder desarrollar aplicaciones escalables y de alto rendimiento utilizando el mismo lenguaje de programación para el lado del servidor y del cliente.

Se basa en el motor de JavaScript V8 de Google, lo que lo hace muy rápido y eficiente. Utiliza un modelo de E/S sin bloqueo y orientado a eventos, lo que significa que es capaz de manejar un gran número de solicitudes simultáneas sin bloquear el proceso. Esto lo hace especialmente adecuado para aplicaciones web en tiempo real y aplicaciones de transmisión de medios.

También cuenta con una amplia variedad de módulos y bibliotecas disponibles a través de su gestor de paquetes NPM (*Node Package Manager*). Esto permite aprovechar funcionalidades existentes, desde la creación de APIs RESTful hasta la manipulación de archivos, la comunicación con dispositivos y bases de datos.

### 2.4.3. Express

Express [22] es un popular framework de Node.js utilizado para la creación de aplicaciones web y APIs RESTful. Fue creado en 2010 y es mantenido por la comunidad de desarrolladores de Node.js.

Proporciona una serie de funcionalidades para simplificar el proceso de creación de aplicaciones web. Entre ellas se incluyen el manejo de rutas, la gestión de middleware, la manipulación de sesiones y cookies, la autenticación de usuarios y la integración con bases de datos.

En Express, el manejo de rutas se realiza mediante la definición estas y los controladores correspondientes. Las rutas son los patrones utilizados para identificar las solicitudes HTTP entrantes que serán atendidas por la aplicación web. Los controladores son las funciones que se encargan de procesar esas solicitudes y generar la respuesta adecuada.

Para definir una ruta, se utiliza el método correspondiente al verbo HTTP que se desea manejar (GET, POST, PUT, DELETE, etc.), seguido de la ruta o patrón que se desea asociar a esa solicitud.

### 2.4.4. Sequelize

Sequelize [23] es una biblioteca de ORM (*Object-Relational Mapping*) para Node.js que permite trabajar con bases de datos relacionales como MySQL, PostgreSQL,

## 2.3. Bases de datos

7

### 2.2.3. Broker Eclipse Mosquitto

Eclipse Mosquitto [18] es un broker MQTT *Open Source* liviano y adecuado para utilizar en todo tipo de dispositivos sobre todo aquellos que cuenten con baja potencia como microcontroladores.

El objetivo de Mosquitto es proporcionar una implementación ligera y de bajo consumo de recursos para permitir la comunicación entre dispositivos IoT en redes con ancho de banda limitado y recursos de memoria.

Mosquitto es compatible con una amplia gama de lenguajes de programación, lo que lo hace fácilmente integrable con diferentes aplicaciones. Además, cuenta con una arquitectura flexible y escalable que permite su implementación en dispositivos con diferentes capacidades de procesamiento y memoria.

Otra característica importante de este broker es su capacidad para manejar conexiones seguras a través del uso de protocolos de seguridad como SSL/TLS y SASL. Esto permite la comunicación segura y cifrada entre dispositivos de IoT en diferentes entornos de red.

## 2.3. Bases de datos

Las bases de datos son una parte esencial de cualquier aplicación IoT, ya que se utilizan para almacenar y gestionar los datos recopilados por los dispositivos IoT. En esta sección, se describen las tecnologías utilizadas en bases de datos.

### 2.3.1. Bases de datos relacionales

Las bases de datos relacionales [19] son un tipo de sistema de gestión de bases de datos (SGBD) que se basa en el modelo de datos relacional. Este modelo se utiliza para organizar y almacenar datos en tablas, donde cada tabla representa una entidad o concepto del mundo real y cada fila representa una instancia de esa entidad.

Las tablas se relacionan entre sí mediante claves primarias y claves externas, lo que permite establecer relaciones entre las entidades y realizar consultas complejas que combinan datos de varias tablas. Además, las bases de datos relacionales utilizan el lenguaje de consulta estructurado (SQL o *Structured Query Language*) para interactuar con los datos almacenados en la base de datos.

En la figura 2.2 se puede apreciar un ejemplo de un diagrama de base de datos relacional con sus tablas, filas y relaciones.

### 2.3.2. Sistema de gestión de base de datos MySQL

MySQL [20] es un sistema de gestión de bases de datos relacional y de código abierto, que utiliza el lenguaje SQL para interactuar con los datos almacenados en la base de datos.

Ofrece una amplia gama de características avanzadas, como soporte para transacciones ACID, índices avanzados, clústeres de alta disponibilidad y replicación. Además, admite múltiples lenguajes de programación, incluyendo C, C++, Python, Java y Ruby, lo que lo hace extremadamente flexible y escalable.

SQLite, y MSSQL. Facilita el acceso a la base de datos y permite la creación de consultas a través de una interfaz de programación de aplicaciones (API) de alto nivel basada en objetos.

Se utiliza junto a Express y Node.js para simplificar el proceso de acceso y manipulación de datos en bases de datos relacionales. Al utilizar Sequelize, se pueden crear modelos de datos que representen las tablas de la base de datos, lo que permite interactuar con la base de datos utilizando objetos en lugar de consultas SQL.

## 2.5. Tecnologías frontend

Las tecnologías frontend son aquellas que se utilizan para crear la parte visual y de interacción de una aplicación web o móvil. Estas tecnologías se enfocan en la presentación y manipulación de la interfaz de usuario, en la interacción con el usuario final y actúa como intermediario entre el usuario final y el backend de la aplicación.

Algunas de las tecnologías frontend más comunes son:

- HTML [12] (*HyperText Markup Language*): es el lenguaje de marcado que se utiliza para estructurar y dar formato al contenido de una página web.
- CSS [24] (*Cascading Style Sheets*): es el lenguaje utilizado para dar estilo y diseño a una página web, permitiendo la personalización de fuentes, colores, márgenes, tamaños y otros aspectos visuales.
- JavaScript [25]: es un lenguaje de programación que se utiliza para hacer que la página web sea interactiva y dinámica, permitiendo la manipulación de elementos del DOM [26] (*Document Object Model*), eventos, animaciones y otras acciones en el lado del cliente.
- Frameworks de JavaScript: son bibliotecas que permiten simplificar el desarrollo de aplicaciones web, proporcionando funcionalidades predefinidas y estructuras de organización.
- Bibliotecas de diseño: son herramientas que ofrecen componentes visuales predefinidos y estilos de diseño que permiten crear páginas web con un aspecto más profesional y elegante.
- Herramientas de gestión de paquetes: son programas que permiten gestionar las dependencias de los proyectos y mantener actualizadas las librerías utilizadas en el desarrollo.

### 2.5.1. Ionic framework

Ionic [27] es un framework de desarrollo de aplicaciones móviles híbridas basado en tecnologías web como HTML, CSS y JavaScript. Permite crear aplicaciones móviles para iOS, Android y la web utilizando un conjunto de herramientas y bibliotecas predefinidas.

Ofrece una gran cantidad de componentes visuales, animaciones y funcionalidades para crear aplicaciones móviles con una apariencia y experiencia de usuario nativa, similar a las aplicaciones desarrolladas con tecnologías nativas como Java

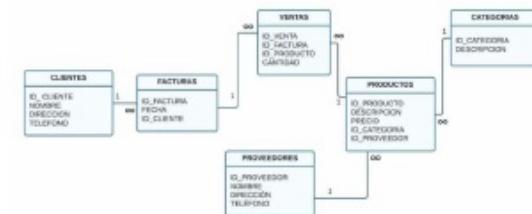


FIGURA 2.2. Diagrama base de datos relacional.

Tiene múltiples capas de seguridad integradas en el sistema, incluyendo autenticación y control de acceso basado en roles.

Debido a su combinación de características avanzadas, flexibilidad y seguridad, MySQL se utiliza ampliamente en aplicaciones de misión crítica y de alta disponibilidad, incluyendo aplicaciones IoT.

## 2.4. Tecnologías backend

Este tipo de tecnologías se utilizan para desarrollar la lógica de la aplicación y gestionar la comunicación entre el servidor y los dispositivos IoT. En esta sección, se describen las tecnologías backend utilizadas.

### 2.4.1. API RESTful

Las APIs RESTful [21] (*Representational State Transfer*) son una arquitectura de diseño de aplicaciones web que utiliza el protocolo HTTP para transferir datos. Las API RESTful están diseñadas para ser escalables, flexibles y fáciles de entender para los desarrolladores y los clientes que las consumen.

Están basadas en el concepto de recursos, que son objetos o conjuntos de datos que se pueden acceder a través de una URI (Identificador de recurso uniforme, por sus siglas en inglés). Cada recurso tiene un conjunto de operaciones que se pueden realizar sobre él, como GET (para obtener los datos del recurso), POST (para crear un nuevo recurso), PUT (para actualizar un recurso existente) y DELETE (para eliminar un recurso).

En la figura 2.3 se puede apreciar un ejemplo de una arquitectura API rest, incluyendo la petición o *request* del usuario en formato JSON, el método HTTP y la respuesta o *response* del servidor.

## 2.6. Tecnologías de servidor

11

para Android o Swift para iOS. Además, permite la integración con otras tecnologías como Angular [28] y React [29].

El uso de tecnologías web permite el desarrollo de aplicaciones móviles de forma más rápida y sencilla que las aplicaciones nativas, ya que se utiliza un único código base que se puede adaptar para cada plataforma. Además, ofrece una gran cantidad de herramientas y servicios para simplificar el proceso de desarrollo, como Ionic Native (para el acceso a las características nativas del dispositivo) y Ionic Appflow (para la implementación continua y la gestión de versiones).

### 2.6. Tecnologías de servidor

#### 2.6.1. Docker y Docker Compose

Docker [30] es una plataforma de software libre que se utiliza para desarrollar, implementar y ejecutar aplicaciones en contenedores. Los contenedores son una forma de virtualización que permiten a los desarrolladores empaquetar una aplicación y todas sus dependencias en una imagen de contenedor, que se puede ejecutar en cualquier entorno que tenga Docker instalado. Docker facilita la implementación de aplicaciones en diferentes plataformas, desde servidores locales hasta nubes públicas.

Docker Compose [31] es una herramienta de Docker que se utiliza para definir y ejecutar aplicaciones de múltiples contenedores. Permite definir todos los servicios necesarios para una aplicación en un archivo de configuración YAML, lo que facilita la implementación y el mantenimiento de la aplicación. Docker Compose puede iniciar todos los contenedores necesarios para la aplicación con un solo comando, lo que ahorra tiempo y reduce los errores.

#### 2.6.2. Servidor web Nginx

Un servidor web es un software que procesa solicitudes HTTP de clientes y responde con contenido estático o dinámico. Los servidores web se utilizan para alojar y servir sitios web y aplicaciones web. Un servidor web típicamente aloja varios sitios web y puede gestionar múltiples solicitudes HTTP simultáneamente.

Nginx [32] es un servidor web de código abierto que se utiliza para alojar y servir sitios web y aplicaciones web. Nginx es conocido por su alta escalabilidad, rendimiento y capacidad de manejar múltiples solicitudes HTTP simultáneamente. Es un servidor web ligero y rápido que se puede utilizar como un proxy inverso para distribuir la carga de trabajo a diferentes servidores y balancear la carga de tráfico.

## 2.7. Hardware utilizado

### 2.7.1. Raspberry Pi

La Raspberry Pi [34] es una pequeña computadora de placa única (*Single Board Computer*, por sus siglas en inglés) diseñada para ser utilizada en proyectos de tecnología, educación y prototipos de hardware. Fue desarrollada por la Fundación Raspberry Pi, una organización benéfica con sede en el Reino Unido.

## 2.4. Tecnologías backend

9



FIGURA 2.3. Arquitectura API Rest.

#### 2.4.2. Node.js

Node.js [22] es un entorno de tiempo de ejecución de JavaScript de código abierto que se ejecuta en el servidor. Fue creado en 2009 con el objetivo de poder desarrollar aplicaciones escalables y de alto rendimiento utilizando el mismo lenguaje de programación para el lado del servidor y del cliente.

Se basa en el motor de JavaScript V8 de Google, lo que lo hace muy rápido y eficiente. Utiliza un modelo de E/S sin bloqueo y orientado a eventos, lo que significa que es capaz de manejar un gran número de solicitudes simultáneas sin bloquear el proceso. Esto lo hace especialmente adecuado para aplicaciones web en tiempo real y aplicaciones de transmisión de medios.

También cuenta con una amplia variedad de módulos y bibliotecas disponibles a través de su gestor de paquetes NPM (*Node Package Manager*). Esto permite aprovechar funcionalidades existentes, desde la creación de APIs RESTful hasta la manipulación de archivos, la comunicación con dispositivos y bases de datos.

#### 2.4.3. Express

Express [23] es un popular framework de Node.js utilizado para la creación de aplicaciones web y APIs RESTful. Fue creado en 2010 y es mantenido por la comunidad de desarrolladores de Node.js.

Proporciona una serie de funcionalidades para simplificar el proceso de creación de aplicaciones web. Entre ellas se incluyen el manejo de rutas, la gestión de middleware, la manipulación de sesiones y cookies, la autenticación de usuarios y la integración con bases de datos.

En Express, el manejo de rutas se realiza mediante la definición estas y los controladores correspondientes. Las rutas son los patrones utilizados para identificar las solicitudes HTTP entrantes que serán atendidas por la aplicación web. Los controladores son las funciones que se encargan de procesar esas solicitudes y generar la respuesta adecuada.

Para definir una ruta, se utiliza el método correspondiente al verbo HTTP que se desea manejar (GET, POST, PUT, DELETE, etc.), seguido de la ruta o patrón que se desea asociar a esa solicitud.

#### 2.4.4. Sequelize

Sequelize [24] es una biblioteca de ORM (*Object-Relational Mapping*) para Node.js que permite trabajar con bases de datos relacionales como MySQL, PostgreSQL,

Principales características del modelo Raspberry Pi 4 seleccionado para este trabajo:

- Procesador Broadcom BCM2711 de cuatro núcleos ARM Cortex-A72 a 1,5 GHz.
- Procesador gráfico VideoCore VI con soporte para OpenGL ES 3.x.
- 8 GB de memoria RAM LPDDR4-3200.
- Bluetooth 5.0.
- Wi-Fi de doble banda 802.11ac.
- Gigabit Ethernet.

Puertos:

- Dos puertos micro-HDMI que pueden soportar dos pantallas con resolución de hasta 4K a 60 fps.
- Dos puertos USB 3.0.
- Dos puertos USB 2.0
- Un puerto GPIO de 40 pines
- Un puerto CSI para la conexión de una cámara
- Un puerto DSI para la conexión de una pantalla táctil
- Un puerto de audio de 3,5 mm.

La placa es compatible con diferentes sistemas operativos, incluyendo Raspberry Pi OS (anteriormente llamado Raspbian), Ubuntu, Windows 10 IoT Core y otros sistemas operativos basados en Linux. También es compatible con diferentes lenguajes de programación como Python, C++, Java y más, lo que la hace ideal para proyectos de IoT, robótica, automatización del hogar, entre otros.

### 2.7.2. NodeMCU Esp32

La NodeMCU ESP32 [35] es una placa de desarrollo basada en el microcontrolador ESP32, que es ampliamente utilizado en proyectos de Internet de las cosas (IoT).

Detalles técnicos del modelo NodeMCU ESP32 WROOM 32 seleccionado para este trabajo:

- Microcontrolador ESP32 de doble núcleo con una velocidad de reloj de hasta 240 MHz y 512 kB de memoria RAM.
- 4 MB de memoria flash integrada para almacenamiento de programas y datos.
- Conectividad inalámbrica Wi-Fi 802.11 b/g/n y Bluetooth 4.2 BLE.
- Interfaz de programación USB integrada para programar la placa y proporcionar una conexión de depuración.
- GPIO de 30 pines para la conexión de sensores, actuadores y otros dispositivos periféricos.

SQLite, y MSSQL. Facilita el acceso a la base de datos y permite la creación de consultas a través de una interfaz de programación de aplicaciones (API) de alto nivel basada en objetos.

Se utiliza junto a Express y Node.js para simplificar el proceso de acceso y manipulación de datos en bases de datos relacionales. Al utilizar Sequelize, se pueden crear modelos de datos que representen las tablas de la base de datos, lo que permite interactuar con la base de datos utilizando objetos en lugar de consultas SQL.

Alguna de las principales ventajas de utilizar Sequelize son:

- Abstracción de la base de datos: proporciona una abstracción de la base de datos que permite a los desarrolladores trabajar con objetos y métodos en lugar de escribir sentencias SQL. Esto facilita el trabajo con la base de datos y reduce la cantidad de código necesario para realizar operaciones CRUD.
- Seguridad: proporciona funciones de seguridad incorporadas, como la prevención de inyecciones SQL y la validación de datos de entrada. Esto ayuda a reducir los riesgos de seguridad y garantiza que los datos almacenados en la base de datos sean confiables.
- Migraciones de base de datos: proporciona una forma fácil de administrar las migraciones de base de datos. Se pueden definir cambios en la estructura de la base de datos utilizando migraciones y aplicarlas en el orden correcto, lo que ayuda a garantizar que la base de datos esté actualizada y que los cambios se realicen de manera controlada.
- Soporte para múltiples bases de datos: admite múltiples bases de datos, lo que significa que se puede trabajar con diferentes bases de datos sin tener que aprender una nueva sintaxis para cada una. Esto hace que sea más fácil trabajar con diferentes bases de datos y reducir el tiempo de aprendizaje.
- Integración con Express: se integra bien con Express, lo que permite trabajar con ambos de manera conjunta. Esto hace que sea más fácil construir aplicaciones web y manejar las operaciones de la base de datos al mismo tiempo.

### 2.4.5. API messenger

Para el envío de mensajes por la aplicación WhatsApp [25] se utilizó una API adicional [26], denominada en este trabajo *API Messenger* a fines de distinguirla de la API de *backend* desarrollada.

Esta API presenta un desarrollo en un patrón modular de tipo DDD (*Domain-Driven Design*). La principal característica en este caso es la creación de capas de infraestructura, servicios y aplicaciones. Esto servirá en caso de cambiar el proveedor de mensajería WhatsApp o agregar otro proveedor en paralelo, lo cual hace el sistema escalable y adaptable a diferentes necesidades y escenarios.

Para los requerimientos de este trabajo se personalizó el código fuente de la *API Messenger* a fines de agregar características adicionales, se detalla esto en el capítulo 3.6.

## 2.7. Hardware utilizado

13

- Interfaces I2C, SPI, UART, PWM y ADC integradas.
- Soporte para el entorno de programación Arduino, así como para MicroPython y Lua.
- Compatible con una amplia gama de bibliotecas y herramientas de desarrollo de código abierto.

### 2.7.3. Identificación por radiofrecuencia

RFID [36] son las siglas en inglés de *Radio Frequency Identification* o identificación por radiofrecuencia en español. Es una tecnología de identificación automática que utiliza ondas de radio para leer y capturar información almacenada en etiquetas o *tags* RFID.

Esta tecnología consta de tres componentes básicos: el tag RFID que comúnmente es una tarjeta o llavero pequeño, el lector RFID y un sistema informático que gestiona la información capturada por el lector.

Los tags RFID contienen antenas y circuitos integrados que permiten la comunicación inalámbrica con los lectores, los cuales envían señales de radio para alimentar y activar los tags, y recibir la información almacenada en ellos. Pueden ser pasivos, activos o semi-activos, dependiendo de si necesitan o no una fuente de alimentación externa.

### 2.7.4. Módulo RC522

El RFID RC522 [37] es un módulo de lectura-escritura RFID que utiliza la tecnología de comunicación de campo cercano (NFC) para la identificación y el intercambio de datos de manera inalámbrica. Entre sus principales características se encuentran:

- Soporte para frecuencias de operación de 13,56 MHz.
- Comunicación mediante el protocolo SPI (*Serial Peripheral Interface*).
- Capacidad para leer y escribir etiquetas RFID de tipo MIFARE, que son ampliamente utilizadas en sistemas de acceso, control de inventario, pago sin contacto, entre otros.
- Funciones de autenticación y encriptación para mayor seguridad en la transmisión de datos.
- Bajo consumo de energía y fácil integración con microcontroladores y otros sistemas embebidos.

### 2.7.5. Buzzer sonoro

El módulo buzzer pasivo KY006 [38] es un pequeño dispositivo electrónico que se utiliza para producir sonidos audibles en una variedad de proyectos electrónicos. Se conecta a la placa de control mediante tres pines.

Entre las características técnicas del módulo KY006 se encuentran:

- Tensión de funcionamiento: 5 V DC.
- Corriente de funcionamiento: <25 mA.

## 2.4. Tecnologías backend

11

### 2.4.6. Portainer

Portainer [27] es una herramienta de administración de contenedores Docker [28] que proporciona una interfaz web para administrar los contenedores y clústeres Docker.

Proporciona una variedad de características útiles, incluyendo:

- Una interfaz web intuitiva y fácil de usar para administrar contenedores Docker, imágenes, volúmenes y redes.
- La posibilidad de crear y configurar contenedores a través de una interfaz gráfica de usuario (GUI).
- La gestión de múltiples hosts desde una única interfaz de usuario, lo que permite administrar varios clústeres o instancias de Docker en diferentes servidores.
- La capacidad de ver y administrar fácilmente el estado de los contenedores, así como el uso de recursos y las estadísticas de rendimiento.
- La gestión de usuarios y roles de acceso, lo que permite controlar el acceso y la autorización de los usuarios.
- La posibilidad de crear y gestionar stacks y servicios de Docker, lo que permite crear aplicaciones complejas y multi-contenedor de manera fácil y rápida.

Para este trabajo se utilizó la herramienta solamente para el monitoreo de los contenedores y el control del uso de recursos de sistema de los mismos.

En las figuras 2.4 y 2.5 podemos observar la interfaz gráfica de la herramienta.

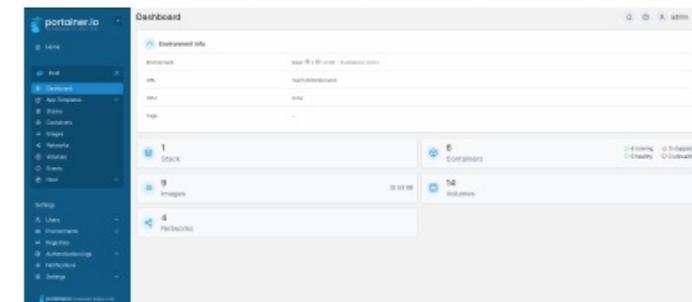


FIGURA 2.4. Arquitectura API Rest.

- Tipo de sonido: continuo y monótono.
- Frecuencia de resonancia:  $2300 \pm 300$  Hz.
- SPL (nivel de presión sonora): >85 dB a 10 cm de distancia.
- Diámetro: 30 mm.
- Altura: 7,5 mm.
- Peso: 4 gramos.

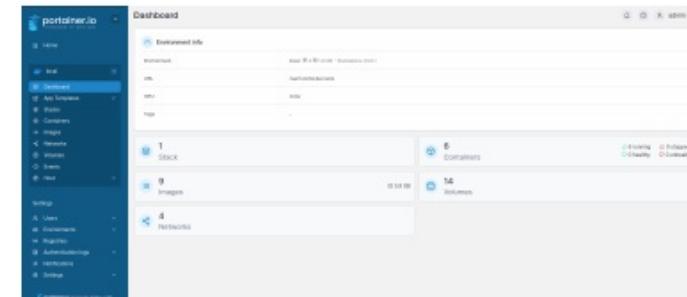


FIGURA 2.5. Arquitectura API Rest.

## 2.5. Tecnologías frontend

Las tecnologías frontend son aquellas que se utilizan para crear la parte visual y de interacción de una aplicación web o móvil. Estas tecnologías se enfocan en la presentación y manipulación de la interfaz de usuario, en la interacción con el usuario final y actúa como intermediario entre el usuario final y el backend de la aplicación.

Algunas de las tecnologías frontend más comunes son:

- HTML [13] (*HyperText Markup Language*): es el lenguaje de marcado que se utiliza para estructurar y dar formato al contenido de una página web.
- CSS [29] (*Cascading Style Sheets*): es el lenguaje utilizado para dar estilo y diseño a una página web, permitiendo la personalización de fuentes, colores, márgenes, tamaños y otros aspectos visuales.
- JavaScript [30]: es un lenguaje de programación que se utiliza para hacer que la página web sea interactiva y dinámica, permitiendo la manipulación de elementos del DOM [31] (*Document Object Model*), eventos, animaciones y otras acciones en el lado del cliente.
- Frameworks de JavaScript: son bibliotecas que permiten simplificar el desarrollo de aplicaciones web, proporcionando funcionalidades predefinidas y estructuras de organización.
- Bibliotecas de diseño: son herramientas que ofrecen componentes visuales predefinidos y estilos de diseño que permiten crear páginas web con un aspecto más profesional y elegante.
- Herramientas de gestión de paquetes: son programas que permiten gestionar las dependencias de los proyectos y mantener actualizadas las librerías utilizadas en el desarrollo.

### 2.5.1. Ionic framework

Ionic [32] es un framework de desarrollo de aplicaciones móviles híbridas basado en tecnologías web como HTML, CSS y JavaScript. Permite crear aplicaciones

## Capítulo 3

# Diseño e implementación

### 3.1. Arquitectura general del sistema

móviles para iOS, Android y la web utilizando un conjunto de herramientas y bibliotecas predefinidas.

Ofrece una gran cantidad de componentes visuales, animaciones y funcionalidades para crear aplicaciones móviles con una apariencia y experiencia de usuario nativa, similar a las aplicaciones desarrolladas con tecnologías nativas como Java para Android o Swift para iOS. Además, permite la integración con otras tecnologías como Angular [33] y React [34].

El uso de tecnologías web permite el desarrollo de aplicaciones móviles de forma más rápida y sencilla que las aplicaciones nativas, ya que se utiliza un único código base que se puede adaptar para cada plataforma. Además, ofrece una gran cantidad de herramientas y servicios para simplificar el proceso de desarrollo, como Ionic Native (para el acceso a las características nativas del dispositivo) y Ionic Appflow (para la implementación continua y la gestión de versiones).

### 2.6. Tecnologías de servidor

#### 2.6.1. Docker y Docker Compose

Docker [28] es una plataforma de software libre que se utiliza para desarrollar, implementar y ejecutar aplicaciones en contenedores. Los contenedores son una forma de virtualización que permiten a los desarrolladores empaquetar una aplicación y todas sus dependencias en una imagen de contenedor, que se puede ejecutar en cualquier entorno que tenga Docker instalado. Docker facilita la implementación de aplicaciones en diferentes plataformas, desde servidores locales hasta nubes públicas.

Docker Compose [35] es una herramienta de Docker que se utiliza para definir y ejecutar aplicaciones de múltiples contenedores. Permite definir todos los servicios necesarios para una aplicación en un archivo de configuración YAML, lo que facilita la implementación y el mantenimiento de la aplicación. Docker Compose puede iniciar todos los contenedores necesarios para la aplicación con un solo comando, lo que ahorra tiempo y reduce los errores.

#### 2.6.2. Servidor web Nginx

Un servidor web es un software que procesa solicitudes HTTP de clientes y responde con contenido estático o dinámico. Los servidores web se utilizan para alojar y servir sitios web y aplicaciones web. Un servidor web típicamente aloja varios sitios web y puede gestionar múltiples solicitudes HTTP simultáneamente.

Nginx [36] es un servidor web de código abierto que se utiliza para alojar y servir sitios web y aplicaciones web. Nginx es conocido por su alta escalabilidad, rendimiento y capacidad de manejar múltiples solicitudes HTTP simultáneamente. Es un servidor web ligero y rápido que se puede utilizar como un proxy inverso para distribuir la carga de trabajo a diferentes servidores y balancear la carga de tráfico.

## 2.7. Hardware utilizado

### 2.7.1. Raspberry Pi

La Raspberry Pi [37] es una pequeña computadora de placa única (*Simple Board Computer*, por sus siglas en inglés) diseñada para ser utilizada en proyectos de tecnología, educación y prototipos de hardware. Fue desarrollada por la Fundación Raspberry Pi, una organización benéfica con sede en el Reino Unido.

Principales características del modelo Raspberry Pi 4 seleccionado para este trabajo:

- Procesador Broadcom BCM2711 de cuatro núcleos ARM Cortex-A72 a 1,5 GHz.
- Procesador gráfico VideoCore VI con soporte para OpenGL ES 3.x.
- 8 GB de memoria RAM LPDDR4-3200.
- Bluetooth 5.0.
- Wi-Fi de doble banda 802.11ac.
- Gigabit Ethernet.

#### Puertos:

- Dos puertos micro-HDMI que pueden soportar dos pantallas con resolución de hasta 4K a 60 fps.
- Dos puertos USB 3.0.
- Dos puertos USB 2.0
- Un puerto GPIO de 40 pines
- Un puerto CSI para la conexión de una cámara
- Un puerto DSI para la conexión de una pantalla táctil
- Un puerto de audio de 3,5 mm.

La placa es compatible con diferentes sistemas operativos, incluyendo Raspberry Pi OS (anteriormente llamado Raspbian), Ubuntu, Windows 10 IoT Core y otros sistemas operativos basados en Linux. También es compatible con diferentes lenguajes de programación como Python, C++, Java y más, lo que la hace ideal para proyectos de IoT, robótica, automatización del hogar, entre otros.

### 2.7.2. NodeMCU Esp32

La NodeMCU ESP32 [38] es una placa de desarrollo basada en el microcontrolador ESP32, que es ampliamente utilizado en proyectos de Internet de las cosas (IoT).

Detalles técnicos del modelo NodeMCU ESP32 WROOM 32 seleccionado para este trabajo:

- Microcontrolador ESP32 de doble núcleo con una velocidad de reloj de hasta 240 MHz y 512 kB de memoria RAM.

## Capítulo 4

### Ensayos y resultados

#### 4.1. Pruebas funcionales del hardware

La idea de esta sección es explicar cómo se hicieron los ensayos, qué resultados se obtuvieron y analizarlos.

17

#### 2.7. Hardware utilizado

15

- 4 MB de memoria flash integrada para almacenamiento de programas y datos.
- Conectividad inalámbrica Wi-Fi 802.11 b/g/n y Bluetooth 4.2 BLE.
- Interfaz de programación USB integrada para programar la placa y proporcionar una conexión de depuración.
- GPIO de 30 pines para la conexión de sensores, actuadores y otros dispositivos periféricos.
- Interfaces I2C, SPI, UART, PWM y ADC integradas.
- Soporte para el entorno de programación Arduino, así como para MicroPython y Lua.
- Compatible con una amplia gama de bibliotecas y herramientas de desarrollo de código abierto.

#### 2.7.3. Identificación por radiofrecuencia

RFID [39] son las siglas en inglés de *Radio Frequency Identification* o identificación por radiofrecuencia en español. Es una tecnología de identificación automática que utiliza ondas de radio para leer y capturar información almacenada en etiquetas o *tags* RFID.

Esta tecnología consta de tres componentes básicos: el tag RFID que comúnmente es una tarjeta o llavero pequeño, el lector RFID y un sistema informático que gestiona la información capturada por el lector.

Los tags RFID contienen antenas y circuitos integrados que permiten la comunicación inalámbrica con los lectores, los cuales envían señales de radio para alimentar y activar los tags, y recibir la información almacenada en ellos. Pueden ser pasivos, activos o semi-activos, dependiendo de si necesitan o no una fuente de alimentación externa.

#### 2.7.4. Módulo RC522

El RFID RC522 [40] es un módulo de lectura-escritura RFID que utiliza la tecnología de comunicación de campo cercano (NFC) para la identificación y el intercambio de datos de manera inalámbrica. Entre sus principales características se encuentran:

- Soporte para frecuencias de operación de 13,56 MHz.
- Comunicación mediante el protocolo SPI (*Serial Peripheral Interface*).
- Capacidad para leer y escribir etiquetas RFID de tipo MIFARE, que son ampliamente utilizadas en sistemas de acceso, control de inventario, pago sin contacto, entre otros.
- Funciones de autenticación y encriptación para mayor seguridad en la transmisión de datos.
- Bajo consumo de energía y fácil integración con microcontroladores y otros sistemas embebidos.

### 2.7.5. Buzzer sonoro

El módulo buzzer pasivo KY006 [41] es un pequeño dispositivo electrónico que se utiliza para producir sonidos audibles en una variedad de proyectos electrónicos. Se conecta a la placa de control mediante tres pines.

Entre las características técnicas del módulo KY006 se encuentran:

- Tensión de funcionamiento: 5 V DC.
- Corriente de funcionamiento: <25 mA.
- Tipo de sonido: continuo y monótono.
- Frecuencia de resonancia:  $2300 \pm 300$  Hz.
- SPL (nivel de presión sonora): >85 dB a 10 cm de distancia.
- Diámetro: 30 mm.
- Altura: 7,5 mm.
- Peso: 4 gramos.

## Capítulo 5

### Conclusiones

#### 5.1. Conclusiones generales

La idea de esta sección es resaltar cuáles son los principales aportes del trabajo realizado y cómo se podría continuar. Debe ser especialmente breve y concisa. Es buena idea usar un listado para enumerar los logros obtenidos.

Algunas preguntas que pueden servir para completar este capítulo:

- ¿Cuál es el grado de cumplimiento de los requerimientos?
- ¿Cuán fielmente se puede seguir la planificación original (cronograma incluido)?
- ¿Se manifestó algunos de los riesgos identificados en la planificación? ¿Fue efectivo el plan de mitigación? ¿Se debió aplicar alguna otra acción no contemplada previamente?
- Si se debieron hacer modificaciones a lo planificado ¿Cuáles fueron las causas y los efectos?
- ¿Qué técnicas resultaron útiles para el desarrollo del proyecto y cuáles no tanto?

#### 5.2. Próximos pasos

Acá se indica cómo se podría continuar el trabajo más adelante.

## Capítulo 3

### Diseño e implementación

#### 3.1. Arquitectura general del sistema

##### 3.1.1. Funcionamiento

Como se puede observar en la figura 3.1, el sistema está compuesto por nodos, los cuales se utilizan para la lectura de tarjetas RFID asignadas a los repuestos de los clientes. Los datos se transmiten en una red local, se procesan y almacenan en un servidor con base de datos y son consultados desde la aplicación web en las terminales (PC o smartphone).

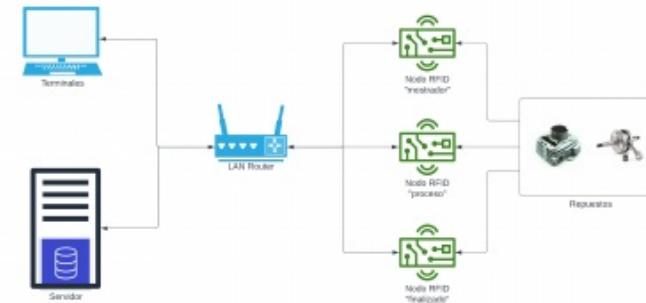


FIGURA 3.1. Diagrama de funciones generales.

##### 3.1.2. Diagrama de bloques

En la figura 3.2 se representa el patrón de modelo conceptual empleado y las tecnologías que se utilizan en cada capa. Se implementó un modelo de 4 capas:

- Capa de percepción.
- Capa de transporte.
- Capa de procesamiento.
- Capa de aplicación.



FIGURA 3.2. Diagrama de bloques y tecnologías del sistema.

En la capa de percepción se utilizan los nodos ESP32 con lector RFID con los cuales se realiza la lectura de las tarjetas correspondientes. Una vez realizada la lectura, los datos son enviados por medio del protocolo MQTT y a través de la red local en la capa de transporte.

La capa de transporte es la encargada de transmitir los mensajes por medio de los protocolos MQTT y HTTP. El broker Eclipse Mosquitto distribuye los mensajes publicados a los suscriptores para su procesamiento.

En la capa de procesamiento se realiza la lógica del *backend*, la base de datos y el *frontend*. Se implementó un servidor central en una Raspberry Pi 4 con todos los servicios. Cada uno de los servicios está desarrollado de manera individual y fue montado en su propio contenedor de Docker. Todos los contenedores se despliegan utilizando Docker Compose.

Por último, la capa de aplicación otorga el acceso web, desarrollado en el framework *Ionic*, para el ingreso, registro, administración y egreso de las órdenes de trabajo. También se utiliza el portal de administración de *Portainer* para el monitoreo completo de Docker y del servidor.

### 3.2. Flujo general del sistema

En esta sección se explica el flujo total de las funciones del sistema desde que el usuario ingresa una nueva orden de trabajo hasta que el producto es retirado por el cliente.

Las comunicaciones y el envío de datos entre los distintos módulos del sistema se realizan en los protocolos HTTP, MQTT y MySQL. Se detallará en cada caso el protocolo utilizado.

#### 3.2.1. Ingreso de repuesto

En la figura 3.3 se puede observar todos los módulos del sistema, el flujo de datos y protocolos que intervienen cuando un usuario carga una nueva orden de trabajo en el sistema.

## Bibliografía

- [1] Lautaro Regis. «Tornería». En: Esc. de Ed. Tec. Nuestra Señora de la Guarda (2018).
- [2] MecanicaFácil. *Rectificado del Bloque del Motor*. [http://www.mecanicafacil.info/Rectificado\\_del\\_Bloque\\_del\\_Motor.html](http://www.mecanicafacil.info/Rectificado_del_Bloque_del_Motor.html). Sep. de 2012. (Visitado 06-09-2022).
- [3] Gilda Liliana Ballivian Rosado. «Reparación del Motor». En: Instituto de Educación Superior Tecnológico Público (2016).
- [4] Robert Bosch. «Manual de la técnica automóvil». En: Editorial Reverte S.A., 1999, págs. 387-389.
- [5] William H. Crouse. *Mecánica de la Motocicleta*. Marcombo, 1992, pág. 321.
- [6] Ediciones Necochea. *Tapas de Cilindros*. Ediciones Necochea, 2021.
- [7] Dipole. ¿Qué es RFID? <https://www.dipolerfid.es/blog-rfid/que-es-rfid>. (Visitado 06-09-2022).
- [8] Enciclopedia EcuRed. *Sistemas radioeléctricos*. <https://www.ecured.cu/UHF>. (Visitado 06-09-2022).
- [9] Electrónica. Web oficial. <https://electronica.com/soluciones-rfid/>. (Visitado 06-09-2022).
- [10] Mozilla. *Generalidades del protocolo HTTP*. <https://developer.mozilla.org/es/docs/Web/HTTP/Overview>. (Visitado 06-09-2022).
- [11] Oscar Blancarte. *Arquitectura cliente-servidor*. <https://reactiveprogramming.io/blog/es/estilos-arquitectonicos/cliente-servidor>. (Visitado 06-09-2022).
- [12] Mozilla. *HTML: Lenguaje de etiquetas de hipertexto*. <https://developer.mozilla.org/es/docs/Web/HTML>. (Visitado 06-09-2022).
- [13] Cloudflare. ¿Qué es el modelo OSI? <https://www.cloudflare.com/es-es/learning/ddos/glossary/open-systems-interconnection-model-osi/>. (Visitado 06-09-2022).
- [14] Rfc. Specification of internet transmission control program. <https://www.rfc-editor.org/rfc/rfc675>. (Visitado 06-09-2022).
- [15] Paessler. ¿Qué es MQTT? <https://www.paessler.com/es/it-explained/mqtt>. (Visitado 06-09-2022).
- [16] Amazon. *Pub/Sub Messaging*. <https://aws.amazon.com/es/pub-sub-messaging/>. (Visitado 06-09-2022).
- [17] Eclipse. *Eclipse Mosquitto™ An open source MQTT broker*. <https://mosquitto.org/>. (Visitado 06-09-2022).
- [18] C. J. Date. *An Introduction to Database Systems*. 8th ed. Boston, MA: Addison-Wesley Professional, 2004.
- [19] Oracle Corporation. *MySQL :: MySQL Documentation*. <https://dev.mysql.com/doc/>. (Visitado 30-03-2023).
- [20] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Irvine, CA: University of California, Irvine, 2000.

### 3.2. Flujo general del sistema

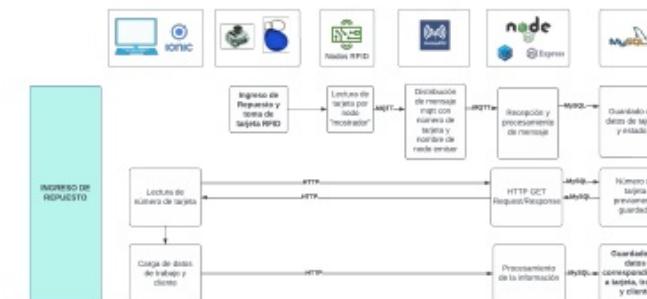


FIGURA 3.3. Flujo en el ingreso de una nueva orden de trabajo.

El flujo inicia cuando el usuario recibe el repuesto y selecciona una tarjeta RFID libre para usarla con ese repuesto. El usuario pasa la tarjeta por el nodo ubicado en la recepción, nodo *mostrador*, y de esta manera se registra el número de tarjeta para ser utilizada. Luego el usuario carga los datos en la aplicación web, donde ya está asignada la tarjeta previamente leída, confirma los datos y estos son guardados en la base de datos. La nueva orden de trabajo tiene su número de tarjeta, los datos del cliente y el estado por defecto *en espera*.

### 3.2.2. Cambio de estado

En la figura 3.4 podemos observar el flujo completo para un cambio de estado de una orden de trabajo.

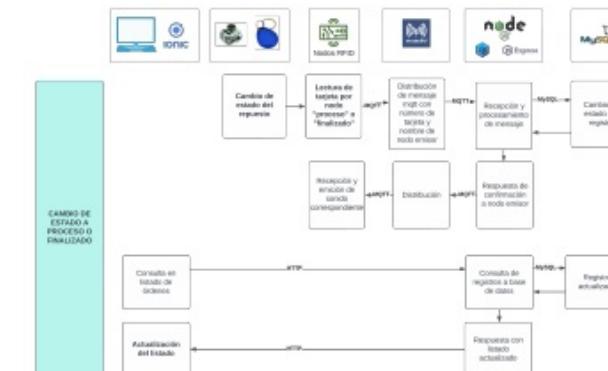


FIGURA 3.4. Flujo en el cambio de estado de una orden de trabajo.

El flujo inicia cuando un trabajador inicia o finaliza una orden, cambiando el estado de la misma a *en proceso* o *finalizada*. Para actualizar el estado de la orden debe pasar la tarjeta RFID asociada al repuesto por el nodo correspondiente, de

- URL: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm> (visitado 27-03-2023).
- [21] Node.js contributors. *Node.js Documentation*. Accessed: 2023-03-27. 2021. URL: <https://nodejs.org/docs/latest-v14.x/api/>.
- [22] Express contributors. *Express Documentation*. Accessed: 2023-03-27. 2021. URL: <https://expressjs.com/>.
- [23] Sequelize contributors. *Sequelize Documentation*. Accessed: 2023-03-27. 2021. URL: <https://sequelize.org/master/>.
- [24] W3C. *Cascading Style Sheets (CSS) specifications*. Accessed: 2023-03-27. 2018. URL: <https://www.w3.org/Style/CSS/>.
- [25] ECMA International. *ECMAScript® Language Specification*. Accessed: 2023-03-27. 2021. URL: <https://www.ecma-international.org/ecma-262/12.0/>.
- [26] World Wide Web Consortium. *Document Object Model (DOM) Specification*. <https://www.w3.org/TR/dom/>. (Visitado 06-09-2022).
- [27] Ionic contributors. *Ionic Framework Documentation*. Accessed: 2023-03-27. 2021. URL: <https://ionicframework.com/docs/>.
- [28] Angular contributors. *Angular Documentation*. Accessed: 2023-03-27. 2021. URL: <https://angular.io/docs>.
- [29] Facebook. *React Documentation*. Accessed: 2023-03-27. 2021. URL: <https://reactjs.org/docs/getting-started.html>.
- [30] Docker contributors. *Docker Documentation*. Accessed: 2023-03-27. 2021. URL: <https://docs.docker.com/>.
- [31] Docker contributors. *Docker Compose Documentation*. Accessed: 2023-03-27. 2021. URL: <https://docs.docker.com/compose/>.
- [32] Nginx, Inc. *Nginx Documentation*. <https://docs.nginx.com/>. (Visitado 27-03-2023).
- [33] Cámara Argentina de IoT. Documentos | Cámara Argentina de IoT. [https://iot.org.ar/es\\_ES/category/documentos/](https://iot.org.ar/es_ES/category/documentos/). (Visitado 05-04-2023).
- [34] Raspberry Pi Foundation. *Raspberry Pi - Teach, Learn, and Make with Raspberry Pi*. <https://www.raspberrypi.org/>. (Visitado 05-04-2023).
- [35] Espressif Systems. *ESP32 Datasheet*. 2021. URL: [https://www.espressif.com/sites/default/files/documentation/esp32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf) (visitado 12-10-2022).
- [36] *RFID Technology*. Impinj website. 2021. URL: <https://www.impinj.com/what-is-rfid>.
- [37] NXP Semiconductors. *MFRC522*. NXP Semiconductors. Eindhoven, The Netherlands, 2011. URL: <https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf>.
- [38] ArduinoModules. *KY-006 Passive Buzzer Module*. <https://arduinomodules.info/ky-006-passive-buzzer-module/>. Accessed: March 30, 2023. 2016.

esta manera se transmite la información por protocolo MQTT hasta la API de *backend*, esta última realiza el la actualización en la base de datos y responde al nodo por medio de MQTT informando si la operación tuvo éxito. El nodo emitirá un sonido que informará al usuario del resultado obtenido (tabla de sonidos en capítulo 3.5.4). Al mismo tiempo el usuario de la aplicación web verá reflejada la actualización del estado de la orden correspondiente en el listado de órdenes, ya que se está constantemente consultando por actualizaciones a la API por medio del protocolo HTTP utilizando el patrón observador de IONIC (detallado en el capítulo 3.7.2).

### 3.2.3. Retiro de repuesto

En la figura 3.5 podemos observar el flujo completo en el retiro de un repuesto.

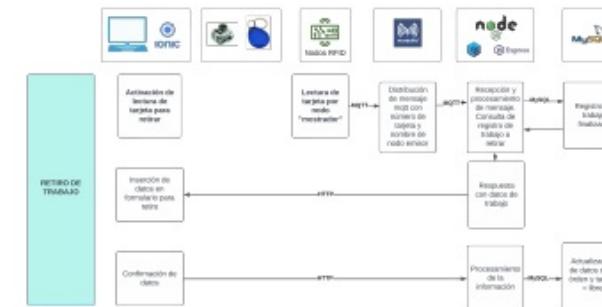


FIGURA 3.5. Flujo en el retiro de una orden de trabajo.

El flujo inicia cuando el usuario de la aplicación web realiza una activación de lectura de tarjeta en la pantalla de retiro. Esto ocasiona que la aplicación active un modo escucha hacia la API, esperando a que el usuario pase la tarjeta asociada al repuesto por el nodo mostrador. Cuando esto sucede la API recibe por MQTT el número de tarjeta y mediante este recupera los datos de la orden correspondiente en la base de datos, luego devuelve los datos obtenidos a la aplicación web por medio del protocolo HTTP. La aplicación inserta todos estos datos en el formulario de retiro y, cuando el usuario confirma la operación, se realiza una petición HTTP con el método PUT para actualizar los datos de los registros. La tarjeta utilizada queda en estado libre para ser reutilizada en una nueva orden de trabajo.

## 3.3. Arquitectura de datos

En la presente sección se desarrollará la arquitectura en la base de datos MySQL, el diseño y la implementación de las funciones con el ORM Sequelize.

### 3.3.1. Diagrama de base de datos

En la figura 3.6 se representa un diagrama UML de la base de datos con sus tablas y relaciones.