Índice general

v

Índice general

Capítulo 2

Introducción específica

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

2.1. Protocolos de comunicación

2.1.1. 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 vídeos.
- Es un protocolo de capa de aplicación del modelo OSI [13].
- 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.

2.1.2. 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.

5

Capítulo 2

Introducción específica

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

2.1. Protocolos de comunicación

2.1.1. 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 vídeos.
- Es un protocolo de capa de aplicación del modelo OSI [13].
- 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.

2.1.2. 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.

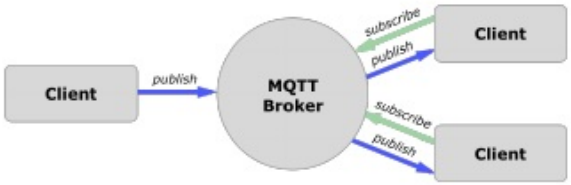

FIGURA 2.1. Arquitectura MQTT publish/subscribe.

La estructura de mensajes en este protocolo se dividen 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.

2.1.3. Ecliplse Mosquitto

Eclipse Mosquitto [17] es un broker MQTT OpenSource 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.

2.2. 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.

6 Capítulo 2. Introducción específica

- 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.

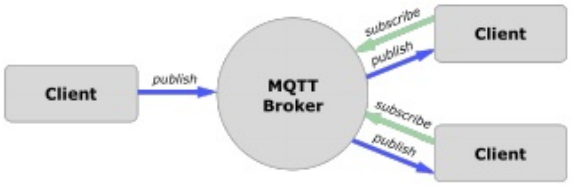

FIGURA 2.1. Arquitectura MQTT publish/subscribe.

La estructura de mensajes en este protocolo se dividen 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.

2.1.3. 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.

2.2. 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.

2.2.1. Bases de datos relacionales

Las bases de datos relacionales 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.

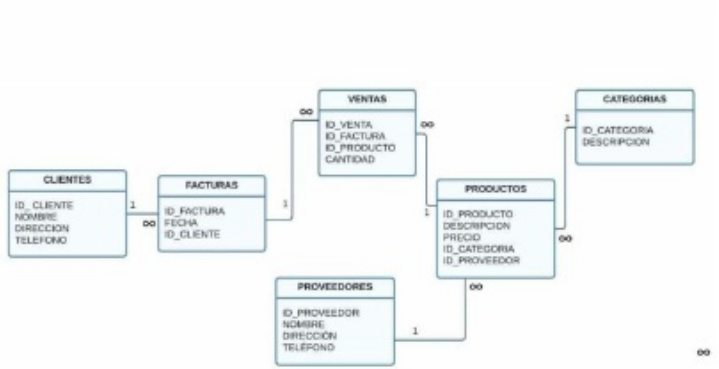

FIGURA 2.2. Diagrama base de datos relacional.

2.2.2. MySql

MySql **[WEBSITE:mysql-site]** 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.

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

2.2.1. 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.

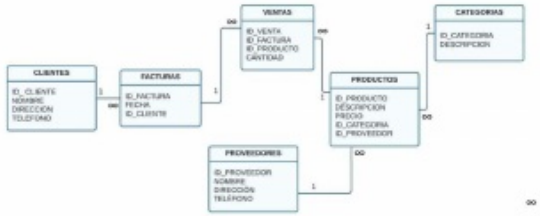

FIGURA 2.2. Diagrama base de datos relacional.

2.2.2. 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.

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.

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.

2.3. Tecnologías backend

Las tecnologías backend 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.

2.3.1. API RESTFul

Las APIs RESTful *(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.

FIGURA 2.3. Arquitectura API Rest.

2.3.2. Node.js

Node.js 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.

2.3. 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.

2.3.1. 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.

FIGURA 2.3. Arquitectura API Rest.

2.3.2. 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.

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

2.3.3. Express

Express 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 de las rutas 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.

2.3.4. Sequelize

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

2.4. 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 *(HyperText Markup Language)*: Es el lenguaje de marcado que se utiliza para estructurar y dar formato al contenido de una página web.

2.4. Tecnologías frontend 9

2.3.3. 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 de las rutas 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.

2.3.4. 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, 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.

2.4. 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

- CSS *(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: 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, eventos, animaciones y otras acciones en el lado del cliente.
- Frameworks de JavaScript: Son librerías 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.

2.4.1. Ionic framework

Ionic 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 librerías 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 y React.

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).

2.5. Tecnologías de servidor

2.5.1. Docker y Docker Compose

Docker 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.

de elementos del DOM [26] *(Document Object Model)*, eventos, animaciones y otras acciones en el lado del cliente.

- Frameworks de JavaScript: son librerías 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.

2.4.1. 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 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).

2.5. Tecnologías de servidor

2.5.1. 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.

Docker Compose 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.

2.5.2. 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 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.

2.5.2. 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.

2.6. Hardware y IoT

2.6.1. Internet of Things

Internet of Things [33] (IoT) 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 que interactuamos 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.

2.6.2. Raspberry Pi

La Raspberry Pi [34] 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.

2.6.3. 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 WiFi 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.

Capítulo 3

Diseño e implementación

3.1. Análisis del software

La idea de esta sección es resaltar los problemas encontrados, los criterios utilizados y la justificación de las decisiones que se hayan tomado.

Se puede agregar código o pseudocódigo dentro de un entorno lstlisting con el siguiente código:

```
\begin{lstlisting}[caption= "un epígrafe descriptivo"]
las líneas de código irían aquí...
\end{lstlisting}
```

A modo de ejemplo:

```
#define MAX_SENSOR_NUMBER 3
#define MAX_ALARM_NUMBER  6
#define MAX_ACTUATOR_NUMBER 6

uint32_t sensorValue[MAX_SENSOR_NUMBER];
FunctionalState alarmControl[MAX_ALARM_NUMBER]; //ENABLE or DISABLE
state_t alarmState[MAX_ALARM_NUMBER];           //ON or OFF
state_t actuatorState[MAX_ACTUATOR_NUMBER];     //ON or OFF

void vControl() {

  initGlobalVariables();

  period = 500 ms;

  while(1) {

    ticks = xTaskGetTickCount();

    updateSensors();

    updateAlarms();

    controlActuators();

    vTaskDelayUntil(&ticks, period);
  }
}
```

CÓDIGO 3.1. Pseudocódigo del lazo principal de control.

2.6.4. RFID

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 comunmente 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.

2.6.5. Módulo RC522

La 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.

2.6.6. 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:

- Voltaje de funcionamiento: 5V DC.
- Corriente de funcionamiento: <25 mA.
- Tipo de sonido: Continuo y monótono.
- Frecuencia de resonancia: 2300 ± 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 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.

Capítulo 3

Diseño e implementación

3.1. Análisis del software

La idea de esta sección es resaltar los problemas encontrados, los criterios utilizados y la justificación de las decisiones que se hayan tomado.

Se puede agregar código o pseudocódigo dentro de un entorno lstlisting con el siguiente código:

```
\begin{lstlisting}[caption= "un epigrafe descriptivo"]
las lineas de código irían aquí...
\end{lstlisting}
```

A modo de ejemplo:

```
#define MAX_SENSOR_NUMBER 3
#define MAX_ALARM_NUMBER  6
#define MAX_ACTUATOR_NUMBER 6

uint32_t sensorValue[MAX_SENSOR_NUMBER];
FunctionalState alarmControl[MAX_ALARM_NUMBER]; //ENABLE or DISABLE
state_t alarmState[MAX_ALARM_NUMBER];           //ON or OFF
state_t actuatorState[MAX_ACTUATOR_NUMBER];     //ON or OFF

void vControl() {

  initGlobalVariables();

  period = 500 ms;

  while(1) {

    ticks = xTaskGetTickCount();

    updateSensors();

    updateAlarms();

    controlActuators();

    vTaskDelayUntil(&ticks, period);
  }
}
```

CÓDIGO 3.1. Pseudocódigo del lazo principal de control.

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 puedo 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 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.

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. 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] Telectrónica. *Web oficial*. https://telectronica.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).

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 puedo 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.