CARRERA DE ESPECIALIZACIÓN EN INTERNET DE LAS COSAS

MEMORIA DEL TRABAJO FINAL

Desarrollo de Aplicación Hospitalaria con MQTT

Autor:
Ing. Gustavo Adrián Bastian

Director:
Mg.Ing. Ericson Joseph Estupiñan Pineda (Surix S.R.L)

Jurados:
Sergio Burgos (Universidad Tecnológica Nacional, Facultad Regional Paraná)
Daniel Iván Cruz Flores 2 (pertenencia)
Luis Mariano Campos (Conicet)

Este trabajo fue realizado en la Ciudad de Santo Tomé, Santa Fé, entre Octubre de 2019 y diciembre de 2022.

Resumen

La presente memoria describe el desarrollo de un sistema para ser utilizado por enfermeros y médicos en el ámbito hospitalario implementado con un protocolo de bajo consumo de recursos. El trabajo se realizó para satisfacer una necesidad de la empresa Surix SRL.

La arquitectura del sistema está compuesta por cinco elementos principales que fueron abordados durante la carrera: un broker que gestiona los mensajes, una base de datos con información de los pacientes, un programa que genera acciones basado en distintos eventos, una aplicación web que permite la gestión de la base de datos y una aplicación móvil.

Agradecimientos

En primer lugar agradezco al director de este trabajo final, Mg. Ing. Ericson Estupiñan por la guía y los consejos brindados durante este tiempo.

Así mismo, agradezco a Sergio Starkoff por proporcionar la idea originaria del trabajo y por la confianza ofrecida desde un primer momento.

No puedo olvidar agradecer a los docentes, por el incalculable valor de realizar muy buenos contenidos para las clases y por la excelente predisposición para responder mis dudas.

Agradezco también a mis compañeros/as por demostrar mucho interés en las asignaturas y participar activamente en las clases.

Especialmente agradezco a mi familia, que comprendieron el tiempo que debía dedicarle al estudio, brindándome apoyo moral y humano. Y a Lucila y Alejo, quienes me brindaron mucho amor y ternura.

Índice general

Índice de figuras

Índice de tablas

Dedicado a todas las personas que participaron en mi formación humana.

Capítulo 1

Introducción general

En este capítulo se presentan las necesidades de los sistemas hospitalarios junto con nociones sobre el Internet de las cosas y el protocolo MQTT. Además se mencionan las motivaciones, el estado de arte y el alcance del trabajo.

1.1. Descripción de los sistemas de Internet de las cosas y su aplicación para gestionar actividades hospitalarias

En la actualidad, el avance de la Internet de las Cosas (IoT de *Internet of Things*) y la disminución de costos asociados a la tecnología hacen factible su incorporación a distintos contextos de la vida cotidiana. Un campo de mucho interés es el de infraestructuras hospitalarias inteligentes[1].

El término Internet de las Cosas fue utilizado por primera vez en 1990 para describir un sistema cuyos componentes del mundo físico se conectaban a la internet mediante sensores. En la actualidad se utiliza el término para referirse a escenarios donde la conectividad en red y la capacidad de computo se extiende a objetos, sensores y elementos cotidianos no considerados computadoras, permitiendoles generar, intercambiar y consumir datos con un mínimo de intervención humana.[2].

Si consideramos a las diferentes personas como un elemento más de un sistema, el ámbito hospitalario cuenta con numerosas entidades que interactuan entre sí como ser: médicos, pacientes, enfermeros, personal de limpieza, personal de seguridad, personal administrativo, dispositivos de iluminación, dispositivos sonoros, herramientas médicas, puertas, ventanas, ventiladores, acondicionadores de aire, termómetros, etc. Un entorno de estas cualidades es ideal para gestionar con IoT, ya que permite mejorar la calidad del servicio de salud[1].

Un sistema de estas características posee un modelo de capas, presentado en [3] que separa los conocimientos en cinco categorías con responsabilidades bien definidas: negocio, aplicación, procesamiento, red y percepción. La tabla 1.1 presenta las funciones reducidas de cada modelo:

TABLA 1.1. Modelo de capas sistema IoT

Capa	Función
Negocio	Establecer reglas y controlar sistema
Aplicación	Interactuar con el usuario
Procesamiento	Almacenar y analizar los datos obtenidos
Red	Transportar datos entre dispositivos
Percepción	Realizar mediciones o acciones

Para mayor información sobre la funcionalidad de cada capa consultar [3].

1.2. Motivación

En Argentina muchos hospitales están retrasados en su progreso tecnológico. Por dicha razón, todos los avances en este campo son necesarios. Por lo explicado anteriormente, el gestionar la institución con un sistema de IoT es de suma utilidad.

Surix S.R.L fabrica un sistema IP de llamado a enfermera que está basado en el protocolo SIP. Este consiste en un servidor central y terminales que se encuentran en las habitaciones del hospital. La aplicación principal se ejecuta en una pc o bien en una tablet y monitorea el estado de las habitaciones. La principal motivación para migrar el protocolo radica en el alto costo de hardware que genera el agregar dispositivos a su sistema actual. En este contexto, se encargó la realización de este trabajo.

Dentro de las múltiples opciones para realizar la comunicación entre los dispositivos IoT, Surix seleccionó el protocolo *Message Queuing Telemetry Transport* (en adelante MQTT) ya que se ha probado como un protocolo confiable, altamente eficiente y ampliamente utilizado [4].

MQTT es un protocolo open source simple, liviano y orientado a dispositivos con pocos recursos y baja velocidad de transmisión. Está basado en la pila TCP/IP(del ingles *Transmission Control Protocol/Internet Protocol*, Protocolo de control de transmisión /Protocolo de Internet), se implementa en la capa de aplicación y utiliza mensajeria bajo el patrón de publicación/subscripción. Las ventajas que provee es la posibilidad de agregar accesorios rápidamente con bajo costo de software, hardware e implementación[4].

En la referencia [4] se realiza un estudio en profundidad del desempeño de varios protocolos de IoT: MQTT, CoAP(*Constrained Application Protocol, Protocolo de Aplicaciones Restringido*), AMQP(*Advanced Message Queuing Protocol*, Protocolo de Encolamiento de Mensajes Avanzado) y HTTP(*Hypertext Transfer Protocol*, Protocolo de Transferencia de Hipertexto). Se menciona que MQTT es ampliamente utilizado pero muy poco estandarizado en comparación con los demas, y se resalta el hecho que tiene limitaciones en lo referido a seguridad.

Cabe resaltar que MQTT no es el único protocolo de la capa de aplicación utilizado en este trabajo, ya que la página de configuración/administración del sistema utiliza HTTP. MQTT es utilizado en la aplicación móvil como única forma de comunicarse con el sistema.

1.3. Estado de arte

En el mercado internacional se encontró un producto similar desarrollado por la firma TigerConnect(antes llamada TigerText), *TigerConnect Clinical Collaboration Platform*(Plataforma de Colaboración Clínica TigerConnect)[5], cuyas principales características detallo a continuación:

- Aplicación de Mensajería para celulares y estaciones de trabajo.
- Solución en la nube asegurando disponibilidad en un 99.99 %.
- Mensajería por texto asegurada con encriptación.
- Homologado por HIPPA(del ingles, *Health Insurance Portability and Accountability Act*,ley de Portabilidad y Responsabilidad del Seguro Médico)[6].
- Certificado HITRUST(es un framework para gestionar riesgos utilizado por muchas redes de salud y hospitales[7]).
- Control administrativo total, permitiendo a los administradores gestionar usuarios, configuraciones y políticas de seguridad por medio de una consola. Los usuarios pueden ser cargados utilizando plantillas csv y, en caso de robo o extravío de dispositivo, prohibir el acceso.
- Posibilidad de incorporar mensajería de voz como servicio extra.
- Mensajes a grupos de personas.

Entre las diferencias que posee la solución presentada en este trabajo con respecto a la disponible en el mercado, el hecho de utilizar MQTT como protocolo base permite incorporar con muy poco esfuerzo dispositivos IoT de mediciones paramétricas de los pacientes ya que la estructura de la red asi lo permite. Por otra parte, la solución desarrollada presenta de base la posibilidad de transmisión de audio.

TigerConnect es una firma nacida en 2010 con 281 empleados cuya principal producto es este sistema[8].

1.4. Objetivos y alcance

El sistema resultante de este trabajo está orientado a gestionar las relaciones entre pacientes, enfermeras y médicos solamente. Un diagrama reducido puede observarse en la figura 1.1.

Los componentes del sistema son:

- broker MQTT: permite gestionar los mensajes entre los elementos del sistema.
- base de datos: donde alojar información de reportes de habitaciones, personal y datos relevantes al paciente
- página web para configuración: permite gestionar usuarios, camas, pacientes, observar estado del sistema

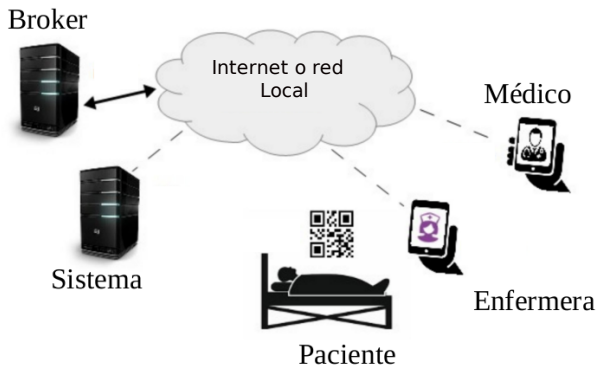

FIGURA 1.1. Figura del sistema.

- aplicación móvil multiplataforma que permite a los usuarios enfermeros interactuar con el sistema y con los médicos. La aplicación es capaz de identificar la cama correspondiente (mediante lectura de sımbolos QR) y de transmitir mensajes de voz en caso de ser necesario.

El alcance del trabajo consiste en:

- Confección de un plan de trabajo.
- Selección y configuración de un broker MQTT.
- Desarrollo local de una página web de configuración, que permite asignar medicos a pacientes, asignar tareas programadas a pacientes, asignar códigos QR a camas, asignar tratamientos a pacientes, asignar especialidades a los usuarios enfermeros.
- Desarrollo local de una aplicación móvil con 3 modos de funcionamiento: modo médico, con envío/recepción de audio/texto/alarmas;modo enfermera con envío/recepción de audio/texto/alarmas y escaneo de QR para identificar paciente; y modo sistema que permite el monitoreo de habitaciones.
- Código fuente/documentación de las aplicaciones realizadas.

El presente trabajo no incluye:

- Manuales de las distintas aplicaciones.
- Traducciones a distintos idiomas de las aplicaciones.
- Sistema llamador.
- Análisis de tráfico en la red.
- Análisis de seguridad(no se certifica).
- Contratación de base de dato remota.
- Contratación e instalación de servidores remotos.

Capítulo 2

Introducción específica

En este capítulo se presentan las bases teóricas que sustentan el trabajo realizado. Se describen distintas soluciones a cada una de las partes del sistemas.

2.1. Descripción del protocolo MQTT

MQTT fue introducido en 1999 por IBM y Eurotech. Es un protocolo de mensajería por suscripción/publicación especialmente diseñado para la comunicación M2M(*Machine to Machine*, Máquina a Máquina) en dispositivos con bajos recursos [9] .

Para poder explicar el funcionamiento del protocolo es necesario incorporar conceptos definidos en la especificación [9]:

- Mensaje: datos transmitidos mediante el protocolo MQTT a través de la red por la aplicación. Cuando un mensaje es transmitido, el mismo contiene datos útiles(*"payload"*), Calidad de Servicio, ciertas propiedades y un nombre de tópico.
- Cliente: programa o dispositivo que utiliza MQTT.
- Servidor o Broker: programa o dispositivo que actúa como intermediario entre Clientes que publican Mensajes y Clientes que han realizado suscripciones.
- Sesión: una interacción entre Cliente y Servidor con estados bien definidos.
- suscripción: una suscripción implica un filtrado de tópicos y una máxima Calidad de Servicio. Una suscripción está asociada a una sola sesión y una sesión puede poseer varias suscripciones.
- Nombre de Tópico: etiqueta que se adjunta a un mensaje la cual es comparada con las suscripciones conocidas en el Servidor.
- Filtro de Tópico: es una expresión contenida dentro de la suscripción para indicar interés, por parte del Cliente, en uno o más Tópicos.
- suscripción con Wildcard: es una expresión contenida dentro de la suscripción que contiene un carácter especial o comodín, como ser '+' o '#', que representan un nivel único y nivel múltiple respectivamente.

En este protocolo, el funcionamiento es el siguiente:

Al iniciar la conexión, el Cliente envía un mensaje CONNECT al Broker y este contesta con un CONNECTACK y un código de estado. El Broker mantiene

abierta la conexión hasta que el cliente envía un comando de desconexión o la conexión se pierde por algún motivo. Los puertos estándar son 1883 para comunicación no encriptada y 8883 para comunicación utilizando TLS/SSL [3].

Luego Cliente MQTT publica mensajes en un Broker MQTT, los cuales son recibidos por los Clientes suscriptos o bien retenidos para una futura suscripción [9]. En la figura 2.1 se presentan todos los componentes de una red MQTT.

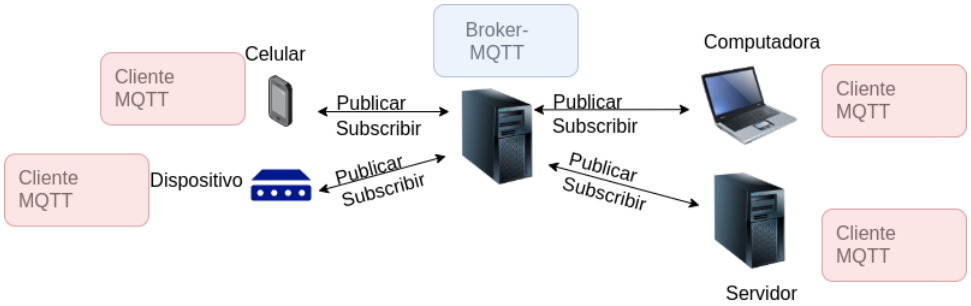

FIGURA 2.1. Esquema de un sistema MQTT.

La conexión MQTT siempre es entre Cliente y Broker, nunca entre Clientes directamente.

Cada mensaje es publicado en una dirección conocida como "tópico". Los clientes pueden subscribirse a múltiples tópicos y recibir todos los mensajes publicados en cada tópico. MQTT es un protocolo binario y normalmente requiere un encabezado fijo de 2-bytes y una dato útil (*"payload"*) de hasta 256 MB[4] .

Ejemplos de tópicos:

- 1: /user/1/question
- 2: /user/1/#
- 3: /user/+/question

En el ejemplo 1, en caso de suscribirse a ese Tópico, el cliente recibirá todos los mensajes publicados en el subtópico "question". En el ejemplo 2, el cliente recibirá todos los mensajes dirigidos al usuario 1. En el ejemplo 3, el cliente que se suscriba a ese Tópico recibirá los mensajes enviados a "questionïndependientemente del número de usuario.

Utiliza usualmente TCP/IP como protocolo de transporte y puede implementar TLS/SLL para seguridad. Por lo que el servicio cliente-broker es del tipo orientado a la conexión. También puede utilizar otros protocolos de red con soporte bidireccional y sin pérdidas de datos[3][9] .

La carga útil del mensaje debe ser codificada en UTF-8(*8-bit Unicode Tranformation Format*, Formato Unicode de Codificación de 8-bits)[9].

MQTT posee tres niveles de calidad de servicio("QoS", del ingles *Quality of Service*) que permite seleccionar la fiabilidad de la entrega de los mensajes[9] .

2.1.1. Uso de WebSockets en MQTT

Actualmente, no es posible utilizar MQTT natural en un browser debido a la imposibilidad de abrir una conexión TCP pura.

Una solución a este problema puede ser transmitir el mensaje MQTT sobre una conexión WebSocket lo que permite que un browser pueda utilizar todas las características del protocolo directamente. Esto es muy útil en el caso de las aplicaciones de telefono móvil[10].

WebSocket es un protocolo de red que provee comunicación bidireccional entre un browser y un servidor web. El protocolo fue estandarizado en 2011 y es soportado por todos los browser modernos. Como MQTT, el protocolo WebSocket está basado en TCP. En la figura 2.2 se muestra conceptualmente como se encapsula la información del protocolo MQTT dentro de una trama WebSocket.

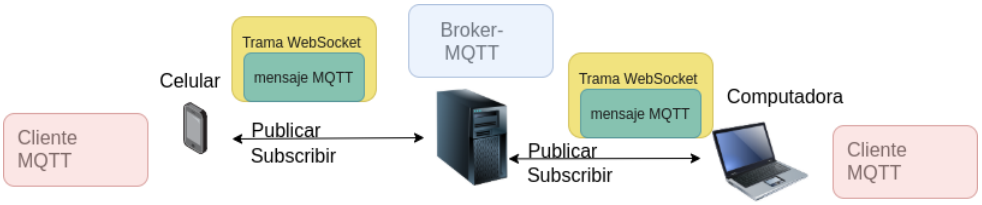

FIGURA 2.2. Uso de WebSockets con MQTT.

En MQTT sobre WebSockets, el mensaje MQTT se transfiere a través de la red y es encapsulado por una o más tramas WebSockets. La ventaja de este método de transmisión es que provee una comunicación bi-direcional, ordenada y sin pérdidas[10] .

Para poder utilizar MQTT sobre WebSockets el Broker debe ser capaz de manejar WebSocket nativos.

Para mejorar la seguridad, se puede implementar TLS utilizando WebSockets seguros para encriptar toda la conexión.

2.1.2. Broker MOSQUITTO

Existen muchos brokers MQTT disponibles. Para este trabajo se seleccionó Eclipse Mosquitto como Broker ya que es de código abierto con licencia EPL/EDL que implementa MQTT en sus versiones 5.0, 3.1.1 y 3.1.[11].

Otra característica que lo hace muy conveniente es que se encuentra disponible en los repositorios de varias distribuciones de Linux como ser Ubuntu, Debian, etc, por lo que su instalación se simplifica.

En mosquitto, el puerto que usualmente se utiliza para WebSockets es el 9001.

2.2. Componentes del sistema

El sistema desarrollado se lo puede descomponer en seis partes definidas:

- Broker MQTT: servidor encargado de gestionar los mensajes
- Base de datos: almacena información relevante de los pacientes, usuarios y sistema.
- Servidor de Base de datos: proporciona una API (*Application Programming Interface*, interfaz de programación de aplicaciones) del tipo REST (*representational state transfer*, transferencia de estado representacional) que permite la modificación/consulta de la base de datos desde un dispositivo web. La API provee *endpoints*(puntos de acceso en español) que permiten realizar operaciones.
- Servidor web/ página web: entrega información a un browser para que pueda mostrar una página web de administración del sistema y de carga de la base de datos.
- Sistema: Recibe los mensajes MQTT y responde/actúa acorde a las peticiones.
- Cliente browser: muestra en la pantalla de una computadora o dispositivo la página web de administración.
- Cliente teléfono: permite a médicos y enfermeras interactuar con el sistema.

Como se puede observar, se trata de un sistema distribuido y sus partes pueden estar ubicadas en distintas localidades.

Con el objetivo de facilitar el desarrollo se utilizó un sistema de contenedores que permite correr tres de las 6 partes: la base de datos, el servidor de base de datos y el sistema propiamente dicho. Además se agrupó los subsistemas que modifican la base de datos en una sola aplicación que provee ambos servicios: el sistema y el servidor REST. En la figura 2.3 se presenta un esquema general según esta división.

2.3. Sistemas de contenedores Docker

Desde los origenes de la programación, existió una necesidad de abstraer la aplicación desarrollada de la plataforma en la que se ejecuta. Multiples avances se fueron dando desde ese entonces: sistemas operativos con interfaces de aplicaciones estandar, librerías portables y lenguajes interpretados en máquinas virtuales son algunos de ellos.

Docker es un proyecto de código abierto que permite automatizar el despliegue de aplicaciones dentro de contenedores de software proporcionando una capa de abstracción y automatización. Fue desarrollado por Docker Inc y su primera versión fue liberada en 2013. Está escrito en el lenguaje Golang y actualmente es un estandar en la industria del Software [12].

Un contenedor de software es un componente aislado que se ejecuta como un proceso mas dentro del sistema operativo del host y posee todas las dependencias y requerimientos que la aplicación necesita para funcionar correctamente. Por cada contenedor en ejecución hay un proceso ejecutandose en el sistema[12].

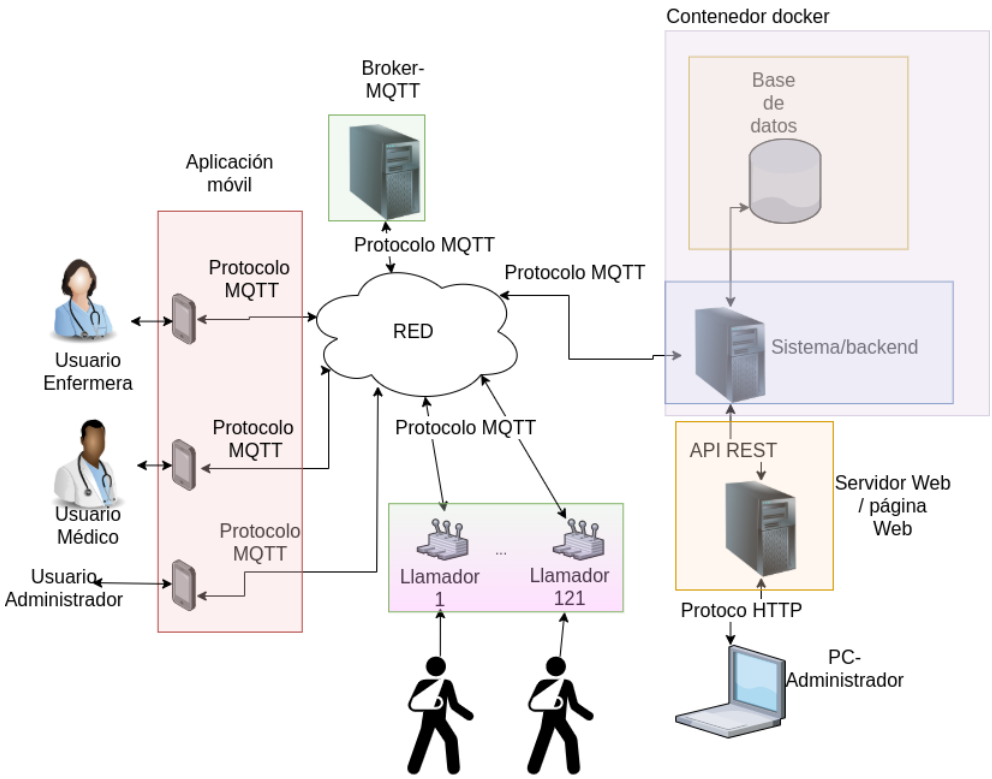

FIGURA 2.3. División del sistema.

Resumo brevemente las partes del ecosistema Docker, para información más detallada consultar[12] o bien [13].

- *Images* : son plantillas que describen el contenedor. Se componen de una base y sobre estas se modifica según necesidad.
- *Containers* : son instancias de ejecución de una imagen. Se pueden iniciar, detener, mover o eliminar mediante comandos. Se puede conectar un contenedor a una o mas redes.
- *Networks* : Docker utiliza una interfaz virtual para comunicarse con la red del equipo host.
- *Volumes* : permiten generar espacios de almacenamiento dentro de cada contenedor, el funcionamiento es similar a las carpetas compartidas en las máquinas virtuales.
- *Registry* : un registro almacena imágenes de docker. Dockerhub es un registro público que cualquiera puede utilizar.

Para poder correr un contenedor docker es necesario ejecutar el comando:

CÓDIGO 2.1. Ejecución de contenedor

```
>docker-run --[parametro1] --[parametro2] ...
nombre_contendor:version
```

2.3.1. Uso de Docker-compose

Docker Compose es una herramienta del ecosistema Docker que sirve para definir aplicaciones multicontenedor en un archivo de texto llamado "docker-compose.yml".

Con esta herramienta se facilita el paso de parámetros al contenedor y posibilita generar imágenes complejas utilizando e interconectando distintos contenedores.

Como ejemplo, en [14] se presenta un archivo para ejecutar un contenedor *Wordpress* junto con una base de datos MySQL.

2.4. Introducción a las bases de datos

Que es un dato? Es un elemento o característica que permite hacer la descripción de un objeto[15].

Una base de datos es una colección de datos (o información) organizados, estructurados y almacenados electrónicamente en un sistema de computadoras. En su mayoría son controladas por un DBMS (*database management system*, sistema gestor de bases de datos). Los datos, el DBMS y la aplicación que está asociada a ellos, conforman el llamado sistema de base de datos, o base de datos en forma abreviada.[14]

Durante los años 80 se popularizó el tipo de base de dato relacional, cuya información se organizaba como un conjunto de tablas con columnas y filas. Este tipo de base de datos provee la forma más eficiente y flexible de acceder a información estructurada.

En 1970, IBM (junto con contribuciones de Oracle) desarrollaron un lenguaje de programación llamado SQL (*Structured Query Language*, Lenguaje de Colas Estructurado). Su principal objetivo era manipular, encolar, definir datos y proveer control de acceso en las bases de datos relacionales. Este lenguaje es apliamente utilizado hoy en día, aún cuando surgen nuevos.

Se podría empezar a clasificar las bases de datos en relacionales y no relacionales. Las primeras utilizan SQL como su lenguaje de programación y entre ellas podemos encontrar: PostgreSQL, MySQL y SQL Server. Las segundas no utilizan un lenguaje SQL, ya que no trabajan en base a estructuras definidas, poseen una gran escalabilidad y estan diseñadas para manejar grandes volumentes de datos[15].

2.4.1. Descripción de MySQL

En este trabajo se seleccionó el DBMS MySQL. Es un sistema de gestión de bases de datos con mas de 10 millones de instalaciones. Fue desarrollado a mediados de la decada de los 90, y es de uso libre. Es potente, rápido y altamente escalable[16].

Existen tres formas de interactuar con una base de datos MySQL: utilizando la consola, a través de una interfaz web como phpMyAdmin y mediante un lenguaje de programación.

En la figura 2.4 se puede observar la interfaz:

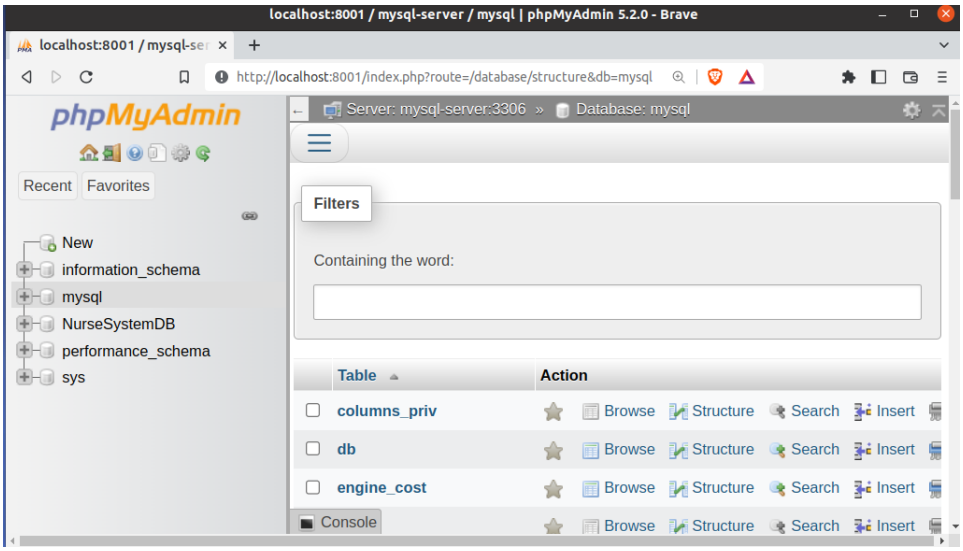

FIGURA 2.4. Página de gestión phpMyAdmin.

Las sentencias SQL finalizan siempre con el símbolo ';'.

Los comandos más utilizados en SQL se observan en la tabla 2.1[16]:

TABLA 2.1. Comandos básicos SQL

Comando	Función
ALTER	Modificar una base de datos o una tabla.
BACKUP	Copia de seguridad de una tabla.
\c	Cancelar la entrada.
CREATE	Crear la base de datos.
DELETE	Borrar una fila de una tabla.
DESCRIBE	Describir columnas de una tabla.
DROP	Eliminar una base de datos o una tabla.
Exit	Salir.
GRANT	Cambiar permisos de un usuario.
INSERT INTO..VALUES	Insertar datos.
RENAME	Cambiar nombre de una tabla.
USE	Usar una base de datos.
UPDATE	Actualizar un registro existente.
SELECT	Consultar una fila
JOIN... ON	Consultar multiples tablas

Ejemplo de uso, suponiendo 2 tablas:

TABLA 2.2. TABLA1 para el ejemplo 1

PacienteId	Nombre
1	Pedro
2	Juan

TABLA 2.3. TABLA2 para el ejemplo 1

PacienteId	Apellido
1	Perez
2	Salvador

Si se quiere obtener el nombre y apellido del paciente con pacienteId igual a 1, una forma de obternerlo es ejecutando el código:

CÓDIGO 2.2. Ejecución de comando sql

```
>SELECT Nombre,Apellido FROM TABLA1 as TB1 \
JOIN TABLA2 as TB2 ON TB1.pacienteId=TB2.pacienteId \
WHERE TB1.pacienteId=1;
>Pedro Perez
```

Uno de las multiples ventajas que poseen los motores de bases de datos relacionales es la posibilidad de realizar transacciones. Las mismas consisten en secuencias de operaciones que se ejecutan en orden y se completan con éxito.

CÓDIGO 2.3. Secuencia de transacción sql

```
>BEGIN;
>SELECT Nombre FROM TABLA1;
>SELECT Apellido FROM TABLA2;
>COMMIT;
>Pedro
>Juan
>Perez
>Salvador
```

Existe muchas referencias sobre programación con MySQL, para un estudio mas profundo se puede utilizar tutoriales como [17].

2.5. Frameworks/librerías para desarrollo web/móvil

En esta sección se presentarán varias librerías que se analizaron y se utilizarán para el desarrollo.

2.5.1. Librerías Node.js, Express y JWT

2.5.2. Librería Eclipse Paho

2.5.3. Framework Ionic/Angular

Capítulo 3

Diseño e implementación

3.1. Generación del entorno base para el desarrollo del sistema de backend

3.2. Broker Mosquitto

3.3. Base de datos del sistema

3.4. Sistema de gestión

3.4.1. Descripción de API REST para aplicación Web

3.4.2. Descripción de API MQTT para la mensajeria de la aplicación móvil

3.5. Página Web(frontend)

3.5.1. Estructura y organización del software

3.5.2. Acceso de usuario

3.5.3. Monitoreo del sistema

3.5.4. Gestión de tareas programadas

3.5.5. Estadísticas del sistema

3.6. Aplicación Móvil

3.6.1. Estructura y organización del software

3.6.2. Configuración del broker y acceso de usuario

3.6.3. Modo administrador

3.6.4. Modo médico

3.6.5. Modo enfermera

3.6.6. Interacción médico-enfermera

3.7. Contraste con los requerimientos

Capítulo 4

Ensayos y resultados

4.1. Generación del sistema de pruebas

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

4.2. Generación/instalación de la aplicación móvil

4.3. Equipo simulador de llamadores

4.4. Resultados de utilizar el sistema

Capítulo 5

Conclusiones

5.1. Notas sobre el sistema desarrollado

5.2. Trabajo futuro

Bibliografía

[1] Sankeerthana Neelam. «Internet of Things in Healthcare». En: (2017).

[2] Lyman Chapin Karen Rose Scott Eldridge. «The Internet of Things: an Overview». En: (2015).

[3] Carlos Ruben Domenje. «Sistema de gestión remota de dispositivo conversor Modbus a MQTT». En: (2021).

[4] Vasil Sarafov. «Comparison of IoT Data Protocol Overhead». En: (2018).

[5] TigerConnect. *Secure Healthcare Communication and Collaboration*. https://tigerconnect.com/products/clinical-collaboration/. Dic. de 2021. (Visitado 25-10-2022).

[6] Juliana De Groot. *What is HIPAA Compliance?* https://digitalguardian.com/blog/what-hipaa-compliance. Abr. de 2022. (Visitado 25-10-2022).

[7] MATT HALBLEIB. *What is HITRUST Compliance?* https://www.securitymetrics.com/blog/what-hitrust-compliance. Dic. de 2021. (Visitado 25-10-2022).

[8] Rocket Reach. *Tigerconnect Information*. https://rocketreach.co/tigerconnect-profile_b4548a1cfc9d885d. Abr. de 2022. (Visitado 25-10-2022).

[9] OASIS. *MQTT Version 5.0. Edited by Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. 07 March 2019. OASIS Standard. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html*. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. Mar. de 2019. (Visitado 02-07-2022).

[10] HiveMQ. *MQTT over WebSockets - MQTT Essentials Special*. https://www.hivemq.com/blog/mqtt-essentials-special-mqtt-over-websockets/. Abr. de 2015. (Visitado 02-04-2022).

[11] Eclipse Foundation. *Eclipse Mosquitto An open source MQTT broker*. https://mosquitto.org/. Ene. de 2022. (Visitado 15-03-2022).

[12] Agustin Bassi. *Introducción al ecosistema Docker*. https://www.gotoiot.com/pages/articles/docker_intro/index.html. Feb. de 2021. (Visitado 15-01-2022).

[13] Mauricio Collazos. *Una guía no tan rápida de Docker y Kubernetes*. https://medium.com/ingeniería-en-tranqui-finanzas/una-guía-no-tan-rápida-de-docker-y-kubernetes-933f5b6709df. Jun. de 2018. (Visitado 15-02-2022).

[14] Oracle. *What is a Database*. https://www.oracle.com/database/what-is-database/. Ene. de 2022. (Visitado 15-10-2022).

[15] Atlantic Technologies. *¿Como se dividen las bases de datos y qué función tiene cada una?* https://en.atlantictech.io/blog/base-de-datos-como-se-dividen. Ene. de 2021. (Visitado 20-10-2022).

[16] Robin Nixon. *Aprender PHP, MySQL y JavaScript, 5ta edición*. Marcombo, 2019.

[17] W3schools. *MySQL Tutorial*. https://www.w3schools.com/mysql/default.asp/. Ene. de 2021. (Visitado 20-01-2022).