UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

SISTEMA DE CONTROL Y MONITOREO DE PLANTAS DESALADORAS UTILIZANDO SERVICIOS DE *CLOUD COMPUTING*

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

GIANLUCA MARRESE TAYLOR

PROFESOR GUÍA:
CLAUDIO ESTEVEZ MONTERO

MIEMBROS DE LA COMISIÓN:
FRANCISCO RIVERA SERRANO
NÉSTOR BECERRA YOMA

SANTIAGO DE CHILE
2021

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: GIANLUCA MARRESE TAYLOR
FECHA: 2021
PROF. GUÍA: CLAUDIO ESTEVEZ MONTERO

SISTEMA DE CONTROL Y MONITOREO DE PLANTAS DESALADORAS UTILIZANDO SERVICIOS DE *CLOUD COMPUTING*

La escasez hídrica es un problema de nivel mundial del cual Chile no queda exento, es más, se encuentra pasando por una de las sequías más severas y extensas de su historia. Esto hace que el agua vaya siendo un recurso cada vez más difícil de conseguir y hasta imposible en algunos lugares. Existen múltiples formas de aportar a la solución de esta problemática, desde cambios culturales hasta el desarrollo de tecnologías capaces de obtener agua de lugares que antes no era posible. Es aquí donde entra Remote Waters, una start-up dedicada a la purificación de agua, específicamente, al tratamiento de agua salada mediante el diseño y desarrollo de plantas de osmosis inversa de pequeña y mediana escala, que permitan abastecer de agua a pequeñas comunidades. Este desarrollo tiene varios desafíos asociados: la selección de componentes, el diseño de la planta, el diseño del sistema de control, el armado, entre otros.

El problema que viene a solucionar este trabajo de titulo corresponde a poder conocer el estado de las plantas instaladas sin necesidad de estar físicamente presentes en el lugar, además de permitir el control de algunos variables de estas, pudiendo diagnosticar y solucionar problemas que se presenten en ellas de manera remota. Para ello, se diseña e implementa una aplicación web capaz de monitorear y controlar remotamente las distintas plantas. Uno de los principales requisitos del sistema es la capacidad que debe tener para admitir plantas que tiene una estructura física distinta, puesto que el diseño de ellas depende de las cualidades del agua a tratar y que, por lo tanto, cada planta tiene una cantidad de variables de monitoreo y control distintas a otra.

La implementación de este sistema se realiza siguiendo una arquitectura sin servidor (*serverless*) y para ello se utilizan los servicios de computación en la nube provistos por Amazon Web Services que entregan la flexibilidad y escalabilidad necesaria para poder llevar a cabo el proyecto. Con esta plataforma como base se diseña e implementa un sistema de IoT para controlar los distintos dispositivos y una aplicación web para interactuar con este sistema, la cual corresponde al principal resultado de este trabajo de título. Específicamente, se presenta el monitoreo y control de una planta desalinizadora que está ubicada en Vichuquén y se verifica que se pueden agregar más plantas al sistema de manera sencilla, pero teniendo que rediseñar algunos componentes de la interfaz gráfica para adaptarse a las características de las nuevas plantas.

Tabla de Contenido

Introducción 1
Objetivo general . . . 3
Objetivos especificos . . . 3
Contenidos de la memoria . . . 3

1. Marco conceptual 5
1.1. Osmosis Inversa . . . 5
1.2. Planta osmosis inversa . . . 5
1.3. Message Queuing Telemetry Transport . . . 7
1.4. REST API . . . 7
1.5. Cloud Computing . . . 9

2. Estado del Arte 11
2.1. *A web-based IoT solution for monitoring data using MQTT protocol* . . . 11
2.2. *Room temperature control and fire alarm/suppression IoT service using MQTT on AWS* . . . 13
2.3. *An MQTT-based scalable architecture for remote monitoring and control of large-scale solar photovoltaic systems* . . . 14

3. Caracterización y diseño de la solución 16
3.1. Características de la solución . . . 16
3.2. Arquitectura general . . . 16
3.3. Tecnologías a utilizar . . . 17
3.4. Diseño de sistema de comunicación (IoT) . . . 19
3.5. Diseño del modelo de datos . . . 20
3.5.1. Diseño *backend* . . . 21
3.6. Diseño de interfaz de usuario . . . 21

4. Implementación 23
4.1. Metodología de desarrollo . . . 23
4.2. Sistema de comunicación (IoT) . . . 24
4.2.1. Comunicación con la nube . . . 24
4.2.2. Comunicación local . . . 26
4.3. Modelo de datos . . . 26
4.4. Backend . . . 28
4.4.1. API endpoints . . . 29

4.4.2. Almacenamiento de datos 30
4.4.3. Autenticación 31
4.5. Frontend 32
4.5.1. Componentes 32
4.5.2. Interfaces 34

Conclusión **40**

Bibliografía **41**

Índice de figuras

1. Nivel de CO2 y efectos del calentamiento global 1
2. Anomalía de la precipitación anual para Chile 2

1.1. Osmosis 5
1.2. Diagrama planta osmosis inversa 6
1.3. Ejemplo de MQTT 8

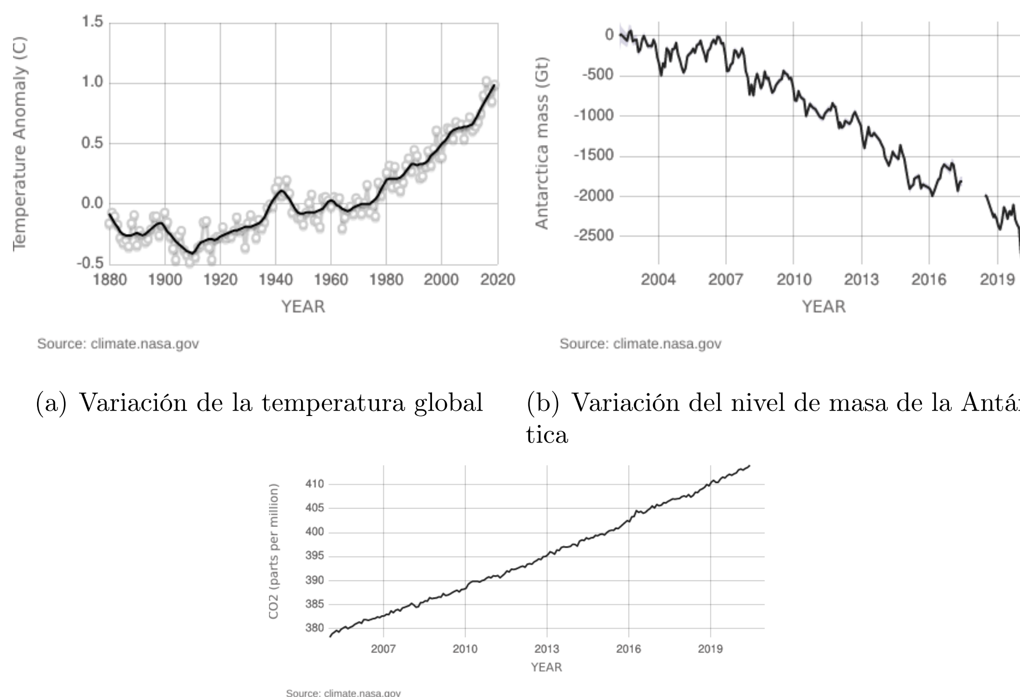
2.1. Arquitectura de la aplicación web [5] 12
2.2. Arquitectura del sistema *Smat Home* [6] 13
2.3. Arquitectura de la red [10] 15

3.1. Arquitectura general de la solución. 17
3.2. Diagrama de comunicación 19
3.3. Diagrama general 21

4.1. Metodología de desarrollo 24
4.2. Diagrama de autenticación 31
4.3. Ciclo de vida de componentes con *hooks* 33
4.4. Componentes de producción de agua 34
4.5. Componente de gráficos en tiempo real 35
4.6. Componente de gráficos en tiempo real 36
4.7. Pop-up en gráficos de tiempo real 37
4.8. Componentes de control 38
4.9. Componente de historial 38
4.10. Componente de gráficos históricos 39

Introducción

El calentamiento global es un problema que afecta a todo el mundo y tiene directa incidencia en el cambio climático que hoy experimenta la Tierra. La lista de estos efectos es extensa; el derretimiento de los glaciares, el aumento del nivel del mar, olas de calor más intensas, sequías más extensas y la lista continua. En la figura 1 se presentan gráficos extraídos de la página de la NASA [8], con algunos de estos efectos. En ella se presenta la variación de la temperatura de la superficie terrestre en comparación con el promedio de los años 1951 y 1980, en la figura 1b se observa la variación del nivel de masa de la Antártica y finalmente en la figura 1c, se presenta el nivel de CO2 presente en la atmósfera.



(a) Variación de la temperatura global

(b) Variación del nivel de masa de la Antártica

Figura 1: Nivel de CO2 y efectos del calentamiento global

En ellos se puede observar como el aumento del CO2 en la atmósfera está ligado al derretimiento de los glaciares de la Antártica y al aumento de la temperatura de la superficie terrestre, evidenciando algunos de los efectos del calentamiento global con datos actualizados al 2020.

Chile no queda exento de los efectos del cambio climático que, sumado a efectos meteoro-

lógicos, han traído como consecuencia que, desde el 2010, Chile esté pasando por una de las sequías más severas y extensas ocurridas en su historia [2], apodada la Megasequía.

Algunos de los efectos de esta megasequía corresponden a la constante disminución de las precipitaciones para la zona centro-sur de Chile, que hasta el año 2018 tenía una tendencia de -23 mm/década en promedio, como se muestra en la figura 2, considerando un período de estudio entre 1961 y el 2018 [2].

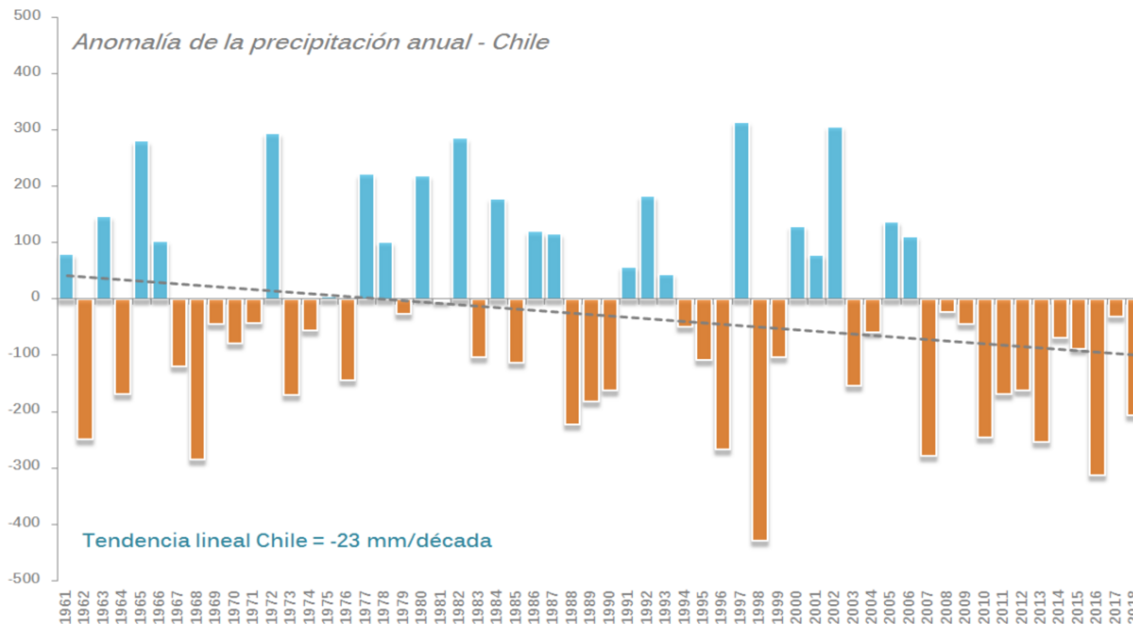


Figura 2: Anomalía de la precipitación anual para Chile

Más aún, el pronóstico para Chile no mejora, de hecho, según estudios del Ministerio del Medio Ambiente [3], las precipitaciones para Santiago caerían un 57 % en el 2030, alcanzando un 66 % para el 2050. En el mismo reporte se señala que se espera un aumento de las temperaturas máximas para Santiago de 2 °C para el 2050.

De este modo, el agua va siendo un recurso cada vez más escaso y el problema de la escasez hídrica siendo cada vez más profundo, resultando imperativo el desarrollar soluciones a este.

Existen muchas formas de enfrentar la crisis hídrica, desde un cambio alimenticio a la inversión en proyectos en pos de solucionar esta crisis. Uno de estos proyectos es Remote Waters, un emprendimiento que trabaja en el desarrollo de plantas de desalanización pequeñas, de modo de poder abastecer de agua a lugares remotos sin acceso a ella.

Esto representa varios desafíos: el diseño de la planta, ya que actualmente no existen plantas desaladoras de agua de pequeña escala, la mantención de dicha planta, el diseño del sistema de control, etc.

Sin embargo el problema que motiva la la realización del presente trabajo es el poder conocer el estado de diversas plantas desalinizadoras en tiempo real de manera remota, de modo de poder detectar y controlar algunas fallas rápidamente, y en caso de no poder solucionar los problemas remotamente, acudir lo más rápido posible al lugar, de modo de disminuir el

tiempo de mal funcionamiento o inoperatividad de la planta.

Objetivo general

El objetivo general consiste en desarrollar una plataforma de control y monitoreo remoto de plantas desaladoras a través de IoT y utilizando los servicios provistos por la computación en la nube, que sea accesible desde internet y de fácil uso.

El primer supuesto con el cual se trabajará es que la planta puede estar ubicada en cualquier lugar, ya que, en principio, lo único necesario para poder instalar una planta es acceso a agua contaminada, ya sea agua de mar, lago, rio, etc. Esto impone una primera limitante sobre el sistema y es que debe existir una forma de enviar mensajes desde la planta al servidor a pesar de que puede poseer mala conectividad.

Uno de los requerimientos es que el servicio de control y monitoreo debe estar funcionando todo el tiempo, ya que si el sistema llegase a fallar, podría tener serias consecuencias sobre la planta, y no hay que olvidar que el sistema provee de agua a las personas y, por lo tanto, no se puede permitir fallas, es por esto que se opta por la opción de utilizar los servicios de *Cloud Computing*, ya que entregan flexibilidad y seguridad al sistema.

Objetivos específicos

Luego, los objetivos específicos de la memoria quedan definidos por:

- Desarrollar un sistema capaz de recibir, procesar, almacenar y enviar mensajes a los distintos dispositivos conectados de manera diferencial, de modo de poder monitorear, en tiempo real, el estado de cada planta y poder controlar algunas variables de ellas por separado.
- Desarrollar un sistema de bases de datos que permita el almacenamiento de información tanto de los clientes como de las distintas plantas, que sea los más eficiente posible de modo de responder rápidamente a las consultas realizadas y disminuir el tiempo de almacenamiento de los datos, creando un sistema de bases de datos capaz de almacenar los datos emitidos en tiempo real por los dispositivos y responder a las consultas realizadas por los usuarios, cuando sea necesario.
- Desarrollar una Web API utilizando los servicios de *Cloud Computing* que responda al tipo de arquitectura REST (Representational State Transfer), respondiendo a los distintos métodos del protocolo HTTP, con el fin de poder hacer solicitudes al sistema desde el exterior.
- Desarrollar la interfaz gráfica para interactuar con la REST API nombrada anteriormente, que permita el acceso a la información de los distintos dispositivos.
- Desarrollar un sistema escalable, es decir, un sistema que, fácilmente, puede ser extendido a partir del sistema ya existente, permitiendo la fácil añadidura de nuevas características.

Contenido de la memoria

El los capítulos siguientes se describe el proceso de diseño e implementación de esta memoria. El capítulo 1 se centra en describir algunos conceptos básicos necesarios para comprender el diseño del sistema. Luego, en el capítulo 2, se presentan algunos trabajos relacionados al presente con el fin de tomar ideas y compararlos con el desarrollo de esta memoria. En el capítulo 3 se presenta el diseño general de la solución, incluyendo el sistema de comunicación, la forma de almacenar la información, el diseño de la API y las interfaces de usuario.

El capítulo 4 se enfoca en la implementación de la plataforma, describiendo el sistema de comunicación con los dispositivos, el modelo de datos, el *backend* y el *frontend*. Finalmente, se presentan las conclusiones y trabajo futuro del la memoria.

Capítulo 1

Marco conceptual

1.1. Osmosis Inversa

El concepto fundamental es el de osmosis inversa, ya que es el principio básico que se utiliza para desalinizar agua. Para comprender el proceso de osmosis inversa, hay que primero comprender en qué consiste la osmosis. La osmosis es el movimiento espontáneo de moléculas de solvente a través de una membrana semipermealble selectiva hacia la región de mayor concentración de soluto, de manera de igualar la concentración a ambos lados de la membrana semipermeable. Al lado izquierdo de la figura 1.1 se observa el fenómeno de la osmosis, en donde en el lado derecho de la membrana se encuentra en mayor concentración de soluto y por lo tanto posee mayor volumen de agua de modo de disminuir su concentración, tratando de igualar las concentraciones a ambos lados de la membrana.

Luego, la osmosis inversa consiste en una técnica de purificación de agua basada en la osmosis. Para ello se presenta un caso similar al anterior pero, mediante presión, se hace fluir el liquido en dirección contraria a la de la osmosis, haciendo que a un lado de la membrana aumente la concentración de soluto y, del otro lado, la concentración de soluto disminuya, obteniendo un agua más pura, como se muestra en el lado derecho de la figura 1.1.

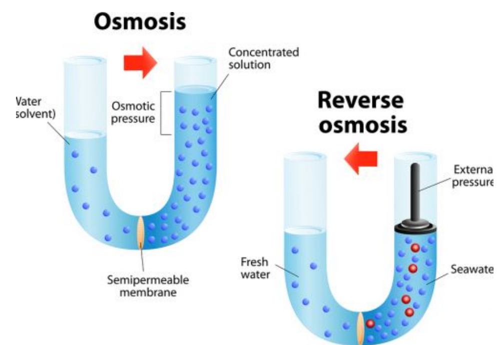


Figura 1.1: Osmosis

1.2. Planta osmosis inversa

Una vez comprendida la osmosis inversa, se debe entender, al menos básicamente, el diseño de una planta desaladora basada en osmosis inversa, de modo de poder comprender el proceso que se pretende controlar. Para ello en la figura 1.2 se presenta un diagrama genérico de una planta desaladora.

En la figura 1.2 se observan las distintas etapas del proceso de desalación; la primera

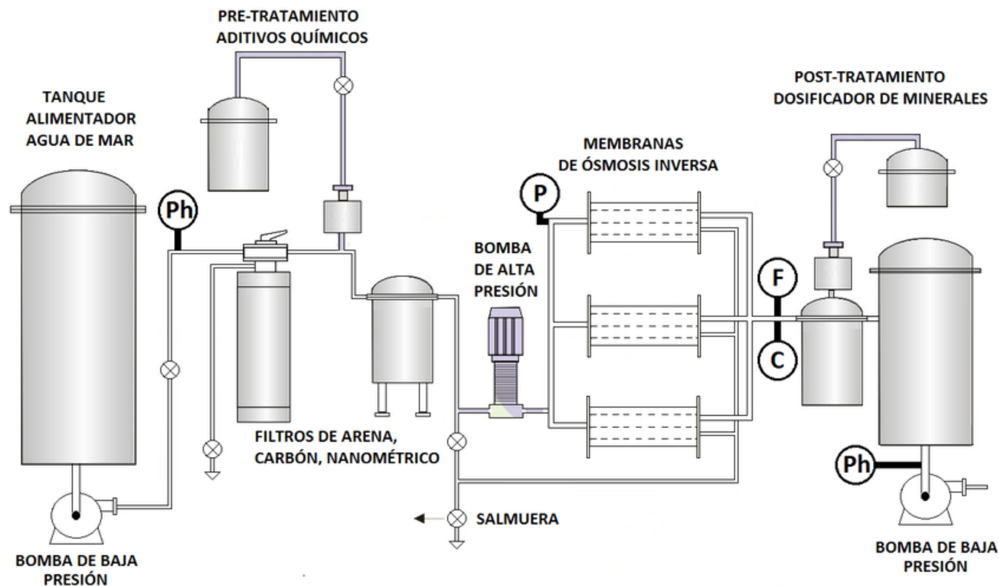


Figura 1.2: Diagrama planta osmosis inversa

etapa consiste en pre-tratar el agua de mar, aplicando filtro de carbón y arena, luego viene el pre-tratamiento químico. Posterior al pre-tratamiento, el agua pasa por una bomba de alta presión, la cual inyecta el agua hacia las membranas de osmosis inversa, proceso del cual salen 2 flujos relevantes para el proceso: el flujo de agua permeada (agua desalinizada) y el flujo de agua concentrada (salmuera). Finalmente, el flujo de agua permeada pasa por un último proceso de potabilización en el cual se le agregan los minerales faltantes.

Una de las características principales de la planta en desarrollo por parte de Remote Waters es la autonomía. Esto hace referencia a que la planta debe poder funcionar sin intervención de personas el mayor tiempo posible, por lo tanto, se desarrolla un sistema de control local, el cual se encarga de monitorear y controlar las variables físicas más relevantes del proceso, utilizando actuadores para ello. Este sistema de control, actualmente, está implementado por un PLC (Controlador Lógico Programable).

El foco del presente, no es explicar el proceso de desalación ni tampoco el sistema de control, pero es extremadamente importante comprender el proceso de manera global, de modo de poder desarrollar una plataforma que cumpla los requisitos impuestos por las características físicas de la planta que se pretende controlar y las limitantes del sistema de control también. Por lo tanto, una vez comprendido el proceso de desalación, es momento de estudiar algunos conceptos básicos de cómo se realiza la comunicación para poder entablar un sistema de control remoto a distancia.

El primer concepto clave es el Internet de las Cosas (*Internet of Things* o IoT de ahora en adelante) que hace referencia a interconectar objetos cotidianos, lo cual se realiza integrando sistemas a dichos objetos y conectándolos a Internet, de modo de crear una red de sistemas distribuidos conectados entre si [11].

Existen varios protocolos, con distintos enfoques y basados en distintos principios de fun-

cionamiento, que se utilizan para implementar el concepto de IoT, pero particularmente es interesante estudiar el protocolo de comunicación *Message Queuing Telemetry Transport* o MQTT de ahora en adelante.

1.3. Message Queuing Telemetry Transport

MQTT es un protocolo de comunicación *machine-to-machine* que se usa sobre TCP/IP y que fue diseñado para ser extremadamente ligero. Está basado en patrón *publish/subscribe* y es útil cuando se desea enviar información desde lugares remotos, donde la capacidad de procesamiento de los dispositivos y/o el ancho de banda son limitados, es decir, es un protocolo de comunicación pensado para *constrained networks*.

Las principales características del protocolo corresponden a

- Usar el patrón *publish/subscribe*, lo cual provee comunicación *one-to-many*.
- Transportar mensajes independiente de la información que se envíe.
- Tres niveles de calidad de servicio (*Quality of Service* o QoS): a los más una vez, a lo menos una vez y solo una vez.
- *Overhead* pequeño.
- Una forma de notificar cuando un dispositivo se desconecta de manera abrupta.

El protocolo define dos entidades: los clientes y un *message broker*. El broker es un servidor que recibe los mensajes de los clientes y lo enruta hacia los clientes apropiados. Los clientes pueden ser cualquier dispositivo, tanto micro-procesadores como servidores completos y de cualquier tipo. Este protocolo funciona en base a tópicos de comunicación y, como se señaló anteriormente, al patrón de publicar/suscribir. Cuando algún cliente desea enviar un mensaje, este manda un mensaje al broker, indicando el tópico en cual desea publicar y es el broker el que envía el mensaje a los clientes que se hayan suscrito a dicho tópico previamente.

En la figura 1.3 se presenta un ejemplo de los paquetes intercambiados entre 2 clientes y el *broker*, en donde se puede apreciar que, inicialmente, el cliente A se conecta al *broker*, intercambiando los paquetes CONNECT y CONNACK, luego el Cliente B publica un mensaje en el tópico *temperature/roof* cuando aun no existe ningún cliente suscrito a ese tópico, con *retain* y por lo tanto el mensaje es entregado al Cliente A por el *broker* en el momento en que se suscribe a *temperature/roof*. Posteriormente, el Cliente A publica en el tópico *temperature/floor* y el paquete es descartado por el *broker*, ya que no existe ningún cliente suscrito a dicho tópico. Después, el cliente B publica en el tópico *temperature/roof* y el mensaje es recibido casi instantáneamente por el Cliente A. Finalmente, el Cliente A se desconecta, enviando el mensaje DISCONNECT.

1.4. REST API

Un concepto fundamental de comprender para el desarrollo del proyecto es el de REST API, que es el concepto básico sobre el cual funciona la mayoría de la Web hoy en día. Para entenderlo, hay que partir por comprender qué es REST.

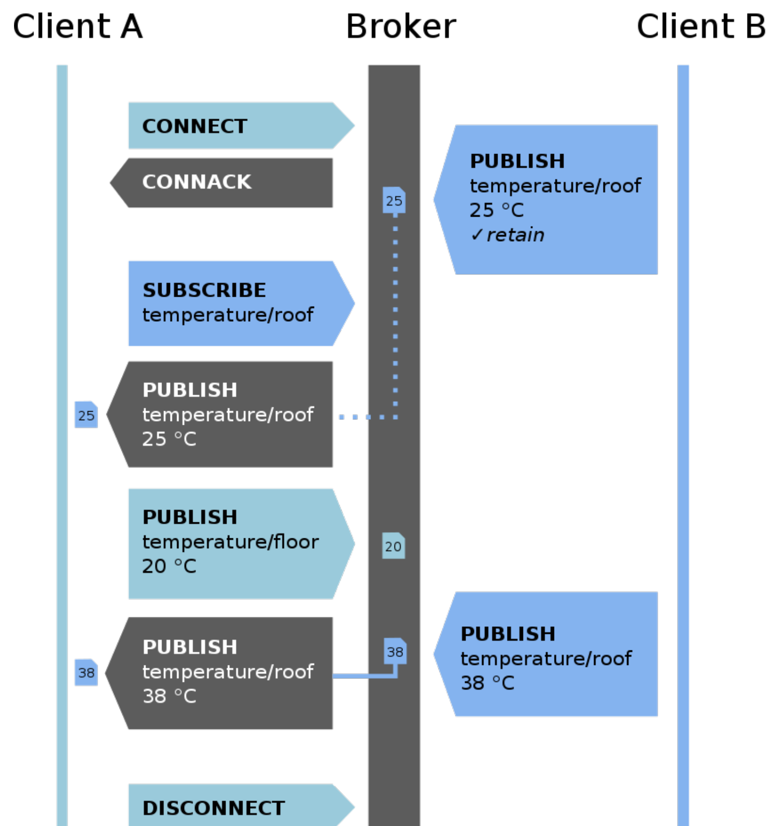


Figura 1.3: Ejemplo de MQTT

REST es una abreviación de *REpresentational State Transfer* y corresponde a un estilo de arquitectura de software que define una serie de reglas (o restricciones) que son útiles al momento de crear servicios Web.

El término REST fue acuñado por primera vez en el 2000 por Roy Fielding en sus tesis de doctorado [4], en donde señala que las restricciones que esta arquitectura impone son seis:

- Arquitectura cliente-servidor: impone el principio de la separación roles, es decir, separar las labores de la interfaz del usuario (cliente) de las labores de almacenamiento de información (servidor). Esto trae consigo mejoras en cuanto a la portabilidad de la interfaz del usuario a múltiples plataformas, además de mejorar el escalamiento de las componentes del servidor.
- Sistema por capas: un cliente no debe ser capaz de diferenciar si está conectado directamente con el servidor o está pasando por un intermediario, permitiendo la existencia de un *proxy* o de un balanceador de cargas (*load balancer*) y, por lo tanto, la existencia de intermediarios no debe afectar la comunicación. La presencia estos intermediarios mejora la escalabilidad del sistema.
- Cache: tanto los clientes como los intermediarios pueden tener respuestas en la memoria cache, lo cual permite eliminar total o parcialmente interacciones cliente-servidor, mejorando la escalabilidad y rendimiento del sistema.
- *Stateless*: la comunicación cliente-servidor está restringida a que ninguna información del contexto del cliente se almacene en el servidor entre solicitudes, por lo tanto, cada

solicitud del cliente debe contener toda la información necesaria para que el servidor pueda responder.

- Código bajo demanda (opcional): los servidores pueden extender o personalizar las funcionalidades del cliente, transfiriéndoles códigos ejecutables, tales como, *scripts* orientados al cliente que están escritos en JavaScript.
- Interfaz uniforme: ésta restricción se descompone en 4 restricciones más:
  - Identificación de los recursos en la solicitud
  - Manipulación de los recursos a través de representaciones
  - Mensajes auto-descriptivos
  - HATEOAS (*Hypermedia as the engine of application state*)

Luego, una REST API, corresponde a una API (*application programming interface*) que utiliza un cliente para comunicarse con un servicio Web y que responde a las restricciones impuestas por la arquitectura REST.

En palabras simples un REST API, son una serie de reglas para realizar un buen diseño de aplicaciones web, donde una aplicación web es una red de recursos Web (*virtual state-machine*) donde el usuario navega a través de identificadores de recursos [URI] y operaciones [GET, POST, PUT, etc] (*application state transition*) dando como resultado que la representación del recurso siguiente (*next application state*) sea transferida al cliente.

Cuando un servicio Web es implementado mediante una REST API se le llama RESTful.

1.5. Cloud Computing

Hasta este punto, se ha hablado de servidor lo cual es de interés para comprender el trabajo a realizar, pero este caso se pretende implementar la plataforma con servicios de *Cloud Computing* y por lo tanto es menester comprender a cabalidad dicho concepto, y los cambios que ello conlleva. Según el Instituto Nacional de Estándares y Tecnología (NIST), *Cloud Computing* es un modelo para habilitar acceso a recursos de computación compartidos de manera ubicua, conveniente y bajo demanda, que pueden ser rápidamente provistos y liberados. [7].

Este concepto esta compuesto de cinco características esenciales:

- Autoservicio bajo demanda: hace referencia a que un consumidor puede proveer capacidades computacionales sin necesidad de que un humano intervenga con cada proveedor de servicio.
- Acceso a través la red: hace referencia a que las capacidades pueden ser accedidas a través de la red, utilizando mecanismos estandarizados.
- *Resource pooling*: los recursos computacionales del proveedor son compartidos por muchos clientes y deben ser dinámicamente asignados y reasignados en base a la demanda de cada cliente.
- Elasticidad rápida: las capacidades del servidor deben ser provistas y desprovistas elásticamente y, en algunos casos, de modo de poder escalar rápidamente en función de la

demanda de los clientes. A ojos del cliente, la capacidad de computación debe parecer ilimitada.

- Servicio medido: los sistemas de *Cloud Computing* deben controlar y optimizar sus recursos automáticamente, teniendo la capacidad de medir la capacidad de un determinado servicio a un nivel de abstracción adecuado (por ejemplo: capacidad de almacenamiento, ancho de banda, procesamiento, etc).

Capítulo 2

Estado del Arte

En este capítulo se presenta un breve estudio de algunos trabajos relacionados a lo propuesto en esta memoria, que resultan relevantes para comprender cómo son implementados y las diferencias de estos con el sistema desarrollado. Si bien el enfoque de ninguno de estos casos de estudio es el control de plantas desalinizadoras, si se enfocan en monitorear dispositivos de IoT mediante diversas técnicas.

2.1. *A web-based IoT solution for monitoring data using MQTT protocol*

En el primer caso de estudio [5] se presenta un sistema que utiliza MQTT para monitorear el proceso de secado de tabaco que muestra la información en un página web. Para la implementación del sistema de monitoreo se utiliza una Raspberry Pi con un sensor de temperatura y humedad conectado que actúa como un cliente de MQTT, publicando información en tópicos establecidos. Esta información es recibida por el *broker*, que en este caso fue implementado con Mosquitto, que es un servidor de MQTT *open source* y posteriormente es transmitida a otros clientes que tienen diferentes roles dentro de la implementación de la aplicación web.

Como se puede ver el la figura 2.1 ésta aplicación web esta segmentada en 3 capas: capa de presentación, capa de aplicación y capa de base de datos. La capa de presentación es la encargada de mostrar la información y está implementada principalmente en HTML, JavaScript/jQuery. La capa de aplicación contiene el servidor web (Apache) y la lógica del sistema, para lo cual se utiliza PHP para su programación. La capa de base de datos se implementa utilizando MySQL y contiene las tablas con la información del usuarios y de los dispositivos.

Este proyecto resulta interesante de estudiar ya que tiene bastante relación con lo que se planea hacer, debido a que implementa un sistema de comunicación mediante MQTT el cual monitorea distintos valores a través de una Raspberry Pi y envía dicha información a otros dispositivos que implementan el servidor web, pero a diferencia del proyecto a implementar, éste no utiliza los servicios de Cloud Computing para su desarrollo. Aún así muestra una de las posibles aplicaciones de la utilización del protocolo MQTT para resolver problemas

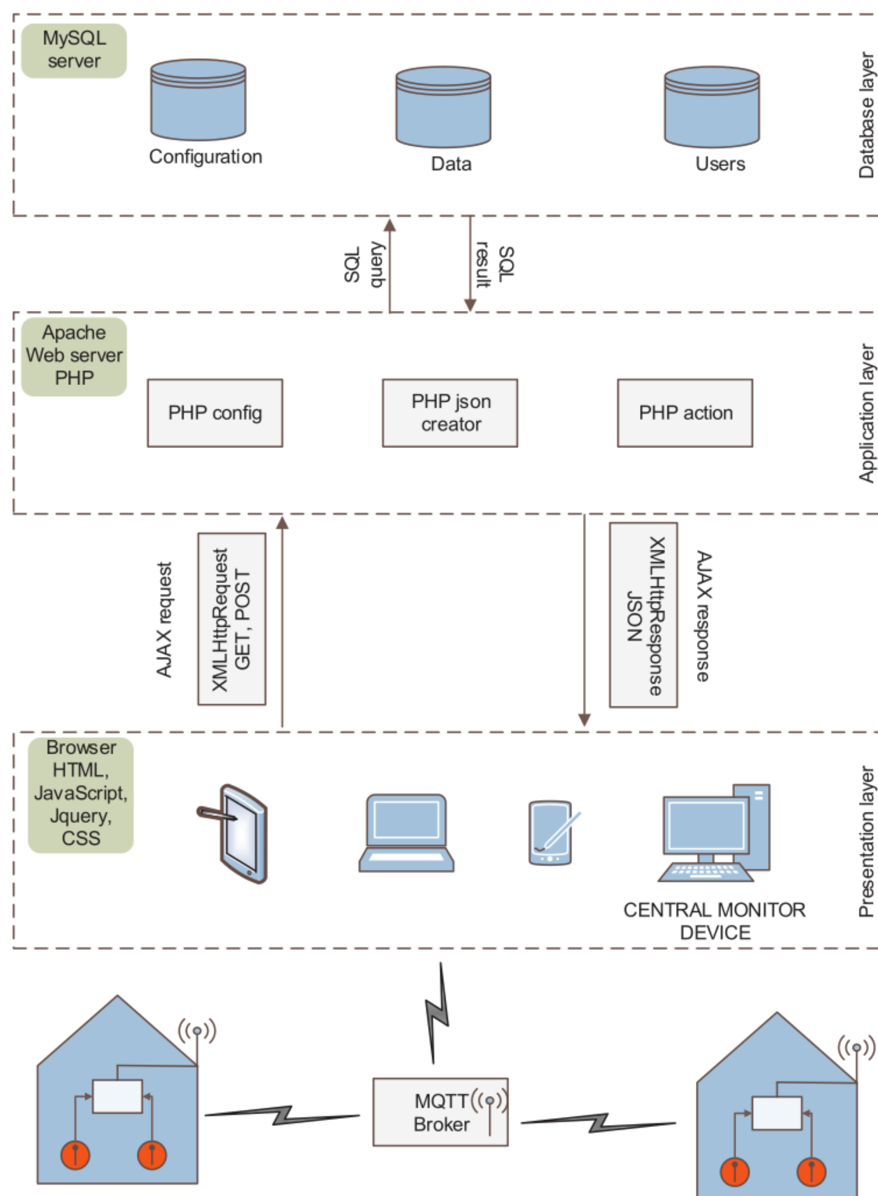


Figura 2.1: Arquitectura de la aplicación web [5]

generados por un proceso que requiere de atención constante, como lo es el proceso de secado de tabaco. También se desarrolla para un escenario similar en cuanto a condiciones físicas del lugar donde se implementa (lugares rurales y remotos, que no necesariamente tienen buena conexión a internet), solucionando el problema de supervisión humana constante del proceso.

Otra diferencia importante de este caso de estudio con el sistema que se propone realizar en esta memoria, consiste en que la Raspberry Pi tiene conectado directamente los sensores, mientras que en el sistema a implementar la Raspberry Pi es solo un intermediario entre el PLC que tiene los sensores conectados y la nube (página web). Además de que el sistema presentado en [5] no se tiene la posibilidad de controlar el proceso, solo de monitorearlo.

2.2. *Room temperature control and fire alarm/suppression IoT service using MQTT on AWS*

El segundo caso de estudio de interés es el presentando en [6], donde también se implementa un sistema de monitoreo y control de la temperatura de una sala y en caso de ser necesario, se activa una alarma. La principal diferencia entre este caso y el anterior, es el propósito por el cual se implementa, donde en el primer caso se realiza para evitar la presencia de personas en todo momento en el monitoreo de la humedad y temperatura de una sala de producción, mientras que en este caso se realiza el sistema orientado a la automatización del hogar o *Smart Home*.

En la figure 2.2 se puede observar la arquitectura utilizada para implementar el sistema, para lo cual se utilizan varios Arduinos como clientes de MQTT, que están conectados a un *broker* (Mosquitto), implementado en un servidor provisto por AWS. Las Arduinos se comunican con un AP a través de WiFi y tienen distintas funciones como: activar las alarmas, controlar el nivel de temperatura, monitorear el nivel de temperatura y mostrar dicho valor en un LCD. Esta arquitectura difiere de la presentada en esta memoria puesto que en [6] se utiliza una red de micro-controladores conectados por WiFi y no un solo micro-controlador que publica información de muchos sensores, como es el caso que se presenta acá.

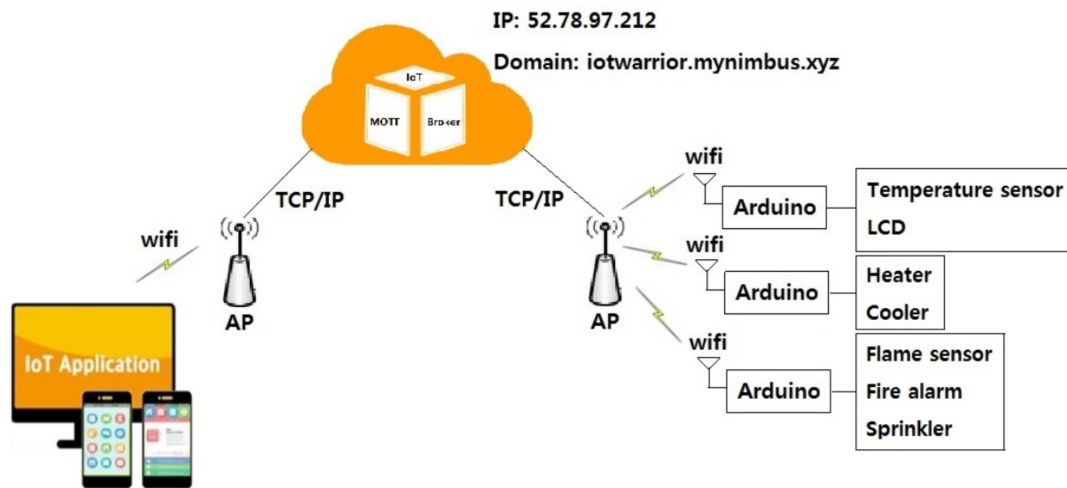


Figura 2.2: Arquitectura del sistema *Smat Home* [6]

Al igual que en el primer caso estudiado, este proyecto también se diferencia del propuesto en que no se utilizan los servicios de *Cloud Computing* para desarrollar el sistema de control y monitoreo, pero resulta interesante de estudiar ya que se logra implementar un sistema de control utilizando MQTT de dispositivos que no se encuentras conectados físicamente y, además, implementan una interfaz gráfica (aplicación móvil) para comunicarse con ellos.

2.3. *An MQTT-based scalable architecture for remote monitoring and control of large-scale solar photovoltaic systems*

El tercer y último caso de estudio [10] se presenta un sistema de monitoreo y control de plantas fotovoltaicas de gran escala a través de IoT, que permite evaluar varios valores de factores ambientales y cómo afectan en la eficiencia de la planta.

Para ello se utiliza MQTT para la comunicación y Rasberrys Pi como dispositivos finales que, en este caso, se utilizan de varias formas distintas, por ejemplo, para transmitir video, para medir distintos valores de los paneles solares y para monitorear el clima. La arquitectura de red propuesta se presenta en la figura 2.3 donde se puede observar que está dividida en 5 capas: capa de percepción, capa de red, capa *middleware*, capa de aplicación y capa de negocio.

En particular, resulta interesante estudiar las cuatro primeras capas nombradas, que en general, son similares a las presentadas en los trabajos anteriores y por ende no se presenta en mayor detalle el funcionamiento de ellas, pero se destaca, que la capa que toma mayor relevancia es la capa *middleware* ya que en ella se encuentra el *broker* y el sistema de base de datos, que permiten establecer la comunicación MQTT y el almacenamiento de los datos.

Este trabajo resulta particularmente interesante ya que se asemeja bastante al sistema a implementar ya que utiliza DynamoDB como motor de base de datos. Los dispositivos finales también corresponden a Raspberrys Pi y se desarrolla una aplicación web para el control y monitoreo de la planta fotovoltaica.

Esta aplicación web está desarrollada con Node.js (Express.js framework) y presenta varias de las características similares al sistema a implementar, tales como: información de la radiación solar que es pública para cualquier usuario, el control y monitoreo en tiempo real de paneles individuales, acceso a información histórica de los valores medidos por los sensores y una interfaz analítica del proceso. Además, la aplicación web posee un sistema de autenticación que posee otorga distintos niveles de privilegios a los usuarios, permitiendo desde usuarios que solo pueden acceder a información pública del nivel de radicación hasta administradores que pueden enviar comandos, ver estadísticas y acceder a cámaras de seguridad.

Algunas de las conclusiones más interesantes del trabajo son que el *delay* de la comunicación MQTT es de aproximadamente 1 s, lo cual hace que el sistema sea suficientemente responsivo, que la utilización de la CPU Raspberry Pi es menor al 5 % y que por lo tanto se podría utilizar un micro-controlador con menor capacidad de procesamiento para ahorrar en costo y en energía.

En fin, todos los trabajos estudiados implementan sistemas de IoT utilizando el protocolo de comunicación MQTT con distintos propósitos y obtienen resultados satisfactorios, implementado plataformas para monitorear y, en algunos casos, controlar algunos parámetros del sistema en cuestión, utilizando micro-controladores que actuan como sensores, actuadores y dispositivos finales de la comunicación.

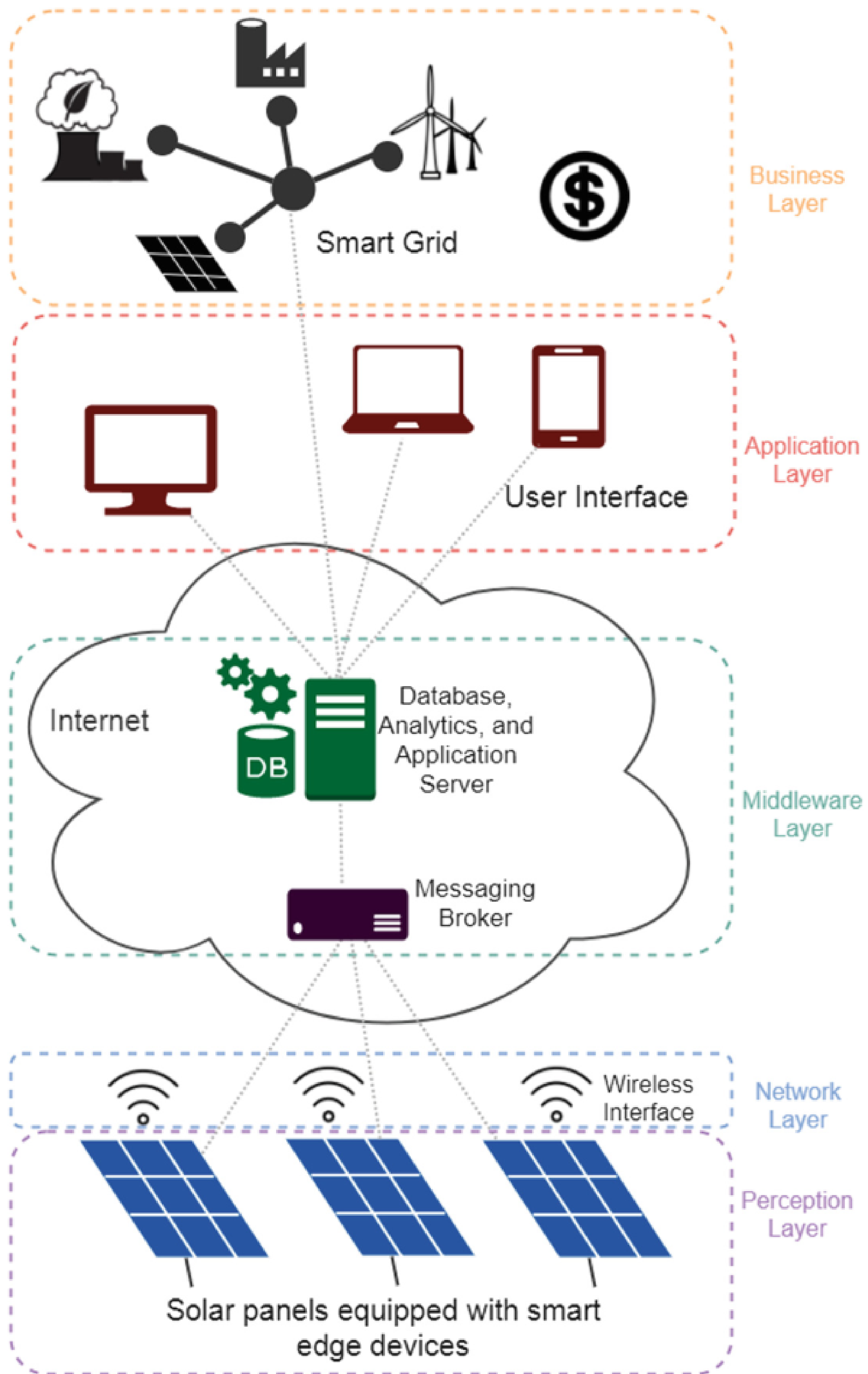


Figura 2.3: Arquitectura de la red [10]

Capítulo 3

Caracterización y diseño de la solución

Para comprender y poder diseñar una solución que se ajuste de manera adecuada al problema, hay que comenzar por describir cuáles son las características de la plataforma a diseñar, para ello, hay que comprender el diseño de la planta a controlar para luego definir una arquitectura que se adecue a estas características, seleccionando las tecnologías a utilizar y, finalmente, plantear los lineamientos del sistema a implementar.

3.1. Características de la solución

Pensando solamente en el dispositivo a controlar, la principal característica del sistema debe ser la flexibilidad al momento de diseñarlo, puesto que no existe un estándar para el diseño de las plantas desalinizadoras (como se explicó en el en los capítulos anteriores) ya que las características de esta vienen dadas por las características del agua a tratar. Por lo tanto, los parámetros de monitoreo y los parámetros de control son distintos en cantidad y cualidad para cada planta.

Otra de las características principales del sistema corresponde a que la actualización de la información para el cliente web y para el dispositivo a controlar, debe ser lo más instantánea posible de modo que frente a una falla en el sistema, el usuario sea capaz de detectarla rápidamente y poder interactuar con la planta, controlando la situación rápidamente de modo de afectar (dañar) lo menos posible las componentes física de la planta o la calidad del agua producida.

3.2. Arquitectura general

El esquema básico para la construcción del sistema de monitoreo y control remoto de una planta desalinizadora de características arbitrarias, se plantea en la figura 3.1, donde se puede observar el principio de separación de intereses entre la capa de presentación y la capa de acceso a datos mediante la presencia de dos servidores en la nube para el frontend y el backend, los cuales se comunican a través de una consulta y respuesta (*Request/Response*) que corresponde a un JSON (JavaScript Object Notation) con la información de la solicitud que realiza el cliente web hacia los servidores, que le responden con otro JSON de vuelta con

la información solicitada.

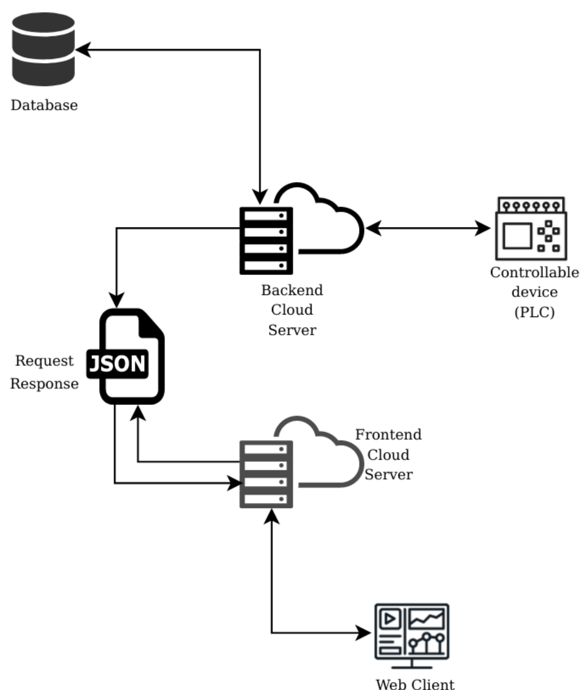


Figura 3.1: Arquitectura general de la solución.

En la figure 3.1 también se puede apreciar que la interacción entre el usuario y el dispositivo se realiza a través del servidor del *backend* y no directamente desde el *frontend*.

Algunos de los beneficios de realizar la implementación de esta forma son:

- Flexibilidad al momento de escoger las tecnologías a utilizar en el *frontend* y en el *backend*
- Desacoplamiento entre la interfaz del usuario, el modelo de datos y el sistema de IoT
- Escalabilidad del sistema, debido a que no se presentan las limitaciones de utilizar un servidor físico, ya que la capacidad de cómputo se puede aumentar fácilmente al ser servicios de computación en la nube.

3.3. Tecnologías a utilizar

Para un buen desarrollo de una aplicación web siempre es importante la selección de herramientas y tecnologías a utilizar, la cual se realiza tomando en consideración las características del sistema a implementar y como se señaló anteriormente, éstas son principalmente dos: la flexibilidad y la instantaneidad.

Otra característica implícita en la figura 3.1 es la utilización de servicios de computación en la nube, por lo tanto el primero paso es seleccionar uno dentro de los distintos proveedores.

El proveedor de servicios de computación en la nube seleccionado corresponde a Amazon Web Services (AWS de ahora en adelante) ya que en la actualidad es el más adoptado por la

comunidad y posee una basta gamma de servicios bien documentados, además de ofrecer un *Free Tier* que permite utilizar algunos de sus servicios de manera gratuita durante un año.

Una vez seleccionado el proveedor de los servicios de computación en la nube, se procede a definir cuáles de estos servicios son los necesarios para llevar a cabo el sistema, los cuales corresponden a:

- Lambda: es un servicio de informática sin servidor que ejecuta código en respuesta a eventos y administra automáticamente los recursos informáticos subyacentes.
- DynamoDB: es un servicio de bases de datos NoSQL totalmente administrado que ofrece un desempeño rápido y previsible, así como una escalabilidad óptima.
- IoT Core: es un servicio en la nube administrado que permite a los dispositivos conectados interactuar de manera fácil y segura con las aplicaciones en la nube y otros dispositivos.
- API Gateway: es un servicio completamente administrado que facilita a los desarrolladores la creación, la publicación, el mantenimiento, el monitoreo y la protección de API a cualquier escala.
- Amazon Cognito: es un servicio que provee de autenticación, autorización y manejo de usuarios para una aplicación web. Esto lo realiza a través de dos componentes principales *User Pools* y *Identity Pools*.
- IAM: es un servicio que ayuda a controlar los accesos a los recursos de AWS de forma segura.
- S3: es un servicio de almacenamiento de objetos en la nube que ofrece escalabilidad, disponibilidad de los datos, seguridad y buen rendimiento.

Comprendiendo que cada uno de estos servicios debe configurarse individualmente (permisos asociados) y, además, se debe programar su interacción con los demás servicios, se decide utilizar un *framework* que simplifique la realización de esto, de modo de no tener que trabajar directamente con la consola de AWS.

Considerando esto se selecciona Serverless Framework para el desarrollo e implementación de la plataforma, puesto que provee de una capa de abstracción sobre los servicios de computación en la nube y en particular sobre AWS.

Luego de seleccionado el *framework* para el *backend* se procede a definir el lenguaje de programación a utilizar para implementar el sistema el cual corresponde a Node.js.

Para el desarrollo del *frontend* se decide utilizar la librería React, principalmente haciendo uso de los *Hooks* [9], en conjunto con *Bootstrap*, *React-icons*, *React-table*, entre otros.

El dispositivo a controlar remotamente corresponde a una Raspberry Pi 4, que toma la función de *gateway* del PLC, ya que posee una fácil integración con todos los servicios de AWS al utilizar el SDK provisto por ellos mismos.

La Raspberry Pi 4 se conecta al PLC mediante una red local y la comunicación se realiza mediante el protocolo MQTT.

Como ya se señaló anteriormente, la principal ventaja de utilizar una Raspberry Pi 4

corresponde a la fácil integración con el sistema desarrollado, ya que si se decidiera no utilizar la Raspberry Pi, se deberían implementar muchas de las funcionalidades que viene ya desarrolladas en el SDK a nivel del PLC; más adelante se detalla el cómo se establece la comunicación entre la Raspberry Pi y la nube. Otra ventaja que tiene la utilización de la Raspberry Pi es que permite seleccionar el lenguaje de programación a utilizar para su desarrollo, y de modo de tener mayor sinergia con el sistema completo, su programa, se desarrolla en Node.js.

También es importante destacar el uso de repositorios GitHub como herramientas de control de versiones durante todo el proceso de desarrollo.

3.4. Diseño de sistema de comunicación (IoT)

En la figura 3.2 se observa el proceso de comunicación para un caso en particular de una planta que cuenta con 24 variables de monitoreo. En ella se observa un PLC que controla el proceso de osmosis inversa, el cual se comunica mediante MQTT (el broker de la comunicación está ubicado en la red local y es un proceso corriendo en la Raspberry Pi) con el proceso device.js que, a su vez, se encarga de comunicar los mensajes hacia la nube de AWS, conectándose con el servicio IoT Core, haciendo uso de las *shadow Things* y con esto permitiendo una total integración con todos los servicios de AWS.

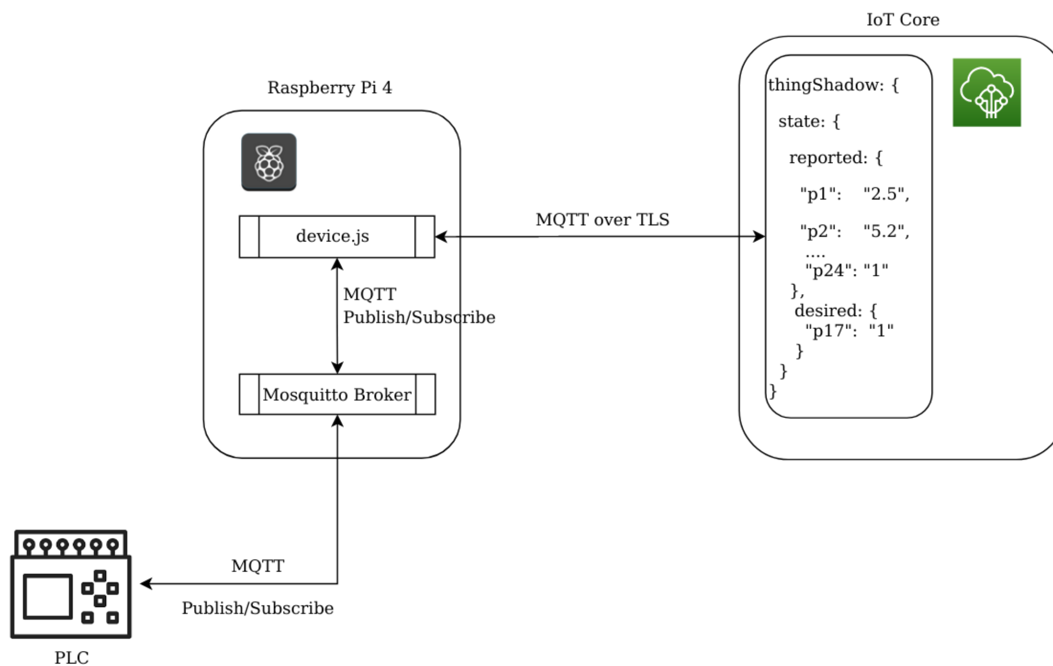


Figura 3.2: Diagrama de comunicación

En palabras simples, las *shadowThings* son una representación del estado del dispositivo que se actualiza al momento de haber un cambio en su estado. Dicha actualización puede ser de cualquier lado de la comunicación, es decir, desde el dispositivo (RPi) o desde el cliente web, que en este caso es representado por el IoT Core.

Esta representación corresponde a un JSON con el estado del dispositivo, almacenando las características y el estado actual de cada una de ellas.

En el estado del *shadowThing* se puede observar que existen dos objetos, uno que representa el estado actual (*reported*) y uno que representa el estado deseado (*desired*) y por lo tanto, en el momento en que el cliente web quiere actualizar un valor del estado debe publicar un mensaje que incluya la propiedad *desired* con el valor que desea actualizar o controlar.

3.5. Diseño del modelo de datos

Como ya se ha mencionado en reiteradas ocasiones, una de las principales características del sistema es la flexibilidad del sistema y, por lo tanto, el almacenamiento de datos también debe ser suficientemente flexible para poder guardar información de plantas con distintas características (número arbitrario de variables de monitoreo y control).

Comprendiendo esta propiedad fundamental del modelo, se descartan las bases de datos relacionales (SQL) y se opta por la opción de una base de datos no relacional (noSQL) que permita esta flexibilidad al momento de almacenar la información, en particular se opta por una base de datos no relacional basada en documentos y además que sea de computación en la nube: DynamoDB.

DynamoDB esta compuesto por tres componentes principales: tablas, ítems y atributos. Una tabla es un conjunto de ítems y cada ítem es un conjunto de atributos. Para identificar cada ítem se utilizan las llaves (llave de partición) como identificador único. También existe la posibilidad de tener una segunda llave llamada llave de orden que, como su nombre indica, permite ordenar los datos que posean una misma llave de partición.

El proceso de diseño de una base de datos no relacional se inicia definiendo cuáles serán las consultas (*queries*) a la base de datos, de modo de almacenar la información que responde a dichas consultas lo más próxima posible, disminuyendo el tiempo de respuesta.

Se identifican principalmente 3 tipos de consultas que se la realizarán a la base de datos: información de los usuarios y dispositivos, donde se presenten las características de cada uno de ellos, para el caso de los dispositivos debe almacenar las propiedades de cada una de las variables de monitoreo y control, por ejemplo, que posee 3 sensores de flujo, 2 de presión, 1 de temperatura, 2 de nivel, etc. Y para el usuario almacena información asociada a su cuenta.

La segunda consulta identificada corresponde a información histórica del dispositivo de interés, por lo tanto se debe almacenar la información de monitoreo de los distintos dispositivos de modo de poder acceder a ella rápidamente.

La tercera y última consulta identificada, corresponde a información histórica de control de los dispositivo, de modo de saber cuáles han sido las variables que se han controlado y el momento en que la instrucción fue dada.

Todas estas consultas se hacen con el fin de poder determinar el estado de la planta en todo momento y, en caso de detectar una falla, poder determinar también cuales fueron los hechos que llevaron a que ocurriera dicha falla.

Finalmente, el diseño de la base de datos queda determinado por tres tablas que respondan a las consultas identificadas:

- **device:** Tabla que almacena la información de las características del dispositivo y de las características del usuario
- **deviceRecords:** Tabla que almacena información histórica de monitoreo de los dispositivos
- **deviceControl:** Tabla que almacena información de control de los dispositivos, de modo de poder llevar un registro de cuáles han sido las acciones tomadas sobre el dispositivo de interés.

3.5.1. Diseño *backend*

El *backend* está basado en eventos. En la figura 3.3 se presenta el diagrama general del sistema, donde se puede observar la interacción entre los distintos servicios y dispositivos.

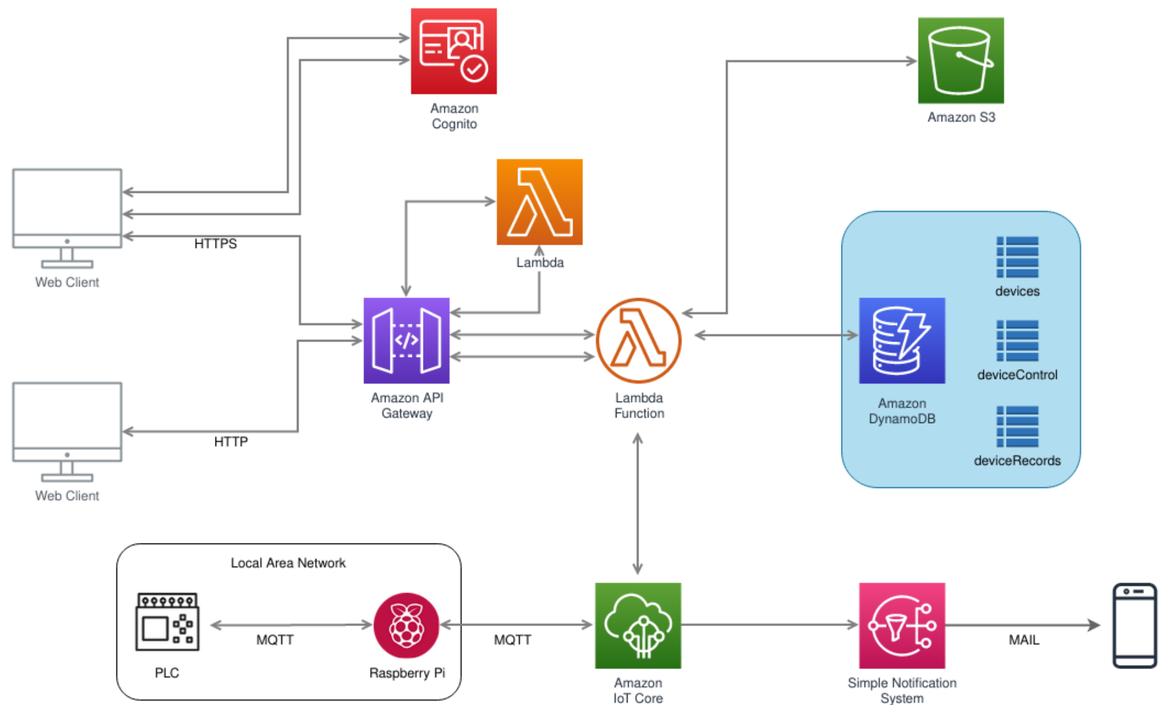


Figura 3.3: Diagrama general

3.6. Diseño de interfaz de usuario

Como ya se mencionó anteriormente, para el diseño de la interfaz de usuario se utilizó React como librería principal, con el uso de Hooks como principio de desarrollo. También es importante recordar que el *frontend* se encuentra en un servidor en la nube aparte de todo el

backend, y que consume datos de la API diseñada a partir del modelo de datos ya explicado anteriormente.

Las aplicaciones diseñadas en React se caracterizan por ser en su mayoría del tipo *Single Page Application* (SPA), que corresponden a aplicaciones web que caben en una sola página y pretenden dar una experiencia más fluida al usuario, similar a una aplicación de escritorio.

Estas aplicaciones están basadas en componentes funcionales como la estructura básica que se combinan para conformar la aplicación completa. Cada componente puede recibir propiedades del componente que lo invoca y también es capaz de manejar su propio estado, teniendo funcionalidad por si solo.

En este caso existen componentes estáticos como dinámicos. Los componentes estáticos corresponden a aquellos que no dependen de la información de la API, sino que solo dependen de su propio estado, como por ejemplo las barras de navegación superior y lateral. Las componentes dinámicas corresponden a gráficos y tablas que dependen directamente de la información de la API.

Es importante destacar que como cada planta es distinta, en caso de necesitar crear una nueva planta distinta a alguna de las ya creadas, se debe desarrollar una nueva vista para dicha información, que puede reutilizar las componentes, pero aun así debe desarrollar el acople y ajuste de dichas componentes a las información de la planta y a las necesidades de visualización.

Capítulo 4

Implementación

En esta sección se describe la metodología y el proceso de implementación de la plataforma de monitoreo y control de plantas desalinizadoras, describiendo el proceso de desarrollo del sistema de comunicación, el modelo de datos, el *backend* y el *frontend*.

4.1. Metodología de desarrollo

El trabajo de título fue realizado en conjunto con integrantes de la empresa Remote Waters por lo cual el diseño y desarrollo se trabajó con una estrategia de desarrollo iterativa e incremental, lo cual facilita la comunicación y la validación de los distintos requisitos.

Las etapas del desarrollo se pueden dividir en:

1. Diseño e implementación de autenticación de usuarios
2. Creación del modelo de datos
3. Creación del sistema de comunicación
4. Generación de REST API (*backend*)
5. Diseño e implementación de interfaces (*frontend*)

En la figura 4.1 se observa un diagrama que representa la metodología de desarrollo de la plataforma web, en donde se destaca el uso de un desarrollo iterativo principalmente para la generación del la API y la interfaz gráfica.

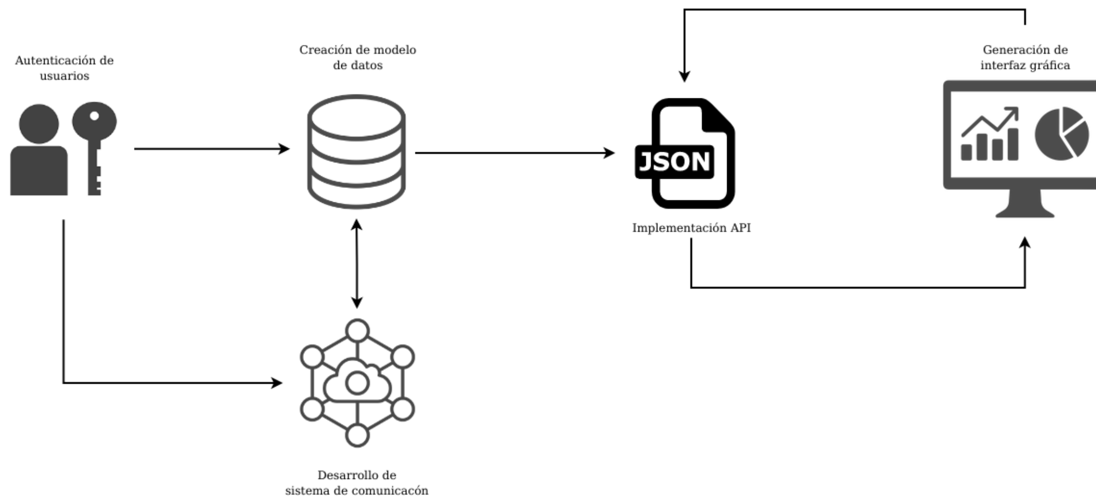


Figura 4.1: Metodología de desarrollo

4.2. Sistema de comunicación (IoT)

Como ya se mencionó anteriormente el sistema de comunicación está conformado por dos redes que interactúan entre sí: la red existente entre las Raspberry Pi y IoT Core, y la red existente entre las Raspberry Pi y el PLC (red local). Primero vamos a obviar el hecho de la existencia de la red local para centrarnos en el proceso de comunicación con la nube.

4.2.1. Comunicación con la nube

El desarrollo del sistema de comunicación con la nube está basado en AWS IoT Core, por lo tanto resulta necesario comprender su funcionamiento.

Como ya se ha mencionado anteriormente, IoT Core provee servicios de computación en la nube que conectan los dispositivos de IoT a los demás servicios de AWS. Esto lo hace a través de distintos protocolos de comunicación; en este caso en particular se realiza mediante MQTT, por lo tanto el IoT Core cumple el rol de broker con algunas características particulares.

El servicio más relevante para este desarrollo es *AWS IoT Device Shadow*, el cual permite mantener el estado del dispositivo en la nube durante todo el tiempo y hacerlo accesible para los demás servicios. Esto da espacio a que los dispositivos se desconecten y vuelvan a conectar sin perder su estado, permitiendo a su vez la interacción con ellos independiente de si estos se encuentra conectados al momento exacto de la interacción.

Técnicamente, el *device shadow* corresponde a un JSON que mantiene el estado del dispositivo que contiene tres propiedades:

- *desired*: permite el control del dispositivo a través de aplicaciones al actualizar esta propiedad.
- *reported*: los dispositivos actualizan su estado actualizando esta propiedad.

- *delta*: AWS IoT reporta las diferencias entre las propiedades *desired* y *reported* en esta propiedad.

Una vez comprendido el principio básico de funcionamiento de los *device shadow* es momento de explicar el desarrollo de la solución, recordando que para el caso de la red de conexión con la nube el dispositivo que se está controlando es una Raspberry Pi, la cual es programada con Node.js.

La Raspberry Pi se encarga de mantener el estado de la planta en todo momento en un JSON, que en la próxima sección se explica cómo y cuándo se actualiza, de momento basta con suponer que ésta siempre se encuentra en sincronía con el estado real de la planta.

El dispositivo se inicializa suscribiéndose a todos los tópicos de comunicación soportados por AWS y luego actualizando su estado al estado que mantiene la nube en el *device shadow*. Al momento de suscribirse a cada uno de los tópicos se deben indicar las acciones a tomar al recibir un mensaje en ese tópico. A continuación se presenta el comportamiento de un mensaje en dos de los tópicos, que son los más relevantes:

- **/get/accepted**: para recibir un mensaje en éste tópico se debe realiza una solicitud previa y al recibirlo se actualizar el estado del dispositivo al estado de la nube, por lo tanto es el que inicializa el estado del dispositivo.
- **/update/delta**: este mensaje se recibe siempre que se actualice el estado en la nube del dispositivo y al momento de recibirlo se actualizan los parámetros presentes en el mensaje recibido de modo de que exista sincronía entre el estado del dispositivo con el estado deseado.

Una vez inicializado el proceso se publica, periódicamente, el estado del dispositivo (se publica en el tópico /update) con el fin de poder monitorerarlo. El periodo de publicación depende de la planta que se esté controlando y de la resolución que se interese tener de los parámetros (depende del ancho de banda disponible para cada caso).

Es importante destacar que las *device shadow* de las distintas plantas mantienen una estructura definida de modo de poder procesar la información contenida en ella independiente de la cantidad de variables. Esta estructura consiste en una ID, un tipo y luego las variables de monitoreo y control llamadas p1, p2, p3 y así hasta la ultima variable. Esta estructura se muestra a continuación en 1.

```
1    {
2        "id": "01478890-4a26-11eb-8ca1-ebb674da09a4",
3        "type": "Eva-v1",
4        "p1": "0.23",
5        ...
6        "p24": "0"
7    }
```

Código 1: Ejemplo de *device shadow*

En resumen, el dispositivo publica en /update periódicamente de modo de poder monitorear su estado desde la plataforma y, desde la plataforma, se publica en /update/delta para controlar el dispositivo.

También es importante notar que la publicación de información desde el cliente web hacia el dispositivo se realiza mediante una API que entrega AWS para este motivo.

Una vez comprendido el proceso de comunicación entre la Raspberry Pi y la nube es momento de comprender cómo se implementa la comunicación local entre la Rasperry Pi y el PLC.

4.2.2. Comunicación local

El proceso de comunicación local escapa del enfoque de la memoria puesto que este proceso guarda relación con la forma en que funciona el PLC y el cómo está programado, por ende se describe superficialmente. Este proceso de comunicación también es mediante MQTT, pero a diferencia de la comunicación presentada en la sección anterior, esta ocurre dentro de una red local.

Para ello, en la Raspberry Pi se encuentra corriendo un proceso llamado Mosquitto que corresponde a servidor de MQTT, es decir, un broker.

En este proceso de comunicación existen dos grandes tópicos: <PLC-name>/control, <PLC-name>/monitoreo. Como su nombre lo indica, en <PLC-name>/control se publican los mensajes de control, y en <PLC-name>/monitoreo se publican los mensajes de monitoreo.

Los mensajes de control son enviados desde la Raspberry Pi hacia el PLC al momento de recibir un mensaje en /update/delta, lo cual desecandena acciones sobre la planta, por ejemplo, darle partida o parada. Y los mensajes de monitoreo son enviados desde el PLC hacia la Raspberry con una frecuencia de envío que depende de cuánto estén variando las variables sensadas; a mayor variación mayor frecuencia de envío, lo cual implica que en momentos de cambio de estado en la planta, mayor es la velocidad de actualización del estado de la Raspberry Pi.

4.3. Modelo de datos

Como se mencionó anteriormente, el motor de base de datos seleccionado corresponde a DynamoDB y el diseño está conformado por tres tablas. Cada tabla almacena información distinta asociada al dispositivo, las cuales se relacionan entre ellas mediante la ID del dispositivo.

La tabla *device* almacena información del dispositivo en items que tienen la forma que se muestra en 2:

```json
{
  "deviceId": "01478890-4a26-11eb-8ca1-ebb674da09a4",
  "userId": "01478890-4a26-11eb-8ca1-ebb674da09a4",
  "isDevice": true,
  "deviceType": "Eva-v1",
  "deviceLocation": "Vichuquen",
  "deviceName": "EvaVichuquen",
  "attachments": {
    "certificate": {
      "name": "Certificate",
      "s3key": "EvaVichuquenTest/certificate.pem"
    },
    "privateKey": {
      "name": "Private Key",
      "s3key": "EvaVichuquenTest/privateKey.pem"
    }
  },
  "createdAt": 1609281265561,
  "parameters": {
    "p1": {
      "name": "pressureOne"
    },
    "p2":{
      "name": "pressureTwo"
    },
    "p3": {
      "name": "pressureThree"
    },
    ...
    "p24": {
      "name": "threewayValveTwo"
    }
  }
}
```

Código 2: Ejemplo de ítem en la tabla *device*

En esta JSON se puede observar algunas de las propiedades del dispositivo. Lo primero que se puede notar es que el dispositivo posee dos IDs, esto se debe que de esta forma se puede almacenar información de los usuarios asociados a los dispositivos de manera similar a la que se almacena información relacionada entre dos tablas en una base de datos relacional, pero de esta forma solo se necesita una tabla.

También se tienen parámetros que indican el tipo de dispositivo, lo cual permite procesar la información de manera diferenciada para distintos tipos de dispositivos. Es importante notar que el tipo de dispositivo se comparte entre la tabla *device* y el *device shadow* correspondiente.

Luego, se almancenan los archivos adjuntos, junto con una forma de identificar dichos archivos en AWS S3. Finalmente, se almacenan las características o parámetros del dispositivo, donde a cada p (p1, p2, p3, etc) se le asigna un nombre, que permite almacenar la información de esa propiedad p con el nombre asociado en la tabla *deviceRecord*.

En el JSON 3 se presenta un ejemplo de un ítem almacenado en la tabla *deviceRecord*

```
{
  "deviceId": "01478890-4a26-11eb-8ca1-ebb674da09a4",
  "deviceTimestamp": 1613422549,
  "state": "Retrolavado Turbidox",
  "inFlux": "583.20",
  "outFlux": "163.80",
  ...
  "pressureOne": "5.11",
  "pressureTwo": "4.88",
  "pressureThree": "3.96",
  "startLight": "1",
  "stopLight": "0"
}
```

Código 3: Ejemplo de ítem de la tabla *deviceRecord*

En este extracto de un ítem se puede observar que se almacena la información del estado de la planta nombrados de la misma forma que en la tabla *device*, con los indetificadores *deviceId* y *deviceTimestamp* que corresponde el valor del tiempo Unix del evento. Además se almacena la información del estado de la planta en el momento en que ocurre el evento. El estado de la planta corresponde a procesos físicos que ocurren en la planta y que depende del control del PLC.

La tabla *deviceControl* almacena la información de manera similar a la tabla *deviceRecord* pero almacenando los parámetros de control de la planta.

4.4. Backend

El desarrollo del *backend* corresponde a la implementación de una REST API que exponga al usuario las interacciones con el sistema de IoT presentado en la sección Sistema de comunicación (IoT) en conjunto con los mecanismos para acceder a la información almacenada en la base de datos.

Su implementación está realizada con el *Serverless Framework* y está basada en eventos que gatillan acciones. Una de las muchas ventajas que brinda esto es que no se necesita de un servidor físico, sino que todo el código corre en la nube, específicamente, en AWS Lambda. Además, cada acción que se realiza corresponde a una función y es totalmente independiente de las demás, permitiendo, si es que así se desea, programar cada una en lenguajes diferentes. En este caso en particular todo el desarrollo está hecho en Node.js, pero para un futuro

desarrollo sería útil tener esta flexibilidad a la hora de decidir el lenguaje de programación a utilizar.

4.4.1. API endpoints

En *Serverless Framework* se configuran los *API endpoints* de la manera presentada en 4, donde para cada *endpoint* se debe definir un *handler* y un evento que invoca dicho *handler*.

```
  functions:
    create-device:
      handler: create-device.main
      events:
        - http:
            path: devices
            method: post
            cors: true
            authorizer: aws_iam
```

Código 4: Extracto de configuración del *Serverless Framework*

Para este caso en particular el *handler* corresponde a *create-device* y el evento consiste en una solicitud HTTP del tipo POST que debe ser realizada a la URL compuesta por <dominio>/<path>, la cual, en este caso, es <dominio>/devices. Además, se indica que soporta CORS ya que la solicitud va a venir desde el *frontend* que tiene otro dominio. Finalmente, se especifica que el acceso a este *endpoint* esta restringido por las credenciales de AWS IAM del usuario.

A continuación se procede a explicar algunos de los *endpoints* más importantes. La configuración de cada uno de ellos tiene la misma estructura a la a la presentada en 4 pero cambiando las variables de la URL (path), el *handler* y el método de la solicitud según corresponda.

Crear dispositivo

La llamada a crear dispositivos (create-device) es del tipo POST y como su nombre lo indica, corresponde a crear un dispositivo, lo cual implica agregar un ítem a la tabla *devices* que contenga las características de dicho dispositivo: nombre, ubicación, fecha de instalación, capacidad, entre otros. Además, crea el dispositivo dentro del IoT Core mediante una llamada a la API de AWS y, luego, procede a actualizar el *device shadow* con las características de este dispositivo, tales como, el tipo y la id, que, posteriormente, permiten que se almacene información del estado en la tabla *deviceRecords*. También, almacena los certificados en un contenedor S3 de AWS para, posteriormente, poder descargarlos y cargarlos a los dispositivos.

Obtener dispositivo

La llamada a obtener el dispositivo (id/get-device) es un consulta del tipo GET y hace una consulta a la base de datos por el dispositivo, respondiendo con la información de este.

Obtener últimos 10 minutos

La llamada a obtener los últimos 10 minutos de un dispositivo (id/get-last-10-mins) es del tipo GET y hace una consulta a la tabla *deviceRecords* por la información de los últimos 10 minutos del dispositivo, respondiendo con una lista con los valores.

Obtener gráficos de los últimos 10 minutos

La llamada a obtener los gráficos de los últimos 10 minutos (id/graph-last-10-min) es una consulta similar a la anterior con la diferencia de que la respuesta está organizada en un objeto que permite graficar la información en el *frontend* sin modificarla en éste.

Obtener producción último día

Consulta (id/get-last-day-production) del tipo GET que responde con la producción de agua del último día, para ello, calcula toma la propiedad nombrada *outFlux*, que debe estar presente en todos los dispositivos y calcula un promedio de L/h y lo multiplica por las 24 horas.

Controlar dispositivo

Consulta (id/control) del tipo POST que permite interactuar con el *device shadow*, actualizando los parámetros del dispositivo correspondiente, es decir, permite controlar el dispositivo. Además, almacena la información de control en la tabla *deviceControl*.

Otros *endpoints*

Existen varias otras consultas que, por ejemplo, permiten adjuntar archivos a los dispositivos, descargar estos archivos, entregar la producción semanal, entregar la producción mensual, obtener el estado del dispositivo en tiempo real, editar las características del dispositivo, obtener la información del estado del dispositivo en un tiempo determinado, gráficos en un tiempo determinado, entre otros.

4.4.2. Almacenamiento de datos

La función encargada de almacenar la información enviada en tiempo real por los distintos dispositivos conectados (device-to-database) no se expone en la API, ya que no corresponde que un usuario pueda acceder a ella.

La estructura que tiene en la configuración del *Serverless Framework* se muestra en 5, donde se puede observar que el evento que gatilla la acción corresponde a una publicación en el tópico /update/accepted de cualquier dispositivo.

```
functions:
  device-to-database:
    handler: device-to-database.main
    events:
      - iot:
          sql: "SELECT * FROM '$aws/things/+/shadow/name/+/update/accepted'"
```

Código 5: Extracto de configuración del *Serverless Framework*

Esta función se encarga de almacenar la información enviada por el dispositivo en la tabla *deviceRecords*, agregando el estado de dicha planta al momento de enviar el mensaje.

4.4.3. Autenticación

Para poder acceder a cualquiera de los *endpoints* del sistema se deben presentar credenciales que acrediten que el usuario está autorizado a realizar esta acción. Para ello se utiliza el servicio Cognito de AWS. En particular se utiliza el servicio *Users Pool* para almacenar los usuarios y permitir el registro de ellos. También, se utiliza el servicio *Identity Pool* el cual le va a asignar a los usuarios, autenticados a través de la *User Pool*, los permisos correspondientes (le va a adjuntar un rol IAM al usuario).

En la figura 4.2 se presenta el esquema que representa el proceso de autenticación, donde se puede observar que el usuario ingresa a su cuenta a través de *Cognito User Pool* y posteriormente se le asigna el rol IAM correspondiente mediante *Cognito Identity Pool* para poder interactuar con el sistema.

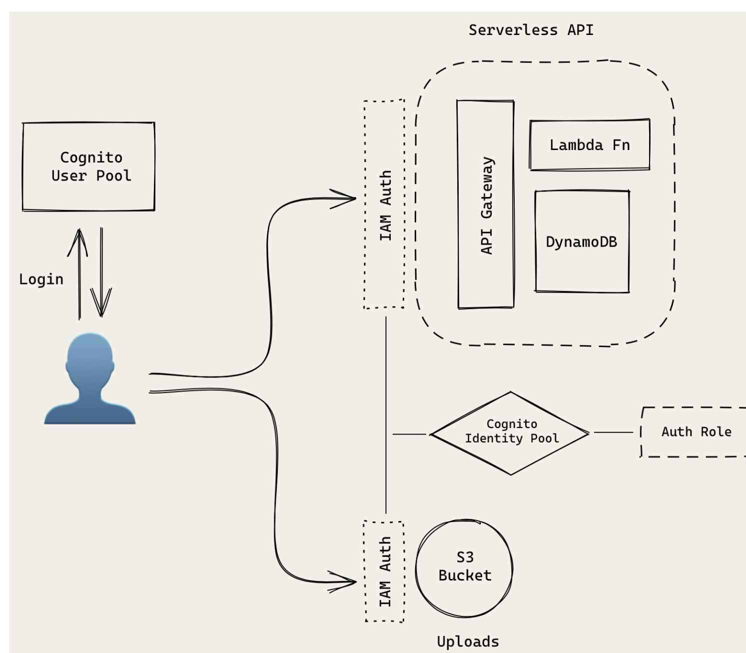


Figura 4.2: Diagrama de autenticación

4.5. Frontend

Como se dijo en la sección de Caracterización y diseño de la solución, el *frontend* se implementa como una aplicación en React cuya estructura y principales características se detallan a continuación.

La aplicación desarrollada es del tipo SPA (*single page application*) y, por lo tanto, posee solo una página que se compone de pequeños componentes (fragmentos de la interfaz) que se actualizan, dinámicamente, generando nuevas páginas. Es importante destacar que la página se carga solo una vez (simulando el comportamiento de una aplicación de escritorio) por lo cual estas nuevas páginas solo se construyen reutilizando, agregando o actualizando, componentes ya existentes.

Otra de las características principales que encontramos dentro de esta página es la forma de ruteo, ya que a diferencia de otros *frameworks*, las rutas no están asociadas a una página, sino a los componentes presentes en la página, es decir, las rutas están asociadas a la combinación de componentes presentes en la página y no a la página propiamente tal.

4.5.1. Componentes

Los componentes son estructuras independientes y autónomas que pueden interactuar con otros elementos o componentes en una página, además, dependiendo de las necesidades de la aplicación, un componente puede estar formado por un conjuntos de otros componentes. Estos pueden ser clases o funciones en JavaScript las cuales poseen propiedades, atributos y un estado local (objeto que representa partes del componente que pueden cambiar en el tiempo). Originalmente, para crear un componente con estado era necesario crear una clase mientras que un componente sin estado se implementaba utilizando funciones, pero desde el lanzamiento de React 16.8 es posible crear componentes funcionales con estado utilizando *hooks*.

Los *hooks* son funciones que permiten utilizar estados dentro de componentes funcionales, proporcionándole a estos las mismas características y utilidades que posee un componente basado en clases. Además del *hook* de estado, existen *hooks* que controlan el contexto del componente, referencias, *callbacks*, entre otros.

Para esta aplicación web se utilizaron componentes funcionales con hooks lo cual permite crear componentes más simples, con una sintaxis concisa y lógica reusable.

Los componentes basado en hooks no son ajenos al ciclo de vida de React, y para mayor comprensión de este en la figura 4.3 se presenta el diagrama propuesto por Dan Abramov [1].

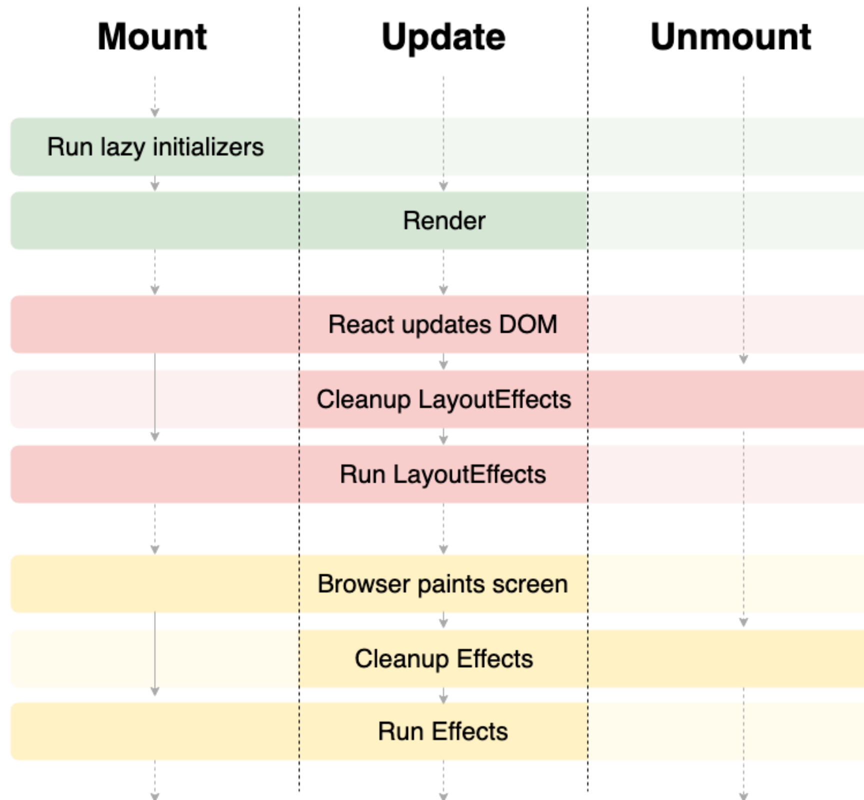


Figura 4.3: Ciclo de vida de componentes con *hooks*

Mount hace referencia al momento cuando se montan por primera vez los componentes en el DOM, cargándose el estado inicial de estos componentes.

Update hace referencia al momento en el que se actualiza el DOM debido a que cambia alguna propiedad o estado del componente. Y *Unmount* hace referencia al momento en que se desmonta algún componente y se ejecutan funciones encargadas de limpiar los ”efectos secundarios” que se producen luego de las actualizaciones.

4.5.2. Interfaces

A continuación se describen las principales funcionalidades e interfaces implementadas.

Interfaz de detalle del dispositivo

Este componente se encarga de mostrar todos los datos asociados a un dispositivo en particular, además de permitir la subida y bajada de archivos asociados a este dispositivo con el fin de almacenar toda la información del dispositivo en un solo lugar.

Interfaz de monitoreo del dispositivo

Esta interfaz está compuesta de varias sub-componentes, de las cuales tres de ellas corresponde a la visualización de la producción diaria, semanal y mensual de agua de la planta, como se muestra en la figura 4.4

Figura 4.4: Componentes de producción de agua

Otra de las sub-componentes que conforman la interfaz de monitoreo corresponden a los gráficos en tiempo real del dispositivo. Esta componente grafica el estado de la planta durante los últimos 10 minutos y se caracteriza por actualizarse cada 5 s brindando una resolución suficiente para visualizar los datos de manera continua sin sobrecargar el navegador. En la figura 4.5 se puede observar esta componente cuando el estanque de acumulación de la planta está lleno y por lo tanto está en estado estacionario.

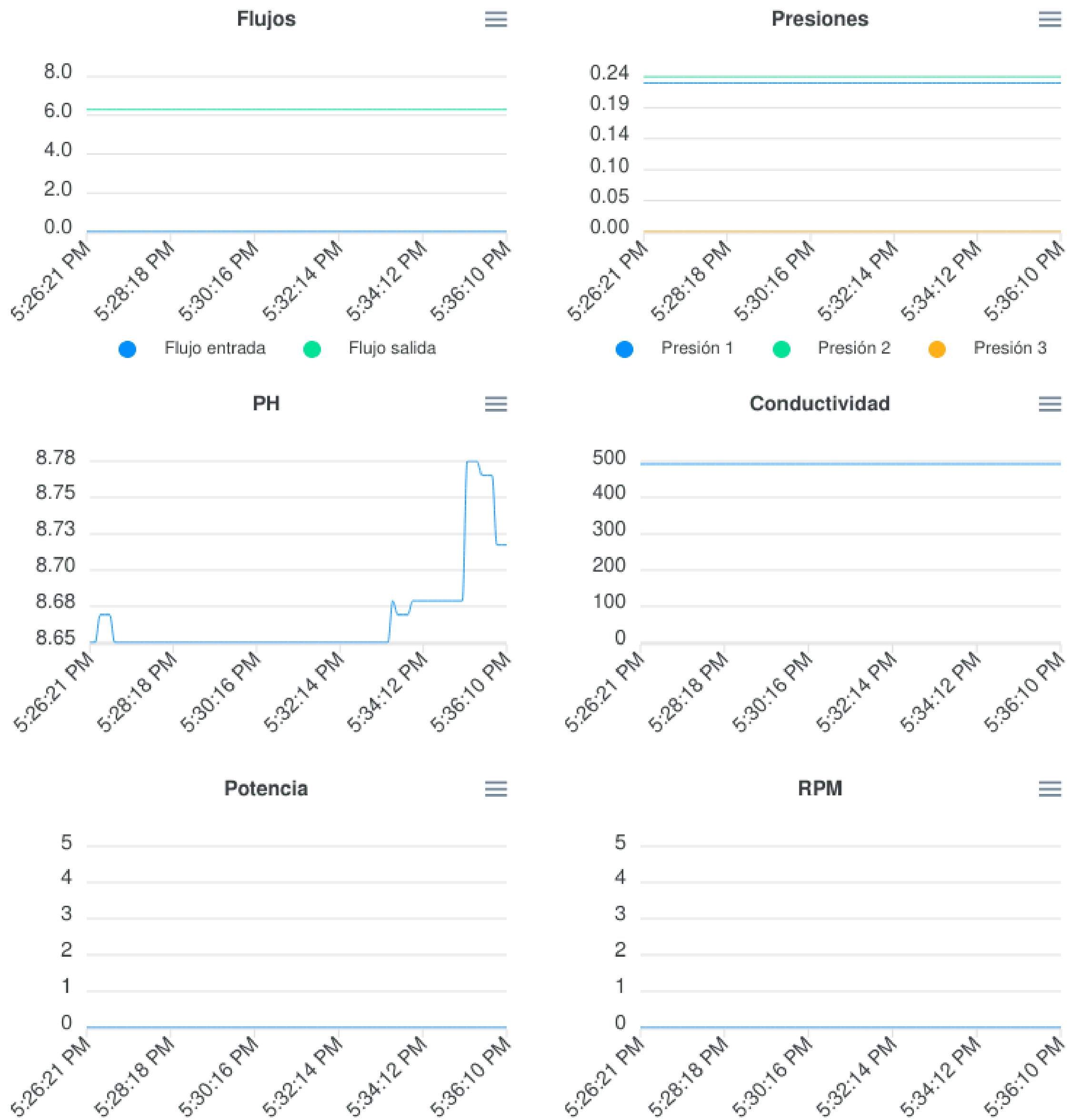


Figura 4.5: Componente de gráficos en tiempo real

En contraste, en la figura 4.6 se presenta la misma componente pero con la planta en estado de retrolavado de los filtros que corresponde a un estado dinámico de la planta y es cuando se aprecia la característica de la instantaneidad de la actualización de los gráficos.

Figura 4.6: Componente de gráficos en tiempo real

Al posicionar el puntero sobre uno de los gráficos se abre un pop-up en cada uno de ellos que indica el valor de las variables sobre el mismo gráfico, como se muestra en la figura 4.7, que permite estudiar el estado de la planta en un momento determinado.

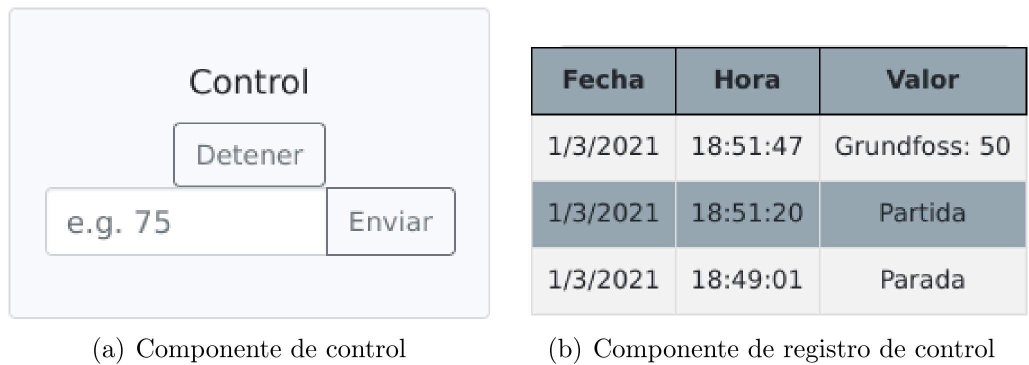
Figura 4.7: Pop-up en gráficos de tiempo real

Otra característica de esta componente es que cada gráfico es descargable como imagen o como CSV.

Interfaz de control del dispositivo

La interfaz de control consiste en, básicamente, dos componentes: la componente de control y la componente de datos de control enviados previamente. La componente de control es implementada como un formulario que permite el control de ciertas características del dispositivo. En la figura 4.8 se observa que el dispositivo posee únicamente dos características controlables: la partida y parada de la planta y la potencia de funcionamiento de una de las

bombas que controla el proceso.



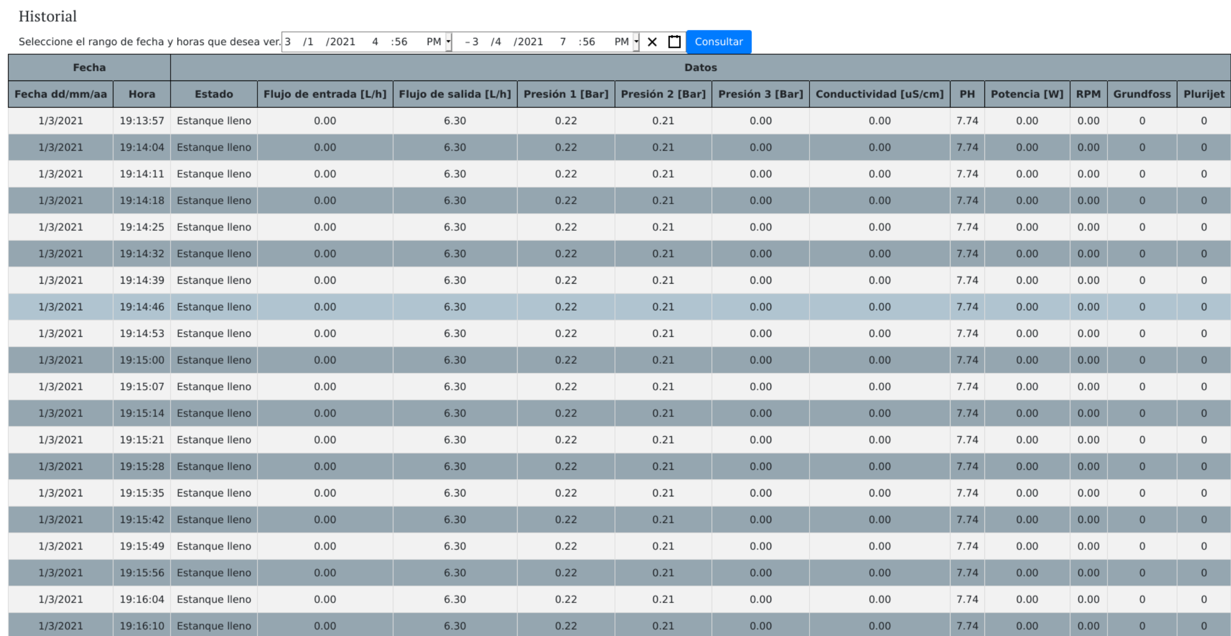
Fecha	Hora	Valor
1/3/2021	18:51:47	Grundfoss: 50
1/3/2021	18:51:20	Partida
1/3/2021	18:49:01	Parada

(a) Componente de control (b) Componente de registro de control

Figura 4.8: Componentes de control

Visualización de datos mediante tablas

Para la implementación de la interfaz de visualización de datos históricos mediante tablas se utilizan técnicas de *memorization* sobre el componente para disminuir el tiempo de carga de éste frente a un nuevo *render* cuando no han cambiado sus propiedades. Esta interfaz se presenta en la figura 4.9, donde se observa el menú de selección de el intervalo de tiempo a observar y la tabla con la información de respondida por la API al realizar la consulta.



Fecha		Datos												
Fecha dd/mm/aa	Hora	Estado	Flujo de entrada [L/h]	Flujo de salida [L/h]	Presión 1 [Bar]	Presión 2 [Bar]	Presión 3 [Bar]	Conductividad [uS/cm]	PH	Potencia [W]	RPM	Grundfoss	Plurijet
1/3/2021	19:13:57	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:14:04	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:14:11	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:14:18	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:14:25	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:14:32	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:14:39	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:14:46	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:14:53	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:15:00	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:15:07	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:15:14	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:15:21	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:15:28	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:15:35	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:15:42	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:15:49	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:15:56	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:16:04	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0
1/3/2021	19:16:10	Estanque lleno	0.00	6.30	0.22	0.21	0.00	0.00	7.74	0.00	0.00	0	0

Figura 4.9: Componente de historial

Visualización histórica de datos mediante gráficos

Esta componente es similar a la de los gráficos en tiempo real, con la diferencia de que en esta se presenta la información del estado del dispositivo en un tiempo determinado por la consulta realizada. En la figura 4.10 se presenta esta interfaz

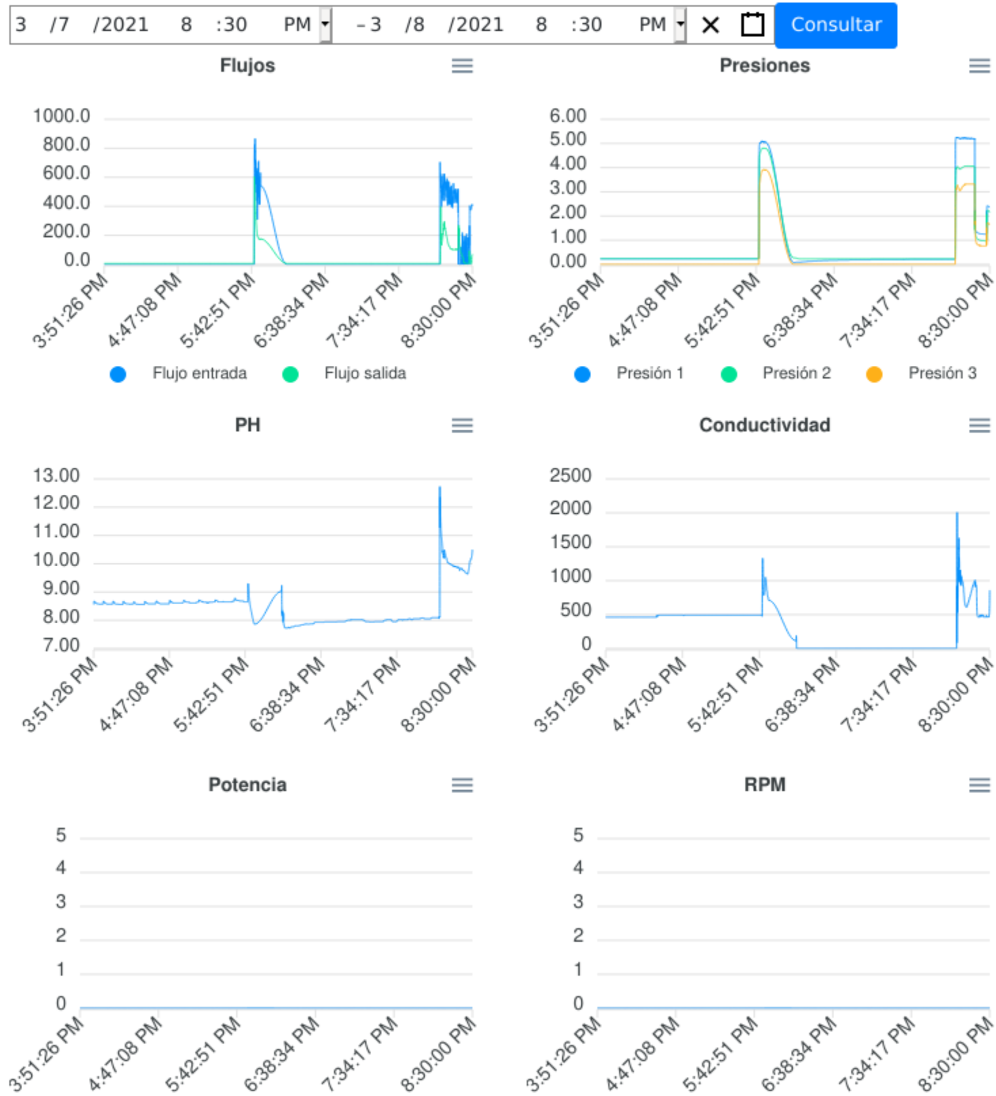


Figura 4.10: Componente de gráficos históricos

Otras componentes

Obviamente, existe otras componentes además de las ya mencionadas como barras de navegación, un menú que permite seleccionar qué dispositivo se desea monitorear y controlar, un formulario de la creación de dispositivos y más pero, al ser componentes bastante estándares para el diseño de una aplicación web, no se muestra su implementación de modo de atenerse a lo componentes relevantes para el trabajo de título.

Conclusión

En el presente trabajo de título se describe la concepción, diseño e implementación de una plataforma web que permite el monitoreo y control en tiempo real de distintas plantas desalinizadoras que poseen un numero arbitrario de variables a monitorear y/o controlar.

Uno de los beneficios más importantes que tiene la implementación de esta plataforma es que al recaudar y almacenar la información del estado de las distintas plantas, permite el estudio del comportamiento de estas a lo largo del tiempo, con el fin de poder detectar fallas remotamente y diagnosticar en base a esta información a qué se debe esta falla.

Otra de los beneficios que tiene la implementación actual es que al estar basada en servicios de computación en la nube, su escalabilidad, en cuanto a infraestructura, es prácticamente ilimitada, puesto que para la asignación de mayores recursos computacionales al sistema basta con acceder a la plataforma de AWS y solicitarlos. También es importante destacar que los costos de mantención de este sistema son extremadamente bajos y dependientes de la cantidad de plantas que se esté monitoreando y, por lo tanto, permiten un crecimiento del sistema en base a la demanda.

Cabe destacar la simplicidad de la agregación de una nueva planta a la plataforma de monitoreo y control ya que basta con agregar el dispositivo a la plataforma, configurar el dispositivo par que se conecte a AWS, que consiste en realizar ligeras modificaciones al programa de la Raspberry Pi (device.js) y, luego, crear las vistas necesarias para visualizar correctamente la información, reutilizando algunos de los componentes ya presentes en el sistema.

La principal dificultad del desarrollo del sistema viene asociada a su característica principal que corresponde a la flexibilidad que tiene el sistema en su totalidad para aceptar plantas que tiene una cantidad arbitraria de variables. Esta dificultad corresponde a que no se pueden definir estructuras fijas para tratar la información, problema que se resuelve mediante la utilización de una base de datos no relacional que permite almacenar información de largo variable y la utilización de una estructura de datos flexible.

Dicho lo anterior, es posible concluir que los objetivos generales y específicos presentados al comienzo del documento se cumplieron ya que se logra implementar una plataforma de monitoreo y control remoto de plantas desalinizadoras mediante IoT.

Queda pendiente la implementación de un sistema de alertas, una mejora en la forma en que se decide cuando enviar un mensaje desde el dispositivo para que no dependa solo de

una frecuencia de envío, sino también de cuánto varían las variables sensadas y la creación de cuenta con permisos de usuario y no administrador como es actualmente.

Finalmente, dada las características y la arquitectura de esta herramienta web, es posible agregar nuevas funcionalidades, características o conexiones a servicios externos, dependiendo de futuros requerimientos. Es por esto, que como trabajo futuro, se plantea la integración con otras plataformas de monitoreo de servicios complementarios, por ejemplo, servicios de monitoreo de la producción de energia solar, puesto que en más de alguna planta desalinizadora se utiliza la energía solar como fuente de alimentación eléctrica.

Bibliografía

[1] D Abramov D. Carlberg West R. West. hook-flow. https://github.com/donavon/hook-flow, Último acceso: 12-04-2021.

[2] BCN. Mega sequía, actualidad territorial. https://www.bcn.cl/siit/actualidad-territorial/mega_sequia, Último acceso: 01-03-2021.

[3] Ministerio del Medio Ambiente. Dirección meteorológica realiza el análisis más detallado del impacto del cambio climático en chile. https://mma.gob.cl/direccion-meteorologica-realiza-el-analisis-mas-detallado-del-impacto-del-cambio-climatico-en-chile, Último acceso: 23-02-2021.

[4] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*, volume 7. University of California, Irvine Irvine, 2000.

[5] Krešimir Grgić, Ivan Špeh, and Ivan Heđi. A web-based iot solution for monitoring data using mqtt protocol. In *2016 international conference on smart systems and technologies (SST)*, pages 249–253. IEEE, 2016.

[6] Do-Hun Kang, Min-Sung Park, Hyoung-Sub Kim, Da-young Kim, Sang-Hui Kim, Hyeon-Ju Son, and Sang-Gon Lee. Room temperature control and fire alarm/suppression iot service using mqtt on aws. In *2017 International Conference on Platform Technology and Service (PlatCon)*, pages 1–5. IEEE, 2017.

[7] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[8] NASA. Global climate change, vital signs of the planet. https://climate.nasa.gov/, Último acceso: 21-09-2020.

[9] React. Introducing hooks. https://reactjs.org/docs/hooks-intro.html, Último acceso: 16-04-2021.

[10] Salsabeel Shapsough, Mohannad Takrouri, Rached Dhaouadi, and Imran Zualkernan. An mqtt-based scalable architecture for remote monitoring and control of large-scale solar photovoltaic systems. In *International Conference on Smart Grid and Internet of Things*, pages 57–67. Springer, 2018.

[11] Feng Xia, Laurence T Yang, Lizhe Wang, and Alexey Vinel. Internet of things. *International journal of communication systems*, 25(9):1101, 2012.