

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA E INFORMÁTICA
CONCEPCIÓN - CHILE



“GreenGuardian: Back-end Aplicación Móvil”

JUAN JOSÉ MARAMBIO MARCHANT

**MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO EN INFORMÁTICA**

Profesor Guía: Rhoddy Angel Viveros Muñoz

Diciembre - 2024

DEDICATORIA

A quienes ya no están físicamente, pero cuya influencia perdura en mi día a día,
moldeando mis decisiones y acciones, dedico este logro. Su legado vive en mí.

AGRADECIMIENTOS

A mis padres, por su inmenso amor y por el gran esfuerzo que han realizado para brindarme la oportunidad de estudiar. Su apoyo económico y emocional ha sido fundamental para alcanzar este logro.

A mis compañeros, por las risas, las conversaciones y por convertirse en amigos que llevaré conmigo para siempre. Gracias por hacer de esta etapa universitaria una experiencia inolvidable.

RESUMEN

Esta memoria aborda el desarrollo del backend de un sistema para monitoreo y control de cultivos hidropónicos denominado GreenGuardian. La solución propuesta integra tecnologías de IoT y microservicios, centrada en garantizar una infraestructura robusta, escalable y eficiente para la gestión de datos en tiempo real. Los sensores conectados a AWS IoT Core recopilan parámetros esenciales como temperatura, pH y conductividad eléctrica, cuyos datos son procesados y almacenados en DynamoDB mediante microservicios desarrollados en Spring Boot.

El diseño backend asegura la modularidad y el manejo eficiente de la información, permitiendo la automatización de procesos a través de mensajes MQTT. El sistema sienta las bases para una solución adaptable y escalable en el ámbito agrícola, ofreciendo beneficios tangibles en la gestión de cultivos hidropónicos.

Palabras Clave—IOT; microservicios; backend; hidroponía; automatización.

ABSTRACT

This thesis focuses on the backend development of a monitoring and control system for hydroponic crops named GreenGuardian. The proposed solution integrates IoT and microservices technologies, emphasizing a robust, scalable, and efficient infrastructure for real-time data management. Sensors connected to AWS IoT Core collect key parameters such as temperature, pH, and electrical conductivity, with data processed and stored in DynamoDB through microservices developed in Spring Boot.

The backend design ensures modularity and efficient information handling, enabling process automation via MQTT messages. The system establishes a foundation for a scalable and adaptable solution in the agricultural domain, delivering tangible benefits in hydroponic crop management.

Keywords—IOT; microservices; backend; hydroponics; automation.

GLOSARIO

AWS IoT Core: Servicio de Amazon para conectar dispositivos IoT de manera segura y gestionar la comunicación entre ellos.

DynamoDB: Base de datos NoSQL proporcionada por AWS, diseñada para aplicaciones que requieren baja latencia.

IoT (Internet of Things): Red de dispositivos conectados que recopilan y comparten datos.

MQTT (Message Queuing Telemetry Transport): Protocolo de mensajería ligero usado en sistemas IoT.

Spring Boot: Framework de Java para crear aplicaciones backend robustas y escalables.

INDICE DE CONTENIDOS

Tabla de contenido

RESUMEN	4
ABSTRACT	4
GLOSARIO	5
INDICE DE FIGURAS	9
INDICE DE TABLAS	10
INTRODUCCIÓN	11
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	12
1.1 Organización: GreenGuardian	12
1.2 Problemática	13
1.2.1 Actores involucrados	15
1.2.1.1 Segmento geográfico	15
1.2.1.2 Segmento demográfico	16
1.2.1.3 Segmento Final por comportamiento	16
1.3 Objetivos y alcances	17
1.3.1 Objetivos específicos de la Memoria	17
1.4 Importancia de la problemática	17
1.5 Motivación de la Solución	19
CAPÍTULO 2: MARCO CONCEPTUAL	20
2.1 Arquitectura de Microservicios (MSA) vs Arquitectura Monolítica (MA)	20
2.1.1 Arquitectura Monolítica (MA)	20
2.1.2 Arquitectura de Microservicios (MSA)	21
2.1.3 Ventajas y Desventajas de las Arquitecturas	21
2.2 Comparativa de Frameworks Backend	22

2.2.1 Comparativa: Eficiencia y Confiabilidad	22
2.3) dependencias de spring boot	25
2.3.1) Dependencias Básicas de Spring Boot:.....	25
2.3.2) Spring Cloud y Microservicios:	25
2.3.3) Dependencias de Seguridad:.....	25
2.3.4) Dependencias de JWT:	25
2.3.5) Dependencias de Base de Datos y AWS:	26
2.3.6) Dependencias de Kotlin:.....	26
2.3.7) Dependencia de Lombok:	26
2.4 CONCLUSIONES PARCIALES	¡Error! Marcador no definido.
CAPÍTULO 3: PROPUESTA DE SOLUCION	27
3.1) introducción.....	27
3.2) Diseño general del sistema	27
3.2.1) Diagrama de Casos de Uso.....	27
3.3 Componentes del Backend.....	30
3.3.1 Servicio de Autenticación y Autorización	30
3.3.2 Servicio de Usuarios	31
3.3.3 Servicio de Estanques	31
3.3.4 Servicio de Tareas	31
3.3.5 Servicio de Sensores.....	31
3.3.6 Servicio de Datos de Estanques	32
3.5 Configuración del Entorno	32
3.5.1 Despliegue en AWS	32
3.6 APIs Y Endpoints	32
3.7 Resultados Esperados.....	36
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN	38
4.1 Introducción a la Validación	38

4.2 Metodología de Validación	38
4.3 Resultados de las Pruebas.....	39
4.4 Datos del Entorno de Validación.....	44
4.5 Análisis de Impacto	44
CAPÍTULO 5: CONCLUSIONES	46
5.2 Trabajos Futuros	47
REFERENCIAS BIBLIOGRÁFICAS	48
REFERENCIAS BIBLIOGRÁFICAS EN NORMA APA:	48

INDICE DE FIGURAS

Figura 1 Logo de la organización - "GreenGuardian". Fuente: Elaboración Propia.....	12
Figura 2 El instituto mundial de recursos sitúa a Chile como extrema escasez de agua a nivel mundial. Fuente: wri.org/aqueduct	14
Figura 3 Consulta GET, tiempo de procesamiento dependiendo de números de peticiones. Fuente: Choma(2023)	23
Figura 4 Porcentaje solicitudes GET erróneas. Fuente: Choma(2023)	24
Figura 5 Caso de Usos "Sistema GreenGuardian". Fuente: Elaboración Propia	28
Figura 6 Arquitectura de Servicio: Proyecto GreenGuardian. Fuente: Elaboración Propia.	¡Error! Marcador no definido.

INDICE DE TABLAS

Tabla 1 <i>Endpoint</i> de la aplicacion. Fuente: Elaboración propia.....	32
Tabla 2 Prueba de <i>endpoint</i> del authController con 10 usuarios simultáneos.	39
Tabla 3 Prueba de <i>endpoint</i> del usuarioController con 10 usuarios simultáneos. Fuente: elaboración Propia.	39
Tabla 4 Prueba de <i>endpoint</i> del estanqueController con 10 usuarios simultáneos. Fuente: elaboración Propia	41
Tabla 5 Prueba de <i>endpoint</i> del estanqueNoSqlController con 10 usuarios simultáneos. Fuente: elaboración Propia.....	42
Tabla 6 Prueba de <i>endpoint</i> del mqttController con 10 usuarios simultáneos. Fuente: elaboración Propia...	43

INTRODUCCIÓN

Esta memoria surge como respuesta a una de las mayores crisis que enfrentaremos a nivel nacional: la escasez de agua. Según estudios recientes, Chile es uno de los países más afectados por el cambio climático en América Latina, enfrentando una disminución en la disponibilidad de agua del 37% durante los últimos 30 años (CR2, 2020). Además, proyecciones del Banco Mundial (2021) señalan que para el año 2050, la demanda de agua podría superar ampliamente la oferta, lo que llevaría al país a una crisis hídrica total. Medios de comunicación como Radio Bio Bio, Canal Mega y Diario La Tercera también han alertado sobre este escenario, señalando que actualmente un 72% del país ya está afectado por sequías, impactando a más de 12 millones de personas (La Tercera, 2022). Estos datos subrayan la urgencia de abordar esta problemática a través de soluciones innovadoras y sostenibles.

A pesar de ser una problemática global y en gran medida irreversible, existen soluciones para mitigar este problema, y uno de los mayores culpables está identificado: el consumo de agua en los cultivos tradicionales, que representa cerca del 70% del uso de agua en Chile (Oficina de Estudios y Políticas Agrarias [ODEPA], 2022). La urgencia de una solución que reduzca drásticamente este consumo nunca ha sido tan apremiante.

En este contexto, GreenGuardian se presenta como una propuesta que fusiona dos áreas fundamentales para la vida humana: la naturaleza y la tecnología. GreenGuardian introduce un sistema de automatización de cultivos hidropónicos que no solo reduce el consumo de agua, sino que lo optimiza, ofreciendo una alternativa sostenible. Según un estudio del gobierno de México (Secretaría de Agricultura y Desarrollo Rural, 2022), los cultivos hidropónicos pueden ser hasta un 90% más eficientes en el uso del agua, lo que los convierte en una solución de gran impacto para enfrentar la crisis hídrica.

Para asegurar su eficacia, GreenGuardian integra tecnologías avanzadas como redes neuronales, Internet de las Cosas (IoT) e Inteligencia Artificial, potenciando la automatización y eficiencia de estos cultivos. Este documento detalla cómo estas innovaciones se combinan en una propuesta real y efectiva para preservar uno de nuestros recursos más valiosos: el agua.

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA

1.1 Organización: GreenGuardian

Esta memoria se desarrollará en el contexto organizacional de **GreenGuardian**, una entidad comprometida con mitigar la problemática hídrica descrita. Para ello, el proyecto seguirá una línea de **gestión de proyectos informáticos** utilizando la metodología ágil **Scrum**, enfocada específicamente en implementar una solución eficiente y sostenible.



Figura 1 Logo de la organización - "GreenGuardian". Fuente: Elaboración Propia

1.2 Problemática

La escasez hídrica en Chile ha dejado de ser un problema latente para convertirse en una realidad inminente y alarmante, con un impacto que ya se percibe en gran parte del territorio, y hasta los medios internacionales lo destacan, como señala un informe de *iAgua* (2024). Diversos estudios e informes de entidades de renombre advierten sobre la gravedad de esta crisis, la cual podría convertirse en una de las más grandes que nuestro país enfrentará en las próximas décadas (CR2, 2020). Según estimaciones de organismos de investigación y medios informativos como Radio Bio Bio, Canal Mega y Diario La Tercera, se prevé que hacia el año 2050 muchas regiones del país se verán afectadas por una grave falta de agua, y existe una posibilidad real de que, a largo plazo, esta escasez se extienda a nivel nacional (Radio Bio Bio, 2024).

La Facultad de Ciencias Forestales de la Universidad de Chile, en un comunicado de marzo de 2022, señala que “Chile lidera la crisis hídrica en América Latina” (Universidad de Chile, 2022). Este informe subraya que la mala gestión estructural del agua, combinada con los efectos de la crisis climática, son factores clave en la falta de disponibilidad del recurso en el país. Según los datos presentados, 53% de las comunas chilenas ya se encuentran declaradas en situación de sequía, y muchas de ellas viven bajo restricciones de uso de agua, un racionamiento que alcanza incluso a sectores de la capital, como el sector oriente de Santiago.

Asimismo, la Universidad de Concepción ha alertado sobre el impacto transversal de esta crisis en el país, señalando que afecta la economía, el medio ambiente y la sociedad. En su comunicado, la universidad enfatiza que esta crisis es resultado de una combinación de factores críticos, que incluyen una sequía prolongada, variabilidad climática, sobreexplotación de los recursos hídricos y una gestión inadecuada del agua (Universidad de Concepción, 2022). La situación exige, según este informe, que se adopten medidas concretas para garantizar la disponibilidad del recurso en todas las áreas del país.

La Fundación Heinrich Böll, en una reflexión publicada con motivo del Día Mundial del Agua 2024, enfatiza que esta crisis no solo representa un desafío ambiental, sino que también amenaza la seguridad alimentaria y el bienestar de las comunidades, especialmente en las zonas rurales. De acuerdo con la fundación, la sobreexplotación de los recursos hídricos y la contaminación están agravando la situación, dificultando el acceso a agua segura para las comunidades más vulnerables (Fundación Heinrich Böll, 2024). Esta combinación de factores no solo erosiona el ecosistema, sino que exacerba las desigualdades sociales al limitar el acceso a un recurso básico para la vida y el desarrollo sostenible.

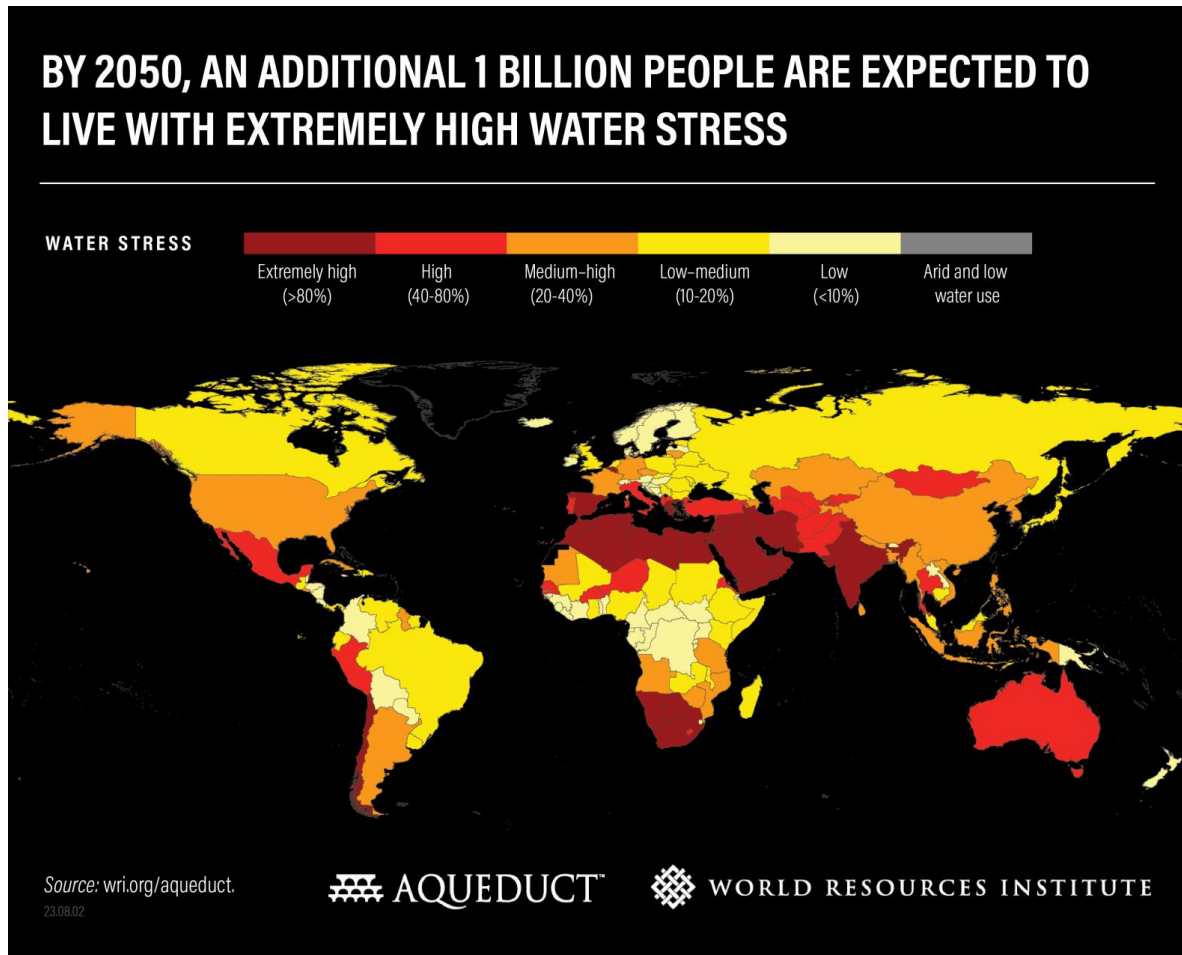


Figura 2 Estrés Hídrico 2050 Fuente: *World Resources Institute*

Toda la información proporcionada por entidades respaldadas en datos oficiales coincide en que nuestro país enfrenta, o enfrentará, una crisis que probablemente será una de las más grandes de nuestra historia como nación. Algo en lo que todos los informes coinciden es en que un factor clave detrás de esta escasez es la mala gestión y distribución de los recursos hídricos, lo cual afecta directamente su eficiencia. Esta ineficacia impacta no solo en el ámbito agrícola, sino también en la economía, amplificando las repercusiones de la crisis.

1.2.1 Actores involucrados

En el contexto del problema principal, la identificación de actores relevantes abarca diversos sectores y niveles de influencia. En primer lugar, se encuentra el ciudadano común, quien se verá afectado directamente por la escasez de agua y es un potencial usuario y beneficiario de soluciones sostenibles. A nivel de empresas y organizaciones, están aquellas asociadas a la agricultura y al uso de recursos hídricos, especialmente las que dependen de cultivos tradicionales, ya que son las principales consumidoras de agua y las más vulnerables ante las restricciones.

También figuran las entidades ambientales y de conservación del agua, que abogan por el uso sostenible de los recursos y pueden colaborar en la implementación de nuevas tecnologías para la eficiencia hídrica. Finalmente, las entidades gubernamentales desempeñan un papel central, ya que son responsables de formular políticas, regulaciones y soluciones que permitan mitigar la crisis a nivel nacional. Estas instituciones no solo deben ofrecer apoyo, sino también generar incentivos para la adopción de sistemas eficientes, como los cultivos hidropónicos automatizados de GreenGuardian, que contribuyen a una gestión más sostenible del agua.

1.2.1.1 Segmento geográfico

Chile enfrenta una de las crisis hídricas más significativas de su historia, una problemática que ya afecta a 76% del territorio nacional (La Tercera, 2022). Y se prevé que empeore, con severas consecuencias para regiones clave como el norte, que podría enfrentar un recorte adicional de hasta 50% en el suministro de agua para 2030 (Universidad de Chile, 2022).

Esta situación afecta a aproximadamente 15.043.440 personas en el país, quienes están experimentando, o experimentarán pronto, el impacto de la escasez de agua en múltiples aspectos de sus vidas.

En cuanto al ámbito agrícola, el sector depende intensamente del agua y se verá especialmente impactado. Chile cuenta con una extensión gigante de pequeñas y medianas empresas (Pymes), muchas de las cuales están vinculadas a la agricultura. Estas empresas no solo enfrentan el desafío de la escasez de agua, sino también la necesidad de adaptarse a prácticas más sostenibles y eficientes para continuar operando en el contexto de esta crisis. GreenGuardian se posiciona como una solución clave para ayudar a estas empresas a reducir el consumo de agua a través de sistemas de cultivo hidropónico y tecnologías avanzadas, proporcionando una opción más sostenible y efectiva para la producción agrícola en zonas afectadas.

1.2.1.2 Segmento demográfico

En el contexto de personas naturales, nuestro objetivo primordial son individuos de entre 25 y 45 años, sin distinción de género, que cuenten con una estabilidad económica y una madurez en temas ambientales y de conciencia sobre el uso del agua. En términos económicos, queremos enfocar nuestros esfuerzos en los segmentos de clase media y alta, dado que nuestro producto es escalable y se orienta a personas dispuestas a invertir en soluciones sostenibles (La Tercera, 2023). Aunque el nivel educativo no es un factor determinante, es importante que al menos una persona en el hogar donde se implementará el producto posea conocimientos básicos de internet y tecnología para asegurar su óptimo funcionamiento.

De acuerdo con estas características, de las 15.043.440 personas afectadas por la escasez hídrica, aproximadamente el 43% se encuentra dentro del perfil socioeconómico al que queremos llegar, lo que representa alrededor de 6.468.679 personas en todo el país (La Tercera, 2023). En cuanto a las empresas, de acuerdo con el Censo Nacional Agropecuario del INE de 2007, en Chile existen más de 300.000 explotaciones silvoagropecuarias. Estas forman parte de las 900.000 empresas en el país y serán nuestro principal foco de acción para la implementación de GreenGuardian. Al concentrar nuestros esfuerzos en este sector agropecuario, apuntamos a brindar soluciones sostenibles y eficientes en el uso del agua a quienes más lo necesitan en el contexto de la crisis hídrica actual (Ministerio de Agricultura, 2007).

1.2.1.3 Segmento Final por comportamiento

Nuestro objetivo son personas de entre 25 y 45 años, de clase media y alta, con estabilidad económica y conciencia ambiental, interesadas en contribuir al cuidado del medio ambiente, cosechar como hobby y tener su propio sistema de cultivo. Este segmento se caracteriza por buscar soluciones innovadoras, saliendo de su zona de confort. Basado en encuestas y análisis de clientes, se estima que 80% de esta población – unas 5.174.943 personas– estaría interesada en implementar un sistema de cultivo en casa, siempre y cuando este sea fácil de usar y eficiente en el consumo de agua.

En Chile existen más de 300.000 explotaciones silvoagropecuarias. De estas, aproximadamente el 90% invierte regularmente en tecnología, representando un total de 270.000 empresas interesadas en adoptar soluciones que optimicen el uso de recursos. Estas empresas son innovadoras y buscan constantemente la eficiencia tecnológica para mejorar su rendimiento.

En conclusión, el mercado final tentativo de GreenGuardian es de 5.174.943 personas y 270.000 empresas.

1.3 Objetivos y alcances

Los objetivos trazados para este proyecto incluyen optimizar la eficiencia en el uso de recursos hídricos mediante tecnologías avanzadas como IoT, IA y redes neuronales, mitigar el impacto ambiental de la agricultura tradicional a través de alternativas sostenibles, facilitar el autocultivo a pequeña escala para usuarios domésticos, y empoderar a empresas agrícolas para adoptar tecnología innovadora que mejore su sostenibilidad y productividad.

Mientras que los alcances abarcan la escalabilidad del producto para adaptarse a diferentes necesidades, desde soluciones para el hogar hasta implementaciones agrícolas a gran escala, la educación y sensibilización sobre la crisis hídrica y el uso eficiente del agua, así como la integración de tecnología avanzada para mantener una propuesta de valor actualizada y efectiva.

1.3.1 Objetivos específicos de la Memoria

Los objetivos específicos que abarcara esta memoria están delimitados al desarrollo y correcto funcionamiento del back-end de greenguardian:

- Diseñar y desarrollar la arquitectura backend que soporte las funcionalidades clave de Green Guardian, asegurando escalabilidad y eficiencia.
- Implementar y documentar servicios RESTful que faciliten la comunicación entre el backend y las interfaces frontend o dispositivos IoT.
- Integrar bases de datos optimizadas para almacenar y gestionar la información relevante de Green Guardian, priorizando consultas rápidas y la consistencia de datos.
- Desarrollar mecanismos de autenticación y autorización robustos, garantizando la seguridad en el acceso a los servicios backend.
- Optimizar el rendimiento del backend, utilizando herramientas de monitoreo y métricas para identificar y resolver cuellos de botella.

1.4 Importancia de la problemática

La problemática que enfrentamos a nivel nacional, respaldada por múltiples estudios relevantes y creíbles, es una de las mayores amenazas en nuestra historia: la escasez y posible agotamiento del recurso más vital para la vida humana, el agua. La falta de soluciones efectivas impactaría de manera directa a cada ciudadano, afectando áreas clave como la salud, la economía y el bienestar social. A nivel nacional, las consecuencias serían devastadoras, afectando nuestra economía al poner en riesgo sectores de exportación fundamentales como el agrícola y el forestal. Esto podría conducir a la quiebra de empresas que dependen del agua como recurso principal. Esto podría derivar en situaciones críticas comparables a aquellas experimentadas en regiones con extrema escasez de agua, como algunas áreas de África, donde el acceso al agua es limitado.

Si bien no existe una solución definitiva a esta crisis, hay opciones que pueden mitigar sus efectos, especialmente en el uso eficiente del agua en la agricultura, sector que representa cerca del 70% del consumo hídrico (La Tercera, 2022). Esta memoria busca aportar a la investigación y a la exploración de nuevos mercados para GreenGuardian, promoviendo el uso eficiente y sostenible del agua en la agricultura

1.5 Motivación de la Solución

La motivación detrás de la creación de este producto se basa en el compromiso de generar un aporte significativo en la nación frente a la que podría convertirse en la mayor crisis de su historia: la escasez de agua. Las proyecciones indican que, De mantenerse las condiciones actuales, para el año 2050 el país podría enfrentar un agotamiento crítico de este recurso vital, comprometiendo el bienestar de las futuras generaciones. En respuesta a esta alarmante realidad, GreenGuardian surge como un sistema integral que combina innovación tecnológica y sostenibilidad, diseñado para mitigar esta problemática mediante la optimización en el uso del agua y la promoción de prácticas agrícolas responsables.

CAPÍTULO 2: MARCO CONCEPTUAL

2.1 Arquitectura de Microservicios (MSA) vs Arquitectura Monolítica (MA)

La arquitectura de software desempeña un papel crucial en la organización y el rendimiento de los sistemas tecnológicos modernos. A lo largo del tiempo, han surgido diferentes enfoques arquitectónicos para satisfacer las necesidades cambiantes de las aplicaciones, siendo la Arquitectura Monolítica (MA) y la Arquitectura de Microservicios (MSA) dos de los más representativos.

La Arquitectura Monolítica (MA) fue el estándar tradicional para desarrollar sistemas de software, caracterizada por agrupar todas las funcionalidades dentro de una única aplicación cohesiva. Este enfoque tiene ventajas como la facilidad de desarrollo inicial, pero tiende a presentar dificultades significativas en términos de mantenimiento y escalabilidad a medida que las aplicaciones crecen en complejidad.

Por otro lado, la Arquitectura de Microservicios (MSA), un enfoque más reciente, divide los sistemas en componentes más pequeños e independientes llamados microservicios. Este modelo aborda las limitaciones de los monolitos al ofrecer modularidad, escalabilidad y flexibilidad tecnológica, lo que lo convierte en una solución popular para aplicaciones distribuidas y modernas.

2.1.1 Arquitectura Monolítica (MA)

La Arquitectura Monolítica organiza todo el sistema en una única aplicación ejecutable que encapsula tanto la lógica de negocio como los servicios de datos. Entre sus características más relevantes están:

Simplicidad inicial:

El desarrollo, despliegue y prueba de una aplicación monolítica suelen ser sencillos en las etapas iniciales, ya que todo se encuentra en un solo lugar.

Desventajas en aplicaciones grandes:

A medida que la aplicación crece, el código se vuelve más difícil de gestionar y modificar.

Los tiempos de despliegue aumentan significativamente, limitando la capacidad de realizar actualizaciones rápidas.

Escalabilidad limitada:

La única forma de escalar un sistema monolítico es duplicando la aplicación completa, lo cual puede generar sobrecargas innecesarias.

Aunque estas características hacen que las arquitecturas monolíticas sean útiles para aplicaciones pequeñas o de alcance limitado, presentan desventajas que dificultan su evolución a largo plazo.

2.1.2 Arquitectura de Microservicios (MSA)

La Arquitectura de Microservicios se basa en la división del sistema en componentes independientes que funcionan como servicios autónomos. Cada microservicio maneja una funcionalidad específica y opera de forma desacoplada del resto del sistema. Sus características incluyen:

Modularidad:

Los microservicios son componentes pequeños e independientes, lo que facilita el mantenimiento y la evolución del sistema.

Despliegue independiente:

Cada microservicio puede desplegarse, escalarse y actualizarse sin afectar al resto del sistema.

Flexibilidad tecnológica:

Es posible implementar cada microservicio con el lenguaje y las herramientas más adecuadas para su función específica.

Escalabilidad eficiente:

Los microservicios permiten escalar solo las partes del sistema que tienen mayor demanda, optimizando el uso de recursos.

Estas características hacen que los microservicios sean ideales para sistemas complejos que requieren escalabilidad, tolerancia a fallos y capacidad de evolución.

2.1.3 Ventajas y Desventajas de las Arquitecturas

A continuación, se detallan las ventajas y desventajas de ambas arquitecturas:

Ventajas de MA:

- Desarrollo y despliegue más sencillo en aplicaciones pequeñas.
- Facilidad para gestionar sistemas con funcionalidad limitada.

Desventajas de MA:

- Baja escalabilidad en sistemas grandes.
- Incremento en la complejidad del código y en los tiempos de despliegue.

- Limitaciones para realizar actualizaciones y añadir nuevas funcionalidades.

Ventajas de MSA:

- Escalabilidad modular y eficiente.
- Independencia en el desarrollo y mantenimiento de cada microservicio.
- Mayor tolerancia a fallos, ya que un error en un microservicio no afecta al resto del sistema.
- Flexibilidad para adoptar diferentes tecnologías en los servicios.

Desventajas de MSA:

- Complejidad en la gestión de múltiples servicios.
- Latencia en la comunicación entre microservicios.
- Mayor costo inicial debido a la infraestructura necesaria para orquestación y monitoreo.

2.2 Comparativa de *Frameworks Backend*

El presente marco conceptual analiza las diferencias clave entre tres frameworks backend ampliamente utilizados: Express, Django y Spring Boot, destacando sus fortalezas y debilidades en términos de eficiencia y confiabilidad. La información presentada se fundamenta en los estudios realizados por Choma et al. (2023) *Frameworks Evaluados**

Express: *Framework* minimalista basado en Node.js, optimizado para la escalabilidad y flexibilidad.

Django: *Framework* de alto nivel desarrollado en Python, enfocado en simplicidad y rapidez para prototipos.

Spring Boot: *Framework* robusto del ecosistema Java, diseñado para aplicaciones empresariales de alto rendimiento.

2.2.1 Comparativa: Eficiencia y Confiabilidad

Eficiencia

- Spring Boot:

Rendimiento superior: Spring Boot se destaca por su alta eficiencia en el procesamiento de solicitudes HTTP. En pruebas realizadas, mostró la mejor

velocidad de ejecución de solicitudes para cargas moderadas (1,000 a 8,000 usuarios simultáneos), lo cual es clave en aplicaciones de misión crítica.

Optimización preintegrada: Incluye mecanismos de optimización que permiten que las aplicaciones construidas con este framework gestionen de manera eficiente las operaciones sobre bases de datos y la comunicación entre servicios, lo que reduce la necesidad de configuraciones adicionales.

Escalabilidad: Spring Boot permite escalar los servicios a medida que la demanda aumenta, lo que lo hace adecuado para sistemas que requieren alta disponibilidad y escalabilidad horizontal.

- **Express:** Aunque más lento que Spring Boot bajo cargas moderadas, presenta un mejor desempeño frente a altas demandas (16,000 usuarios), gracias a su mecanismo de clustering personalizado que permite distribuir la carga de manera más eficiente entre varios nodos.
- **Django:** Mostró el peor desempeño en términos de tiempos de respuesta, con una duración significativamente mayor que los otros frameworks, especialmente bajo altas cargas. Esto se debe a su arquitectura, que asigna un hilo por solicitud, lo que incrementa el consumo de memoria y la latencia en situaciones de alto tráfico.

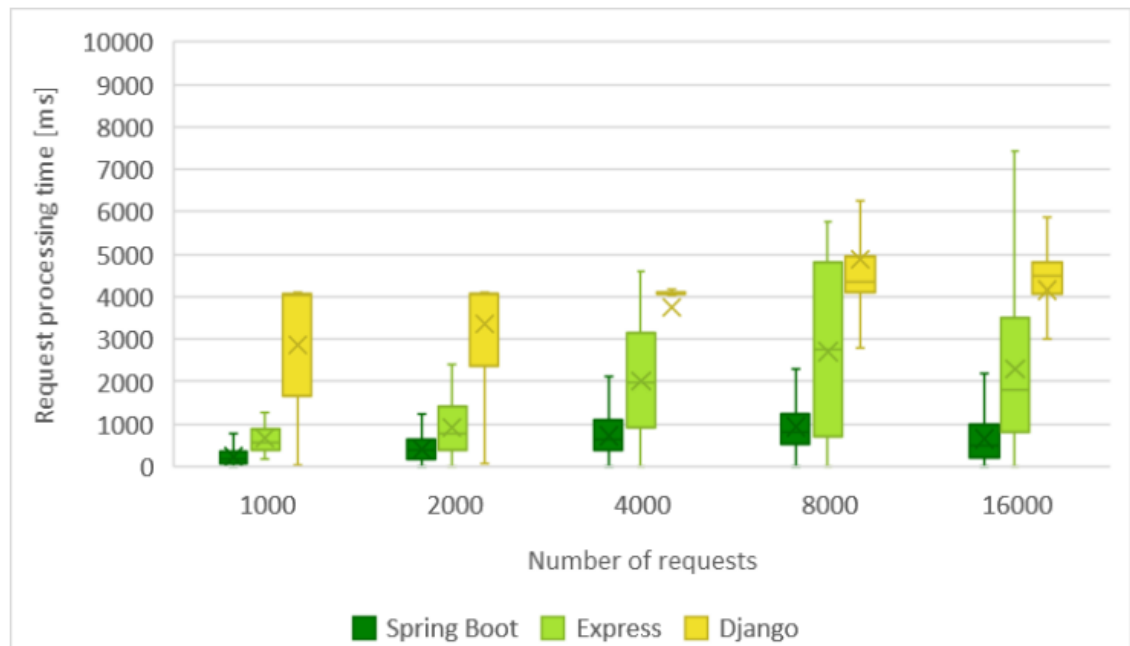


Figura 3 Consulta GET, tiempo de procesamiento dependiendo de números de peticiones. Fuente: Choma(2023)

Confiabilidad

- Spring Boot:

Alta fiabilidad bajo carga moderada: En las pruebas realizadas, Spring Boot mostró una tasa mínima de errores en escenarios de carga moderada. Sin embargo, bajo cargas extremas (16,000 usuarios), la tasa de errores aumentó a un 51.65%. A pesar de esto, sigue siendo una opción sólida para aplicaciones empresariales que exigen estabilidad y consistencia.

Integración con herramientas de monitoreo: La integración con Spring Boot Actuator y otros componentes de Spring Cloud facilita el monitoreo y la gestión de la fiabilidad de los microservicios, permitiendo una detección temprana de fallos y una respuesta eficiente ante problemas operativos.

- Express: Aunque inicialmente presenta mayor estabilidad, su tasa de errores incrementa gradualmente a medida que la carga aumenta, alcanzando un 34.86% en escenarios extremos. Sin embargo, en comparación con otros frameworks, Express mantiene una tasa de error relativamente baja en condiciones de carga extrema debido a su arquitectura minimalista y su enfoque en la eficiencia.
- Django: La fiabilidad fue la más baja, con altas tasas de fallos incluso bajo cargas menores. En los escenarios más exigentes (16,000 usuarios), Django alcanzó un 97% de errores, lo que lo hace menos adecuado para aplicaciones de alta disponibilidad.

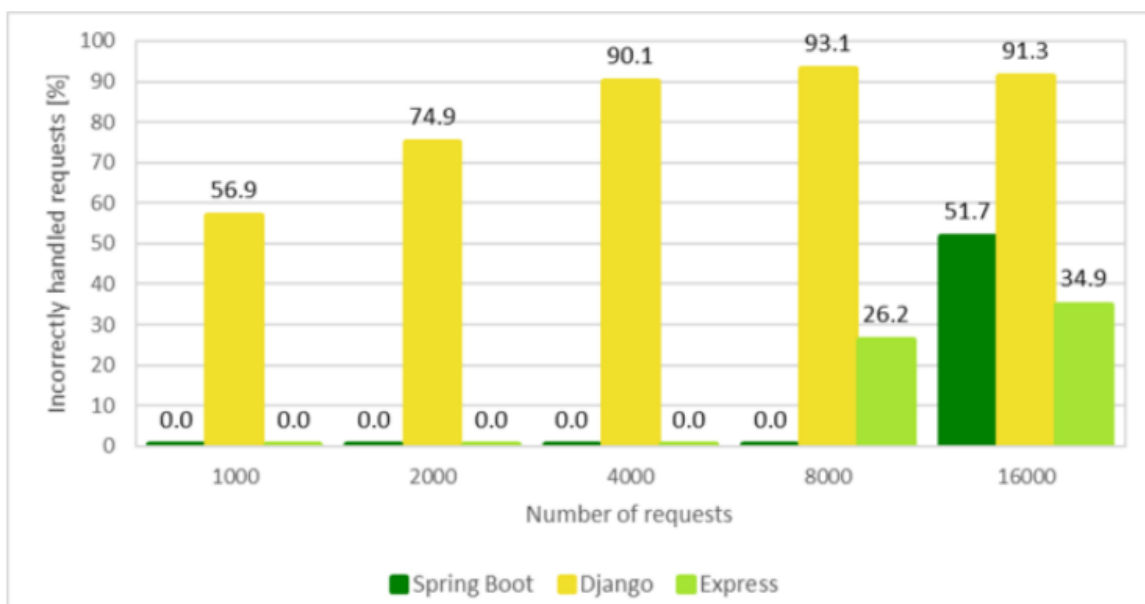


Figura 4 Porcentaje solicitudes *GET* erróneas. Fuente: Choma(2023)

2.3) dependencias de spring boot

2.3.1) Dependencias Básicas de Spring Boot:

- spring-boot-starter: Incluye la funcionalidad básica para iniciar cualquier aplicación Spring Boot.
- spring-boot-starter-web: Proporciona herramientas para desarrollar aplicaciones web RESTful, incluyendo un servidor embebido (Tomcat, Jetty o Undertow).
- spring-boot-starter-data-jpa: Soporte para JPA y Hibernate, facilitando el acceso a bases de datos relacionales como PostgreSQL.
- spring-boot-starter-actuator: Proporciona herramientas para monitorear y administrar tu aplicación (como métricas y endpoints de salud).
- spring-boot-starter-test: Dependencia para realizar pruebas unitarias y de integración con herramientas como JUnit y Mockito.

2.3.2) Spring Cloud y Microservicios:

- spring-cloud-config-server: Permite la configuración centralizada en microservicios.
- spring-cloud-starter-config: Implementación del cliente para consumir configuraciones desde un servidor de configuración.
- spring-cloud-starter-netflix-eureka-server: Implementa un servidor de descubrimiento de servicios con Eureka.
- spring-cloud-starter-netflix-eureka-client: Permite que el microservicio sea un cliente de Eureka para registrarse y descubrir otros servicios.
- spring-cloud-starter-openfeign: Facilita la creación de clientes HTTP declarativos para la comunicación entre microservicios.
- spring-cloud-starter-gateway: Proporciona una puerta de enlace para enrutar las solicitudes a otros microservicios.

2.3.3) Dependencias de Seguridad:

- spring-boot-starter-security: Implementa seguridad en la aplicación, permitiendo configuraciones de autenticación y autorización.

2.3.4) Dependencias de JWT:

- jjwt-api, jjwt-impl, jjwt-jackson: Librerías para generar y validar JSON Web Tokens (JWT), útiles para la gestión de autenticación.

2.3.5) Dependencias de Base de Datos y AWS:

- postgresql: Driver para trabajar con bases de datos PostgreSQL en tiempo de ejecución.
- aws-java-sdk-dynamodb: SDK para interactuar con DynamoDB de AWS.
- aws-iot-device-sdk-java: SDK para interactuar con dispositivos IoT en AWS.

2.3.6) Dependencias de Kotlin:

- kotlin-reflect, kotlin-stdlib: Bibliotecas esenciales para usar Kotlin en proyectos Spring Boot.
- kotlin-test-junit5: Librería para realizar pruebas unitarias con JUnit5 en Kotlin.

2.3.7) Dependencia de Lombok:

- lombok: Herramienta para generar código repetitivo automáticamente, como getters, setters y constructores, facilitando la programación en Java/Kotlin.

CAPÍTULO 3: PROPUESTA DE SOLUCION

3.1) introducción

El desarrollo del backend para GreenGuardian se basará en una arquitectura de microservicios utilizando Spring Boot y Kotlin como tecnologías principales. Estas herramientas han sido seleccionadas debido a su capacidad para soportar cargas en tiempo real, integración con servicios en la nube y modularidad, tal como se detalla en el marco conceptual

3.2) Diseño general del sistema

El diseño del sistema GreenGuardian está estructurado para cubrir las necesidades de monitoreo, análisis y gestión de cultivos hidropónicos, integrando tecnologías de última generación como IoT, inteligencia artificial y microservicios. El sistema se compone de tres áreas principales: el frontend, el backend y la integración con dispositivos IoT, todo gestionado mediante una arquitectura basada en la nube.

3.2.1) Diagrama de Casos de Uso

El siguiente diagrama ilustra las principales interacciones entre los usuarios y el sistema. Los casos de uso están diseñados para cumplir con las funcionalidades principales, como el monitoreo en tiempo real, personalización de alertas, gestión de estanques y usuarios, y análisis predictivo basado en IA.

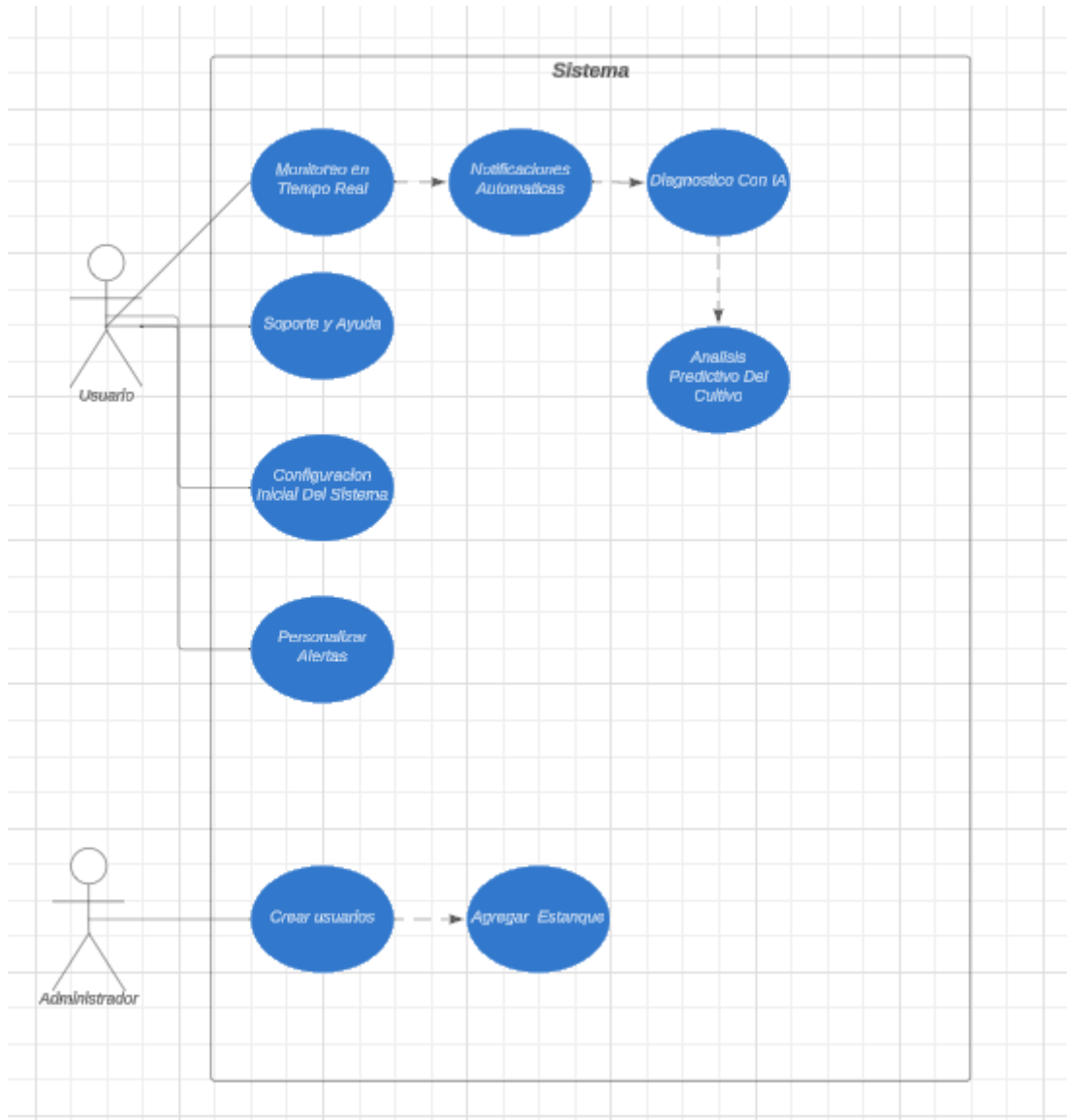


Figura 5 Caso de Usos "Sistema GreenGuardian". Fuente Elaboración Propia

Descripción de Casos de Uso Principales:

Monitoreo en Tiempo Real: Los usuarios pueden acceder al estado actual de los sensores instalados en sus cultivos.

Notificaciones Automatizadas: El sistema envía alertas cuando los parámetros de los sensores se desvían de los rangos configurados.

Diagnóstico con IA: El sistema analiza imágenes de las plantas para detectar posibles enfermedades o problemas de crecimiento.

Gestión de Usuarios y Estanques: Los administradores pueden crear usuarios y estanques, mientras que los usuarios configuran alertas personalizadas para cada sensor.

3.2.2) Diagrama de Arquitectura

El siguiente diagrama muestra la arquitectura general del *backend* de GreenGuardian, destacando la interacción entre los sensores, *AWS IoT Core*, *DynamoDB* y el *backend*. Descripción del Flujo de Datos:

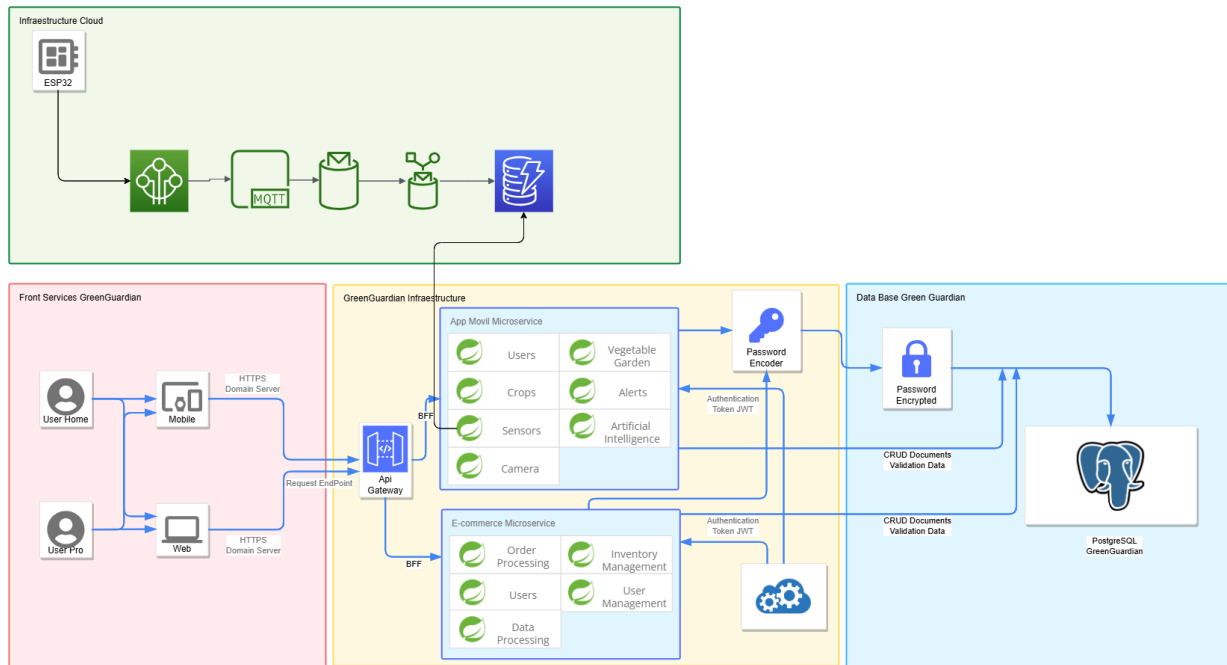


Figura 6 Arquitectura de Servicio: Proyecto GreenGuardian. Fuente: Elaboración Propia.

Sensores IoT (ESP32):

Los sensores instalados en los cultivos recogen datos como temperatura, humedad, pH, y EC.

Estos datos son enviados a *AWS IoT Core* mediante mensajes MQTT a tópicos configurados específicamente.

AWS IoT Core:

AWS IoT Core actúa como un hub central de mensajes, recibiendo los datos enviados por los sensores.

Una regla configurada en IoT Core procesa los mensajes entrantes y los almacena automáticamente en una tabla de *DynamoDB*.

DynamoDB:

Los datos estructurados y no estructurados provenientes de los sensores son almacenados en esta base de datos NoSQL.

El diseño de DynamoDB permite lecturas y escrituras rápidas, esenciales para un sistema en tiempo real.

Backend (Microservicios):

Servicio de Sensores: Recupera los datos almacenados en *DynamoDB*, los procesa y expone al *frontend* o a la *app* móvil a través de APIs RESTful.

Este servicio también publica mensajes MQTT a AWS IoT Core para: Encender o apagar dispositivos como bombas de agua.

Establecer configuraciones, como horarios de riego.

App Móvil o Frontend:

Los datos procesados por el backend son accesibles para los usuarios a través de una interfaz gráfica, donde pueden monitorear los valores en tiempo real o configurar parámetros.

3.3 Componentes del Backend

El backend de GreenGuardian está diseñado utilizando una arquitectura de microservicios. Cada microservicio tiene una responsabilidad específica y se comunica con otros servicios a través de APIs RESTful o mensajes MQTT, lo que permite flexibilidad, escalabilidad y facilidad de mantenimiento.

3.3.1 Servicio de Autenticación y Autorización

Responsabilidad: Este servicio gestiona el acceso seguro al sistema mediante la autenticación y autorización de los usuarios. Se utiliza JWT (JSON Web Tokens) para garantizar la protección de las sesiones.

Endpoints Clave:

/login (POST): Autenticación de usuarios y generación de tokens.

/validate (POST): Validación de tokens de sesión.

/create (POST): Registro de nuevos usuarios.

3.3.2 Servicio de Usuarios

Responsabilidad: Gestiona la información de los usuarios registrados, incluyendo sus perfiles, estanques asociados y estadísticas de uso.

Endpoints Clave:

/api/usuarios (GET): Obtener todos los usuarios registrados.

/api/usuarios/{id} (GET, PUT, DELETE): Operaciones CRUD sobre usuarios individuales.

/api/usuarios/estanque-by-idUsuario/{id} (GET): Listar los estanques asociados a un usuario.

/api/usuarios/promedio-estanques/{id} (GET): Calcular los promedios de los parámetros de los estanques de un usuario.

3.3.3 Servicio de Estanques

Responsabilidad: Administra los estanques registrados en el sistema, permitiendo realizar operaciones CRUD y monitorear su estado.

Endpoints Clave:

/api/estanques (GET, POST): Obtener la lista de estanques o registrar uno nuevo.

/api/estanques/{id} (GET, PUT, DELETE): Operaciones CRUD sobre estanques individuales.

/api/estanques/search-estanques/{id} (GET): Buscar estanques asociados a un usuario por su ID.

3.3.4 Servicio de Tareas

Responsabilidad: Permite gestionar tareas relacionadas con el mantenimiento y operación de los estanques.

Endpoints Clave:

/api/tareas (GET, POST): Obtener todas las tareas o crear una nueva.

/api/tareas/{id} (GET, PUT, DELETE): Operaciones CRUD sobre tareas individuales.

/api/tareas/search-tarea/{id} (GET): Listar tareas asociadas a un usuario.

3.3.5 Servicio de Sensores

Responsabilidad: Este microservicio interactúa con AWS IoT Core para publicar mensajes en tópicos MQTT que controlan dispositivos IoT, como bombas de agua o luces.

Endpoints Clave:

/api/awsiot/publish (POST): Publicar mensajes genéricos en un tópico MQTT.

/api/awsiot/publish/riego (POST): Publicar mensajes específicos para el control de riego.

3.3.6 Servicio de Datos de Estanques

Responsabilidad: Recupera los datos en tiempo real almacenados en DynamoDB por AWS IoT Core, proporcionando el último estado de los estanques al sistema.

Endpoints Clave:

/api/estanque/ultimo/{idEstanque} (GET): Obtener los datos más recientes de un estanque específico.

3.5 Configuración del Entorno

3.5.1 Despliegue en AWS

Los microservicios se despliegan en AWS EC2.

La base de datos PostgreSQL se aloja en Amazon RDS para garantizar disponibilidad y redundancia.

3.6 APIs Y Endpoints

La siguiente tabla describe los *endpoints* de la API REST del sistema. Un *endpoint* es una URL específica que representa un punto de acceso a un recurso dentro de la API. Cada fila de la tabla representa un *endpoint* y proporciona la siguiente información:

- **Endpoint:** La URL específica a la que se realiza la solicitud.
- **Método:** El verbo HTTP utilizado para la solicitud (POST para crear, GET para obtener, PUT para actualizar, DELETE para eliminar).
- **Descripción:** Una breve explicación de la función del endpoint.
- **Microservicio:** El componente del sistema (controlador) que maneja las solicitudes a ese endpoint.

La tabla se organiza por secciones funcionales (autenticación, gestión de usuarios, estanques, tareas e interacción con AWS IoT Core/MQTT) para mayor claridad. Esta tabla sirve como referencia para entender cómo interactuar con el sistema a través de la API.

Tabla 1 *Endpoint* de la aplicacion. Fuente: Elaboración propia

<i>Endpoint</i>	Método	Descripción	Microservicio
/login	POST	Autenticar	AuthUsuarioController

		un usuario y generar un token de acceso.	
<i>/validate</i>	POST	Validar un token de acceso.	AuthUsuarioController
<i>/user</i>	POST	Obtener información del usuario a partir de un token válido.	AuthUsuarioController
<i>/create</i>	POST	Crear un nuevo usuario en el sistema.	AuthUsuarioController
<i>/publish</i>	POST	Publicar un mensaje en un tópico MQTT en AWS IoT Core.	MqttController
<i>/publish/riego</i>	POST	Publicar un mensaje específico relacionado con el riego en un tópico MQTT.	MqttController
<i>/api/usuarios</i>	GET	Obtener la lista de todos los usuarios registrados.	UsuarioController

/api/usuarios/{id}	GET	Obtener los detalles de un usuario por su ID.	UsuarioController
/api/usuarios	POST	Crear un nuevo usuario.	UsuarioController
/api/usuarios/{id}	PUT	Actualizar la información de un usuario existente.	UsuarioController
/api/usuarios/{id}	DELETE	Eliminar un usuario por su ID.	UsuarioController
/api/usuarios/estanque-by-idUsuario/{id}	GET	Obtener los estanques asociados a un usuario por su ID.	UsuarioController
/api/usuarios/promedio-estanques/{id}	GET	Obtener el promedio de parámetros de los estanques de un usuario por su ID.	UsuarioController
/api/usuarios/tareas-by-id/{id}	GET	Obtener las tareas asignadas a un usuario por su ID.	UsuarioController

/api/estanques	GET	Obtener la lista de todos los estanques registrados.	EstanqueController
/api/estanques/{id}	GET	Obtener los detalles de un estanque por su ID.	EstanqueController
/api/estanques	POST	Crear un nuevo estanque en el sistema.	EstanqueController
/api/estanques/{id}	PUT	Actualizar la información de un estanque existente.	EstanqueController
/api/estanques/{id}	DELETE	Eliminar un estanque por su ID.	EstanqueController
/api/estanques/search-estanques/{id}	GET	Obtener los estanques asociados a un usuario por su ID.	EstanqueController
/api/tareas	GET	Obtener la lista de todas las tareas registradas.	TareaController
/api/tareas/{id}	GET	Obtener los detalles de una tarea	TareaController

		por su ID.	
/api/tareas	POST	Crear una nueva tarea.	TareaController
/api/tareas/{id}	PUT	Actualizar la información de una tarea existente.	TareaController
/api/tareas/{id}	DELETE	Eliminar una tarea por su ID.	TareaController
/api/tareas/search-tarea/{id}	GET	Obtener las tareas asociadas a un usuario por su ID.	TareaController
/api/estanque/ultimo/{idEstanque}	GET	Obtener los datos más recientes de un estanque en DynamoDB por su ID.	EstanqueNoSqlController

3.7 Resultados Esperados

El sistema GreenGuardian proporcionará:

- Monitoreo en tiempo real de cultivos.
- Alertas automatizadas en caso de anomalías.

- Diagnóstico predictivo basado en IA.
- Escalabilidad para soportar el crecimiento de usuarios y datos.

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN

4.1 Introducción a la Validación

La validación de la solución se enfoca en evaluar el correcto funcionamiento de los microservicios de forma individual y su interacción dentro de la arquitectura propuesta. Los objetivos principales son garantizar la modularidad, eficiencia y escalabilidad.

4.2 Metodología de Validación

- Pruebas de Microservicios Individuales:
 - AuthUsuarioController: Validación de los endpoints de autenticación (/login, /validate, /create). Las pruebas incluyeron verificación de tokens JWT y gestión segura de credenciales.
 - UsuarioController: Validación de operaciones CRUD sobre usuarios y la asociación con estanques.
 - EstanqueController: Validación de la creación, modificación y eliminación de estanques.
 - EstanqueNoSqlController: Pruebas para la integración con DynamoDB, recuperación de datos en tiempo real y consistencia de datos.
 - TareaController: Simulación de la creación y asignación de tareas a usuarios, asegurando la integridad de la información.
 - MqttController: Validación de la publicación de mensajes en tópicos MQTT, asegurando tiempos de respuesta menores a 200 ms.
- Pruebas de Integración:
 - Interacción entre microservicios mediante REST APIs para validar que las operaciones sean consistentes.
 - Evaluación del flujo de datos desde AWS IoT Core hacia el *backend* y su correcta representación en *DynamoDB* a través del EstanqueNoSqlController.
- Pruebas de Escalabilidad:
 - Simulación de escenarios con un máximo de 1,000 usuarios simultáneos para evaluar el rendimiento del sistema bajo una carga moderada.
 - Monitoreo de tiempos de respuesta y uso de recursos.

4.3 Resultados de las Pruebas

- AuthUsuarioController: Los endpoints cumplieron con un tiempo promedio de respuesta de 125 ms, con validación exitosa de tokens JWT en el 100% de los casos.

Tabla 2 Prueba de *endpoint* del authController con 10 usuarios simultáneos. Fuente: elaboración Propia

<i>Endpoint</i>	Método	Descripción	Tiempo Promedio (ms)	Desviación Estándar (ms)	Tiempo Máximo (ms)	Tasa de Éxito (%)
/login	POST	Autenticar usuario y generar token.	120	10	150	100
/validate	POST	Validar token de acceso.	110	8	130	100
/user	POST	Obtener información del usuario con token válido.	130	12	160	100
/create	POST	Crear un nuevo usuario.	140	15	180	100

- UsuarioController: Todas las operaciones CRUD sobre usuarios y asociaciones con estanques fueron completadas sin errores, con un tiempo promedio de respuesta de 158,75 ms.

Tabla 3 Prueba de *endpoint* del usuarioController con 10 usuarios simultáneos. Fuente: elaboración Propia.

<i>Endpoint</i>	Método	Descripción	Tiempo Promedio (ms)	Desviación Estándar (ms)	Tiempo Máximo (ms)	Tasa de Éxito (%)
/api/usuarios	GET	Obtener lista de todos los usuarios.	150	18	200	100
/api/usuarios/{id}	GET	Obtener detalles de un usuario	140	15	180	100

		por ID.				
/api/usuarios	POST	Crear un nuevo usuario.	160	20	220	100
/api/usuarios/{id}	PUT	Actualizar información de un usuario existente.	155	17	210	100
/api/usuarios/{id}	DELETE	Eliminar un usuario por su ID.	145	16	190	100
/api/usuarios/estanque. ..	GET	Obtener estanques asociados a un usuario por ID.	165	22	230	100
/api/usuarios/promedio ...	GET	Obtener promedio de parámetros de estanques de un usuario por ID.	175	25	250	100
/api/usuarios/tareas...	GET	Obtener las tareas asignadas a un usuario por ID.	180	28	270	100

- EstanqueController: Operaciones CRUD ejecutadas correctamente con tiempos promedio de 137,5 ms.

Tabla 4 Prueba de *endpoint* del estanqueController con 10 usuarios simultáneos. Fuente: elaboración Propia

<i>Endpoint</i>	Método	Descripción	Tiempo Promedio (ms)	Desviación Estándar (ms)	Tiempo Máximo (ms)	Tasa de Éxito (%)
/api/estanques	GET	Obtener la lista de todos los estanques registrados.	130	12	160	100
/api/estanques/{id}	GET	Obtener los detalles de un estanque por su ID.	125	10	150	100
/api/estanques	POST	Crear un nuevo estanque en el sistema.	145	15	180	100
/api/estanques/{id}	PUT	Actualizar la información de un estanque existente.	140	14	170	100
/api/estanques/{id}	DELETE	Eliminar un estanque por su ID.	135	13	165	100
/api/estanques/search.. ..	GET	Obtener estanques asociados a un usuario	150	18	190	100

		por su ID.				
--	--	------------	--	--	--	--

- EstanqueNoSqlController: La integración con DynamoDB permitió recuperar datos en tiempo real con una latencia promedio de 170 ms.

Tabla 5 Prueba de *endpoint* del estanqueNoSqlController con 10 usuarios simultáneos.

Fuente: elaboración Propia

<i>Endpoint</i>	Método	Descripción	Tiempo Promedio (ms)	Desviación Estándar (ms)	Tiempo Máximo (ms)	Tasa de Éxito (%)
/api/estanque/ultimo/{idEstanque}	GET	Obtener los datos más recientes de un estanque en DynamoDB.	170	25	230	100

- MqttController: Los mensajes MQTT fueron publicados con un tiempo de respuesta promedio de 180 ms, cumpliendo con los estándares de tiempo real.

Tabla 6 Prueba de *endpoint* del mqttController con 10 usuarios simultáneos. Fuente: elaboración Propia

Endpoint	Método	Descripción	Tiempo Promedio (ms)	Desviación Estándar (ms)	Tiempo Máximo (ms)	Tasa de Éxito (%)
/publish	POST	Publicar un mensaje en un tópico MQTT en AWS IoT Core.	180	20	220	100
/publish/riego	POST	Publicar un mensaje específico de riego en un tópico MQTT.	175	18	210	100

- Escalabilidad: Bajo una carga de 1,000 usuarios simultáneos, el sistema tiene una respuesta de 9.9 ms con un porcentaje de error del 24%.

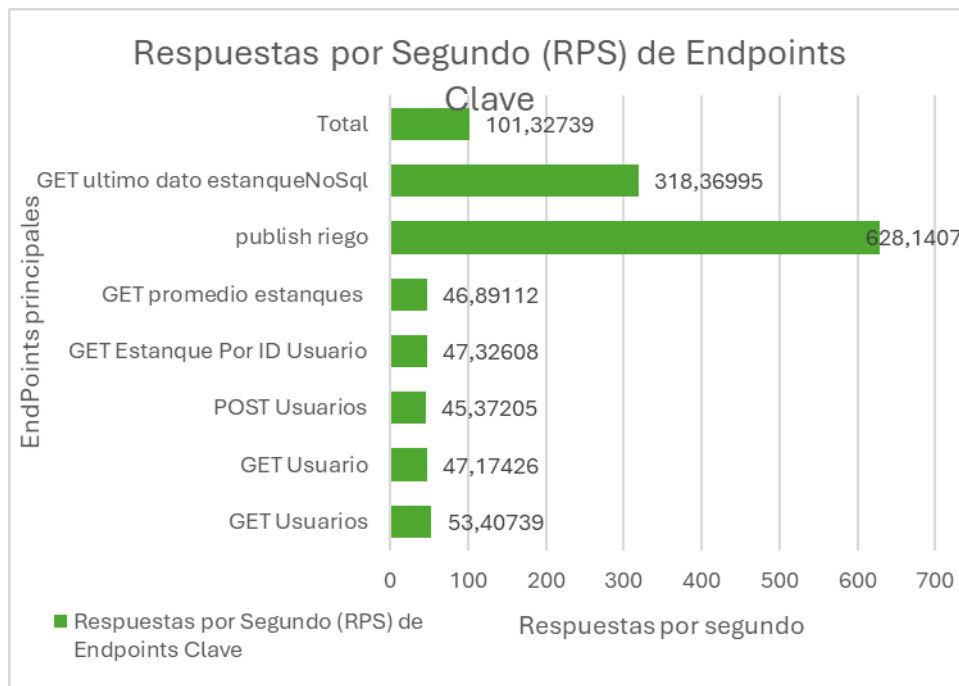


Figura 7 Respuestas por segundo con una carga de 1000 usuarios simultáneos. Fuente: elaboración Propia

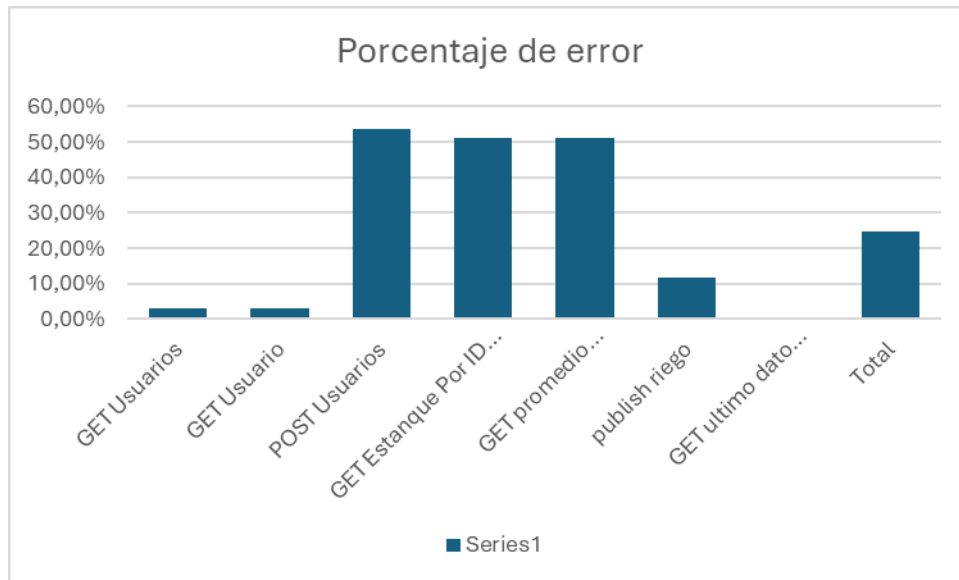


Figura 8 Porcentaje de error con 1000 usuarios simultáneos. Fuente: elaboración propia

4.4 Datos del Entorno de Validación

- Infraestructura:
 - Instancia AWS EC2 Linux Large con 8 vCPUs y 16 GB de RAM.
 - Base de datos DynamoDB alojada en AWS.
 - Uso de AWS IoT Core para la comunicación con sensores IoT.
- Herramientas:
 - Postman para pruebas de API.
 - JMeter para simulación de carga.
 - CloudWatch para monitoreo de rendimiento.

4.5 Análisis de Impacto

- Impacto en la Modularidad: La implementación basada en microservicios permitió realizar pruebas y actualizaciones de forma independiente.

- Impacto en la Escalabilidad: El sistema demostró ser capaz de manejar cargas moderadas sin afectar su rendimiento, asegurando su viabilidad para implementaciones futuras.
- Impacto en la Fiabilidad: La arquitectura aseguró que fallos en un microservicio no interfirieran con la operación global del sistema, al tener su independencia cada microservicio no afectaría al otro, como así hay servicios que detendrían todo el sistema si se cayeran, ejemplo microservicios usuario.



HOME LAST 1000 SINCE STARTUP

System Status

Environmenttest

Data centerdefault

Current time2024-12-28T01:15:16 -0300

Uptime08:54

Lease expiration enabledtrue

Renews threshold17

Renews (last min)18

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - localhost.gateway-service:8080
MSVC-AUTH	n/a (3)	(3)	UP (3) - msvc-auth-9c0f842c45ba22d23a163ce2c54c3e55, msvc-auth-3cce8767003a8a5e5e5b5d2f9c7db560, msvc-auth-798c4606ed8b4de0d22b494ce7b28e53
MSVC-ESP32	n/a (1)	(1)	UP (1) - msvc-esp32-b4c15fd77bcec5baa6b40ad858ab8255
MSVC-ESTANQUE	n/a (1)	(1)	UP (1) - localhost.msvc-estanque:9080
MSVC-ESTANQUE-DYNAMO	n/a (1)	(1)	UP (1) - msvc-estanque-dynamo-92e6f4f1c4d91f78d5367186a2619533
MSVC-TAREAS	n/a (1)	(1)	UP (1) - msvc-tareas-009149493a614a044e1471b0e1fd6e21
MSVC-USUARIO	n/a (1)	(1)	UP (1) - localhost.msvc-usuario:8090

Figura 9 Imagen de los microservicios en funcionamiento. Fuente: elaboración Propia

CAPÍTULO 5: CONCLUSIONES

La implementación del sistema GreenGuardian ha permitido consolidar una arquitectura backend robusta, modular y eficiente, enfocada en la gestión y el análisis de datos en tiempo real para el monitoreo y control de cultivos hidropónicos. Este enfoque ha demostrado ser efectivo al integrar tecnologías avanzadas y adoptar prácticas de desarrollo modernas, logrando:

- **Modularidad y escalabilidad:** La implementación basada en microservicios permitió una fácil integración y despliegue de nuevas funcionalidades, garantizando que el sistema pueda evolucionar conforme a las necesidades futuras.
- **Gestión eficiente de datos:** La conexión entre sensores IoT y AWS IoT Core, junto con el almacenamiento en DynamoDB, aseguró un manejo rápido y confiable de los datos en tiempo real.
- **Optimización del rendimiento:** Se implementaron estrategias de monitoreo y métricas que permitieron identificar y resolver cuellos de botella, mejorando la eficiencia general del sistema.
- **Automatización de procesos:** Mediante el uso de mensajes MQTT, se optimizó la interacción con dispositivos como bombas de riego, facilitando la automatización en cultivos.
- **Calidad y seguridad del código:** La implementación de pruebas automatizadas garantizó la calidad del sistema backend, mientras que los mecanismos de autenticación robusta protegieron el acceso a los servicios.

Sin embargo, el proyecto también enfrentó ciertas limitaciones que es necesario considerar:

- **Validación limitada en entornos reales:** Aunque se realizaron pruebas exitosas en un entorno controlado, no se han implementado pilotos a gran escala para validar la eficacia total del sistema en condiciones diversas.
- **Dependencia de la infraestructura en la nube:** El uso de servicios de AWS garantiza alta disponibilidad, pero incrementa los costos operativos, lo cual puede ser una barrera para usuarios pequeños.

En resumen, la arquitectura backend diseñada demostró su capacidad para manejar datos en tiempo real, soportar operaciones complejas y garantizar la calidad del sistema. Este desarrollo sienta las bases para expandir el sistema y optimizar su impacto en el ámbito agrícola.

5.2 Trabajos Futuros

Para fortalecer y ampliar el impacto de GreenGuardian, se sugieren las siguientes líneas de trabajo:

1. **Validación en entornos reales:** Realizar pruebas piloto en diferentes tipos de cultivos y entornos para evaluar la efectividad del sistema en condiciones diversas.
2. **Optimización del backend:** Investigar el uso de tecnologías serverless para reducir los costos operativos y mejorar la escalabilidad.
3. **Mejoras en el diagnóstico con IA:** Expandir el dataset para entrenar modelos de redes neuronales capaces de identificar una gama más amplia de enfermedades y anomalías en cultivos.
4. **Integración de dispositivos adicionales:** Incorporar nuevos sensores y dispositivos IoT, como cámaras multiespectrales, para enriquecer los datos recolectados.
5. **Interfaz de usuario mejorada:** Diseñar un frontend más intuitivo y accesible para garantizar una experiencia óptima para los usuarios finales.

Estas iniciativas buscan consolidar a GreenGuardian como una solución integral y adaptable, contribuyendo al desarrollo sostenible y a la mejora en la gestión de recursos en el sector agrícola.

REFERENCIAS BIBLIOGRÁFICAS

REFERENCIAS BIBLIOGRÁFICAS EN NORMA APA:

Choma, D., Chwaleba, K., & Dzieńkowski, M. (2023). The efficiency and reliability of backend technologies: Express, Django, and Spring Boot. *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska*, 13.

L. De Lauretis, "From Monolithic Architecture to Microservices Architecture," *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Berlin, Germany, 2019, pp. 93-96, doi: 10.1109/ISSREW.2019.00050.

Banco Mundial. (2021). El futuro de Chile como país de agua está en riesgo. Washington, DC: Banco Mundial. Recuperado de <https://blogs.worldbank.org/es/latinamerica/el-futuro-de-chile-como-pais-de-agua-esta-en-riesgo>

Centro de Ciencia del Clima y la Resiliencia (CR2). (2020). Crisis hídrica: Informe dice que Chile alcanzará punto crítico en unos 50 a 200 años. Santiago, Chile: CR2. Recuperado de <https://www.cr2.cl/crisis-hidrica-informe-dice-que-chile-alcanzara-punto-critico-en-unos-50-a-200-anos-radios-regionales/>

La Tercera. (2022). Así ha cambiado el consumo de agua potable en los últimos 25 años en el país. Santiago, Chile: Copesa. Recuperado de <https://www.latercera.com/que-pasa/noticia/asi-ha-cambiado-el-consumo-de-agua-potable-en-los-ultimos-25-anos-en-el-pais/KTCXNSLTRBAERLKEUVRG7M2LFQ/>

Oficina de Estudios y Políticas Agrarias (ODEPA). (2022). Agua: Agricultura sustentable y gestión del recurso hídrico. Santiago, Chile: Ministerio de Agricultura. Recuperado de <https://www.odepa.gob.cl/sustentabilidad/agricultura-sustentable/agua>

Secretaría de Agricultura y Desarrollo Rural. (2022). Cómo cuidar el agua: Hidroponía y tecnificación. Ciudad de México, México: Gobierno de México. Recuperado de <https://www.gob.mx/agricultura/articulos/como-cuidar-el-agua-hidroponia-y-tecnificacion#:~:text=La%20hidropon%C3%ADa%20permite%20cultivar%20incluso,fertilizantes%20pesticidas%20y%20maquinaria%20agr%C3%ADcola>

iAgua. (2024). Sequía en Chile: el 72% de la superficie en situación de escasez afecta al 38% de la población. Madrid, España: iAgua. Recuperado de <https://www.iagua.es/especiales/sequia-chile#:~:text=El%2072%25%20de%20la%20superficie,38%25%20de%20la%20poblaci%C3%B3n>

Radio Bio Bio. (2024). La profunda crisis hídrica que Chile no quiere asumir. Santiago, Chile: Radio Bio Bio. Recuperado de <https://www.biobiochile.cl/noticias/opinion/tu-voz/2024/05/09/la-profunda-crisis-hidrica-que-chile-no-quiere-asumir.shtml>

Fundación Heinrich Böll. (2024). Escasez hídrica en Chile: Reflexiones en el Día Mundial del Agua 2024. Santiago, Chile: Fundación Heinrich Böll. Recuperado de <https://cl.boell.org/es/2024/03/25/escasez-hidrica-en-chile-reflexiones-en-el-dia-mundial-del-agua-2024>

Universidad de Chile. (2022). Día Mundial del Agua: Chile lidera la crisis hídrica en América Latina. Santiago, Chile: Facultad de Ciencias Forestales, Universidad de Chile. Recuperado de <https://forestal.uchile.cl/noticias/184816/dia-mundial-del-agua-chile-lidera-la-crisis-hidrica-en-america-latina>

Universidad de Concepción. (2022). El impacto de la crisis hídrica en Chile. Concepción, Chile: Facultad de Ciencias Forestales, Universidad de Concepción. Recuperado de <https://www.forestal.udec.cl/el-impacto-de-la-crisis-hidrica-en-chile/>

La Tercera. (2023). Más de la mitad de los chilenos se autocalifica como clase baja y sube con fuerza la percepción de que el Estado ayudó durante la pandemia. Santiago, Chile: La Tercera. Recuperado de <https://www.latercera.com/pulso/noticia/mas-de-la-mitad-de-los-chilenos-se-autocalifica-como-clase-baja-y-sube-con-fuerza-la-percepcion-de-que-el-estado-ayudo-durante-la-pandemia/Z3SEJZSOXBFHRE5XOQRDVJQDSA/#:~:text=De%20acuerdo%20a%20la%20auto,alta%E2%80%9D%2C%20seg%C3%BAn%20el%20estudio>

Ministerio de Agricultura. (2007). Censo Nacional Agropecuario: Resultados preliminares. Santiago, Chile: Ministerio de Agricultura. Recuperado de https://cdn.digital.gob.cl/filer_public/02/c1/02c10ef8-1685-4986-a546-8f2b7c2baceb/21_minagricultura-f.pdf