| [EMPRESA DE PANELES SOLARES]  GREEN ENERGY SPA. |
| --- |

**Contenido**

[**1. Definición del problema 4**](#_v50uycbho2ye)

[**1.1 Descripción del sistema monolítico actual 4**](#_lt2tlm5vhj4z)

[**1.2 Identificación del problema principal 4**](#_aqjc07xg3zjc)

[**1.3 Impacto de dicho problema en el negocio 4**](#_wavo66luku5f)

[**1.4 Objetivo general y objetivos específicos del proyecto 5**](#_6hoaxbcqy2cv)

[**2. Estrategia de microservicios 5**](#_y06ro6h6okig)

[**2.1. Tipo de Estrategia Usada (Por dominio o por funcionalidad) 5**](#_5lritjupjfa)

[**2.2. Justificación técnica y de diseño de la estrategia elegida. 6**](#_jllcqqajek5v)

[**2.3. Beneficios de adoptar esta estrategia en comparación al sistema actual. 6**](#_7qaaq7im2h2f)

[**3. Herramientas utilizadas 8**](#_2knqlf68klhe)

[**3.1 Listado de herramientas 8**](#_54m3xjkhbxwp)

[**3.2 Funcionalidad específica de cada herramienta 9**](#_opnjhsbg4hlu)

[**3.3 Justificación de la elección de estas herramientas 9**](#_nte88lqdvhdz)

[**4. Herramientas de trabajo colaborativo 9**](#_z7jagxfbjghu)

[**4.1 Herramientas utilizadas 10**](#_a1z0yfz3kmcj)

[**4.2 Evidencias del uso de herramientas colaborativas 10**](#_kkwq8zywdxma)

[**5. Enfoque ético en el desarrollo 10**](#_flshzjgoarlz)

[**5.1 Evaluación de los principales desafíos éticos 10**](#_1bu7tillboat)

[**5.2. Soluciones y buenas prácticas adoptadas 11**](#_ef7j4nl6jbep)

[**6. Análisis de requerimientos 12**](#_k40elqqlk8ur)

[**6.1 Requisitos Funcionales 12**](#_w7bx5hehosaj)

[**6.2 Requisitos No Funcionales 15**](#_gjkxvmjexmuc)

[**6.3 Síntesis de necesidades de cada perfil 16**](#_gwv9tu5ekq2k)

[**7. Análisis del sistema actual 16**](#_je7057bldgu4)

[**7.1 Arquitectura general del sistema monolítico 16**](#_o8ly53jehhui)

[**7.2 Puntos débiles detectados 17**](#_l24fasxr3lk8)

[**8. Diseño de la nueva arquitectura 17**](#_d69j6ydis975)

[**8.1 Descripción general de la arquitectura basada en microservicios 17**](#_ezh8yqu881wn)

[**8.2. Diagrama de Actores de Alto Nivel (quién interactúa con qué servicio) 19**](#_u2wnfs3zgwa4)

[**8.3 Diagrama de Clases 19**](#_5k97vl8aacpv)

[**8.4. Diagramas de Casos de Uso 19**](#_7lnfrxrwxmzq)

[**9. Planificación de la migración 23**](#_7s4jekpuyamh)

[**9.1. Fases de Migración (por módulo o servicio) 23**](#_t8d2hi6x1v4x)

[**9.2. Consideraciones de Compatibilidad 24**](#_idw4gqqbos0l)

[**9.3. Identificación de Riesgos y su plan de mitigación 24**](#_5ioi0o66hkqt)

[**10. Planificación Ágil y Cronograma 25**](#_tkgnvog88xf4)

[**10.1. Breve Explicación del Enfoque Ágil Usado (Scrum, Kanban, etc.) 25**](#_jc7fjbbvd9or)

[**10.2. Roles asumidos por los integrantes. 26**](#_7ncmtffiv4e6)

[**10.3. Carta Gantt con cronograma de implementación y revisión. 26**](#_14x6g42c680b)

[**11. Conclusiones y recomendaciones 26**](#_6e2kow5bymes)

[**11.1 Lecciones aprendidas 26**](#_6p91gdepbvze)

[**11.2 Potencial escalabilidad del sistema 27**](#_oubk5f38u1qm)

[**11.3 Posibles mejoras futuras 27**](#_u03aoeh42gia)

*Profesor a cargo: Victor Isidro Rosendo Lugo*

*Asignatura: Desarrollo Fullstack I*

**Fecha de entrega: 31-03-2025**

**Integrantes:**

| **Nombre Integrante del Equipo** |
| --- |
| *Omar Hurtado* |
| *Valeria Fonseca* |
| *Gerardo Soto* |

## Definición del problema

## 1.1 Descripción del sistema monolítico actual

GreenEnergy nos ha solicitado ayuda para modificar o derechamente cambiar el sistema actual que tienen debido varios problemas, pero principalmente uno, el cual es debido a la alta demanda de sus servicios de instalación y mantención de paneles solares. Ellos nos han compartido una diferenciación de perfiles y funciones con las que quieren contar en su nuevo sistema.

Su sistema actual está dividido en módulos: Gestión de Usuarios, Gestión de Instalaciones, Gestión de mantenimientos, Facturación y Pagos, Soporte y Atención al Cliente e Inventario y Logística. Sin embargo, ellos tienen una arquitectura monolítica en su sistema. La arquitectura monolítica es un modelo tradicional de diseño de software donde toda la aplicación se desarrolla y ejecuta como un único bloque. Esto significa que los diferentes módulos (como la interfaz de usuario, la lógica de negocio y la base de datos) están fuertemente acoplados en un solo código fuente y en una sola unidad de despliegue, esto puede traer problemas debido a la alta dependencia entre sus componentes. Si falla algo puede fallar toda la aplicación.

## 1.2 Identificación del problema principal

El principal problema que enfrenta GreenEnergy SPA es la falta de un sistema de gestión eficiente para hacer frente a la creciente demanda de sus servicios. La empresa tiene dificultades para administrar adecuadamente sus proyectos, realizar un seguimiento efectivo de las instalaciones y gestionar de manera organizada las solicitudes de mantenimiento, lo que afecta su operatividad y la experiencia del cliente.

Esta deficiencia impacta tanto en la operatividad interna de la empresa como en la experiencia del cliente, quien no dispone de una plataforma para monitorear sus solicitudes, recibir notificaciones o acceder a información clave sobre su consumo energético.

**1.3 Impacto de dicho problema en el negocio**

El problema impacta negativamente en el negocio al generar insatisfacción en los clientes, lo que disminuye la confianza y lealtad hacia la empresa. Además, afecta la eficiencia operativa al dificultar la gestión de proyectos y solicitudes de mantenimiento, lo que puede generar retrasos en las instalaciones y servicios. Esta ineficiencia reduce la capacidad de respuesta ante imprevistos, afecta la toma de decisiones estratégicas y puede provocar la pérdida de oportunidades de negocio, afectando el crecimiento y la competitividad de la empresa.

## **1.4 Objetivo general y objetivos específicos del proyecto**

**Objetivo General:** Desarrollar e implementar una solución tecnológica integral que optimice la administración de proyectos, el seguimiento de instalaciones y la gestión de solicitudes de mantenimiento, con el fin de mejorar la experiencia del cliente, aumentar la eficiencia operativa y fortalecer la competitividad de la empresa.

**Objetivos Específicos:**

1. Diseñar una plataforma centralizada que permita gestionar los proyectos de instalación y mantenimiento de manera eficiente.
2. Implementar un sistema de seguimiento en tiempo real para mejorar la visibilidad del estado de cada proyecto.
3. Agilizar la comunicación entre clientes, técnicos y coordinadores mediante notificaciones automáticas.
4. Optimizar la asignación de recursos y técnicos para evitar retrasos en las instalaciones.
5. Desarrollar un módulo de generación de reportes para facilitar la toma de decisiones basada en datos.
6. Garantizar la seguridad y confidencialidad de los datos de clientes y proyectos mediante medidas de protección adecuadas.

## Estrategia de microservicios

## 2.1. Tipo de Estrategia Usada (Por dominio o por funcionalidad)

En el desarrollo de microservicios, existen diversas estrategias para dividir un sistema en servicios independientes. Dos de los enfoques más utilizados son la separación por funcionalidad y por dominio. La división por funcionalidad organiza los microservicios según tareas específicas, como autenticación, notificaciones o pagos. Por otro lado, la división por dominio, basada en Domain-Driven Design (DDD), estructura los microservicios en torno a contextos delimitados o Bounded Contexts, que agrupan la lógica relacionada con diferentes áreas del negocio. En nuestro caso, adoptaremos la estrategia por dominio, permitiendo que cada Bounded Context encapsule sus propias reglas de negocio, modelos de datos y operaciones específicas, asegurando una arquitectura más modular y escalable.

## 2.2. Justificación técnica y de diseño de la estrategia elegida.

El enfoque basado en dominios se justifica desde varias perspectivas: a. acoplamiento y cohesión, b. escalabilidad, c. mantenibilidad, d. resiliencia, e. comunicación eficiente y f. alineación con el negocio.

1. En el enfoque basado en dominio, los servicios agrupan toda la lógica relacionada con un contexto de negocio, reduciendo dependencias externas y asegurando que los datos y reglas de negocio estén encapsulados dentro de un único servicio.
2. Se pueden escalar únicamente los microservicios que realmente lo necesitan, porque cada servicio tiene toda la lógica relevante para su dominio.
3. Los cambios en la lógica de negocio pueden realizarse dentro del mismo microservicio, sin impactar en otros servicios.
4. Al encapsular lógica de negocio en contextos delimitados, un fallo en un microservicio no interrumpe todo el sistema.
5. Los microservicios por dominio agrupan toda la lógica de un contexto, reduciendo la necesidad de llamadas remotas innecesarias.
6. Los servicios se diseñan en función de contextos de negocio reales, facilitando que los equipos trabajen de forma independiente sobre áreas específicas del sistema.

## 2.3. Beneficios de adoptar esta estrategia en comparación al sistema actual.

Vamos a enumerar punto por punto de comparación para que queden bien claros los beneficios de la estrategia seleccionada considerando los problemas de la arquitectura monolítica:

**1. Escalabilidad**

* Problema en la arquitectura monolítica:
* Todo está en un solo sistema, por lo que escalar un módulo específico (como Facturación) implica escalar toda la aplicación.
* Si la carga en "Gestión de Instalaciones" aumenta, afecta el rendimiento de "Soporte" y "Facturación".
* Beneficio con microservicios:
* Escalabilidad independiente: Se puede aumentar solo los microservicios que necesiten más capacidad sin afectar los demás.
* Uso eficiente de recursos: Cada servicio puede ejecutarse en diferentes instancias o servidores.

**2. Mantenimiento y Desarrollo Ágil**

* Problema en la arquitectura monolítica:
* Un pequeño cambio en un módulo puede afectar toda la aplicación.
* Se necesita re-deploy completo al hacer una actualización.
* Equipos de desarrollo dependen unos de otros, generando bloqueos.
* Beneficio con microservicios:
* Equipos independientes pueden trabajar en diferentes servicios sin afectar a otros.
* Se pueden desplegar cambios en un solo servicio sin reiniciar todo el sistema.
* Mayor velocidad para agregar nuevas funciones sin afectar lo existente.

**3. Resiliencia y Disponibilidad**

* Problema en la arquitectura monolítica:
* Si un módulo falla, toda la aplicación podría dejar de funcionar.
* Un error en "Facturación" podría hacer que "Soporte" o "Inventario" se caigan.
* Beneficio con microservicios:
* Aislamiento de fallos: Si un microservicio falla, el resto sigue funcionando.
* Se pueden usar circuit breakers para evitar que errores propaguen fallos a todo el sistema.
* Alta disponibilidad, ya que se pueden levantar instancias solo cuando se necesiten.

**4. Flexibilidad Tecnológica**

* Problema en la arquitectura monolítica:
* Todo debe usar la misma tecnología (Java y MySQL), lo que limita la evolución del sistema.
* No se pueden probar nuevas herramientas sin modificar todo el código.
* Beneficio con microservicios:
* Cada microservicio puede usar la mejor tecnología para su propósito (Ejemplo: MySQL para Finanzas, MongoDB para logs).
* Se pueden probar nuevas herramientas sin afectar otros módulos.

**5. Optimización del Rendimiento**

* Problema en la arquitectura monolítica:
* Todas las consultas a la base de datos están en un mismo servidor, lo que ralentiza las respuestas.
* "Inventario" y "Facturación" dependen de "Mantenimiento", lo que hace que las consultas sean lentas y complejas.
* Beneficio con microservicios:
* Cada microservicio maneja sus propias consultas, reduciendo la sobrecarga en la base de datos.
* Se pueden hacer consultas optimizadas y usar caché para acelerar las respuestas.

**6. Seguridad y Control de Accesos**

* Problema en la arquitectura monolítica:
* Todos los módulos comparten la misma base de datos, lo que expone información sensible.
* No se pueden restringir permisos con precisión sin afectar a toda la aplicación.
* Beneficio con microservicios:
* Cada servicio tiene su propia base de datos, reduciendo el acceso innecesario a información sensible.
* Se pueden implementar autenticación con OAuth2/JWT por microservicio para mayor seguridad.

En resumen, adoptar microservicios por dominio en GreenEnergy ofrece mayor escalabilidad, flexibilidad, seguridad y resiliencia. En comparación con la arquitectura monolítica actual, permite un desarrollo más rápido, mantenimiento más sencillo y mejor rendimiento, asegurando que el sistema pueda crecer sin problemas en el futuro.

## Herramientas utilizadas

## 3.1 Listado de herramientas

Las herramientas utilizadas en este proyecto serían: Java, Spring Boot, MySQL y como plataforma cloud AWS.

## 3.2 Funcionalidad específica de cada herramienta

* Java: Lenguaje de programación principal debido a su escalabilidad, seguridad, su capacidad de implementación y que es un lenguaje reconocido por su uso en el desarrollo de software empresarial.
* Spring Boot: Framework que se usará para el desarrollo de aplicaciones web y microservicios, que facilita la gestión de cambios y permite una rápida implementación.
* MySQL: Sistema de gestión de base de datos que garantiza una gestión fácil y segura.
* AWS: Plataforma en la nube que ofrece una infraestructura segura, escalable, confiable para guardar aplicaciones y gestionar bases de datos.

## 3.3 Justificación de la elección de estas herramientas

La elección de estas herramientas responde a su facilidad de uso, escalabilidad y compatibilidad con los procesos existentes dentro de la empresa. Su combinación permite una integración ágil y flexible, asegurando que el sistema pueda adaptarse con rapidez a cualquier cambio o desafío futuro.

Además, estas tecnologías cuentan con un amplio respaldo en la industria, lo que garantiza estabilidad, soporte a largo plazo y una optimización de recursos. Java y Spring Boot permiten un desarrollo eficiente sin comprometer el rendimiento, MySQL

ofrece una gestión confiable de los datos, y AWS proporciona una infraestructura escalable y segura. Su adopción facilita la conectividad con otras soluciones, reduce los tiempos de implementación y minimiza los costos operativos.

Al utilizar herramientas ampliamente adoptadas y probadas, la empresa no solo garantiza una base tecnológica sólida, sino que también se asegura de contar con un ecosistema de desarrollo dinámico y en constante evolución, preparado para enfrentar nuevas oportunidades y desafíos.

## Herramientas de trabajo colaborativo

## 4.1 Herramientas utilizadas

Para la planificación y desarrollo de este proyecto utilizamos herramientas colaborativas que facilitaron la organización, comunicación y gestión del trabajo en equipo. Trello se empleó para la planificación y seguimiento de tareas, permitiendo una distribución eficiente del trabajo. Discord sirvió como plataforma de comunicación en tiempo real, favoreciendo la coordinación del equipo. Además, utilizamos WhatsApp para una comunicación rápida y directa entre los miembros del equipo. Para la edición y gestión de documentos, empleamos Google Drive, lo que nos permitió poder editar el documento de manera simultánea y visualizar los cambios en tiempo real. Finalmente, GitHub se utilizará para el almacenamiento de la documentación final del proyecto.

## 4.2 Evidencias del uso de herramientas colaborativas

Carpeta de capturas: <https://drive.google.com/drive/folders/1RzUv9ZCr57MMwRNmtUt6wFRptsG-wprY?usp=drive_link>

## Enfoque ético en el desarrollo

## 5.1 Evaluación de los principales desafíos éticos

Existen diversos temas a considerar a la hora de trabajar con una estrategia que plantean desafíos éticos: la privacidad de los datos, la seguridad general del sistema, la responsabilidad en el despliegue y el impacto en los puestos de trabajo.

¿En qué medida se entienden como desafíos? Revisemos cada uno:

**Privacidad de los datos:** El sistema basado en microservicios podría aumentar la exposición de datos sensibles debido a una mayor cantidad de interacciones entre servicios. Se debe garantizar que la segmentación de datos y los accesos sean controlados adecuadamente.

**Seguridad General del sistema:** Los microservicios aumentan la superficie de ataque del sistema, ya que hay más puntos de entrada y comunicación entre servicios. Se requiere una estrategia de seguridad robusta para proteger el sistema y los datos de los usuarios.

**Responsabilidad en el despliegue:** plantea desafíos éticos porque la forma en que se liberan, actualizan y monitorean estos servicios puede impactar la seguridad, privacidad y estabilidad del sistema. Un error en la gestión del despliegue podría derivar en fallas críticas, exposición de datos sensibles o incluso manipulación indebida de información.

**Impacto en los puestos de trabajo:** La transición a microservicios suele implicar automatización y nuevos paradigmas de trabajo, lo que puede hacer que ciertos roles queden obsoletos o que se exija a los empleados nuevas habilidades sin la capacitación adecuada.

## 5.2. Soluciones y buenas prácticas adoptadas

¿Cómo podemos resolver o hacer frente a los desafíos éticos según cada uno en particular?

* Privacidad de los datos

- Principio de minimización de datos: Cada microservicio solo debe manejar la información estrictamente necesaria.

- Autenticación y autorización centralizada: Uso de OAuth2 o JWT para gestionar accesos de manera segura.

- Cifrado de datos sensibles en tránsito y en reposo.

* Seguridad

- Firewall de API: Implementación de un Gateway API con reglas de seguridad.

- Control de acceso basado en roles (RBAC) para limitar la exposición de servicios críticos.

- Registros y auditoría de acceso para rastrear actividad sospechosa.

* Responsabilidad en el despliegue

- Automatización con CI/CD: Implementar pipelines para pruebas, despliegue y rollback seguro.

- Monitorización activa: Integración de herramientas como Prometheus y Grafana para detectar fallos en tiempo real.

* Impacto en los puestos de trabajo

- Capacitación del equipo en DevOps y microservicios para mejorar su adaptación al nuevo modelo.

- Redefinición de roles en desarrollo y soporte para gestionar eficientemente los nuevos servicios.

## 5.3. Consideraciones de cumplimiento normativo (ej. Ley de Protección de Datos en Chile)

Leyes que regulan y cómo podemos apegarnos a la ley en el desarrollo de software.

-> Leyes de privacidad y seguridad de datos en Chile

* Ley 19.628 sobre Protección de la Vida Privada

- Regula el uso de datos personales en Chile.

- Esta ley indica que se deben establecer políticas de privacidad claras y obtener consentimiento informado para el tratamiento de datos.

* Ley 20.393 sobre responsabilidad penal de empresas

- Obliga a implementar sistemas de control para evitar delitos informáticos y filtraciones de datos.

¿Cómo hacer en virtud de la ley para desarrollar tranquilamente software? La estrategia de microservicios puede cumplir con estas normativas mediante:

1. Gestión de accesos controlada (OAuth2, JWT).

2. Trazabilidad de operaciones en la base de datos.

3. Cifrado de datos sensibles (AES-256 en reposo, TLS en tránsito).

## Análisis de requerimientos

## 6.1 Requisitos Funcionales

R.1 Inicio de sesión  
 Actores: Administrador, técnico, cliente, coordinador, soporte técnico  
 Descripción: Permitir a los usuarios iniciar sesión y acceder a sus funcionalidades de acuerdo a su rol.

R.2 Gestión de usuarios  
 Actores: Administrador  
 Descripción: Crear, modificar y eliminar cuentas de clientes y empleados con distintos niveles de acceso.

R.3 Configuración de roles y permisos  
 Actores: Administrador  
 Descripción: Definir permisos para técnicos, administradores y clientes dentro del sistema.

R.4 Monitorización del sistema  
 Actores: Administrador, soporte técnico  
 Descripción: Visualizar el estado del sistema, recibir alertas sobre posibles fallos y monitorizar el rendimiento.

R.5 Respaldo y restauración de datos  
 Actores: Administrador  
 Descripción: Realizar copias de seguridad periódicas y restaurar datos en caso de pérdida o fallo.

R.6 Gestión de proyectos  
 Actores: Coordinador  
 Descripción: Crear nuevos proyectos de instalación, asignar recursos, técnicos y definir plazos.

R.7 Seguimiento de proyectos  
 Actores: Coordinador, cliente, técnico  
 Descripción: Visualizar el estado de avance de cada proyecto, generar alertas ante retrasos y actualizar el progreso.

R.8 Coordinación de técnicos y recursos  
 Actores: Coordinador  
 Descripción: Asignar técnicos y materiales a cada proyecto en función de disponibilidad y necesidades.

R.9 Generación de reportes  
 Actores: Coordinador  
 Descripción: Crear informes sobre tiempos de instalación, eficiencia y demanda de servicios.

R.10 Gestión de perfil  
 Actores: Administrador, técnico, cliente, coordinador, soporte técnico  
 Descripción: Actualizar información personal y contraseñas.

R.11 Registro de problemas  
 Actores: Técnico, cliente, soporte técnico  
 Descripción: Enviar consultas o problemas durante la instalación y solicitar soporte técnico.

R.12 Gestión de inventario  
 Actores: Coordinador, técnico  
 Descripción: Registrar el uso de materiales y herramientas en cada proyecto e informar niveles de stock.

R.13 Atención de solicitudes de soporte  
 Actores: Soporte técnico, cliente, coordinador  
 Descripción: Gestionar consultas o problemas enviados por los clientes o empleados, mediante tickets o chat.

R.14 Registro de cliente  
 Actores: Cliente  
 Descripción: Registrarse en la plataforma web proporcionando la información necesaria para su inscripción.

R.15 Portal de cliente  
 Actores: Cliente  
 Descripción: Espacio donde los clientes puedan ver os servicios disponibles, su historial de consumo energético y el estado de sus instalaciones, mantenimientos y solicitudes.

R.16 Solicitud de instalación  
 Actores: Cliente  
 Descripción: Solicitar evaluaciones y presupuestos para instalaciones de paneles solares.

R.17 Gestión y agendamiento de mantenimiento  
 Actores: Cliente, coordinador, técnico  
 Descripción: El sistema debe permitir a los clientes solicitar y programar mantenimientos preventivos o correctivos desde su portal, y a los coordinadores gestionar dichas solicitudes, asignar técnicos y hacer seguimiento de su cumplimiento.

R.18 Sistema de reseñas y calificaciones  
 Actores: Cliente  
 Descripción: Evaluar servicios contratados y dejar comentarios en el sitio web.

R.19 Notificaciones automáticas  
 Actores: Técnico, cliente, coordinador, soporte técnico  
 Descripción: Enviar correos, SMS o notificaciones push sobre actualizaciones de proyectos y mantenimientos.

R.20 Panel de administración  
 Actores: Administrador  
 Descripción: Herramienta para gestionar usuarios, roles, permisos, monitoreo del sistema y supervisión de operaciones clave.

R.21 Panel de técnico  
 Actores: Técnico  
 Descripción: Panel donde los técnicos consultan sus asignaciones, reportan incidencias, actualizan el estado de sus tareas y gestionan el inventario.

R.22 Portal de coordinación  
 Actores: Coordinador  
 Descripción: Portal para coordinadores donde pueden gestionar proyectos y recursos, notificar sobre avances a los clientes y crear informes.

R.23 Portal de soporte técnico  
 Actores: Soporte técnico  
 Descripción: Espacio para recibir, atender y gestionar solicitudes de soporte técnico enviadas por clientes y empleados.

R.24 Historial de consumo energético  
 Actores: Cliente, Administrador, Soporte Técnico  
 Descripción: Permitir a los clientes consultar su historial de consumo energético en una tabla con registros por fecha, kWh y costos. La tabla deberá incluir filtros por período y opción de exportación a Excel.

R.25 Generación de facturas  
 Actores: Administrador, Cliente  
 Descripción: El sistema debe emitir facturas automáticamente después de contratado un servicio.

R.26 Historial de transacciones  
 Actores: Cliente, Administrador  
 Descripción: Permitir a los clientes y administradores consultar pagos realizados y facturas emitidas.

## 6.2 Requisitos No Funcionales

R.27 Escalabilidad  
 Actores: Administrador, equipo de desarrollo  
 Descripción: El sistema debe crecer sin afectar el rendimiento.

R.28 Disponibilidad  
 Actores: Administrador, cliente, técnico  
 Descripción: El sistema debe estar disponible cuando sea necesario.

R.29 Rendimiento  
 Actores: Equipo de desarrollo  
 Descripción: El sistema debe responder rápidamente aún cuando haya alta demanda.

R.30 Mantenibilidad  
 Actores: Administrador, equipo de desarrollo  
 Descripción: El sistema debe ser fácil de mantener y actualizar.

R.31 Seguridad  
 Actores: Administrador, equipo de desarrollo  
 Descripción: Los datos de usuarios y proyectos deben protegerse mediante autenticación segura.

R.32 Usabilidad  
 Actores: Equipo de desarrollo, cliente, técnico  
 Descripción: El sistema debe ser fácil de usar y comprender, con una interfaz intuitiva.

R.33 Compatibilidad  
 Actores: Equipo de desarrollo  
 Descripción: El sistema debe ser compatible con distintos navegadores web y dispositivos móviles sin pérdida de funcionalidad o diseño.

R.34 Testing continuo  
 Actores: Administrador, equipo de desarrollo  
 Descripción: La plataforma debe contar con monitoreo constante para detectar fallos, cuellos de botella y comportamientos inusuales.

## 6.3 Síntesis de necesidades de cada perfil

El Administrador del Sistema necesita gestionar usuarios mediante la creación, modificación y eliminación de cuentas, definir roles y permisos, monitorear el estado del sistema, recibir alertas sobre fallos, supervisar su rendimiento, así como realizar copias de seguridad y restauración de datos en caso de fallos.

El Coordinador de Proyectos requiere administrar las instalaciones, asignando recursos y plazos, dar seguimiento al estado de los proyectos notificando avances a los clientes, coordinar los equipos técnicos asegurando la disponibilidad de materiales y generar reportes sobre tiempos de instalación, eficiencia y demanda de servicios.

El Técnico de Instalación debe poder acceder al sistema, gestionar su perfil, consultar asignaciones, actualizar el estado de los proyectos en tiempo real, registrar incidencias encontradas durante la instalación y gestionar el inventario mediante el registro del uso de materiales y herramientas.

El equipo de Soporte Técnico necesita atender solicitudes de clientes o empleados sobre problemas en las instalaciones y monitorear el sistema para analizar métricas de rendimiento y prevenir fallos.

Finalmente, el Cliente debe contar con la posibilidad de registrarse en la plataforma, iniciar sesión, explorar y buscar servicios, solicitar instalaciones, realizar seguimiento de sus proyectos, gestionar y programar mantenimientos, actualizar su información personal, solicitar soporte técnico en caso de inconvenientes y dejar reseñas y calificaciones sobre los servicios recibidos. Además, debe poder realizar la facturación de sus compras y recibir el comprobante.

## Análisis del sistema actual

## 7.1 Arquitectura general del sistema monolítico

El sistema actual de GreenEnergy SPA es monolítico, lo que significa que todas las funciones están integradas en una sola aplicación. Este sistema gestiona la administración de proyectos, la asignación de técnicos y la gestión de instalaciones y mantenimiento, pero no está diseñado para crecer de manera flexible ni adaptarse fácilmente a nuevas necesidades.

Características principales del sistema:

* Todos los módulos dependen entre sí, por lo que cualquier cambio afecta a toda la aplicación.
* Se usa una base de datos centralizada, lo que genera tiempos de carga elevados y sobrecarga el sistema con el aumento de solicitudes.
* La gestión de instalaciones impacta directamente en mantenimientos, facturación e inventario, causando bloqueos y fallas en cascada.
* Modificar o agregar nuevas funcionalidades requiere ajustes en múltiples módulos, haciendo el sistema rígido y poco flexible.
* Si hay alta carga en otros módulos, la atención al cliente se vuelve lenta o inaccesible.

## 7.2 Puntos débiles detectados

El sistema monolítico de GreenEnergy SPA presenta varias deficiencias que afectan su eficiencia:

1. Problemas en la administración de proyectos (Módulo de Gestión de Instalaciones)

* Existen retrasos en la asignación de técnicos y en la ejecución de instalaciones debido a la falta de un proceso estructurado.

1. Deficiencias en el seguimiento de instalaciones (Módulo de Clientes y Módulo de Gestión de Instalaciones)

* Ausencia de una plataforma centralizada para monitorear en tiempo real el estado de cada instalación, dificultando el control y la supervisión interna de los proyectos.

1. Limitaciones en la gestión de solicitudes (Módulo de Mantenimiento y Soporte)

* No hay un proceso eficiente para registrar, organizar y dar seguimiento a las solicitudes, lo que provoca demoras y respuestas inconsistentes.
* Falta de comunicación automatizada para informar a clientes y empleados sobre el estado de sus solicitudes o servicios, generando incertidumbre y retrasos en la atención.

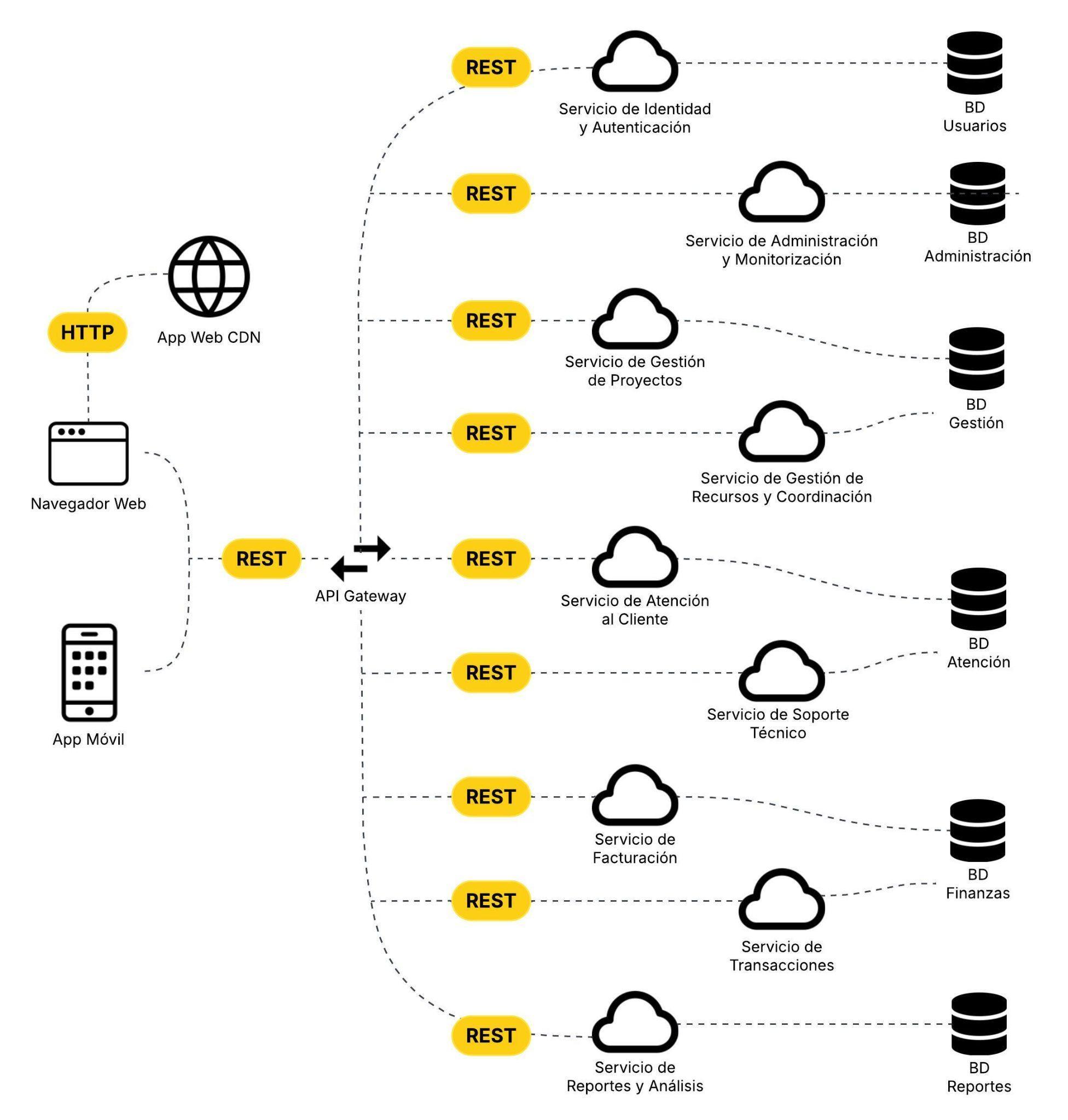
## Diseño de la nueva arquitectura

## 8.1 Descripción general de la arquitectura basada en microservicios

La nueva arquitectura de GreenEnergy SPA usará microservicios para mejorar la flexibilidad, escalabilidad y seguridad del sistema. Se organizará por dominios, con módulos independientes y bases de datos separadas para mayor autonomía y protección de datos.

Se aplicará cifrado, OAuth 2.0, JWT, firewalls y detección de anomalías, además de pruebas de seguridad antes del despliegue. El desarrollo se basará en Java, Spring Boot, MySQL y AWS, y se capacitará a los empleados en su uso.

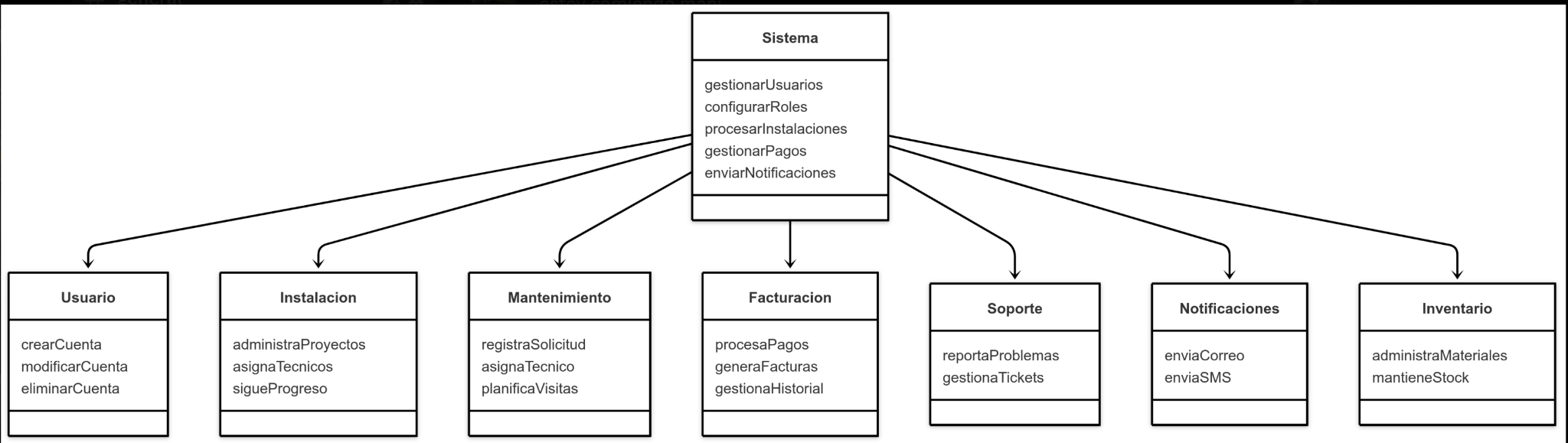
Cumplirá con la Ley de Protección de Datos de Chile y, para clientes internacionales, se podrá aplicar GDPR.



**8.2. Diagrama de Actores de Alto Nivel**

| # | Requisito Principal | Administrador | Técnico | Cliente | Coordinador | Soporte Técnico |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Inicio de sesión | x | x | x | x | x |
| 2 | Gestión de usuarios | x |  |  |  |  |
| 3 | Gestión de proyectos |  |  |  | x |  |
| 4 | Seguimiento de proyectos |  | x | x | x |  |
| 5 | Gestión de perfil | x | x | x | x | x |
| 6 | Atención de solicitudes de soporte |  |  | x | x | x |
| 7 | Solicitud de instalación |  |  | x |  |  |
| 8 | Notificaciones automáticas |  | x | x | x | x |
| 9 | Historial de consumo energético | x |  | x |  | x |
| 10 | Generación de facturas | x |  | x |  |  |
| 11 | Escalabilidad | x |  |  |  |  |
| 12 | Disponibilidad | x | x | x |  |  |
| 13 | Seguridad | x |  |  |  |  |

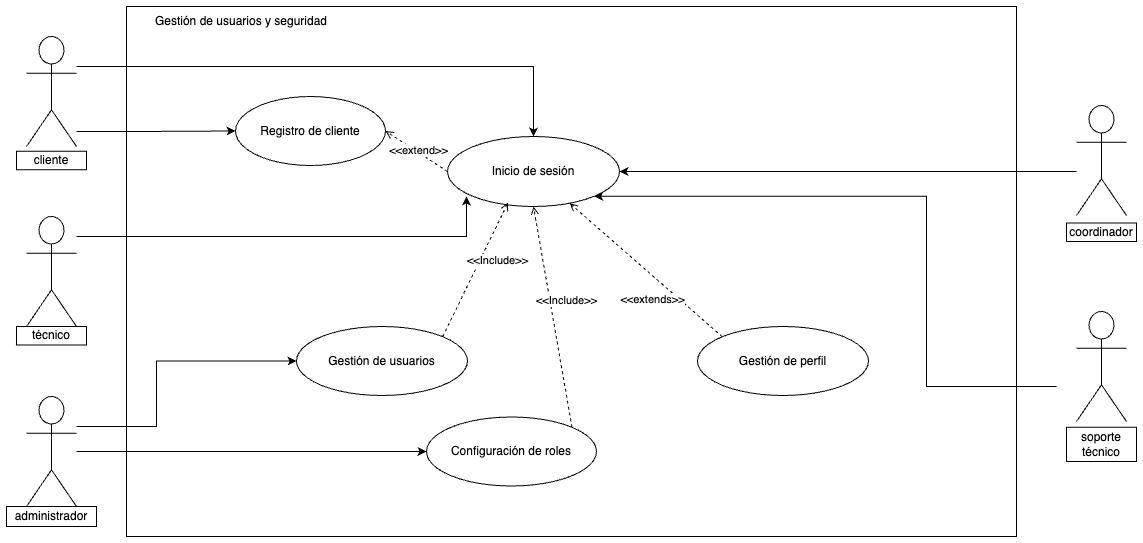
## 8.3 Diagrama de Clases



[De clases.png](https://drive.google.com/file/d/15rcwR6IbidEUdm8CxlqEaDPnOUP2baNN/view?usp=sharing)

## 8.4. Diagramas de Casos de Uso

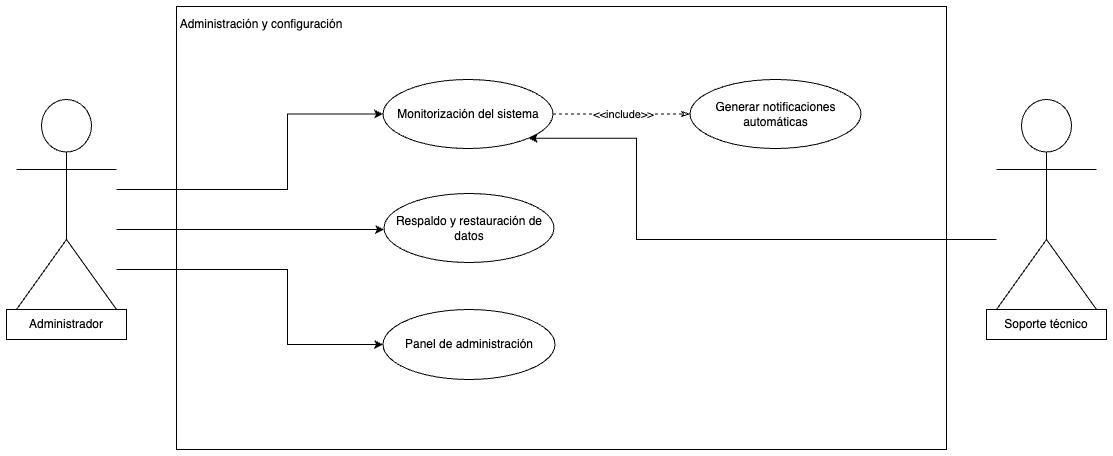
**Diagrama N°1**



Casos de uso:

* Registro de cliente: Permite al cliente registrarse en la plataforma web proporcionando la información necesaria. Después del registro, el cliente puede iniciar sesión en el sistema.
* Inicio de sesión: Permite a todos los usuarios autenticarse en el sistema y acceder a sus funcionalidades de acuerdo a su rol.
* Gestión de perfil: Permite a todos los usuarios actualizar su información personal y contraseña. Se accede a esta función después de iniciar sesión.
* Gestión de usuarios: Permite al administrador crear, modificar y eliminar cuentas de clientes y empleados con distintos niveles de acceso. El administrador podrá acceder a esta funcionalidad después de iniciar sesión.
* Configuración de roles: Permite al administrador definir permisos para los usuarios dentro del sistema. El administrador podrá acceder a esta funcionalidad después de iniciar sesión.

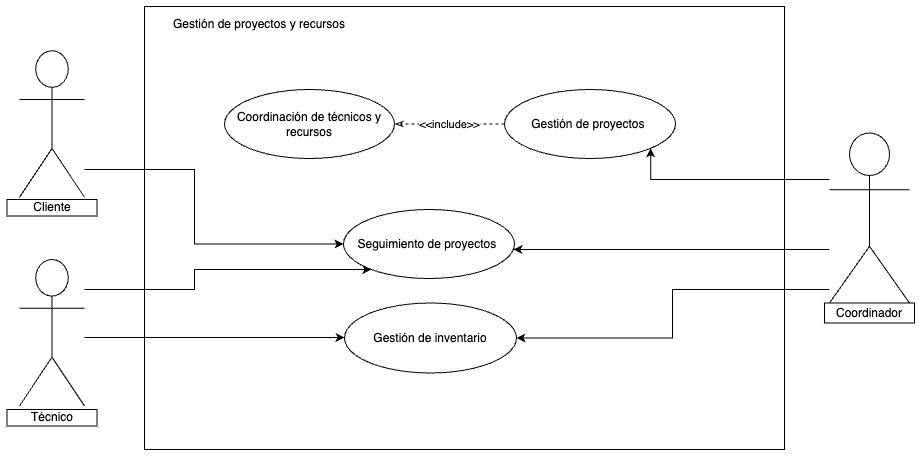
**Diagrama N°2**



Casos de uso:

* Monitorización del sistema: Permite al administrador y al soporte técnico supervisar el estado del sistema, recibir alertas sobre posibles fallos y monitorizar el rendimiento.
* Generar notificaciones automáticas: Se encarga de enviar notificaciones cuando se detectan eventos importantes en la monitorización del sistema.
* Respaldo y restauración de datos: Permite al administrador realizar copias de seguridad periódicas y restaurar datos en caso de pérdida o fallo
* Panel de administración: Espacio que permite al administrador gestionar usuarios, roles, permisos y supervisar el estado del sistema.

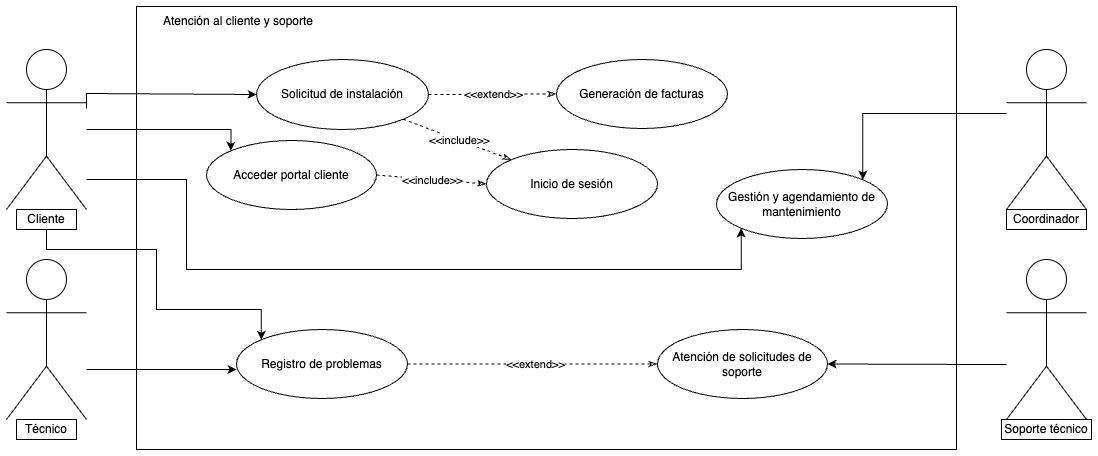
**Diagrama N°3**



Casos de uso:

* Gestión de proyectos: Permite al coordinador crear nuevos proyectos de instalación, asignar recursos, técnicos y definir plazos.
* Coordinación de técnicos y recursos: Permite al coordinador asignar técnicos y materiales a cada proyecto en función de disponibilidad y necesidades. Este se ejecuta junto a la gestión de proyectos.
* Seguimiento de proyectos: Permite al técnico y coordinador actualizar el estado del proyecto. Permite al coordinador y cliente visualizar dicho estado. Permite al coordinador notificar a los clientes sobre avances o alertas de retraso.
* Gestión de inventario: Permite al técnico registrar el uso de materiales y herramientas en cada proyecto e informar niveles de stock. También permite al coordinador asegurar la disponibilidad de materiales necesarios para cada instalación.

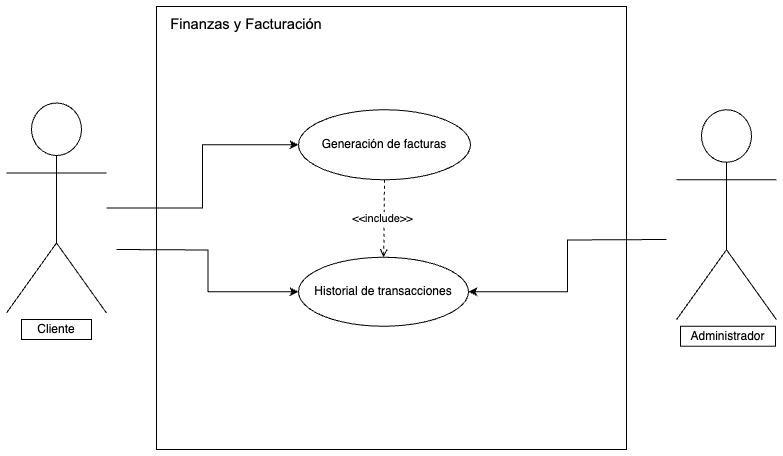
**Diagrama N°4**



Casos de uso:

* Solicitud de instalación: Permite al cliente solicitar evaluaciones y presupuestos para instalaciones de paneles solares. Para esto, el cliente debe iniciar sesión.
* Generación de facturas: El sistema debe emitir facturas automáticamente después de contratado un servicio.
* Acceder al portal cliente: Luego de iniciar sesión, permite al cliente acceder al espacio donde pueden ver los servicios disponibles, su historial de consumo energético y el estado de sus instalaciones, mantenimientos y solicitudes.

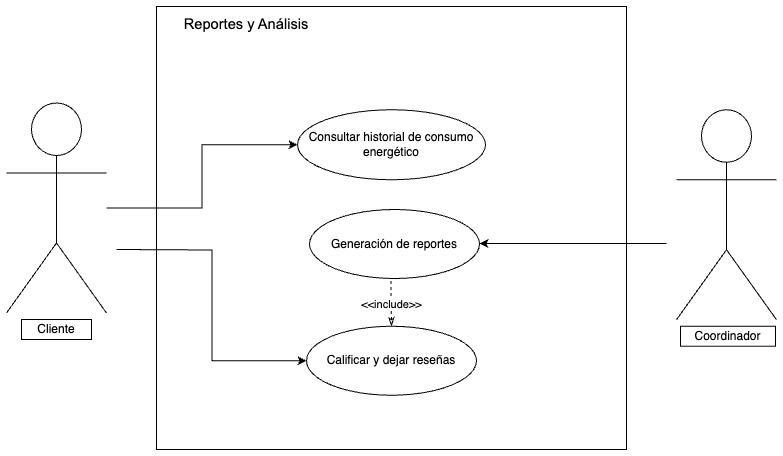
**Diagrama N°5**



Casos de uso:

* Generación de facturas: Luego de contratado un servicio (por el cliente) el sistema emitirá automáticamente la factura. Esta transacción se registrará en el historial para que pueda ser consultada en el futuro.
* Historial de transacciones: Permite al cliente y administrador consultar pagos realizados y facturas emitidas.

**Diagrama N°6**



Casos de uso:

* Consultar historial de consumo energético: Permite al cliente consultar su historial de consumo energético en una tabla con registros por fecha, kWh y costos.
* Generación de reportes: Permite al coordinador crear informes sobre tiempos de instalación, eficiencia y demanda de servicios. Para estos informes se utilizarán las reseñas y calificaciones de los clientes.
* Calificar y dejar reseñas: Permite al cliente evaluar servicios contratados y dejar comentarios en el sitio web.

## 

## Planificación de la migración

## 9.1. Fases de Migración (por módulo o servicio)

\* Estas son las fases que seguiremos para la migración de monolito a microservicios:

**1. Análisis y Planificación**

- Evaluar dependencias y definir el orden de migración (se recomienda empezar con servicios menos críticos).

- Identificar módulos con alta cohesión y bajo acoplamiento.

- Diseñar APIs para comunicación entre los nuevos microservicios y el monolito.

- 1era Semana

**2. Extracción Progresiva de Microservicios**

- Primero, migrar Gestión de Usuarios y Seguridad porque es fundamental y puede operar independiente. (2da Semana)

- Luego, mover Atención al Cliente y Soporte, ya que tiene menos impacto en el resto del sistema (3era Semana).

- Posteriormente, extraer Facturación y Finanzas para mejorar rendimiento en cálculos financieros (4ta Semana).

- Finalmente, dividir Inventario y Logística, Gestión de Proyectos y Reportes (5ta Semana).

**3. Sincronización de Datos y Comunicación**

- Implementar mensajería asíncrona (Kafka, RabbitMQ, AWS SQS) para evitar bloqueos en la base de datos.

- Mantener un modo híbrido donde el monolito y los microservicios coexistan mientras se migran datos.

**4. Desconexión del Monolito**

- Después de migrar todas las funcionalidades, eliminar los módulos redundantes en el monolito.

- Hacer pruebas de carga y rendimiento en cada microservicio antes de finalizar la transición.

## 

## 

## 9.2. Consideraciones de Compatibilidad

- Lenguaje: Ambos usan Java, pero hay que revisar si el monolito está en una versión antigua y actualizarlo.

- Base de Datos: MySQL sigue siendo la misma, pero cada microservicio puede tener su propia base de datos o compartir una según necesidades.

- Autenticación: Si el monolito usa sesiones, hay que migrar a JWT u OAuth2 para que microservicios y monolito compartan autenticación.

- Comunicación: Se usará APIs REST y mensajería para evitar dependencia directa entre sistemas.

- Despliegue: Spring Boot 3 es compatible con AWS, incluyendo EC2, ECS, Lambda y RDS para la base de datos.

Además, hay alta compatibilidad entre Spring Boot 3 y las herramientas de AWS.

Puede usarse con:

- Amazon RDS (para MySQL): Maneja bases de datos de forma escalable.

- AWS Lambda: Para tareas pequeñas sin necesidad de servidores.

- Amazon API Gateway: Maneja el tráfico y facilita la integración entre microservicios.

- ECS o Kubernetes (EKS): Para orquestación y escalabilidad de microservicios.

- S3 y CloudFront: Para almacenamiento y distribución de archivos.

## 9.3. Identificación de Riesgos y su plan de mitigación

Identificamos algunos riesgos y cómo mitigarlos:

**1**.

- Riesgo: Pérdida de datos durante la transición.

- Impacto: Alto.

- Forma de mitigar: Implementar un proceso de sincronización de datos antes de cortar el monolito. Hacer backups diarios.

**2.**

- Riesgo: Tiempo de inactividad durante la migración

- Impacto: Medio

- Forma de mitigar: Migrar en fases y probar cada microservicio antes de desconectar el monolito.

**3.**

- Riesgo: Errores de integración entre microservicios y monolito

- Impacto: Medio

- Forma de mitigar: Usar pruebas automatizadas y mock services antes del despliegue.

**4.**

- Riesgo: Mayor complejidad operativa

- Impacto: Alto

- Forma de mitigar: Capacitar al equipo en herramientas de monitoreo y gestión de microservicios (Prometheus, Grafana, AWS CloudWatch).

**5.**

- Riesgo: Problemas de seguridad al exponer múltiples APIs

- Impacto: Alto

- Forma de mitigar: Implementar autenticación con OAuth2 y API Gateway para proteger los endpoints.

## Planificación Ágil y Cronograma

## 10.1. Breve Explicación del Enfoque Ágil Usado (Scrum. Kanban, etc.)

La metodología ágil es un conjunto de principios y prácticas para la gestión de proyectos de software que busca entregar valor de

manera rápida e incremental. En lugar de seguir un plan rígido desde el inicio, se adapta a cambios y necesidades del cliente durante

el desarrollo. Se basa en el Manifiesto Ágil, que prioriza: Individuos e interacciones sobre procesos y herramientas, Software funcionando

sobre documentación extensa, colaboración con el cliente sobre negociación de contratos y respuesta ante el cambio sobre seguir un

plan rígido.

Hay varios enfoques ágiles, entre ellos, Scrum y Kanban. Scrum organiza su trabajo en sprints de 1 a 4 semanas. Se termina el sprint

se tiene un entregable. También existe una combinación de enfoques llamado Scrumban. Kanban organiza su trabajo en un tablero: to-do,

in progress y done.

Nosotros ocupamos Scrumban organizamos nuestro trabajo ocupando ambas estrategias, ocupamos sprint y también distribuimos nuestras tareas

en un tablero, en nuestro caso ocupamos trello.

## 10.2. Roles asumidos por los integrantes.

- Scrum Master: Omar Hurtado

- Product Owner: Valeria Fonseca

- Arquitecto de Soluciones: Gerardo Soto.

- Además todos colaboramos en la construcción de entregables como equipo de desarrollo.

## 10.3. Carta Gantt con cronograma de implementación y revisión.

Link de acceso: https://docs.google.com/spreadsheets/d/1P9WUB9gyvAG6AS9K9VNsMXa-tV3O5e2W/edit?usp=sharing&ouid=105300964347349510334&rtpof=true&sd=true

## Conclusiones y recomendaciones

## 11.1 Lecciones aprendidas

Las lecciones aprendidas en este trabajo se vieron reflejadas en los siguientes puntos:

**1- Trabajo en equipo y comunicación fluida:** La colaboración entre los diferentes miembros del equipo fue vital para el éxito. La comunicación continua y la actualización constante sobre el avance de las tareas ayudó a evitar malentendidos y aseguró que todos estén alineados.

**2- Manejo de tiempos y recursos:** El control de los plazos ha sido una de las dificultades de este proyecto. Aprender a priorizar tareas y ajustar el plan de trabajo según sea necesario fue una habilidad esencial para mantener el proyecto en curso.

**3- Trabajo colaborativo y adaptabilidad del equipo**: El trabajo en equipo fue esencial para superar los desafíos del proyecto. Al ser un equipo pequeño, la comunicación constante y la colaboración estrecha permitieron aprovechar las fortalezas de cada miembro. Este punto fue añadido por un problema que surgió en las etapas finales de este informe.

**4- Resolución de dudas y aprendizaje constante:** La resolución de dudas fue clave para avanzar sin bloqueos. La comunicación abierta y el intercambio de conocimientos dentro del equipo nos permitió encontrar soluciones rápidas y fomentar el aprendizaje continuo, lo que mejoró el enfoque en los desafíos del proyecto. En este punto destacamos la participación de los 3 miembros del equipo, los cuales participaron activamente en la resolución de dudas de sus compañeros.

## 

## 

## 

## 

## 

## 11.2 Potencial escalabilidad del sistema

El sistema ha sido diseñado con una arquitectura flexible que permite su crecimiento y adaptación a medida que aumentan las necesidades de la empresa. Gracias al uso de Spring Boot y AWS, la aplicación puede escalar tanto vertical como horizontalmente, asegurando un rendimiento óptimo sin afectar la estabilidad. MySQL, por su parte, ofrece opciones de replicación y división de datos que facilitan la gestión de grandes volúmenes de información sin degradación del servicio.

Además, la combinación de estas tecnologías permite una distribución eficiente de la carga de trabajo, optimizando el uso de recursos y garantizando una respuesta rápida ante picos de demanda. Este enfoque asegura que el sistema pueda expandirse de manera sostenible, manteniendo su eficiencia operativa y reduciendo costos innecesarios a largo plazo.

La arquitectura de microservicios permite escalar servicios individualmente, es por eso que el sistema puede escalar sin problemas.

## 11.3 Posibles mejoras futuras

Aunque el sistema es sólido, aún se pueden hacer mejoras para optimizar la compatibilidad entre microservicios y garantizar una escalabilidad más eficiente. Adoptar una arquitectura basada en microservicios con Spring Boot y desplegada en AWS permitiría una mayor flexibilidad y facilidad de crecimiento, ya que cada servicio podría escalar de manera independiente según la demanda, sin afectar al resto del sistema.

El uso de Java como lenguaje de programación aporta estabilidad y rendimiento, facilitando la integración con diversas herramientas y bases de datos. Además, Spring Boot simplifica la creación y gestión de microservicios, reduciendo la complejidad del desarrollo y mejorando la mantenibilidad del código.

Aprovechar la infraestructura en la nube de AWS brindaría beneficios como el ajuste automático de recursos, balanceo de carga y alta disponibilidad, lo que garantizaría un sistema más ágil y resistente a fallos. Estas mejoras permitirían una respuesta más rápida ante aumentos de tráfico, optimizando costos y asegurando un crecimiento sostenible del sistema.

Asimismo, se podría implementar inteligencia artificial para mejorar la atención a clientes. Se podrá ocupar para la resolución de dudas y consultas 24 horas.

**Enlaces anexados:**

Trello:

<https://trello.com/invite/b/67e1ff5b22aab6f7a5719475/ATTIad5067a3ec92ca102e54bcf9c671504d7CC0C1B3/entrega-1>

Carpeta Drive:

https://drive.google.com/drive/folders/1c86lMK17CpbbHh2-HaYpoaV\_4myizcQs?usp=drive\_link

GitHub:

<https://github.com/TraucoMaldito/FullStack->