Diseño de una arquitectura de software modular, escalable y segura para una plataforma de servicios de cuidado infantil

Yessica Marcela Gualdrón Gómez
Rusbell Wbeimar Ruiz Portocarrero

Proyecto de grado entregado para obtener el título de
Magister en Ingeniería de Software

Dirigida por
MSc. Juan Felipe Córdoba Victoria

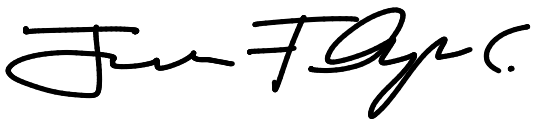
Pontificia Universidad Javeriana Cali
Facultad de Ingeniería y Ciencias
Maestría en Ingeniería de Software
Santiago de Cali
30 de noviembre de 2025

Santiago de Cali, 30 de noviembre de 2025.

Señores
Pontificia Universidad Javeriana Cali.
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software.
Cali.

Cordial Saludo.

Por medio de la presente hago constar que en mi calidad de director de trabajo de grado he revisado el proyecto titulado "Diseño de una arquitectura de software modular, escalable y segura para una plataforma de servicios de cuidado infantil" realizado por los estudiantes de Magister en Ingeniería de Software Yessica Marcela Gualdrón Gómez (cod: 9018668) y Rusbell Wbeimar Ruiz Portocarrero (cod: 9019031), el cual se encuentra terminado y considero que cumple con los requisitos para ser sustentado.
Atentamente,



MSc. Juan Felipe Córdoba Victoria

Santiago de Cali, 30 de noviembre de 2025.

Señores
Pontificia Universidad Javeriana Cali
Ph.D. Luisa Rincón
Directora Maestría en Ingeniería de Software
Cali.

Cordial Saludo.

Nos permitimos presentar a su consideración el proyecto de grado titulado "Diseño de una arquitectura de software modular, escalable y segura para una plataforma de servicios de cuidado infantil" con el fin de cumplir con los requisitos exigidos por la Universidad y para que sea sometido a revisión del jurado y cumpla su aprobación, para conseguir posteriormente el título de Magister en Ingeniería de Software.

Atentamente,

Yessica Marcela Gualdrón Gómez
Código: 9018668

Rusbell Wbeimar Ruiz Portocarrero
Código: 9019031

Ficha Resumen
Trabajo de Grado Maestría en Ingeniería de Software

TíTULO: Diseño de una arquitectura de software modular, escalable y segura para una plataforma de servicios de cuidado infantil

1. Énfasis: Ingeniería de Software
2. Área de trabajo: Tecnología
3. Tipo de proyecto (Aplicado, Innovación, Investigación): Aplicado
4. Estudiante: Yessica Marcela Gualdrón Gómez, Rusbell Wbeimar Ruiz Portocarrero
5. Correo electrónico: ygualdron@javerianacali.edu.co, rusbellruizp@javerianacali.edu.co
6. Dirección y teléfono: Cra 52a N° 174b-08, Torre 2 Apto 204, conjunto Residencial Parque Baviera, Villas del Prado Bogotá.
7. Director: Juan Felipe Córdoba Victoria
8. Vinculación del director: (Planta/Cátedra/Externo)
9. Correo electrónico del director: juan.cordobavictoria@javerianacali.edu.co
10. Co-Director (Si aplica):
11. Grupo o empresa que lo avala (Si aplica):
12. Otros grupos o empresas:
13. Palabras clave(al menos 5): Arquitectura de Software, atributos de calidad, multiplataforma, cuidado infantil, Modelo C4.
14. ODS que aplica al proyecto (Agenda 2030): ODS 8, Meta 8.2
15. Fecha de inicio:
16. Resumen: En este proyecto se desarrolló SuricatosCare, una plataforma en la nube creada bajo una arquitectura sólida para responder a la creciente demanda de servicios de cuidado infantil en Colombia, donde se requiere ampliar el acceso a opciones de calidad y asequibles. La solución se desarrolló con una arquitectura modular y escalable basada en AWS (Lambda, API Gateway, Cognito, DynamoDB y RDS), aplicando principios de arquitectura hexagonal que fortalecen la seguridad, la mantenibilidad y el desempeño del sistema. La validación del MVP evidenció una integración eficiente entre componentes y un funcionamiento confiable de los flujos esenciales. Los resultados confirman la viabilidad técnica de la propuesta y su potencial

para mejorar la experiencia de las familias, ampliando el acceso a servicios de cuidado infantil mediante una plataforma flexible y preparada para su futura expansión. Palabras Clave: Arquitectura de Software, integrabilidad, escalabilidad, seguridad, cuidad infantil, Modelo C4.

Agradecimientos

Queremos expresar nuestra gratitud a nuestras familias, quienes con su apoyo, paciencia y confianza nos acompañaron durante todo este proceso. Su presencia constante nos brindó la fortaleza necesaria para avanzar incluso en los momentos más exigentes.

Agradecemos al profesor Juan Felipe Córdoba Victoria, director de este trabajo de grado, por su guía académica y por el compromiso demostrado en cada sesión de revisión. Sus observaciones y orientaciones fueron fundamentales para el desarrollo y la consolidación de este trabajo.

Extendemos también un agradecimiento especial a la profesora Luisa, directora de la Maestría en Ingeniería de Software, por su disposición permanente y por promover un entorno formativo que favorece la creatividad, el pensamiento crítico y el crecimiento profesional.

Reconocemos el trabajo colaborativo que consolidamos como equipo. La dedicación, el compromiso y el apoyo mutuo hicieron posible superar los desafíos del proyecto y mantener la motivación a lo largo de este camino.

A todas las personas que, de una u otra manera, contribuyeron al desarrollo de este trabajo, nuestro sincero agradecimiento.

Resumen

En este proyecto se desarrolló SuricatosCare, una plataforma en la nube creada bajo una arquitectura sólida para responder a la creciente demanda de servicios de cuidado infantil en Colombia, donde se requiere ampliar el acceso a opciones de calidad y asequibles. La solución se desarrolló con una arquitectura modular y escalable basada en AWS (Lambda, API Gateway, Cognito, DynamoDB y RDS), aplicando principios de arquitectura hexagonal que fortalecen la seguridad, la mantenibilidad y el desempeño del sistema. La validación del MVP evidenció una integración eficiente entre componentes y un funcionamiento confiable de los flujos esenciales. Los resultados confirman la viabilidad técnica de la propuesta y su potencial para mejorar la experiencia de las familias, ampliando el acceso a servicios de cuidado infantil mediante una plataforma flexible y preparada para su futura expansión.

Palabras Clave: Arquitectura de Software, integrabilidad, escalabilidad, seguridad, cuidado infantil, Modelo C4.

Abstract

In this project, SuricatosCare was developed, a cloud platform created under a solid architecture to respond to the growing demand for childcare services in Colombia, where it is necessary to expand access to quality and affordable options. The solution was developed with a modular and scalable architecture based on AWS (Lambda, API Gateway, Cognito, DynamoDB and RDS), applying hexagonal architecture principles that strengthen the system's security, maintainability, and performance. The validation of the MVP showed efficient integration between components and reliable operation of the essential flows. The results confirm the technical feasibility of the proposal and its potential to improve the experience of families by expanding access to childcare services through a flexible platform prepared for its future expansion.

Keywords: Software Architecture, integrability, scalability, security, childcare, C4 Model.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Los primeros años de vida constituyen una etapa determinante en el desarrollo de las personas, pues en ellos se configuran las bases físicas, emocionales y cognitivas que acompañarán al individuo durante su vida adulta (UNESCO (2022); UNICEF (sf)). Sin embargo, brindar una atención integral en esta etapa exige tiempo, recursos y acceso a servicios especializados, algo que muchas familias no pueden garantizar debido a las exigencias laborales y a la escasez de programas de apoyo. En Colombia, el Instituto Colombiano de Bienestar Familiar (ICBF) ofrece modalidades de atención que han ampliado la cobertura, pero estas siguen siendo insuficientes para responder a la creciente demanda. Los servicios privados, por su parte, suelen estar asociados a costos elevados, lo que limita aún más el acceso. Ante este panorama, y en medio de una transformación digital acelerada, surge la necesidad de explorar alternativas tecnológicas que faciliten la conexión entre familias y profesionales capacitados en el cuidado de la primera infancia.

A partir de este problema, la investigación se orientó al diseño de una arquitectura de software modular, escalable y segura que sirva como base para el desarrollo de artefactos de software destinados a ofrecer servicios de cuidado infantil. El objetivo general se acompañó de cuatro metas específicas: analizar las expectativas de familias en Bogotá frente a los servicios de cuidado infantil, identificar los requerimientos arquitectónicos más apropiados, diseñar la arquitectura siguiendo el modelo C4 hasta el nivel de componentes y, finalmente, desarrollar un prototipo funcional que validara las decisiones adoptadas.

Para alcanzar estos propósitos, se optó por una metodología de carácter aplicado y exploratorio. En primera instancia, se revisaron antecedentes y referentes que dieran sustento teórico al proyecto. Luego, se realizó un levantamiento de información con familias de la ciudad de Bogotá, lo cual permitió precisar necesidades y traducirlas en requerimientos funcionales y no funcionales. Con estos insumos se diseñó la arquitectura del sistema y se construyó un prototipo que funcionó como prueba de concepto para demostrar la integración adecuada entre los componentes y el comportamiento confiable de los flujos esenciales.

El aporte de este trabajo se manifiesta en dos planos. En el ámbito social, ofrece una alternativa que puede contribuir a mejorar el acceso a servicios de cuidado infantil, impactando directamente en la calidad de vida de las familias. En el plano tecnológico, aporta al campo de la ingeniería de software al mostrar cómo una arquitectura bien definida, validada a través de un prototipo,

puede responder a drivers de calidad relevantes en contextos de alta demanda. De esta manera, la investigación busca articular un beneficio comunitario con una contribución académica y técnica.

La tesis se estructura en cinco capítulos que recorren el proyecto de manera progresiva. Inicia con la definición del problema, los objetivos planteados, el alcance y la metodología que guió el trabajo. Continúa con un marco de referencia que reúne los fundamentos teóricos y el contexto tecnológico asociado a las plataformas de cuidado infantil, además de las tendencias arquitectónicas relevantes. El desarrollo del proyecto se presenta en el tercer capítulo, donde se describe el análisis de necesidades, la identificación de requisitos y la construcción del diseño arquitectónico bajo el modelo C4, junto con las decisiones adoptadas, la implementación del prototipo y la gestión de riesgos. El cuarto capítulo expone el proceso de evaluación de la arquitectura, incluyendo la validación mediante escenarios, el análisis con el AWS Well-Architected Framework y los resultados obtenidos. El documento cierra con un capítulo final que recoge las conclusiones, las recomendaciones para trabajos futuros y las principales lecciones aprendidas durante el desarrollo de la propuesta.

1.1. Definición del problema

1.1.1. Planteamiento del problema

Los primeros años de vida de una persona son considerados como una etapa crucial e importante para el desenvolvimiento del ser humano en su adultez. Se considera que la estimulación adecuada y temprana, acompañada con un ambiente sano para el desarrollo y crecimiento de los niños durante los primeros años de vida; son la fase de desarrollo más importante de todo el ciclo vital (UNESCO, 2022). Un desarrollo adecuado de la primera infancia donde se abarque no solo el dominio físico; sino también aspectos afectivos, motrices, de lenguaje y desarrollo sensorial, se constituyen un aspecto importante para el futuro de la persona; debido a que garantiza beneficios notables sobre el bienestar durante la vida (UNICEF, sf).

Sin embargo, alcanzar un desarrollo integral en esta etapa exige la disponibilidad de tiempo, recursos y acceso a servicios especializados. En muchos hogares colombianos, gran parte de los ingresos proviene del trabajo remunerado, lo que dificulta que los cuidadores principales, en especial las madres, puedan dedicar el tiempo suficiente a la atención de sus hijos sin comprometer la estabilidad económica del hogar. Esta situación ha derivado en una creciente demanda de servicios de cuidado infantil que, además de favorecer el desarrollo de los menores, ofrezcan a las familias la flexibilidad necesaria para cumplir con sus responsabilidades laborales (Araujo, 2015).

Actualmente en Colombia existen ciertos programas que brindan el acceso a la educación y cuidado de niños los cuales se encuentran tipificados en programas gubernamentales y programas o servicios privados. Dentro de los gubernamentales se destacan los liderados por el Instituto Co-

lombiano de Bienestar Familiar (ICBF), que ofrece cuatro modalidades de atención: Institucional, Familiar, Propia e Intercultural y Comunitaria (ICBF, 2021). Sin embargo, la cobertura continúa siendo insuficiente frente a la demanda real, especialmente en zonas urbanas como Bogotá, donde se concentra un gran número de familias en búsqueda de servicios de cuidado infantil. De acuerdo con el DANE (2022), cerca del 47 % de los niños menores de cinco años en el país no acceden a programas formales de cuidado o educación inicial, lo que refleja la magnitud de la brecha en el acceso.

Por otro lado, los servicios privados como jardines infantiles, programas empresariales y plataformas digitales emergentes suelen estar asociados a costos elevados que limitan el acceso de los sectores más vulnerables (CEPAL, 2016). La desigualdad en la cobertura y el costo de los servicios genera consecuencias importantes: limita el desarrollo integral de los niños, incrementa la sobrecarga en las familias y perpetúa brechas sociales y de género, ya que las mujeres siguen asumiendo en mayor medida el rol de cuidadoras principales (ONUMujeres, 2024).

Si la situación actual se mantiene, es previsible que aumenten las dificultades de las familias para conciliar el cuidado infantil con la vida laboral, con efectos negativos tanto en el desarrollo de los niños como en la participación equitativa de las mujeres en el mercado de trabajo. Esta realidad plantea la necesidad urgente de diseñar soluciones innovadoras, apoyadas en el uso de tecnologías de la información, que permitan ampliar el acceso a servicios de cuidado infantil de calidad, adaptados a las necesidades de las familias urbanas.

En este contexto, cobra relevancia explorar el desarrollo de arquitecturas de software flexibles, escalables, seguras y con capacidad de integración, que sirvan de base para plataformas digitales orientadas a conectar de manera confiable y eficiente a las familias con profesionales capacitados en la atención a la primera infancia.

1.1.2. Formulación del problema

En el marco del déficit de servicios de atención a la primera infancia y la acelerada transformación digital que ha dado lugar al surgimiento de nuevas plataformas de intermediación que buscan establecer un puente de comunicación entre familias y cuidadores, se plantean las siguientes preguntas de investigación:

- ¿Cuáles son las principales necesidades y expectativas de las familias de Bogotá en cuanto a los servicios de cuidado infantil?
- ¿Qué características debe tener una arquitectura escalable, segura y con capacidad de integrabilidad, sobre la cual se puedan construir aplicaciones de servicios de cuidado infantil?

- ¿Cómo debe estructurarse la arquitectura para asegurar una adecuada comunicación entre sus diferentes componentes y módulos?
- ¿Cómo se podrá evaluar que la arquitectura propuesta garantiza los drivers arquitectónicos identificados?
- ¿Qué tecnologías y herramientas son apropiadas para implementar la arquitectura propuesta?

1.2. Objetivos del proyecto

1.2.1. Objetivo General

Diseñar una arquitectura modular, escalable y segura que facilite el acceso de familias a profesionales capacitados en atención a la primera infancia.

1.2.2. Objetivos específicos

- Analizar las necesidades y expectativas de una muestra representativa de familias ubicadas en la ciudad de Bogotá respecto a los servicios de cuidado infantil, con el fin de identificar las características clave que debe incluir el software.
- Identificar los requerimientos arquitectónicos apropiados para una arquitectura modular, escalable, segura y con capacidad de integración.
- Diseñar la arquitectura de software bajo el modelo C4, hasta el nivel de componentes.
- Desarrollar un Prototipo Funcional (MVP) como prueba de la arquitectura planteada.

1.3. Delimitaciones y alcances

El desarrollo de este trabajo de grado se delimitó a la ciudad de Bogotá, tomando como referencia una muestra representativa de familias que permitió identificar de manera directa sus necesidades y expectativas frente a los servicios de cuidado infantil. Esta delimitación geográfica facilitó el acceso a información pertinente y a una población diversa, propia de un entorno urbano con alta conectividad digital. La ciudad de Bogotá, por su infraestructura tecnológica, disponibilidad de servicios de internet y concentración significativa de familias que demandan soluciones de cuidado infantil, ofreció condiciones ideales para recopilar insumos relevantes y orientar de manera acertada el diseño de la plataforma.

El análisis de necesidades se dirigió a la identificación de requerimientos funcionales y no funcionales desde la perspectiva de los usuarios, lo que permitió caracterizar con claridad los aspectos esenciales para el diseño del sistema. Este proceso brindó una base sólida para orientar las decisiones arquitectónicas, las cuales posteriormente fueron evaluadas mediante pruebas controladas y escenarios estructurados en el prototipo funcional. La validación se realizó en un entorno controlado que permitió comprobar el comportamiento de las funcionalidades implementadas y contrastar la arquitectura frente a atributos como rendimiento, seguridad, mantenibilidad y fiabilidad, asegurando coherencia entre las necesidades identificadas y los resultados observados.

En cuanto al diseño arquitectónico, el alcance se centró en la construcción de una propuesta modular, escalable, segura y con capacidad de integración, modelada bajo el enfoque C4 hasta el nivel de componentes. Se implementaron los módulos necesarios para evaluar la viabilidad técnica de la arquitectura propuesta y demostrar su coherencia con los objetivos del proyecto.

Finalmente, el prototipo funcional (MVP) permitió demostrar la factibilidad técnica de la arquitectura planteada, validando los drivers más relevantes para esta fase según el proceso de priorización, en especial aquellos asociados a seguridad y estructura modular. Los demás drivers, como escalabilidad e integrabilidad, fueron incorporados en el diseño arquitectónico, sentando las bases para futuras iteraciones. Este MVP se consolidó como una prueba de concepto que verifica los elementos esenciales de la arquitectura y establece un fundamento sólido para la evolución del sistema.

1.4. Justificación del trabajo de grado

El cuidado infantil constituye un desafío estructural para las familias, ya que implica no solo atender las necesidades básicas de los niños, sino también garantizar una estimulación adecuada en cada etapa de la infancia con el fin de promover un desarrollo óptimo. Diversos estudios han señalado que las experiencias en los primeros años de vida impactan de manera decisiva en el desempeño escolar, la integración social y el bienestar futuro de los individuos (UNESCO, 2022; UNICEF, sf). Sin embargo, en Colombia la oferta de servicios de cuidado infantil sigue siendo insuficiente y heterogénea. Los programas gubernamentales, aunque han ampliado cobertura, no logran responder a la creciente demanda, mientras que los servicios privados suelen estar asociados a altos costos, lo que limita el acceso de gran parte de la población (ICBF, 2021; CEPAL, 2016).

Frente a este escenario, el presente proyecto se justificó en la necesidad de diseñar una solución tecnológica que facilitara el acceso de las familias a servicios de cuidado infantil seguros, escalables y adaptables a sus necesidades. La propuesta se centró en el diseño de una arquitectura de software que permitiera el desarrollo de aplicaciones capaces de conectar a las familias con profesionales capacitados, contribuyendo así a cerrar la brecha entre la demanda y la limitada oferta de servicios especializados en el país.

Los impactos de esta investigación se manifiestan en varios planos. En el ámbito social, ofrece a las familias la posibilidad de acceder a servicios de cuidado más personalizados y confiables, lo que favorece el bienestar infantil y reduce la sobrecarga de cuidado, especialmente en las mujeres, quienes históricamente han asumido la mayor parte de esta responsabilidad (ONUMujeres, 2024). En el plano económico, la solución contribuye a que los cuidadores primarios puedan mantenerse activos en el mercado laboral sin sacrificar el bienestar de sus hijos, fortaleciendo así la productividad y la estabilidad de los hogares. Desde una perspectiva tecnológica, la investigación aporta al campo de la ingeniería de software aplicada al sector social, mostrando cómo los atributos de calidad arquitectónicos de integrabilidad, escalabilidad y seguridad, pueden materializarse en un prototipo funcional que sirva como base para futuras aplicaciones en este y otros contextos.

La solución desarrollada representa un avance significativo en la construcción de alternativas tecnológicas para ampliar el acceso a servicios de cuidado infantil de calidad. El prototipo permitió validar que es posible materializar una plataforma capaz de conectar familias con cuidadores confiables, fortalecer la seguridad en los procesos de intermediación y apoyar dinámicas orientadas al bienestar infantil. Su diseño y evaluación demuestran el potencial de la ingeniería de software para responder de forma efectiva a desafíos sociales reales, ofreciendo bases concretas para futuras implementaciones y evoluciones del sistema. En esta medida, el trabajo de grado se justifica plenamente al aportar una propuesta funcional, escalable y alineada con las necesidades actuales de las familias y del ecosistema de cuidado infantil.

1.5. Metodología de la investigación

El proyecto se enmarca dentro de un enfoque de investigación aplicada con carácter exploratorio. La naturaleza del problema y el objetivo de construir una arquitectura robusta para un sistema complejo justificaron la adopción de un modelo iterativo que permitiera gestionar la incertidumbre, validar decisiones tempranas y ajustar el rumbo de forma progresiva. En este contexto, se eligió el Modelo en Espiral, en su variante Incremental Commitment Spiral Model (ICSM), dado que permite integrar la construcción gradual de soluciones con un control riguroso de riesgos técnicos y de negocio (Boehm et al., 2014).

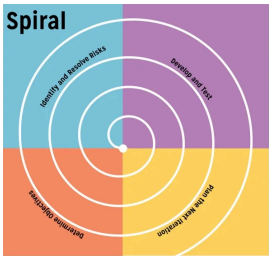


Figura 1.1: Modelo Espiral. Fuente: (Medium, 2019)

Como se observa en la Figura 1.1, este modelo organiza el proceso en ciclos sucesivos que integran formulación de objetivos, evaluación de alternativas, análisis de riesgos, desarrollo, validación y planificación del siguiente ciclo. En este proyecto, se aplicaron cuatro iteraciones, cada una con el propósito de refinar y mejorar la versión anterior, incorporando ajustes derivados de la validación técnica, retroalimentación del entorno y análisis de riesgos. Esta dinámica permitió evolucionar gradualmente desde una propuesta conceptual hasta una solución técnica validada, manteniendo la coherencia entre los objetivos del sistema y las decisiones de diseño tomadas en cada fase.

- **Objetivos y determinación alternativas:** Se inició con un proceso de levantamiento de información en campo, orientado a comprender las necesidades y expectativas de una muestra representativa de familias de Bogotá con relación a los servicios de cuidado infantil. Para ello, se diseñó un formulario de recolección de requisitos que permitió sistematizar tanto aspectos funcionales como no funcionales del sistema esperado. Esta técnica cualitativa facilitó la identificación de patrones comunes, así como casos de uso específicos que sirvieron como base para el diseño arquitectónico. Con estos insumos, se plantearon varias alternativas de solución tecnológica, valorando criterios como flexibilidad, interoperabilidad y sostenibilidad. Link Formulario

- **Análisis y evaluación de riesgos:** Con base en la información recolectada, se evaluaron las diferentes alternativas de diseño arquitectónico. Se analizaron riesgos relacionados con interoperabilidad, seguridad de la información, complejidad de implementación y sostenibilidad técnica. Esta etapa incluyó el análisis de decisiones arquitectónicas críticas (trade-offs), utilizando técnicas de documentación como los Architectural Decision Records (ADR), y se apoyó en principios de ingeniería de requisitos (Wiegers and Beatty, 2013). Además, se identificaron, priorizaron y se propusieron alternativas de mitigación para cada riesgo, utilizando una plantilla estandarizada y una matriz de riesgos basada en probabilidad e impacto. Este enfoque se alinea con las buenas prácticas de gestión de riesgos descritas por Sommerville (2016) en el capítulo 22 de su libro, donde se detallan los pasos para identificar, evaluar y planificar la respuesta a riesgos técnicos dentro del ciclo de vida del software. La matriz permitió visualizar de forma clara la severidad de cada riesgo y orientar las decisiones de diseño hacia soluciones que maximizaban la calidad sin comprometer la viabilidad técnica del sistema.

- **Desarrollo y prueba:** Durante la fase de desarrollo y prueba del ciclo metodológico, se abordó la construcción de un MVP funcional con el fin de poner a prueba la arquitectura planteada. Previo a esta implementación, se había elaborado un prototipo tipo mockup que facilitó la discusión inicial de componentes clave y permitió anticipar la estructura general del sistema. A partir de ese insumo, se definieron los principales bloques funcionales, lo que dio paso al diseño formal de la arquitectura.

  El diseño de la arquitectura se estructuró mediante el modelo C4, abarcando los niveles de contexto, contenedores y componentes. Esta representación sirvió como base para organizar la solución de forma modular y anticipar las responsabilidades de cada parte del sistema. Con

esta definición completa, se inició el montaje del backend sobre servicios administrados de AWS: se implementaron funciones Lambda para la lógica de negocio, Cognito para autenticación, DynamoDB como almacenamiento principal, API Gateway para la gestión de endpoints y S3 tanto para almacenamiento como para el despliegue del frontend.

Una vez configurada la infraestructura, se desarrolló una aplicación web responsive que permitió validar aspectos como la integración entre servicios, el manejo de sesiones, el rendimiento bajo carga básica y la coherencia entre la arquitectura prevista y la implementación real. Este MVP buscaba validar las decisiones arquitectónicas tomadas y preparar el camino para ajustes en los siguientes ciclos del modelo.

- **Planificación del siguiente ciclo:** Finalizada la validación técnica del MVP, se llevó a cabo un ejercicio de revisión integral para identificar aprendizajes, puntos de mejora y ajustes necesarios en la arquitectura. Esta evaluación no solo consideró los resultados obtenidos en pruebas funcionales y no funcionales, sino también la experiencia de implementación, las decisiones arquitectónicas tomadas y los riesgos emergentes durante el despliegue.

  Con base en estos hallazgos, se documentaron los cambios requeridos para fortalecer la modularidad, adecuar el uso de servicios cloud y mejorar la experiencia de usuario. Las funcionalidades definidas desde el inicio del proyecto se fueron puliendo e implementando de manera progresiva a lo largo de las iteraciones, ajustando aspectos técnicos, de interfaz y de integración según los resultados de validación obtenidos en cada ciclo.

  Este proceso de planificación y ajuste se repitió durante cuatro ciclos completos, en coherencia con el enfoque iterativo del ICSM, permitiendo evolucionar gradualmente desde un diseño inicial hasta un producto funcional validado. En cada ciclo, se actualizaron los artefactos arquitectónicos (como el modelo C4 y el ADR) para reflejar el estado actualizado de la solución y mantener la trazabilidad de las decisiones. Finalmente, se definió un plan de acción para cada nueva iteración, priorizando actividades de refinamiento técnico, validaciones con usuarios y mejoras incrementales orientadas a robustecer la solución final.

1.6. Resultados obtenidos

La aplicación del proceso metodológico iterativo, basado en el modelo ICSM, permitió un refinamiento progresivo de la solución y facilitó la consolidación de una arquitectura de software modular, escalable y segura, adecuada para el desarrollo de una plataforma digital orientada a la gestión de servicios de cuidado infantil. Esta arquitectura fue validada mediante la construcción de un MVP funcional desplegado sobre infraestructura cloud, lo que permitió verificar su viabilidad técnica y su alineación con los atributos de calidad definidos desde el inicio del proyecto. Entre los principales resultados alcanzados se destacan:

- La definición formal de la arquitectura utilizando el modelo C4 en sus tres primeros niveles

(contexto, contenedores y componentes), lo que permitió una representación clara, comprensible y trazable de la solución.

- La implementación y configuración de servicios AWS para el backend, incluyendo Lambda, API Gateway, DynamoDB, S3, y Cognito, lo cual permitió validar la capacidad de escalar, desacoplar responsabilidades y garantizar seguridad en el acceso a los recursos.
- El desarrollo y despliegue de una aplicación web responsive como canal de interacción principal, alojada en S3 con configuración para hosting estático complementada con Amazon CloudFront como red de distribución de contenido.
- La aplicación de pruebas funcionales para los flujos críticos del sistema, como autenticación, registro de usuarios, agendamiento de servicios y visualización de perfiles.
- La actualización progresiva de los artefactos arquitectónicos (modelo C4, ADR y matriz de riesgos), garantizando la coherencia entre el diseño propuesto, las decisiones tomadas y la solución desarrollada.

Estos resultados permitieron demostrar que la arquitectura diseñada no solo es viable técnicamente, sino que constituye una base sólida para el crecimiento funcional futuro del sistema y su eventual puesta en producción.

Capítulo 2

Marco de referencia

A continuación, se mencionan diferentes fundamentos teóricos, técnicos y contextuales que respaldan la construcción de la plataforma de cuidado infantil propuesta. Este apartado se enmarca en el contexto de las tecnologías modernas y los servicios en la nube, los cuales posibilitan la creación de sistemas escalables, seguros y de fácil acceso. Se abordan conceptos claves relacionados con la arquitectura de software, las herramientas y consideraciones relevantes empleadas en el desarrollo de aplicaciones. Asimismo, se analizan diferentes soluciones existentes, sus componentes funcionales y limitaciones, con el fin de establecer las bases que orientan el diseño y la implementación de la propuesta.

2.1. Marco Teórico

2.1.1. Bases Teóricas

- **Cuidado infantil en Colombia:** En Colombia, a lo largo de las últimas tres décadas, se ha venido fortaleciendo la Política Nacional de Infancia y Adolescencia, brindando al país y a las distintas entidades gubernamentales elementos que contribuyen al desarrollo integral de los niños, niñas y adolescentes. Este marco normativo busca garantizar que cada menor cuente con oportunidades que les permitan potenciar sus capacidades y hacer efectivos sus derechos.

  Para ello, se han venido implementando diversos mecanismos, como las leyes 1098 de 2006, 1804 de 2006 y 1753 de 2015, en las que se establece la protección integral de la niñez y promueven el ejercicio efectivo de sus derechos y libertades. Asimismo, estas normativas fortalecen y consolidan condiciones y capacidades en las familias, la sociedad y las instituciones, facilitando la construcción de trayectorias de vida dignas para los niños, niñas y adolescentes (ICBF, 2018).

- **Tipos de servicios de cuidado infantil existentes en Colombia:** En Colombia, el cuidado infantil está organizado en diversas modalidades, por medio de las cuales se busca garantizar el desarrollo integral de los niños, en cada una de las etapas de desarrollo infantil (ICBF, 2021). A continuación, se detallan las modalidades existentes y los principales tipos de servicios de cada una:

- Modalidad Institucional:
  - Centros de Desarrollo Infantil
  - Hogares Infantiles
  - Hogares Empresariales
  - Hogares Múltiples
  - Jardines sociales
  - Preescolar Integral
  - Desarrollo Infantil en Establecimientos de Reclusión
- Modalidad Propia e intercultural
- Modalidad Familiar
  - Desarrollo Infantil en Medio Familiar (DIMF)
  - Hogares Comunitarios de Bienestar (FAMI)
- Modalidad Comunitaria
  - HCB Familiar
  - HCB Agrupados
  - Unidades Básicas de Atención (UBA)
  - HCB Cualificados o Integrales

- **Contexto de las Plataformas digitales de prestación de servicios y de cuidado:** A raíz de la llamada "crisis social de los cuidados", un fenómeno global que surge debido a diversos factores, como el aumento de empleos bien remunerados para mujeres, la limitada participación de los hombres en las tareas domésticas y de cuidado, y la insuficiencia de recursos públicos para satisfacer todas las necesidades de cuidado, se ha generado un impacto significativo en la digitalización de los hábitos de consumo y de la economía.

  Esta acelerada transformación ha dado lugar al surgimiento de plataformas digitales de trabajo, que actúan como nuevas intermediarias entre las familias y los empleados domésticos, cuidadores y niñeros. Como resultado, a nivel global, "las plataformas digitales de empleo doméstico y de servicios de cuidados se han multiplicado por ocho en tan solo 10 años" (Blanchard, 2024).

- **Levantamiento de requisitos:** Los requisitos corresponden a la especificación de todo lo que se desea implementar en un proyecto y, además, detallan el comportamiento esperado del sistema o de sus componentes específicos. Existen diferentes tipos de requisitos como empresariales, de negocio, funcionales, no funcionales, de sistema y de usuario y también algunos aspectos importantes que ayudan a determinar de manera más precisa su objetivo, algunos de ellos son: reglas de negocio, restricciones, atributos de calidad y/o características esenciales.

  Existen dos aspectos clave asociados a los requisitos. El primero de ellos es el desarrollo del requisito, cuyo objetivo es lograr una comprensión suficiente para construir un producto con

un nivel de riesgo aceptable. Se divide en cuatro disciplinas: elicitación, análisis, especificación y validación.

El segundo aspecto es la gestión de requisitos, que tiene como objetivo anticiparse y adaptarse a los posibles cambios que puedan surgir durante el proceso de desarrollo, con el fin de minimizar su impacto en el proyecto (Karl Wiegers, 2013).

- **Estilos de arquitectura:** Los estilos de arquitectura de software son un conjunto de características que definen la estructura general, organización e interacción de los componentes dentro de un sistema. Existen diversos estilos, y cada uno de ellos establece reglas y patrones específicos que sirven como guía para organizar y estructurar el software. Un estilo arquitectónico describe aspectos como la topología y organización de los componentes, la arquitectura física, el despliegue, el estilo de comunicación y la topología de datos, con el fin de ayudar a mejorar la calidad, flexibilidad y escalabilidad del sistema (Mark Richards, 2025).

  Entre los estilos de arquitectura más comunes se destacan:

  - **En capas:** Este estilo organiza el sistema en capas horizontales con responsabilidades específicas, no determina un número fijo de capas, aunque típicamente se manejan cuatro.
  - **Monolítico:** Como su nombre indica, este estilo de arquitectura se basa en un sistema monolítico donde todos los componentes se organizan bajo una sola unidad. Favorece la implementación; pero es un estilo que dificulta la escalabilidad de la aplicación.
  - **Microservicios:** Es un estilo de arquitectura muy reciente que busca desacoplar los servicios que hacen parte de una solución de software, con el fin de permitir soluciones sencillas, fáciles de escalar y más rápidas de desarrollar.
  - **Basado en eventos:** hace referencia a un estilo arquitectónico distribuido y de particularidad asíncrono, suele ser usado para crear aplicaciones altamente escalables y de alto rendimiento. Su base de comportamiento se da por componentes de procesamiento que accionan ante un evento específico.

- **Escenarios de calidad:** Constituyen una herramienta fundamental para describir y evaluar los atributos de calidad de un sistema. Su enfoque es transformar requerimientos abstractos, como rendimiento, seguridad o disponibilidad, en situaciones concretas, observables y medibles que facilitan su análisis y validación durante el diseño arquitectónico. De acuerdo con Bass et al. (2021), un escenario de atributo de calidad especifica cómo debe responder el sistema ante un estímulo particular bajo ciertas condiciones. La aplicación de esta estrategia dentro del diseño de arquitectura contribuye para garantizar que las decisiones arquitectónicas se alineen con las expectativas del sistema (Len Bass, 2021).

- **Atributos de calidad:** Un atributo de calidad es una característica medible y verificable de un sistema. Estas medidas permiten garantizar y evaluar que las funcionalidades y utilidades del software cumplan con los requisitos y expectativas establecidas.Los atributos de calidad

pueden abordarse desde diversas perspectivas, asegurando aspectos que impactan directamente al usuario, como la usabilidad, disponibilidad y funcionalidad (Len Bass, 2021). Además, existen otros enfoques orientados a mantener la calidad, seguridad y desempeño del software, considerando aspectos como el rendimiento, la escalabilidad, la mantenibilidad, entre otros, tal y como se muestra en la Figura 2.1.

Figura 2.1: Atributos de calidad. Fuente: (ISO25000, sf).

- **Mecanismo de validación de una arquitectura:** "La evaluación de la arquitectura es el proceso que determina el grado de idoneidad de una arquitectura para el propósito previsto"(Len Bass, 2021).

  La arquitectura en sí es portadora de las decisiones más tempranas e importantes asociadas al sistema; pues dentro de su definición aborda aspectos como atributos de calidad, restricciones de implementación, estructura y organización de componentes. Dada su contribución al proyecto, es importante asegurarse de que sea robusta, eficaz y alineada con los objetivos del sistema, para ello existen diversos mecanismos de validación como:

  - **Frameworks de validación:** Métodos estructurados que permiten analizar atributos claves de la arquitectura (Len Bass, 2012).
  - **Revisión por expertos:** Involucra personas altamente capacitadas en temas de arquitectura para detectar inconsistencias, iterar y aportar a la mejora de calidad del diseño (Varma, 2009a).
  - **Pruebas de viabilidad:** Se basa en prototipos simples que permitan evaluar el posible comportamiento en un entorno real (Richard N. Taylor, 2009).
  - **AWS Well-Architected Tool:** Permite evaluar si una arquitectura y sus componentes siguen buenas prácticas de diseño en AWS (Amazon Web Services, 2025).
  - **AWS Trusted Advisor:** Auditor automático que actúa como revisor en aspectos tales como seguridad, costos, tolerancia a fallos y rendimiento; detectando posibles acciones de mejora y proponiendo medidas correctivas (Amazon Web Services, 2024).

- **Modelo C4:** El modelo C4 nace como una propuesta para homogenizar los diagramas de estructuras de software y sus componentes con el propósito de poder transmitir mediante una imagen, una visión desde diferentes enfoques de una propuesta de software (Model, sf). Dentro de sus principales características se destacan:

  - Jerarquía de diagramas ofreciendo una visión desde diferentes niveles de abstracción.
  - Facilita la comunicación clara y efectiva, adaptándose a distintas audiencias.
  - Evita la ambigüedad en los diagramas usando un diseño estandarizado en textos descriptivos y notaciones.

  Este modelo se divide en cuatro niveles, los cuales detallan a continuación:

  - **Contexto del sistema:** Proporciona una visión general de cómo el sistema se integra en su entorno y cómo interactúa con otros sistemas o actores. A continuación se detalla un ejemplo de este nivel en la Figura 2.2.

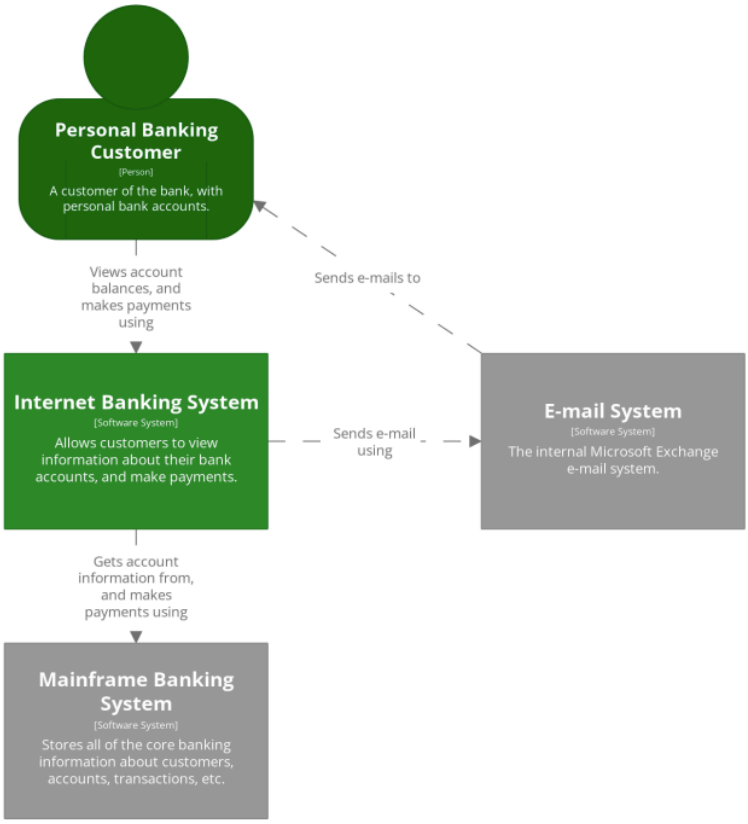


Figura 2.2: Contexto del Sistema. Fuente: (Model, sf).

- **Contenedores:** Amplía la visión del sistema, representando sus principales componentes, como el modelo de datos y las aplicaciones asociadas dentro de su alcance, como se puede evidenciar en la Figura 2.3.

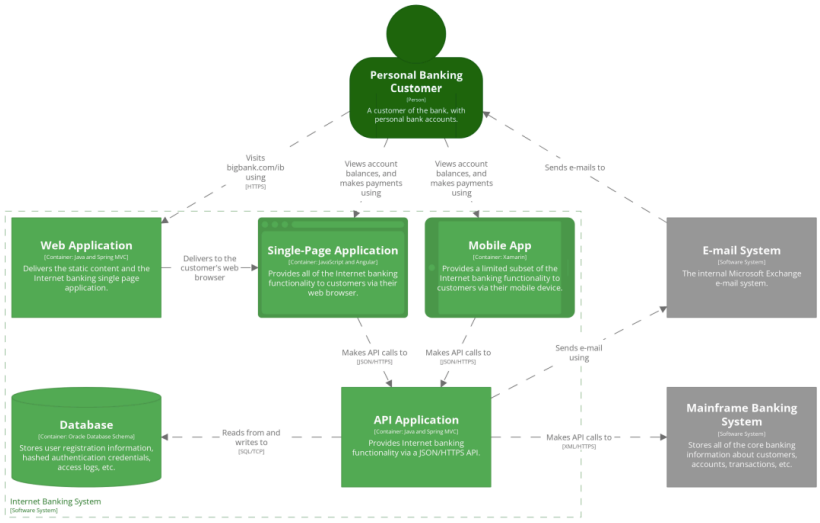


Figura 2.3: Diagrama contenedores. Fuente: (Model, sf).

- **Componentes:** Ofrece una visión más detallada de un contenedor específico, describiendo los componentes individuales que lo conforman. A modo de ejemplo de este nivel, se incluye la figura 2.4.

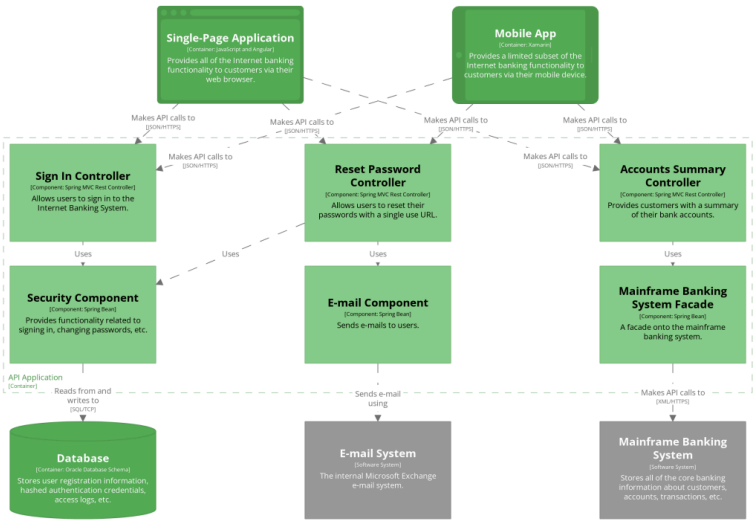


Figura 2.4: Diagrama componentes. Fuente: (Model, sf).

- **Código:** Focaliza en un componente particular, detallando su implementación a nivel de código. A continuación, se ilustra este nivel mediante un ejemplo en la Figura 2.5.

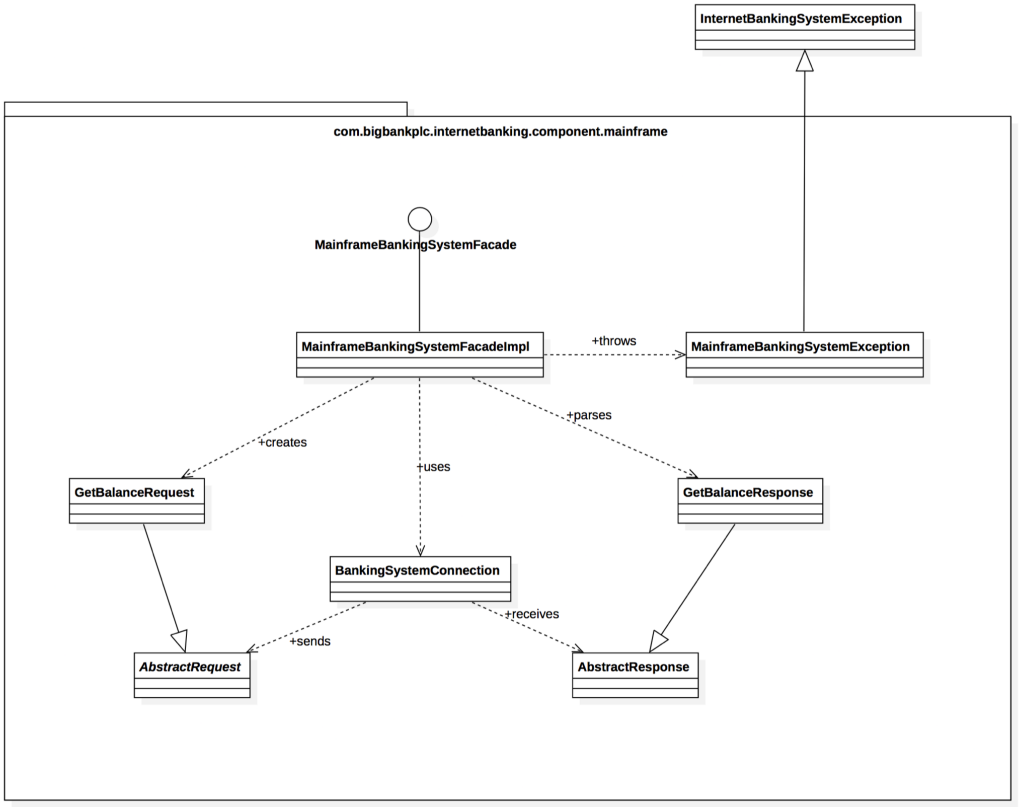


Figura 2.5: Diagrama código. Fuente: (Model, sf).

- **Computación en la nube:** Se ha consolidado como uno de los pilares fundamentales para el desarrollo de soluciones tecnológicas escalables y eficientes. Este modelo permite el acceso bajo demanda a recursos informáticos configurables que pueden ser aprovisionados y liberados con un esfuerzo mínimo de gestión o interacción con el proveedor de servicios (Mell and Grance, 2011).

  En la actualidad existen diversos proveedores de servicios en la nube, entre ellos se destacan Amazon Web Services (AWS), Microsoft Azure y Google Cloud Platform (GCP), todo estos ofrecen infraestructuras globales y herramientas que facilitan la creación de aplicaciones. Estos proveedores operan bajo diferentes modelos de servicio —Infraestructura como Servicio (IaaS), Plataforma como Servicio (PaaS) y Software como Servicio (SaaS)— que definen el nivel de control y responsabilidad del usuario sobre los recursos desplegados (Varma, 2009b).

Para la ejecución de esta tesis se tomó la decisión de usar los servicios que presta Amazon Web Services - AWS.

- **Amazon Web Services (AWS):** Actualmente es el proveedor líder en el mercado de computación en la nube, reconocido por su enfoque en la escalabilidad, seguridad y confiabilidad de sus servicios. Ofrece más de 200 servicios en áreas como cómputo, almacenamiento, bases de datos, redes, inteligencia artificial y desarrollo sin servidor (Amazon Web Services, 2024).

  Amazon Web Services (AWS) es uno de los principales proveedores de servicios en la nube, ofreciendo soluciones escalables, seguras y de alta disponibilidad. Entre sus herramientas destaca AWS Lambda, un servicio serverless que permite ejecutar funciones sin administrar servidores, optimizando costos y facilitando la integración con servicios como API Gateway, S3, DynamoDB y Cognito entre otros (Amazon Web Services, 2024). Algunos de los beneficios que se identificaron y fueron parte importante para su adopción para esta tesis son:

  - **Ejecución bajo demanda:** pago solo por uso.
  - **Escalabilidad automática:** adaptación al volumen de solicitudes.
  - **Alta disponibilidad:** infraestructura distribuida globalmente.
  - **Reducción de mantenimiento:** sin gestión de servidores.
  - **Integración nativa:** conexión directa con otros servicios AWS.

2.2. Estado del Arte

Durante el proceso de investigación para la elaboración de la tesis se realizó una revisión de las principales tendencias, plataformas y tecnologías asociadas al desarrollo de plataformas para el cuidado infantil. De acuerdo a la información recolectada y analizada, esta sección se divide en tres apartados: contexto tecnológico del sector, enfoques arquitectónicos empleados en las soluciones actuales y los aspectos funcionales y sociales que influyen en la adopción de este tipo de plataformas.

2.2.1. Contexto tecnológico de las plataformas de cuidado infantil

En los últimos años, el avance de la economía digital ha impulsado el surgimiento de múltiples plataformas que buscan conectar cuidadores y familias mediante soluciones tecnológicas accesibles y seguras. En Colombia, destacan iniciativas como *Pandas Cuidado Infantil*, *Kanguritos*, *Funnana*, *Ada Niñeras*, *Sitly* y *Nannies Colombia*, cada una con diferentes enfoques y niveles de madurez tecnológica (Sitly, 2025; Funnana, 2025; Niñeras, 2025; Colombia, 2025).

Pandas Cuidado Infantil y *Kanguritos* representan modelos de intermediación digital que priorizan la experiencia del usuario mediante plataformas web. En contraste, *Funnana* y *Ada Niñeras* operan como agencias híbridas, combinando procesos manuales con herramientas tecnológicas para la verificación de perfiles y asignación de servicios. Por su parte, *Sitly* adopta un modelo tipo *marketplace*, permitiendo a cuidadores y padres interactuar directamente a través de perfiles públicos y reseñas.

Si bien estas plataformas responden a la creciente demanda de servicios de cuidado confiables, aún se evidencian desafíos en materia de escalabilidad, trazabilidad de información, autenticación avanzada y automatización de procesos, lo cual abre oportunidades de mejora para propuestas basadas en arquitecturas más flexibles y escalables.

2.2.2. Tendencias arquitectónicas y tecnológicas

El desarrollo de soluciones digitales modernas ha migrado progresivamente hacia arquitecturas distribuidas, orientadas a eventos y altamente escalables. En este contexto, los modelos *serverless* y basados en microservicios se consolidan como una tendencia clave para el diseño de sistemas resilientes y sostenibles (Bass et al., 2021).

Proveedores de servicios en la nube como *Amazon Web Services (AWS)*, *Google Cloud Platform (GCP)* y *Microsoft Azure* han impulsado esta transición mediante la oferta de herramientas administradas que reducen la complejidad operativa. Entre ellas, se destacan las funciones bajo demanda (*AWS Lambda*), las pasarelas de APIs (*API Gateway*), los sistemas de autenticación federada (*Cognito*) y las bases de datos escalables (*DynamoDB*) (Mell and Grance, 2011; Amazon Web Services, 2024).

El enfoque *serverless*, en particular, representa una evolución significativa respecto a la infraestructura tradicional, ya que elimina la necesidad de mantener servidores dedicados, optimiza los costos mediante el pago por ejecución y facilita la integración entre componentes distribuidos. Este modelo se alinea con las necesidades de plataformas de cuidado infantil, que requieren disponibilidad constante, adaptabilidad y seguridad en el manejo de datos personales.

2.2.3. Dimensión funcional y social del cuidado infantil digital

Además de la infraestructura tecnológica, las plataformas de cuidado infantil deben considerar aspectos éticos, sociales y de usabilidad. La confianza es un factor central: las familias buscan servicios que garanticen la protección de los datos, la verificación de antecedentes y la trazabilidad de las interacciones. Iniciativas como *Sitly* y *Funnana* han incorporado mecanismos de reputación y validación, aunque la mayoría de los servicios todavía dependen de procesos manuales o poco

automatizados (Sitly, 2025; Funnana, 2025).

En este contexto, la adopción de arquitecturas basadas en la nube no solo responde a criterios de eficiencia técnica, sino también a la construcción de entornos digitales confiables. El uso de servicios administrados como *Cognito* para la autenticación segura, y de bases de datos cifradas para la gestión de información sensible, contribuye a fortalecer la privacidad y la responsabilidad social de las plataformas tecnológicas.

Finalmente, la propuesta de tesis se enmarca dentro de este panorama como una evolución hacia una plataforma de cuidado infantil que integre escalabilidad, sostenibilidad y confianza, aprovechando la infraestructura *serverless* de AWS para ofrecer una solución innovadora y adaptable a las necesidades del contexto colombiano.

Capítulo 3

Desarrollo del Proyecto

Este capítulo presenta el desarrollo de la solución propuesta, estructurada bajo una metodología iterativa y alineada con los objetivos del proyecto. El proceso abarcó desde el levantamiento de necesidades hasta la validación final del prototipo, documentando las decisiones tomadas en el diseño arquitectónico, la definición de requisitos, la implementación del prototipo funcional y los mecanismos utilizados para evaluar su efectividad técnica y funcional.

3.1. Análisis de necesidades y expectativas

3.1.1. Objetivo del análisis

El propósito de este análisis fue identificar las necesidades, preferencias y preocupaciones de los usuarios potenciales frente al cuidado infantil, con el fin de que sirvieran como insumo para el diseño funcional y arquitectónico de la solución. En particular, esta caracterización buscó fundamentar la construcción de una arquitectura de software modular, escalable y segura, capaz de soportar una solución de software que facilite el acceso de las familias a profesionales capacitados en atención a la primera infancia. Al comprender el contexto de uso y las expectativas reales de los usuarios, se favoreció la toma de decisiones arquitectónicas alineadas con las condiciones y necesidades del entorno objetivo.

3.1.2. Metodología aplicada

Para la identificación de las necesidades y expectativas de los usuarios prospecto del sistema se empleó una metodología de tipo cuantitativo, basada en la aplicación de una encuesta estructurada como instrumento de recolección de datos. La encuesta fue diseñada y aplicada mediante la plataforma Google Forms, lo que permitió una distribución eficiente, recolección automática y acceso remoto a una muestra diversa de participantes.

La elección de la encuesta como técnica principal responde a su capacidad para recolectar

datos de forma sistemática, estandarizada y escalable, lo cual resulta especialmente útil cuando se busca describir patrones de comportamiento, percepciones o necesidades dentro de una población determinada (Sampieri et al., 2014). Además, al tratarse de un estudio exploratorio orientado al diseño de una solución tecnológica, el enfoque cuantitativo permitió identificar tendencias generales que sirvieron como base para la toma de decisiones arquitectónicas fundamentadas.

El cuestionario incluyó preguntas cerradas de opción múltiple y escala Likert (Likert, 1932), estructuradas en torno a cinco dimensiones clave: (i) Información demográfica, (ii) Situación actual de cuidado, (iii) Factores de confianza y preferencia, (iv) Seguridad y preocupaciones, y (v) Disposición al pago. Esta organización buscó capturar información relevante para el diseño funcional del sistema, así como para identificar requisitos no funcionales asociados con la experiencia de usuario, la seguridad y la confianza, y finalmente tener una visión general de la intención de las personas de pagar por servicios como el propuesto.

La encuesta fue aplicada a una muestra no probabilística intencional compuesta por 36 personas residentes en la ciudad de Bogotá, en su mayoría madres o padres de niños entre 0 y 5 años. Esta población fue seleccionada por su relación directa con el objetivo del análisis y conforme a la delimitación geográfica establecida en el alcance del proyecto, siguiendo un criterio de pertinencia más que de representatividad estadística (Sampieri et al., 2014).

Como soporte teórico que permite evidenciar la pertinencia y el nivel de precisión asociado al tamaño de muestra seleccionado, se procedió a calcular el margen de error estadístico estimado con base en la fórmula clásica para la estimación de proporciones en muestras finitas:

$$n = \frac{Z^2 \cdot p \cdot (1 - p)}{E^2}$$

Donde:

- $n = 36$ es el tamaño muestral,
- $Z = 1{,}44$ corresponde al valor crítico asociado a un nivel de confianza del **85 %**, seleccionado en concordancia con el carácter exploratorio del estudio, balanceando entre precisión y factibilidad operacional,
- $p = 0{,}5$ representa la proporción poblacional asumida para el diseño, la cual maximiza la varianza muestral y, por ende, el tamaño de muestra requerido. Este valor se adopta en ausencia de una estimación previa robusta sobre la proporción esperada, siguiendo recomendaciones estándar en estudios de sondeo inicial (Salant and Dillman, 1994).
- E es el error máximo permitido o margen de error deseado.

Despejando E, se obtiene:

$$E = \sqrt{\frac{Z^2 \cdot p \cdot (1-p)}{n}} = \sqrt{\frac{1{,}44^2 \cdot 0{,}5 \cdot 0{,}5}{36}} \approx 0{,}12$$

Esto implica un **margen de error estadístico estimado de aproximadamente ±12 puntos porcentuales**. Dado que la población objetivo estimada (hogares con niños entre 0 y 5 años en Bogotá) es muy extensa —del orden de decenas o incluso cientos de miles, muy superior al tamaño muestral ($n = 36$)— la fracción muestreada resulta mínima. En estos casos, el *factor de corrección por población finita* (FPC) tiende a 1, lo que implica que su impacto sobre la varianza y el error estándar es estadísticamente insignificante. Según Glen (sf), cuando la proporción muestreada representa menos del 5 % de la población, el FPC puede omitirse sin comprometer la validez del análisis. Por tanto, los resultados derivados de esta muestra ofrecen un nivel de certeza suficiente para respaldar decisiones arquitectónicas preliminares.

Ficha técnica de la encuesta:

- **Tipo de estudio:** Exploratorio, cuantitativo.
- **Instrumento:** Encuesta estructurada (Google Forms).
- **Diseño de muestreo:** No probabilístico intencional.
- **Tamaño de muestra:** 36 personas.
- **Población objetivo:** Padres/madres de niños entre 0 y 5 años en Bogotá.
- **Nivel de confianza:** 85 %.
- **Margen de error estimado:** ±12 %.
- **Fecha de recolección:** Octubre de 2025.

3.1.3. Análisis descriptivo de resultados

A partir de los datos recolectados, se realizó un análisis descriptivo que permitió observar tendencias generales en las respuestas obtenidas. Este análisis no solo ofrece una visión panorámica del contexto actual de cuidado infantil según la percepción de los usuarios, sino que también aporta evidencia empírica para fundamentar decisiones de diseño, tanto funcionales como arquitectónicas. A continuación, se presentan los resultados más relevantes mediante gráficas seleccionadas.

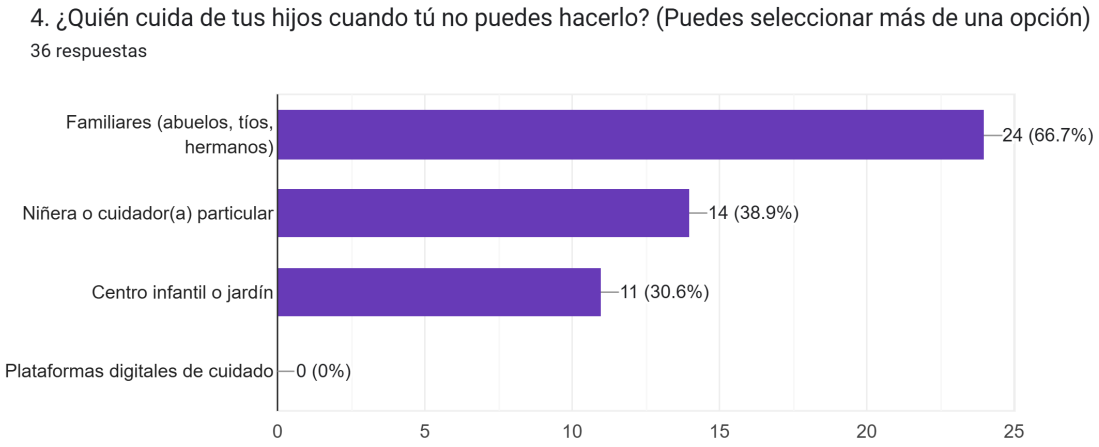


Figura 3.1: Responsables habituales del cuidado infantil. Fuente: elaboración propia a partir de encuesta (n=36).

La Figura 3.1 muestra que, ante la imposibilidad de cuidar personalmente a sus hijos, la mayoría de los encuestados recurre al apoyo de su entorno familiar cercano. Un 66,7 % indicó que deja esta responsabilidad en manos de abuelos, tíos o hermanos, lo que refleja una fuerte dependencia de redes de cuidado informales y de confianza. En segundo lugar se encuentran las niñeras o cuidadores particulares, con un 38,9 %, mientras que los centros infantiles o jardines son utilizados por el 30,6 % de los participantes.

Un aspecto particularmente llamativo es que ninguna persona manifestó utilizar actualmente plataformas digitales especializadas para este fin. Esta ausencia, considerada a la luz de los resultados posteriores de la encuesta, sugiere que más que una resistencia por parte de los usuarios, lo que existe es una falta de soluciones accesibles, confiables o suficientemente visibles en el contexto local. Es importante destacar, además, que una proporción significativa de los encuestados ya recurre al servicio de cuidadores particulares, lo cual evidencia que existe apertura hacia este tipo de apoyo, pero a través de canales informales o no digitalizados. Esto refuerza el propósito central de la solución propuesta: facilitar, a través de una arquitectura tecnológica robusta, la conexión segura entre familias y cuidadores calificados.

Desde una perspectiva arquitectónica, este hallazgo resalta la necesidad de diseñar un sistema que priorice atributos como la seguridad y la escalabilidad. Por un lado, se deben incorporar mecanismos que refuercen la percepción de confianza en la plataforma; por otro, es fundamental prever una infraestructura capaz de soportar un eventual crecimiento acelerado en la base de usuarios, especialmente considerando que, al no haber muchas soluciones consolidadas en el mercado, una

porción significativa de la población podría adoptar rápidamente una alternativa bien diseñada. Adicionalmente, la solución tecnológica no debe pretender reemplazar las dinámicas tradicionales de cuidado, sino integrarse de manera complementaria con las redes de apoyo ya establecidas.

Figura 3.2: Frecuencia con la que los padres requieren apoyo en el cuidado infantil. Fuente: elaboración propia a partir de encuesta (n=36).

La Figura 3.2 revela que una porción significativa de los encuestados requiere apoyo frecuente en el cuidado infantil. Un 41,7 % afirma necesitar ayuda a diario, y un 25 % adicional lo requiere varias veces por semana. Estas cifras indican que más de dos tercios de los participantes enfrentan una demanda recurrente de apoyo, lo cual resalta la carga constante que representa el cuidado de los niños en la vida cotidiana de muchas familias.

En contraste, solo un 13,9 % señala que requiere apoyo de forma muy esporádica, mientras que el 16,7 % lo necesita ocasionalmente y apenas un 2,7 % reporta una necesidad mensual. Esta distribución sugiere que el acompañamiento en el cuidado infantil no es un requerimiento puntual, sino una necesidad estable y presente con alta frecuencia.

Desde una perspectiva funcional y arquitectónica, este hallazgo respalda la importancia de que el sistema propuesto esté diseñado para responder de manera ágil y eficiente a solicitudes de cuidado frecuentes. Requiere, por tanto, mecanismos que garanticen alta disponibilidad, gestión dinámica de la oferta y demanda, y una experiencia de usuario optimizada para interacciones repetidas. Además, valida el potencial de uso sostenido de la solución, lo cual puede ser determinante en términos de adopción y escalabilidad.

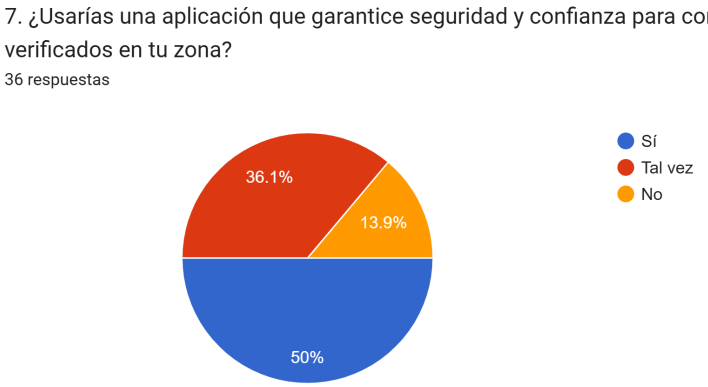


Figura 3.3: Disposición de los encuestados a utilizar una aplicación con cuidadores verificados. Fuente: elaboración propia a partir de encuesta (n=36).

La Figura 3.3 evidencia una disposición mayoritariamente favorable hacia el uso de una aplicación que conecte a padres con cuidadores verificados. Donde el 50 % de los encuestados manifestaron que la utilizarían, mientras que un 36,1 % respondió "tal vez", lo cual sugiere un interés condicionado por factores como la confianza, la percepción de seguridad, entre otros. Solo el 13,9 % indicó que no la usaría. Estos resultados respaldan la viabilidad de una solución tecnológica como la propuesta, siempre que su diseño arquitectónico priorice aspectos como la protección de los datos personales, la confidencialidad de la información sensible y una experiencia de usuario clara y confiable, capaces de generar en los usuarios una sensación de seguridad y confianza que incentive su adopción progresiva.

Figura 3.4: Disposición de pago por el servicios de cuidado infantil. Fuente: elaboración propia a partir de encuesta (n=36).

La Figura 3.4 revela una disposición significativa a pagar por servicios de cuidado infantil, con el 36,1 % de los encuestados dispuestos a pagar entre $30.000 y $40.000 COP por hora, y un 27,8 % indicando que pagarían menos de $30.000 COP. Aunque solo un 5,6 % consideraría tarifas superiores a los $40.000, es relevante que el 30,6 % manifestó no estar seguro aún sobre cuánto estaría dispuesto a invertir, lo que sugiere la necesidad de fortalecer el valor percibido del servicio. Este resultado ofrece un punto de partida útil para definir estrategias de monetización realistas, que contemplen tanto la disposición de pago de los usuarios como las tarifas estándar del mercado, garantizando condiciones adecuadas para los cuidadores. Asimismo, es fundamental que dichas estrategias consideren el contexto económico del público objetivo y promuevan una percepción de valor sustentada en atributos como la calidad, la seguridad y la confianza que pueda transmitir la solución tecnológica.

3.1.4. Conclusiones y perfil de usuario objetivo

A partir del análisis cuantitativo realizado, se identificaron patrones clave que permitieron orientar de manera informada las decisiones de diseño funcional y arquitectónico de la solución tecnológica. En primer lugar, se evidenció una alta dependencia de redes de apoyo familiares para el cuidado infantil, lo cual reforzó la necesidad de que la plataforma funcionara como una alternativa confiable que ayudara a aliviar la carga que muchas familias enfrentan actualmente. Además, la frecuencia con la que se requiere apoyo, en muchos casos diaria, implicó el diseño de una infraestructura tecnológica con alta disponibilidad, capacidad de respuesta oportuna y estabilidad operativa.

Aunque en el momento del estudio no se evidenció el uso de plataformas digitales especializadas para este fin, la mayoría de los encuestados manifestó una disposición favorable a adoptar una solución tecnológica, siempre que ofreciera garantías claras de confianza y protección de la información personal. A esto se sumó una disposición significativa a pagar por el servicio, lo que dejó abierta la posibilidad de estructurar un modelo de valor que beneficie a ambas partes: condiciones laborales dignas para los cuidadores y servicios de calidad para las familias. Estas conclusiones validaron la pertinencia del proyecto y constituyeron un insumo clave para definir los requisitos del sistema y orientar la priorización de atributos arquitectónicos fundamentales. A continuación, se presenta una caracterización sintética del perfil del usuario objetivo, construida con base en los hallazgos de la encuesta y que sirvió como insumo para la formulación de las conclusiones previamente expuestas.

Tabla 3.1: Perfil del usuario objetivo según resultados de la encuesta

Característica	**Descripción**
Tipo de usuario	Padres, madres o cuidadores de niños entre 0 y 5 años
Ubicación geográfica	Residentes en Bogotá, Colombia

Característica	Descripción
Necesidad de apoyo	Requieren apoyo frecuente (diario o varias veces por semana) en el cuidado infantil
Soluciones actuales	Dependen principalmente de familiares o cuidadores particulares
Uso de tecnología	No usan plataformas digitales actualmente, pero muestran disposición a adoptar una solución segura y confiable
Disposición de pago	Entre $30.000 y $40.000 COP por hora
Condicionantes clave	Seguridad de datos, confianza en los cuidadores, facilidad de uso, disponibilidad del servicio

3.2. Identificación de requisitos

En esta fase, la identificación y priorización de los requisitos no se limitó únicamente al análisis de los resultados obtenidos mediante la encuesta aplicada a familias en Bogotá. Se aplicó una estrategia de triangulación metodológica que integró múltiples fuentes de información con el fin de lograr una caracterización más robusta y coherente de las necesidades funcionales y no funcionales del sistema. En particular, se consideraron: (i) los hallazgos derivados del análisis de necesidades expresadas por las familias encuestadas, (ii) la revisión de plataformas existentes de cuidado infantil y aplicaciones digitales con funcionalidades comparables, (iii) la experiencia directa de los autores como padres de familia, lo cual permitió interpretar con mayor profundidad las expectativas reales de los usuarios en contextos cotidianos, y (iv) la definición explícita del alcance del proyecto, considerando las restricciones temporales y técnicas propias de un trabajo de grado. A partir de esta integración, se establecieron relaciones significativas entre las necesidades detectadas y las funcionalidades propuestas. Como resultado, se optó por clasificar los requisitos en dos grupos: aquellos esenciales para la construcción del MVP y aquellos proyectados para versiones futuras. Esta distinción respondió a criterios de viabilidad técnica, criticidad funcional y aporte al valor percibido por los usuarios finales.

3.2.1. Requisitos funcionales

Tabla 3.2: Requisitos funcionales del sistema

ID	Requerimiento funcional	Versión
RF-01	El sistema debe permitir el registro de cuidadores mediante un formulario que incluya datos personales y profesionales.	MVP
RF-02	El sistema debe permitir el registro de usuarios responsables del cuidado de niños (padres, madres o familiares) interesados en contratar servicios.	MVP

ID	Requerimiento funcional	Versión
RF-03	El sistema debe permitir a los usuarios agendar servicios de cuidado infantil especificando fecha y lugar.	MVP
RF-04	El sistema debe permitir al titular de la reserva cancelar un servicio de cuidado previamente agendado, antes de la fecha programada.	MVP
RF-05	El sistema debe permitir a los cuidadores aceptar o rechazar solicitudes de servicio agendadas.	Futura
RF-06	El sistema debe ofrecer un buscador de cuidadores disponibles, filtrable por ubicación, experiencia y horarios.	Futura
RF-07	El sistema debe mostrar a los cuidadores el historial de servicios prestados y las valoraciones recibidas.	MVP
RF-08	El sistema debe mostrar a los usuarios el historial de servicios solicitados, incluyendo fechas, cuidadores y estado.	MVP
RF-09	El sistema debe permitir a los usuarios valorar y dejar comentarios sobre el servicio recibido por parte del cuidador.	MVP
RF-10	El sistema debe enviar notificaciones automáticas ante confirmaciones, cancelaciones o recordatorios de servicios.	Futura
RF-11	El sistema debe permitir la comunicación entre usuarios y cuidadores a través de un canal interno seguro.	Futura

3.2.2. Requisitos no funcionales

Tabla 3.3: Requisitos no funcionales del sistema

ID	Requerimiento no funcional	Atributo de calidad
RNF-01	El sistema debe estar disponible para los usuarios al menos el 99.5 % del tiempo durante el año.	Disponibilidad
RNF-02	Los tiempos de respuesta de los servicios no deben exceder los 2 segundos en el 95 % de las solicitudes, bajo condiciones normales.	Rendimiento
RNF-03	El sistema debe asegurar el cifrado de los datos personales en tránsito y en reposo.	Seguridad
RNF-04	El acceso a funcionalidades sensibles debe requerir autenticación mediante correo electrónico y contraseña segura.	Seguridad
RNF-05	El sistema debe ser capaz de escalar horizontalmente para soportar incrementos en el número de usuarios.	Escalabilidad
RNF-06	El sistema debe ser accesible desde dispositivos móviles y desktop.	Usabilidad
RNF-07	La arquitectura del sistema debe permitir la incorporación de nuevos módulos funcionales sin afectar los existentes.	Modularidad

3.2.3. Drivers arquitectónicos

Durante el análisis de requisitos se identificó un conjunto de atributos de calidad que actuarían como drivers arquitectónicos, al influir directamente en las decisiones de diseño estructural del sistema. Estos atributos fueron definidos con base en las necesidades y expectativas expresadas por los usuarios, así como en los requisitos no funcionales priorizados. Su identificación temprana permitió establecer criterios claros para orientar el diseño arquitectónico hacia los aspectos más críticos para la viabilidad, confiabilidad y sostenibilidad de la solución.

Tabla 3.4: Drivers arquitectónicos

Atributo de calidad	**Justificación como driver arquitectónico**
Disponibilidad	La necesidad frecuente de apoyo (en muchos casos diaria) exige una plataforma disponible en horarios amplios o incluso en todo momento.
Seguridad	La solución debe garantizar confidencialidad, integridad y protección de los datos personales para generar confianza y cumplir con normativas.
Escalabilidad	Se espera un crecimiento progresivo en el número de usuarios, lo cual exige una arquitectura capaz de escalar horizontalmente sin degradación del servicio.
Usabilidad	La adopción del sistema depende en gran medida de su facilidad de uso, ya que está orientado a usuarios no técnicos y con distintos niveles de alfabetización digital, por tanto se espera una solución intuitiva con diseños claros y de fácil manejo y acceso.
Mantenibilidad	La arquitectura debe permitir ajustes, correcciones y evolución de funcionalidades sin necesidad de grandes reestructuraciones del sistema.
Modularidad	La posibilidad de incorporar nuevas funcionalidades o perfiles (por ejemplo, cuidadores profesionales, validadores, etc.) requiere una arquitectura modular.
Integrabilidad	Dado que el sistema está concebido como una arquitectura base que se puede adaptar al desarrollo de aplicaciones, se requiere que la solución sea capaz de integrarse fácilmente con servicios externos como pasarelas de pago, módulos de verificación o sistemas de mensajería, facilitando su evolución y expansión futura.
Interoperabilidad	Dado que a futuro se proyecta la integración con sistemas externos como plataformas gubernamentales para verificación de identidad y antecedentes, el sistema debe ser capaz de intercambiar información utilizando estándares y protocolos comunes que aseguren una comunicación efectiva entre plataformas heterogéneas.

3.3. Diseño de la arquitectura

3.3.1. Descripción general de la solución

La solución propuesta consiste en una plataforma moderna, escalable y altamente disponible basada en servicios administrados de AWS, diseñada bajo principios de arquitectura modular, desacoplamiento funcional y mejores prácticas de ingeniería de software en la nube (Len Bass, 2021; Amazon Web Services, 2024). El objetivo principal de la solución es habilitar el procesamiento seguro, eficiente y trazable de las operaciones del sistema incluyendo registro de usuarios, gestión de reservas, validación de documentos, actualización de estados y demás funcionalidades transversales; mediante un backend serverless y un frontend distribuido globalmente Mell and Grance (2011).

La arquitectura se implementa siguiendo el patrón Serverless First, donde los componentes son completamente administrados y permiten minimizar la operación, incrementar la elasticidad y optimizar los costos (Amazon Web Services, 2024). Para la capa de presentación, se utiliza Amazon S3 como repositorio estático y Amazon CloudFront como CDN para distribuir el frontend con baja latencia y alta disponibilidad. Este componente asegura que las solicitudes del usuario final sean entregadas de forma segura mediante protocolo HTTPS y políticas de acceso (Amazon Web Services, 2025).

Las interacciones del frontend con el backend se realizan a través de Amazon API Gateway, que actúa como puerta de entrada controlada y segura de todas las peticiones (Amazon Web Services, 2024). API Gateway implementa autenticación mediante tokens firmados e integración con CORS controlado. Cada endpoint invoca de manera desacoplada funciones AWS Lambda, las cuales encapsulan reglas de negocio específicas siguiendo un enfoque modular orientado a responsabilidades únicas (Len Bass, 2021). Esto permite que cada función Lambda represente un caso de uso independiente, facilitando mantenibilidad, escalabilidad horizontal y despliegues granulares.

Para la capa de persistencia, la solución combina dos modelos de datos complementarios. Ambas capas de datos se consumen mediante repositorios especializados, asegurando que los servicios de negocio se mantengan libremente acoplados al tipo de almacenamiento utilizado (Len Bass, 2021):

- Amazon DynamoDB, utilizado para operaciones que requieren accesos de baja latencia y consultas basadas en claves, como validación de existencia de usuarios por identificadores (por ejemplo, documento de identidad) (Amazon Web Services, 2024).
- Amazon RDS for MySQL, empleado para operaciones transaccionales, relaciones complejas y persistencia estructurada (Amazon Web Services, 2024).

El flujo completo asegura integridad y consistencia operacional, desde que un usuario en el navegador genera una petición, pasando por la red global de CloudFront, entrando por API Gateway,

ejecutando la lógica en las Lambdas, y persistiendo o consultando los datos en DynamoDB o RDS según corresponda. Para esto la solución incorpora diferentes mecanismos de seguridad end-to-end (Amazon Web Services, 2025):

- Autorización de peticiones desde CloudFront hacia API Gateway mediante restricciones de origen.
- Mecanismos de validación de entrada a nivel de API Gateway y Lambda (DTOs, sanitización, control de excepciones).
- Manejo seguro de errores evitando exposición de trazas internas.
- Gestión de secretos mediante AWS Secrets Manager.
- Roles IAM con privilegios mínimos para cada componente.

Finalmente, la arquitectura favorece una operación automatizada mediante CI/CD, observabilidad mediante CloudWatch Logs y métricas, y monitoreo continuo de los servicios. Esto garantiza un ciclo de vida sostenible, medible y alineado a los estándares profesionales de una solución en la nube para entornos de producción (Amazon Web Services, 2025; Sommerville, 2016).

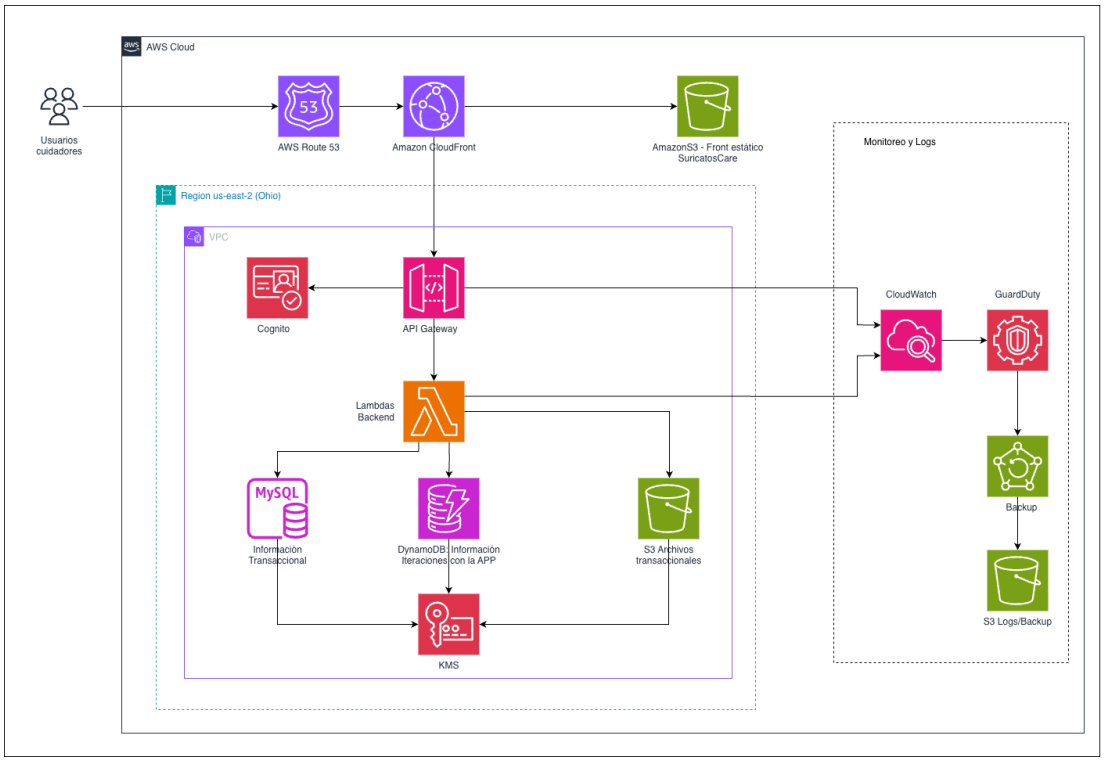


Figura 3.5: Vista Arquitectura. Fuente: Elaboración propia acorde a la arquitectura propuesta.

3.3.2. Diagramas C4

Los diagramas bajo el Modelo C4 permiten describir la arquitectura del sistema de manera estructurada y jerárquica, facilitando la comprensión del contexto y los componentes que conforman la solución propuesta (Model, sf).

Al emplear el Modelo C4 en este proyecto, se logra representar de forma clara cómo se relacionan los usuarios con el sistema (Nivel 1 - Contexto), cómo se distribuyen los principales bloques tecnológicos que lo soportan (Nivel 2 - Contenedor), y cómo se organizan internamente los módulos y servicios que componen la solución (Nivel 3 - Componentes). Este enfoque visual no solo facilita la comunicación con stakeholders técnicos y no técnicos, sino que también respalda el diseño modular y la arquitectura serverless adoptada en AWS, en coherencia con los principios arquitectónicos descritos por Len Bass (2021) y las buenas prácticas del AWS Well-Architected Framework (Amazon Web Services, 2025).

Los diagramas presentados a continuación resumen la estructura técnica del sistema y sirven como guía para entender la distribución lógica, las dependencias entre componentes y los flujos de interacción entre la capa de presentación, las APIs, las funciones Lambda y las capas de persistencia.

3.3.2.1. Diagrama de Contexto

La figura 3.6 muestra al sistema como una unidad completa y las interacciones que mantiene con los actores externos. En este proyecto, se tienen dos tipos de actores el usuario que representa al Padre/Madre y el cuidador quien es la persona capacitada para ofertar servicios de cuidado infantil. Ambos acceden a SuricatosCare como una aplicación que permite la interacción entre ambas partes. Este nivel permite comprender de manera global cómo el sistema se integra con los actores humanos y con servicios externos, sin exponer los detalles internos de la arquitectura.

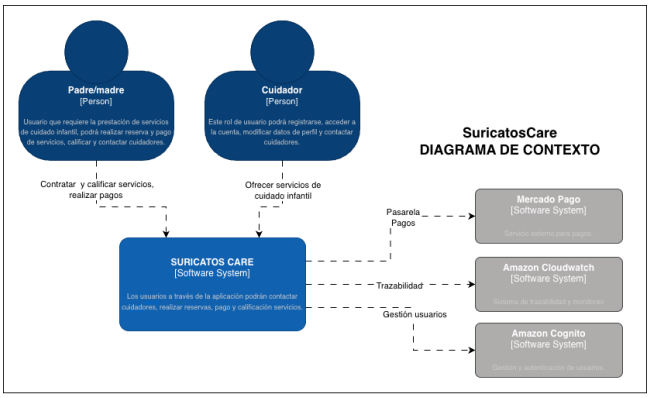


Figura 3.6: Diagrama contexto. Fuente: Elaboración propia acorde a la arquitectura propuesta.

3.3.2.2. Diagrama de Contenedor

En la Figura 3.7 se detalla los principales bloques tecnológicos que componen la solución y cómo colaboran entre sí. El frontend está alojado en Amazon S3 y entregado mediante CloudFront. La capa de backend está compuesta por múltiples funciones AWS Lambda que ejecutan la lógica de negocio de manera desacoplada y escalable. API Gateway actúa como orquestador entre el frontend y las Lambdas. Los datos se almacenan en dos servicios: DynamoDB y Aurora Serverless MySQL. Este nivel evidencia claramente cómo se distribuyen las responsabilidades, cómo se garantiza la escalabilidad y cómo se reduce la gestión operativa mediante servicios totalmente administrados.

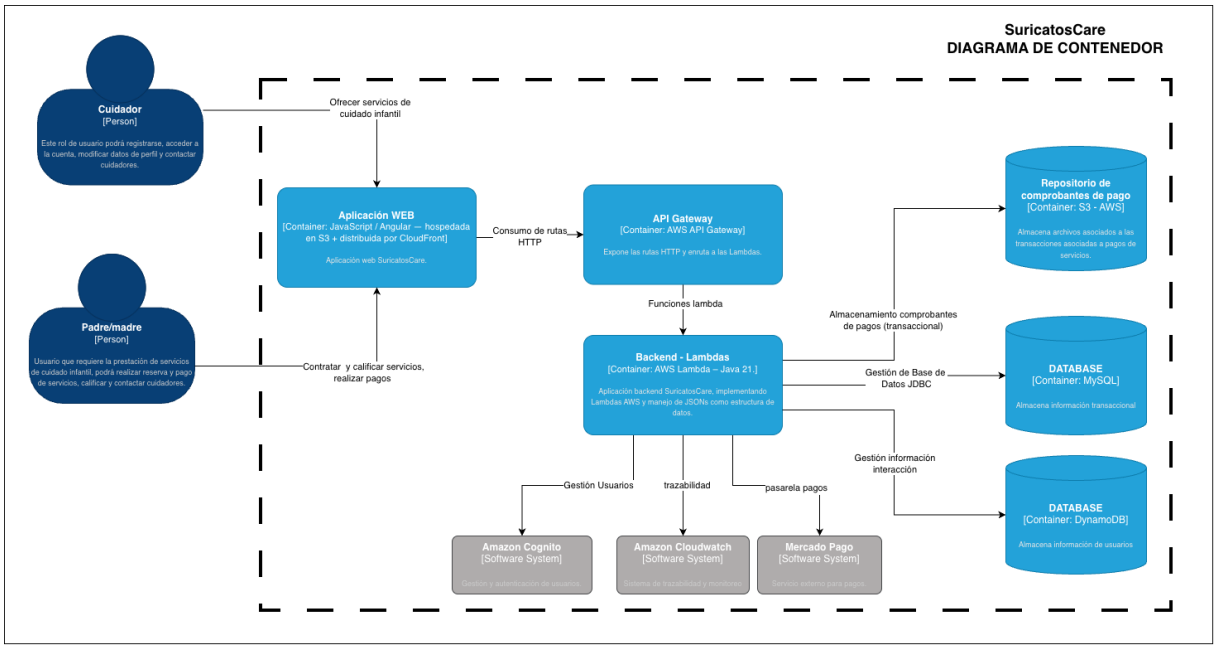


Figura 3.7: Diagrama contenedor. Fuente: Elaboración propia acorde a la arquitectura propuesta.

3.3.2.3. Diagrama de Componentes

En la Figura 3.8 se detalla la estructura interna de cada contenedor, principalmente las funciones AWS Lambda. Cada Lambda implementa un caso de uso específico y sigue la arquitectura modular aplicada en el proyecto, con capas separadas para el controlador (handler), la lógica de negocio (services) y la capa de persistencia (repositories). Esta organización facilita la mantenibilidad, el versionamiento independiente y el despliegue aislado de cada componente. Además, muestra cómo cada función Lambda interactúa con DynamoDB, Aurora o Secrets Manager, reforzando la trazabilidad de la lógica de negocio dentro de la nube.

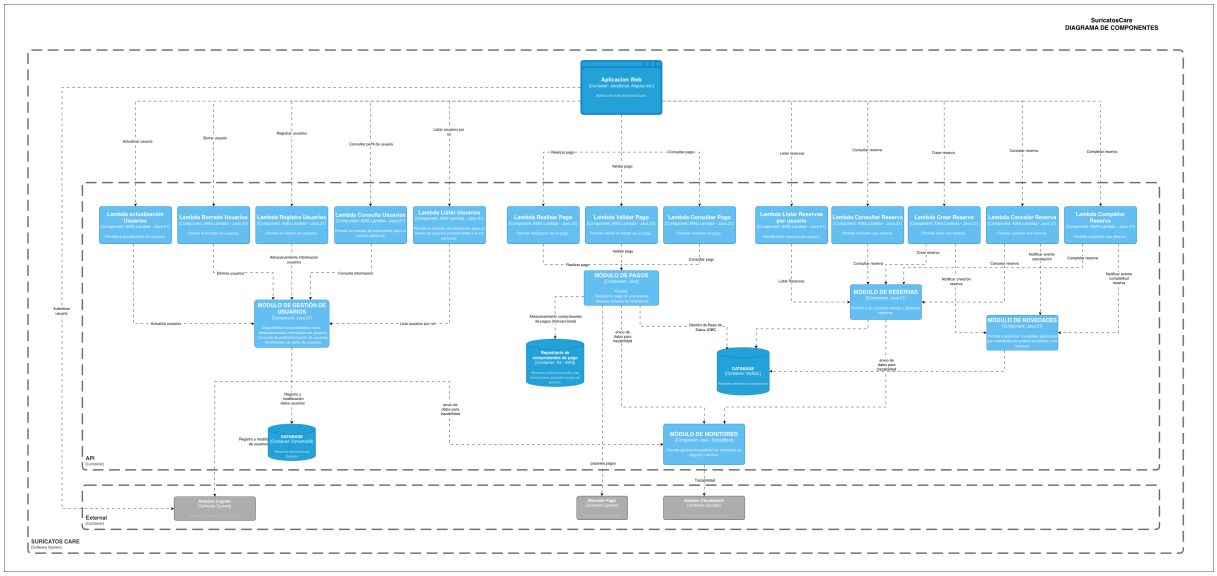


Figura 3.8: Diagrama componentes. Fuente: Elaboración propia acorde a la arquitectura propuesta.

3.3.3. Estilos arquitectónicos aplicados

La solución desarrollada se fundamenta en una combinación de estilos arquitectónicos modernos que permiten garantizar resiliencia, escalabilidad, mantenibilidad y una operación costo eficiente en la nube. Estos estilos se seleccionaron teniendo en cuenta el dominio del sistema, las restricciones operativas, los requisitos de disponibilidad y la necesidad de desacoplamiento entre componentes, siguiendo las recomendaciones de la literatura especializada en arquitectura de software y lineamientos de arquitectura cloud-native.

3.3.3.1. Arquitectura Basada en Servicios

La estructura general del sistema se sustenta en los principios de la Arquitectura Basada en Servicios (SOA), la cual promueve la exposición de funcionalidades a través de interfaces bien definidas y desacopladas (Lanman et al., 2013). En el contexto de la solución, cada funcionalidad principal como gestión de usuarios y reservas se implementa como un servicio independiente expuesto mediante API Gateway. Esta aproximación facilita la reutilización, reduce el acoplamiento entre módulos y habilita la evolución independiente de cada servicio sin afectar al resto de la aplicación.

3.3.3.2. Arquitectura de Microservicios

Aunque el sistema no adopta microservicios en su expresión más estricta, sí incorpora varios de sus principios fundamentales, entre ellos: despliegue independiente, límites claros de responsabilidad y elasticidad horizontal (Mark Richards, 2025). Las funciones AWS Lambda representan unidades de computación autónomas que encapsulan lógica específica (por ejemplo, creación de reservas, consulta de usuarios o actualización de estados). Esto facilita una escalabilidad fina y una asignación eficiente de recursos, alineada con las buenas prácticas de arquitectura serverless descritas por AWS (Amazon Web Services, 2025).

3.3.3.3. Arquitectura Serverless

El elemento predominante del diseño es la adopción del estilo *serverless*, donde la gestión de servidores, capacidad y disponibilidad es delegada completamente al proveedor cloud (Amazon Web Services, 2024). Las funciones AWS Lambda, combinadas con API Gateway y DynamoDB/RDS, proporcionan un modelo altamente desacoplado, sin gestión de infraestructura y con escalado automático. Este estilo arquitectónico reduce la complejidad operativa, mejora la resiliencia y favorece un modelo de costos basado en uso.

3.3.3.4. Arquitectura Hexagonal

Para conservar la mantenibilidad y aislar la lógica del dominio de los detalles de infraestructura, se aplicaron principios de la Arquitectura Hexagonal descrita por Len Bass (2012). Esto se evidencia en la separación entre:

- Lógica de negocio - capa de dominio.
- Adaptadores de entrada - Lambda handlers.
- Adaptadores de salida - repositorios hacia DynamoDB o RDS.

Este estilo permite reemplazar o modificar tecnologías sin afectar el núcleo del dominio, habilitando la posibilidad futura de migraciones tecnológicas; como cambiar DynamoDB por otro almacén de datos sin impactos significativos.

3.3.3.5. Arquitectura Cliente-Servidor

Dado que el frontend se despliega en Amazon CloudFront con origen en un bucket S3, el sistema mantiene características del estilo Cliente-Servidor tradicional (Len Bass, 2012). El cliente (navegador del usuario final) solicita recursos estáticos y realiza peticiones HTTP hacia el backend expuesto por API Gateway. Este patrón no solo simplifica la interacción, sino que se integra adecuadamente con los estilos cloud native adoptados en el backend.

3.3.3.6. Integración de los Estilos

La combinación de los estilos descritos permite una arquitectura robusta, moderna y alineada con los principios de *Cloud Native*. SOA y microservicios aportan modularidad y desacoplamiento; serverless aporta elasticidad y eficiencia operacional y la arquitectura hexagonal garantiza mantenibilidad..

3.3.4. Decisiones de diseño y trade offs

El diseño arquitectónico de la plataforma se fundamentó en la necesidad de lograr un equilibrio entre escalabilidad, mantenibilidad, costo–eficiencia y simplicidad operativa. Tomar decisiones arquitectónicas implica aceptar compromisos entre atributos de calidad, tal como expone Len Bass (2012). En este sentido, cada una de las decisiones clave adoptadas en la solución busca maximizar los beneficios esperados, reconociendo y gestionando de manera explícita sus implicaciones y trade offs.

3.3.4.1. Adopción del modelo Serverless

La elección de un modelo *serverless* basado en AWS Lambda se justificó por su capacidad de eliminar la gestión directa de infraestructura, habilitar un escalado automático y optimizar costos bajo un esquema estrictamente orientado al consumo. De acuerdo con AWS (Amazon Web Services, 2024; Amazon Web Services, 2025), este enfoque permite a los equipos de desarrollo concentrarse en la lógica de negocio en lugar de en aspectos operativos como capacidad, servidores o parches del sistema.

Trade-off: La principal desventaja radica en la complejidad asociada a la observabilidad y depuración distribuida, dado que las funciones son efímeras y altamente desacopladas. También implica dependencia del proveedor y límites estrictos de ejecución y memoria. Estos aspectos fueron mitigados mediante la integración con CloudWatch y el diseño de funciones atómicas alineadas con

buenas prácticas serverless.

3.3.4.2. Uso combinado de DynamoDB y Amazon RDS

La solución combina dos modelos de persistencia: DynamoDB para usuarios y RDS Aurora MySQL para reservas. Esta decisión responde a la necesidad de balancear rendimiento, flexibilidad en consultas y costos. DynamoDB permite trabajo de baja latencia y escalabilidad inmediata para entidades con acceso directo por llave, mientras que RDS ofrece consistencia transaccional y consultas SQL necesarias para las operaciones complejas asociadas a reservas.

Trade-off: El uso de dos bases de datos introduce complejidad operativa y lógica adicional en el dominio. No obstante, tal como argumenta Mark Richards (2025), seleccionar tecnologías especializadas según los requerimientos del componente es una práctica recomendable en arquitecturas modernas, siempre que se mantengan límites claros entre contextos.

3.3.4.3. Separación en capas mediante Arquitectura Hexagonal

Se adoptaron principios de Arquitectura Hexagonal para separar la lógica de dominio de los adaptadores de entrada y salida. Esto facilita la mantenibilidad y la posibilidad de reemplazar tecnologías sin afectar el núcleo del sistema. Según Len Bass (2012), este tipo de desacoplamiento estructural incrementa la vida útil del software y reduce la deuda técnica.

Trade-off: La adopción de este estilo implica una inversión inicial mayor en diseño y estructura de carpetas, así como mayor esfuerzo para desarrolladores sin experiencia previa en patrones de puertos y adaptadores. Sin embargo, este costo se ve compensado por la reducción de dependencias tecnológicas a largo plazo.

3.3.4.4. Exposición del backend mediante API Gateway

La plataforma se expone a través de Amazon API Gateway, lo cual permite gestionar autenticación, límites de petición, seguridad y control de acceso. Esto habilita un desacoplamiento completo entre frontend y backend, consistente con estilos orientados a servicios (Lanman et al., 2013).

Trade-off: API Gateway introduce latencias adicionales por la capa de intermediación entre el cliente y las Lambdas, especialmente bajo alto volumen de solicitudes. Sin embargo, su capacidad de gestión, seguridad y monitoreo justifican los milisegundos de latencia añadidos.

3.3.4.5. Entrega del frontend mediante S3 y CloudFront

La decisión de hospedar el frontend en Amazon S3 y distribuirlo mediante CloudFront obedece a la necesidad de disponibilidad global, baja latencia y reducción de costos. Este diseño se articula naturalmente con el estilo cliente-servidor descrito por Len Bass (2012).

Trade-off: El despliegue estático limita la ejecución de lógica dinámica fuera del backend y requiere políticas estrictas de CORS con API Gateway. Esta limitación fue mitigada mediante configuración explícita de encabezados de seguridad y pruebas exhaustivas de comunicación entre capas.

3.3.4.6. Balance general de decisiones

Todas las decisiones arquitectónicas reflejan un balance entre simplicidad, desempeño, costo y mantenibilidad. Siguiendo a Len Bass (2012), un diseño arquitectónico robusto no elimina los trade-offs, sino que los hace explícitos y gestionables. En este proyecto, la combinación de estilos permitió construir una solución moderna, escalable y alineada con los requerimientos funcionales y no funcionales del dominio.

3.4. Implementación del prototipo

La implementación del prototipo se desarrolló siguiendo principios de arquitectura moderna, priorizando la escalabilidad, el desacoplamiento y la operación costo–eficiente. Para ello, se seleccionó un conjunto de tecnologías cloud y frameworks que permiten soportar adecuadamente los atributos de calidad definidos durante la etapa de diseño. Esta sección describe detalladamente las tecnologías empleadas, así como su rol dentro de la solución final.

3.4.1. Tecnologías Utilizadas

La construcción del prototipo combina servicios administrados de Amazon Web Services (AWS), herramientas de desarrollo backend basadas en Java y mecanismos de almacenamiento orientados a distintos modelos de acceso. La selección tecnológica se fundamenta en criterios de adecuación arquitectónica, buenas prácticas cloud-native y recomendaciones del AWS Well-Architected Framework (Amazon Web Services, 2024; Amazon Web Services, 2025).

3.4.1.1. Amazon Web Services (AWS)

AWS constituye la base de infraestructura del prototipo, permitiendo una operación bajo demanda, escalado automático y mínima gestión de servidores. Siguiendo la definición de computación en la nube formulada por el NIST (Mell and Grance, 2011), la plataforma aprovecha las características esenciales del modelo cloud: autoservicio bajo demanda, asignación elástica de recursos y servicios administrados.

AWS Lambda: Las funciones AWS Lambda ejecutan la lógica de negocio de manera completamente *serverless*. Este servicio elimina la necesidad de aprovisionar o administrar servidores, ajustando automáticamente la capacidad en respuesta al volumen de solicitudes. Su adopción se alinea con los principios de arquitectura serverless recomendados por Amazon Web Services (2024) y con prácticas modernas de diseño de sistemas descritas por Richards y Ford (Mark Richards, 2025).

Amazon API Gateway: Se utiliza como punto de entrada al backend, exponiendo rutas REST para las funcionalidades de usuarios y reservas. Este componente proporciona mecanismos integrados de autenticación, control de tráfico y CORS, facilitando una arquitectura basada en servicios, consistente con los principios para entornos orientados a servicios (Lanman et al., 2013).

Amazon DynamoDB: Se emplea como base de datos NoSQL para la gestión de usuarios. Su modelo clave-valor y su escalabilidad automática permiten atender cargas variables con baja latencia. Esta elección responde al patrón de acceso directo por llave y al requerimiento de ofrecer consultas predictivas en tiempo constante.

Amazon Aurora MySQL (RDS): El sistema de reservas requiere relaciones entre entidades y operaciones transaccionales complejas. Para ello se utilizó Aurora MySQL, un motor relacional administrado que ofrece compatibilidad con MySQL y escalabilidad automática. Según Amazon Web Services (2024), Aurora permite alta disponibilidad sin la carga operativa de administrar instancias tradicionales.

Amazon S3 y CloudFront: El frontend de la aplicación se hospeda como contenido estático en Amazon S3, distribuido globalmente mediante Amazon CloudFront. Este patrón cliente-servidor, descrito ampliamente en literatura de arquitectura de software (Len Bass, 2012), mejora la entrega de contenido y reduce los tiempos de respuesta.

3.4.1.2. Backend en Java con Arquitectura Hexagonal

La capa de negocio se implementó en Java 21, aplicando principios de Arquitectura Hexagonal para separar lógica de dominio, adaptadores de entrada (Lambda handlers) y adaptadores de salida

(repositorios hacia DynamoDB y RDS). Esta estructura sigue las recomendaciones de modularidad y mantenibilidad planteadas por Len Bass (2012) y por Richard N. Taylor (2009).

Maven: Se utilizó Maven como gestor de dependencias para estructurar y empaquetar el proyecto, garantizando repetibilidad y facilidad en la integración continua.

3.4.1.3. Frontend Web

El frontend se desarrolló como una aplicación web estática que interactúa exclusivamente con API Gateway mediante peticiones HTTP. Su despliegue en S3 y distribución vía CloudFront garantiza baja latencia y compatibilidad con las arquitecturas orientadas a servicios.

3.4.1.4. Herramientas de Monitoreo y Seguridad

AWS CloudWatch: Se empleó para monitoreo de logs, métricas de ejecución de Lambda y alertas operativas. Su integración directa con los servicios serverless facilita la trazabilidad en arquitecturas distribuidas.

AWS IAM: Se configuraron roles y políticas de acceso siguiendo el principio de mínimo privilegio, conforme a las recomendaciones del Well-Architected Framework (Amazon Web Services, 2025).

3.4.1.5. Control de versiones y desarrollo colaborativo

Para garantizar trazabilidad y versionamiento, el proyecto utiliza Git y repositorios remotos. Esto permite seguir buenas prácticas de ingeniería recomendadas por Sommerville (2016) y Wiegers and Beatty (2013) en cuanto al manejo disciplinado del ciclo de vida del software.

3.4.2. Alcance funcional del MVP

El desarrollo del Producto Mínimo Viable (MVP) se orientó a validar las funcionalidades esenciales del sistema y demostrar la viabilidad técnica y operativa de la solución propuesta. En esta fase se priorizaron aquellos procesos que generan mayor valor para los usuarios finales y que permiten evaluar el flujo completo de interacción entre el frontend, los servicios backend y la infraestructura cloud desplegada. El MVP busca ofrecer una versión operativa que permita probar los componentes críticos, verificar hipótesis de diseño y establecer una base sólida para futuras iteraciones. La

definición del alcance funcional se basó en criterios de impacto, factibilidad y alineación con los objetivos del proyecto, las funcionalidades que hacen parte del MVP son:

- Registro de usuarios.
- Inicio de sesión.
- Creación de reserva.
- Cancelación de reserva.
- Completitud de reserva.

3.4.2.1. Diagramas de flujo

Los diagramas de flujo permiten representar de manera clara y estructurada la secuencia de operaciones y decisiones dentro del sistema SuricatosCare. A continuación se describen los flujos que se desarrollaron para el MVP objetivo de esta tesis.

- Registro: describe la secuencia de pasos necesarios para que un Padre/Madre y/o cuidador; pueda registrarse dentro de la plataforma de SuricatosCare.

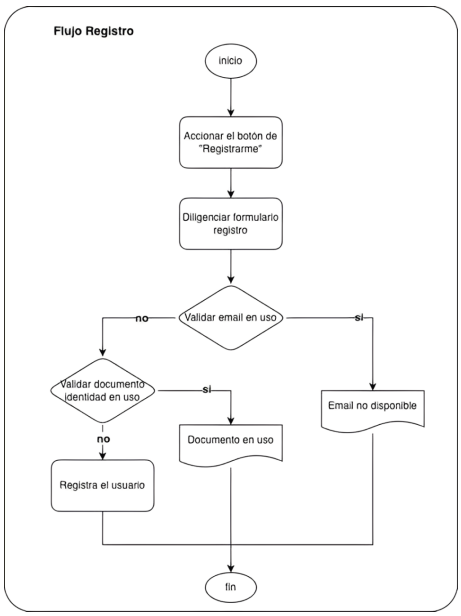


Figura 3.9: Flujo registro de usuario. Fuente: Elaboración propia acorde al mvp construido.

- Iniciar sesión: este flujo detalla la forma en que un usuario puede acceder a la aplicación mediante el uso de sus credenciales de acceso.

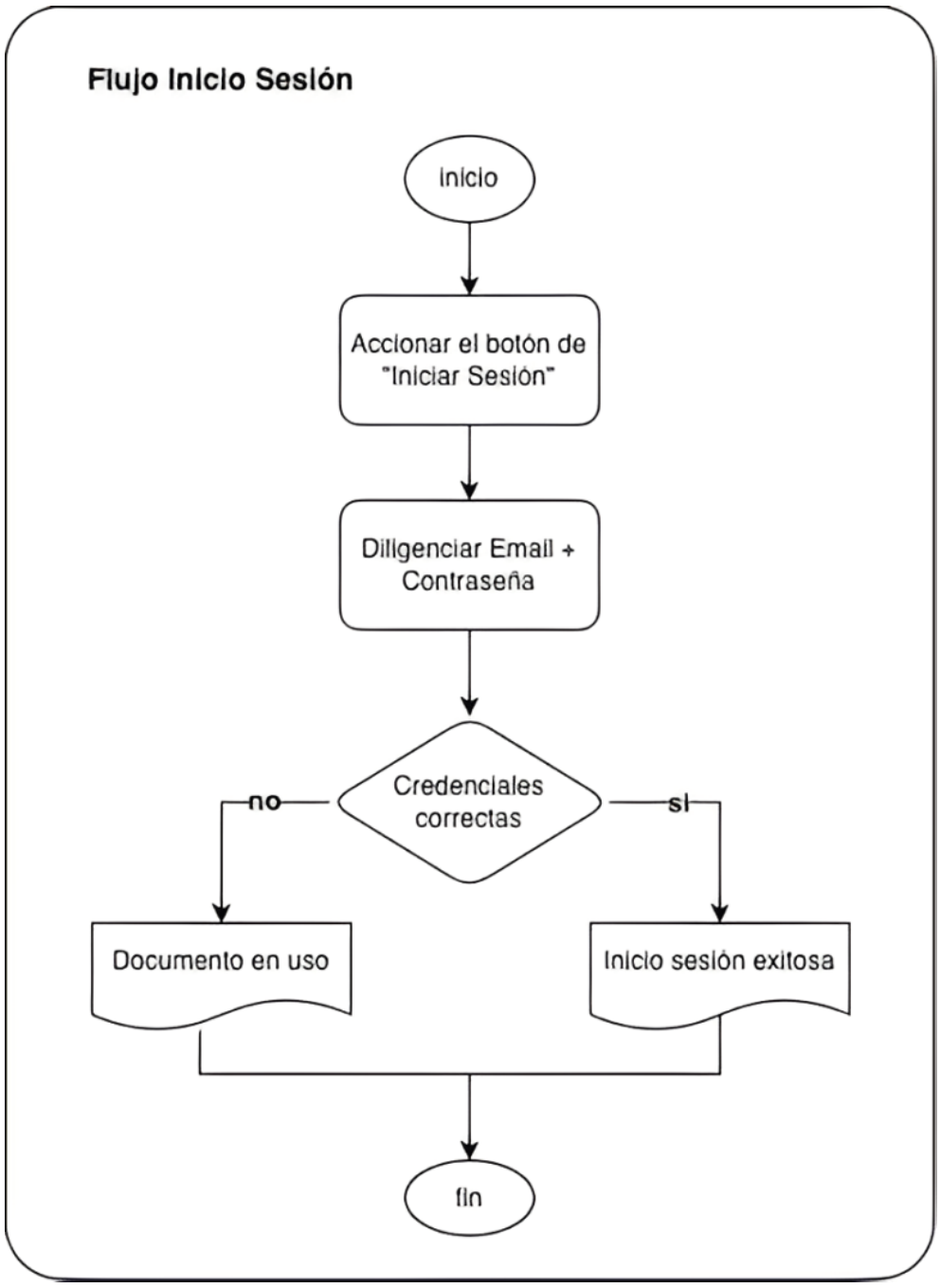


Figura 3.10: Flujo inicio de sesión. Fuente: Elaboración propia acorde al mvp construido.

- Crear reserva: se presenta la forma como un usuario ya autenticado dentro de SuricatosCare puede crear una reserva; mediante la cual realizará la solicitud de un servicio de cuidado para un cuidador y en una fecha específica.

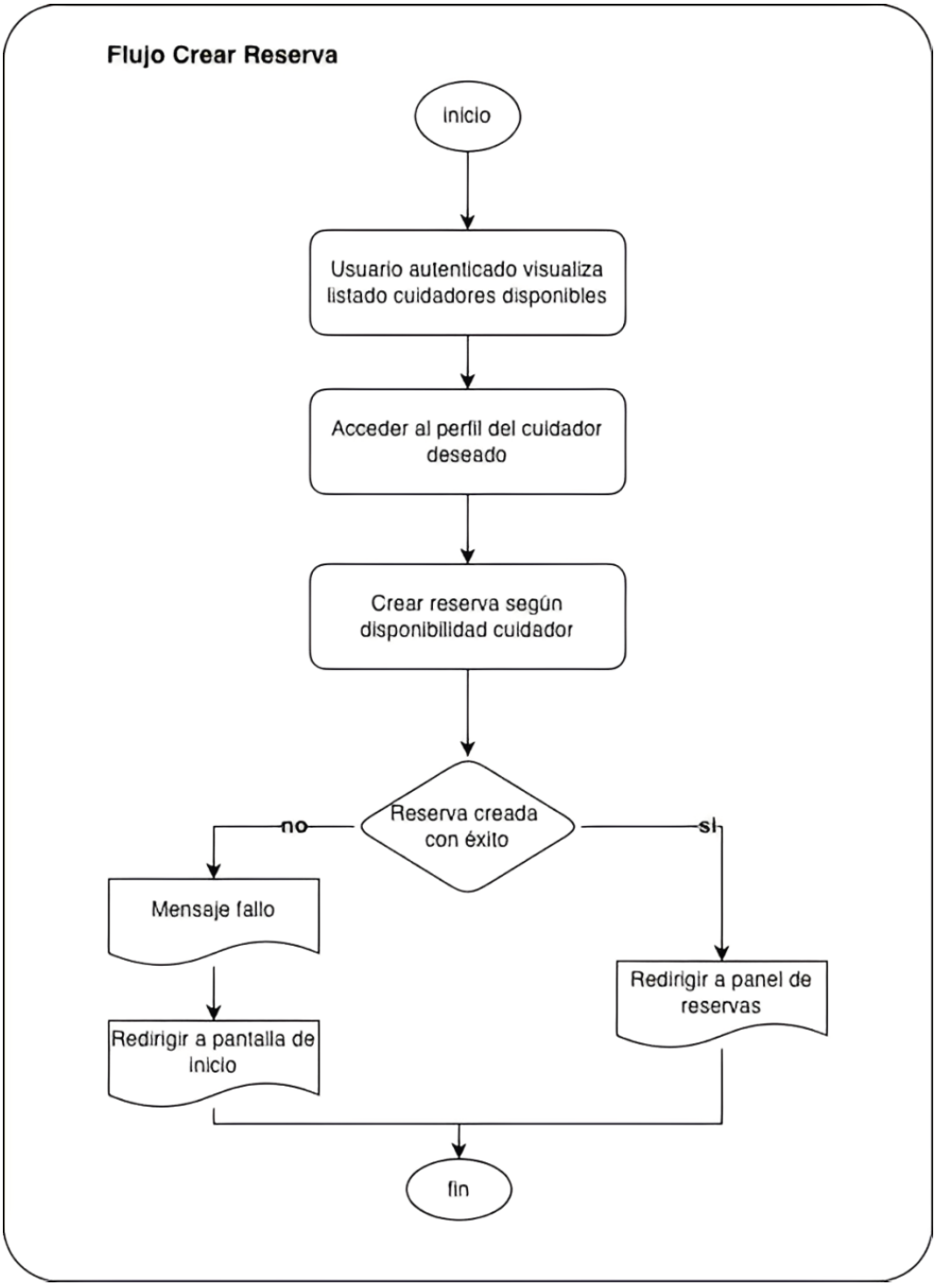


Figura 3.11: Flujo creación reserva. Fuente: Elaboración propia acorde al mvp construido.

- Cancelar Reserva: este flujo muestra los pasos para que una reserva creada; pueda ser cancelada por la persona que la creó (Padre/Madre).

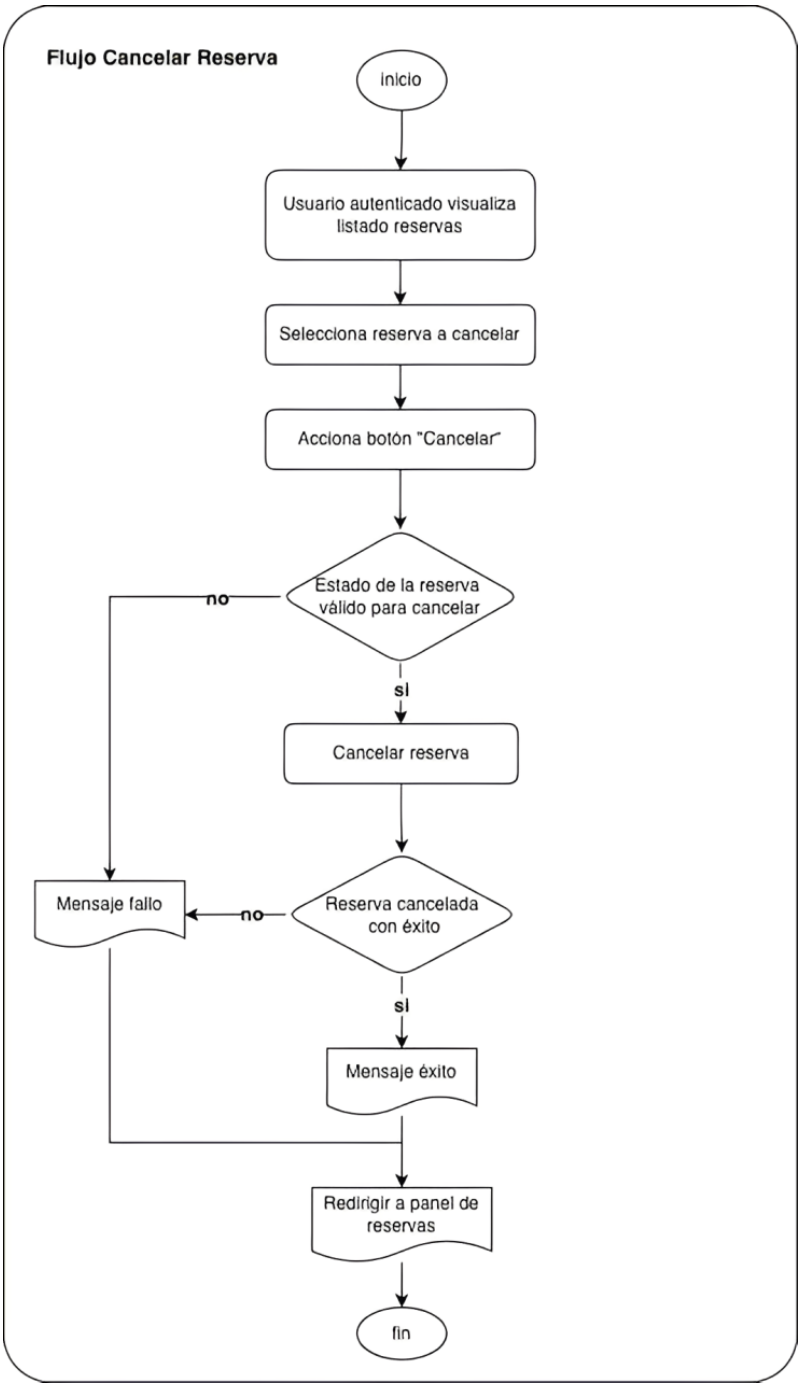


Figura 3.12: Flujo cancelación reserva. Fuente: Elaboración propia acorde al mvp construido.

- Completar reserva: consiste en el caso de uso donde un servicio reservado ha sido completado, refleja el paso a paso mediante el cual un usuario puede dar por completado un servicio incluyendo la calificación del cuidador asociado.

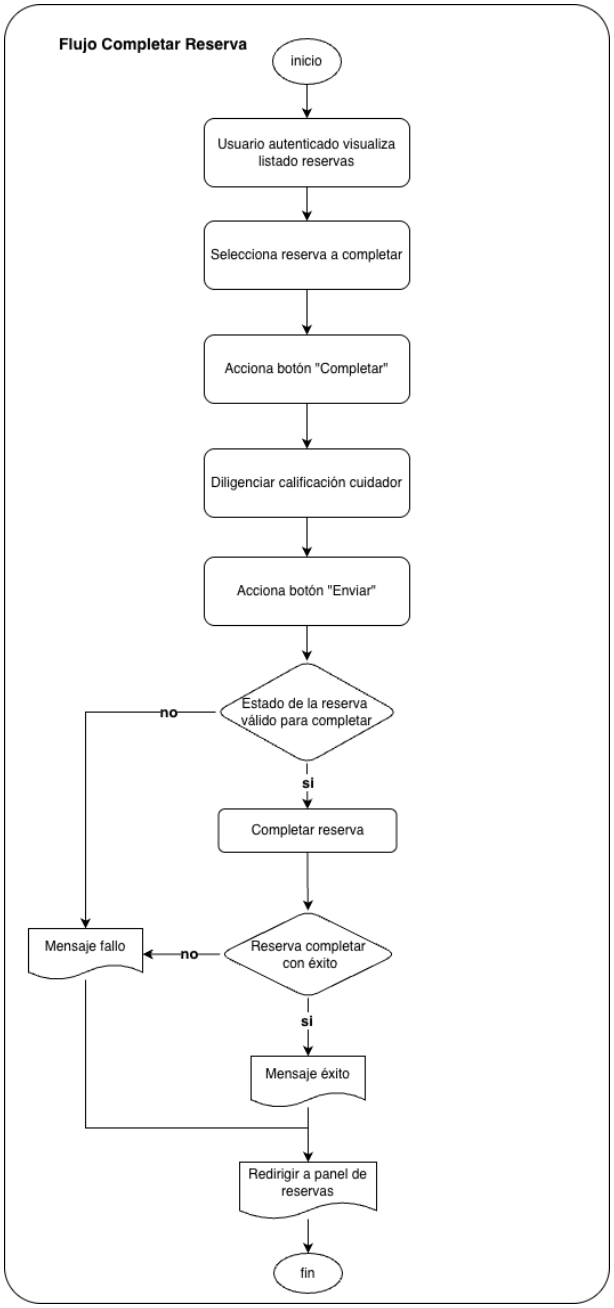


Figura 3.13: Flujo completar reserva. Fuente: Elaboración propia acorde al mvp construido.

3.4.3. Codificación

La fase de codificación corresponde a la implementación concreta de la arquitectura definida para la aplicación SuricatosCare, siguiendo principios de modularidad, mantenibilidad y desacoplamiento. Para este proyecto, el backend se desarrolló utilizando Java 21 como lenguaje principal y Maven como sistema de gestión de dependencias y construcción. La solución se implementó bajo un enfoque serverless, donde cada funcionalidad es ejecutada mediante AWS Lambda, lo que permite desplegar unidades de cómputo altamente escalables, de bajo costo y sin necesidad de administrar servidores.

El código se organizó siguiendo una estructura por dominios y basada en los principios de la Arquitectura Hexagonal (Ports and Adapters), lo cual facilita la separación entre lógica de negocio, casos de uso, adaptadores de entrada (funciones Lambda) y adaptadores de salida (accesos a DynamoDB y RDS MySQL). Esta organización fomenta la mantenibilidad, el testeo independiente por capa y la posibilidad de reemplazar tecnologías sin afectar el núcleo del sistema.

En la Figura 3.14 se muestra la estructura final del código fuente del backend, organizada por dominios y capas conforme a la arquitectura definida. Link GitHub

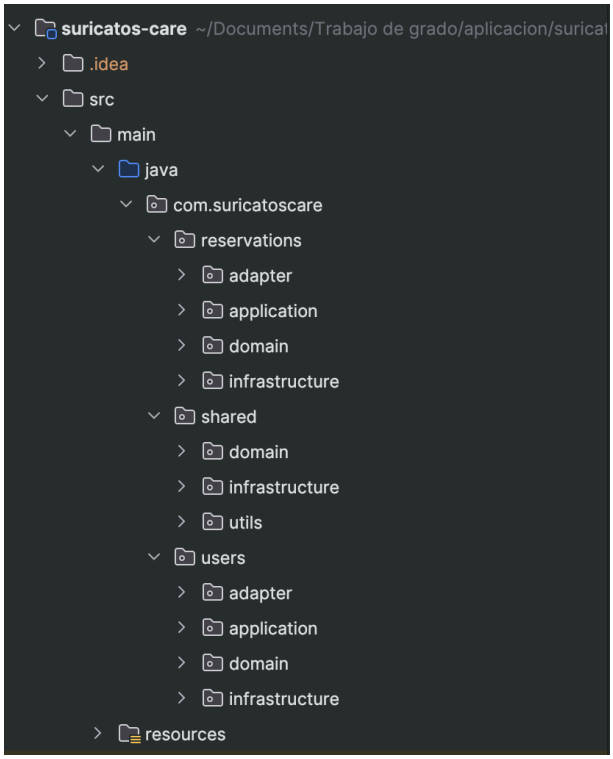


Figura 3.14: Estructura código backend. Fuente: Elaboración propia acorde al mvp construido.

Por otra parte, la implementación del frontend se desarrolló utilizando React, un framework que ofrece capacidades para construir interfaces de usuario dinámicas, modulares y reutilizables. Esta elección tecnológica se alinea con los principios de escalabilidad y modularidad definidos en la arquitectura general del sistema. El frontend fue diseñado como una aplicación de página única (Single Page Application, SPA), lo que permite una navegación fluida entre vistas sin recargar el navegador y mejora la experiencia del usuario. La estructura del proyecto se organiza en componentes, páginas, servicios, tipos y rutas, lo que facilita la encapsulación de responsabilidades, el testing por unidad funcional y el crecimiento modular de la interfaz.

Para su despliegue, se aprovechó el uso de servicios gestionados de AWS. El código del frontend se compila y se aloja como contenido estático en un bucket de Amazon S3, configurado con políticas de acceso seguro. La distribución se realiza mediante Amazon CloudFront, que actúa como red de entrega de contenido (CDN), optimizando el rendimiento, reduciendo la latencia y garantizando una experiencia estable para usuarios distribuidos geográficamente.

En la Figura 3.15 se presenta la estructura de carpetas del frontend implementado, donde se evidencian los componentes clave y la organización del código. Link GitHub

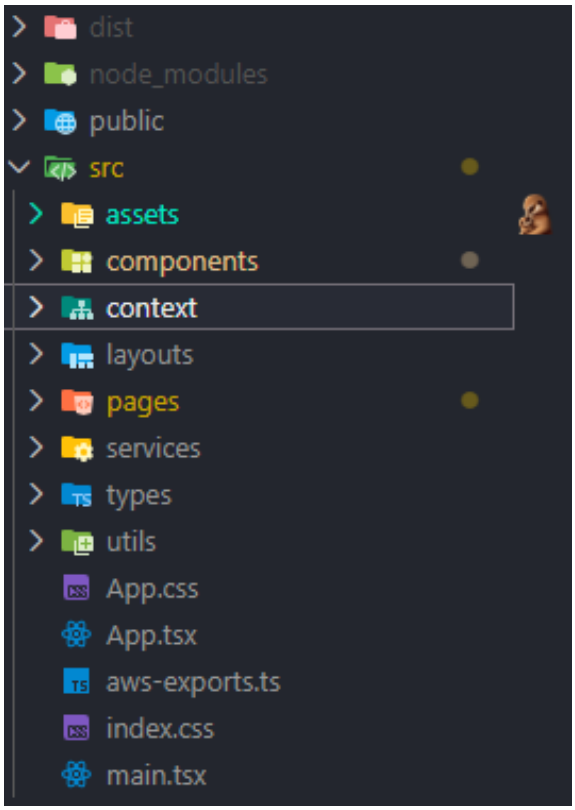


Figura 3.15: Estructura código frontend. Fuente: Elaboración propia acorde al mvp construido.

3.4.4. Capturas o funcionalidades clave

- Registro de usuarios.

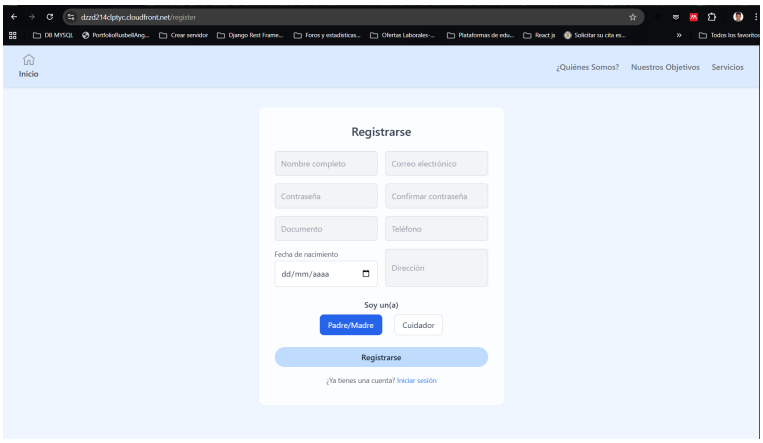


Figura 3.16: Pantalla de registro de usuarios. Fuente: Elaboración propia acorde al mvp construido.

La Figura 3.16 presenta la pantalla de registro, donde los nuevos usuarios pueden crear su cuenta dentro de la plataforma. Esta vista reúne los datos indispensables para identificar al usuario, establecer sus credenciales de acceso y definir su rol dentro del sistema, diferenciando entre cuidadores y padres. Su función principal es habilitar un ingreso controlado y coherente con las necesidades del MVP, garantizando que cada registro cumpla con los criterios previstos para la operación inicial de la plataforma.

- Inicio de sesión.

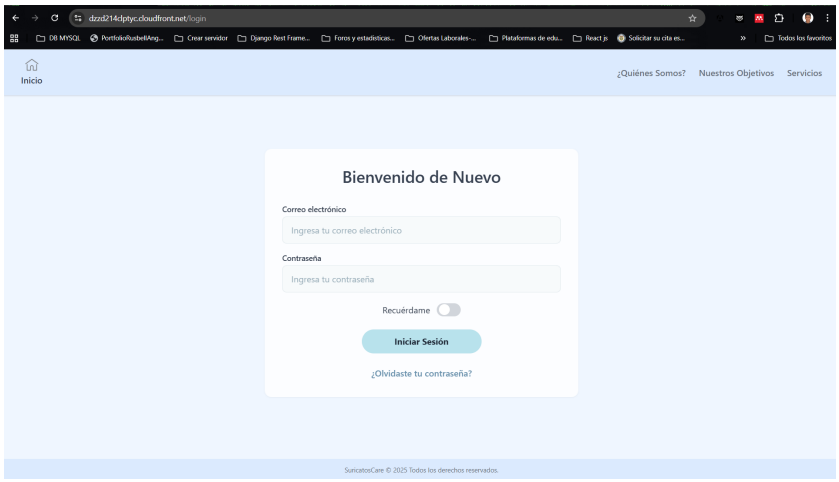


Figura 3.17: Pantalla de inicio de sesión. Fuente: Elaboración propia acorde al mvp construido.

La Figura 3.17 corresponde a la pantalla de inicio de sesión, donde el usuario accede a la plataforma mediante sus credenciales registradas. Esta interfaz concentra el flujo de autenticación y habilita el ingreso seguro al sistema, garantizando que solo usuarios válidos puedan interactuar con las funcionalidades internas del MVP.

- Dashboard con cuidadores.

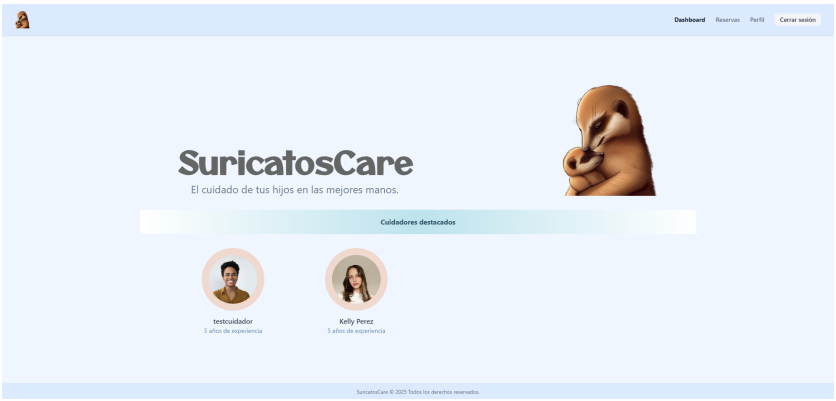


Figura 3.18: Dashboard con cuidadores disponibles. Fuente: Elaboración propia acorde al mvp construido.

La Figura 3.18 muestra la vista inicial disponible para los usuarios con rol padre. En esta pantalla se presenta el listado de cuidadores disponibles dentro del sistema, permitiendo al usuario identificar rápidamente perfiles destacados y continuar con la exploración o selección del cuidador que mejor se ajuste a sus necesidades.

- Creación de reserva.

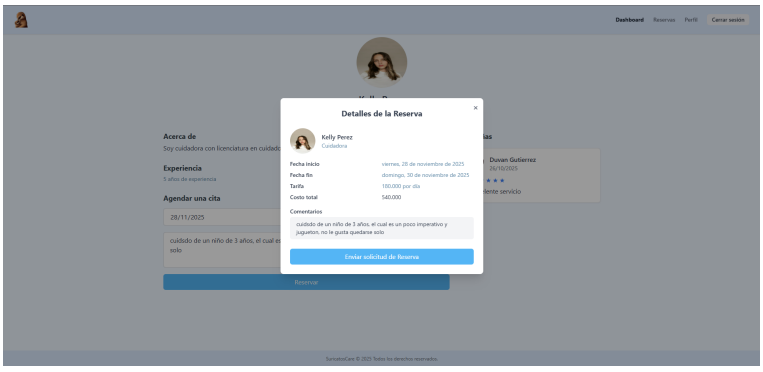


Figura 3.19: Creación de reserva. Fuente: Elaboración propia acorde al mvp construido.

La Figura 3.19 muestra la ventana emergente que aparece durante el proceso de creación de una reserva. En esta vista se resumen los datos esenciales del servicio solicitado, incluida la

información del cuidador, las fechas seleccionadas, la tarifa diaria y el costo total calculado automáticamente según la duración del servicio. También se visualizan los comentarios adicionales registrados por el padre o familiar. Desde este punto, el usuario puede confirmar la solicitud y enviarla para su posterior gestión en la plataforma.

- Lista de reservas.

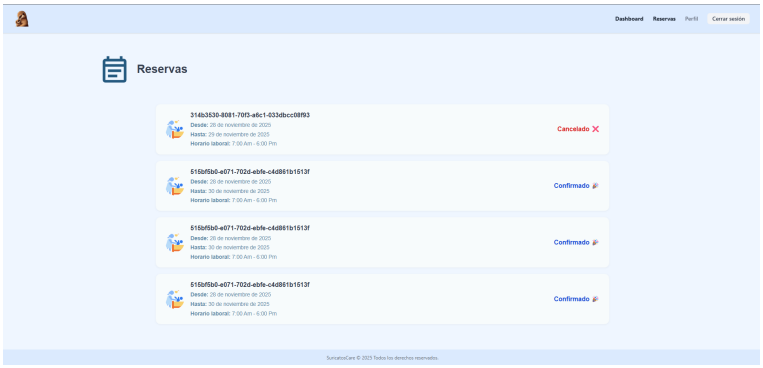


Figura 3.20: Listado de reservas agendadas. Fuente: Elaboración propia acorde al mvp construido.

La Figura 3.20 corresponde a la vista donde el usuario con rol padre puede consultar el historial de reservas realizadas. En esta pantalla se muestran las solicitudes junto con su fecha de inicio, fecha de finalización, horario y estado actual, lo que facilita el seguimiento de los servicios programados y permite identificar rápidamente reservas confirmadas o canceladas dentro del sistema.

- Cancelación de reserva.

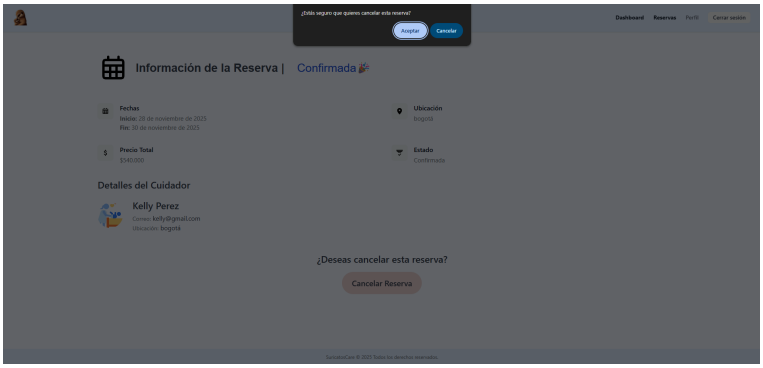


Figura 3.21: Cancelar una reserva. Fuente: Elaboración propia acorde al mvp construido.

La pantalla presentada en la Figura 3.21 permite al usuario gestionar la cancelación de una reserva que se encuentra en estado confirmado. En esta vista se muestran los detalles principales de la reserva, junto con la información del cuidador asignado. Al seleccionar la opción

de cancelación, el sistema despliega un cuadro de confirmación que solicita validar la acción antes de proceder, evitando cancelaciones accidentales.

3.5. Gestión de riesgos

Durante el desarrollo del presente proyecto se identificaron posibles eventos que podrían afectar negativamente el cumplimiento de los objetivos propuestos. En este sentido, se propuso una gestión de riesgos estructurada mediante un proceso de identificación, evaluación y priorización, con el fin de anticipar escenarios adversos y establecer estrategias de mitigación adecuadas. Esta sección presenta los principales riesgos detectados, su análisis cuantitativo y cualitativo, así como las medidas adoptadas para gestionar su impacto y probabilidad.

3.5.1. Identificación de riesgos

La identificación de riesgos constituyó el primer paso del proceso de gestión, permitiendo reconocer los eventos potenciales que podrían afectar la calidad, la seguridad y el éxito operativo del proyecto. En esta etapa se analizaron distintos escenarios que podrían presentarse durante la operación de la plataforma SuricatosCare. La Tabla 3.5 presenta los riesgos detectados, documentando para cada caso su identificador (ID), descripción, escenario y tipo de riesgo.

Tabla 3.5: Identificación de riesgos

ID	**Descripción del riesgo**	**Escenario del riesgo**	**Tipo de riesgo**
R1	Fuga de datos personales de cuidadores y/o padres por almacenamiento inseguro.	Durante un un proceso de auditoría se detecta que los datos sensibles de cuidadores y padres (direcciones, teléfonos, correos electrónicos entre otros) están almacenados en una base de datos sin cifrado. Esto permite que un atacante pueda acceder a la información ocasionando perdida de confidencialidad e incluso sanciones por fallas en la protección de datos.	Tecnológico
R2	Acceso no autorizado a las APIs de la aplicación.	Un atacante descubre que las APIs de la aplicación no cuentan con mecanismos robustos de autenticación y autorización. aprovechando esta debilidad, logra realizar llamadas a servicios a través de los cuales no solo obtiene información; sino que también manipula datos, comprometiendo así la integridad y confidencialidad de la información, afectando la prestación del servicio.	Tecnológico

Continúa en la siguiente página

Tabla 3.5 – *continuación de la página anterior*

ID	Descripción del riesgo	Escenario del riesgo	Tipo de riesgo
R3	Autenticación débil o mal implementada para acceso a la plataforma.	Durante un proceso de registro un usuario puede crear contraseña sin cumplir los requisitos mínimos de seguridad (longitud, caracteres especiales). Un atacante aprovecha la debilidad de este factor y accede a los la cuenta de un usuario; logrando de esta manera visualizar y modificar información., ocasionando vulneración de la privacidad.	Tecnológico
R4	Configuraciones erróneas de artefactos que hacen parte de la arquitectura de la aplicación. Ej Bucket S3 público en la nube.	Durante el proceso de configuración de un bucket, un miembro del equipo de desarrollo lo deja por error con acceso publico. Este descuido es detectado por tercero que aprovecha la falla de seguridad y descarga fotos, datos e información de los usuarios compartiéndola de manera no autorizada, afectando de esta manera la reputación de SuricatosCare y exponiendo la empresa a sanciones legales.	Tecnológico
R5	Validación deficiente en campos de los formularios que permitan la inyección de código.	Un usuario mal intencionado ingresa código malicioso en la sección comentarios en calificación de servicios debido a que no existen validaciones de entrada adecuadas. Esto le permite ejecutar scripts en el sistema redirigiendo a los usuarios a a páginas fraudulentas, ocasionando afectación en la disponibilidad del servicio.	Tecnológico
R6	Errores en la gestión de roles y permisos que permitan accesos no autorizados.	Durante la operación diaria de la plataforma, un error en la configuración de roles permite que un usuario con perfil básico acceda a módulos reservados para administradores. Esto sucede porque no se aplicaron correctamente las reglas de autorización en el backend y no existen validaciones adicionales en el frontend. Como resultado, el usuario puede visualizar y modificar información sensible, generando pérdida de confidencialidad, incumplimiento de la normativa de protección de datos, afectación a la reputación de la plataforma y posibles sanciones legales o económicas.	Tecnológico
R7	Alta dependencia con servicios externos para servicios claves dentro de la aplicación (ej: pasarla de pagos).	En un día de alta demanda, la pasarela de pagos integrada en la plataforma presenta una falla generalizada en sus servidores, impidiendo que los usuarios puedan completar transacciones. Como resultado, los procesos de pago quedan pendientes o cancelados, afectando directamente la confianza de los clientes y generando pérdidas económicas por ventas no concretadas.	Tecnológico

Continúa en la siguiente página

Tabla 3.5 – *continuación de la página anterior*

ID	Descripción del riesgo	Escenario del riesgo	Tipo de riesgo
R8	Sobrecarga del sistema ante aumento repentino de usuarios.	Durante una campaña de lanzamiento en redes sociales, la plataforma recibe un flujo de usuarios muy superior al promedio, lo que provoca que los servidores se saturen y el tiempo de respuesta aumente drásticamente. Esto ocasiona que algunos usuarios no puedan iniciar sesión o completar procesos, afectando su experiencia y generando quejas en línea.	Tecnológico
R9	Perdida de integridad de la información en la comunicación entre sistemas interoperables.	En un proceso automatizado de integración con una API de terceros para gestionar pagos, se omiten validaciones de integridad en los mensajes intercambiados. Esto permite que, ante una falla temporal del servicio, se guarden en la plataforma transacciones incompletas o con montos erróneos, generando reclamaciones por parte de los usuarios.	Operacional
R10	Incumplimiento de normatividad relacionada con el tratamiento de datos personales (financieros).	Durante el almacenamiento de datos financieros de los usuarios, el equipo de desarrollo no aplica el cifrado requerido por la normativa vigente. Este descuido se detecta en una auditoría externa, evidenciando que la plataforma incumple la Ley de Protección de Datos Personales y normativas asociadas, lo que podría derivar en sanciones económicas y pérdida de confianza de los clientes.	Legal
R11	Interrupciones prolongadas del servicio por fallas en la infraestructura de hosting.	Durante un día de operación normal, el proveedor de hosting experimenta una falla crítica en su centro de datos, dejando la plataforma web y móvil fuera de servicio por más de 12 horas. La empresa no cuenta con un plan de contingencia ni servidores redundantes en otro proveedor, lo que impide restablecer la disponibilidad rápidamente. Esto afecta a todos los usuarios, interrumpe transacciones y genera una ola de reclamos en redes sociales, impactando la confianza en el servicio.	Operacional
R12	Falsificación de identidad o antecedentes de cuidadores.	Un cuidador se registra en la plataforma con documentos falsos o antecedentes penales ocultos. Al ser contratado, pone en riesgo la seguridad física y emocional del menor, generando un incidente grave que afecta la reputación y responsabilidad legal de la plataforma.	Operacional
R13	Abuso de confianza o conductas inapropiadas durante el servicio.	Un cuidador, tras ser contratado, comete actos inapropiados (verbales, físicos o psicológicos) contra el menor durante la prestación del servicio. La plataforma no cuenta con mecanismos de monitoreo ni protocolos de denuncia efectivos.	Legal

Continúa en la siguiente página

Tabla 3.5 – *continuación de la página anterior*

ID	Descripción del riesgo	Escenario del riesgo	Tipo de riesgo
R14	Manipulación de reseñas o calificaciones falsas.	Usuarios (cuidadores o padres) crean cuentas falsas o manipulan el sistema de calificaciones para inflar su reputación o perjudicar a otros, erosionando la confianza en el sistema de reputación de la plataforma.	Operacional
R15	Falta de cobertura legal en jurisdicciones donde opera la app.	La plataforma opera en múltiples ciudades o países sin adaptar sus términos de servicio, políticas de privacidad o cumplimiento regulatorio a las leyes locales (ej: GDPR en Europa, COPPA en EE.UU., Ley de Protección de la Infancia en LATAM), exponiéndose a multas y bloqueos.	Legal
R16	Dependencia crítica de un solo proveedor de autenticación (ej: Google/Facebook Login).	La plataforma depende exclusivamente de OAuth de terceros (Google, Facebook) para el inicio de sesión. Si estos servicios sufren una caída o cambian sus políticas, los usuarios no pueden acceder, generando pérdida de confianza y transacciones canceladas.	Tecnológico
R17	Fuga de datos por dispositivos móviles comprometidos (usuarios finales).	Un padre o cuidador accede a la app desde un dispositivo infectado con malware. El atacante roba credenciales o datos sensibles en tránsito, comprometiendo cuentas sin que la plataforma tenga responsabilidad directa, pero afectando su reputación.	Tecnológico
R18	Incumplimiento de obligaciones fiscales o tributarias por pagos a cuidadores.	La plataforma actúa como intermediaria de pagos, pero no retiene impuestos ni emite comprobantes fiscales según la legislación local, generando responsabilidad tributaria para la empresa y sanciones por evasión o elusión fiscal.	Legal
R19	Desinformación o mal uso de la plataforma por usuarios no capacitados.	Padres o cuidadores malinterpretan funcionalidades (ej: confunden reserva con contratación inmediata, no entienden políticas de cancelación), generando conflictos, reclamos y sobrecarga del soporte.	Operacional
R20	Ataques de ingeniería social a empleados o soporte técnico.	Un atacante se hace pasar por usuario legítimo y engaña al equipo de soporte para restablecer contraseñas, acceder a datos o modificar perfiles, aprovechando la falta de protocolos de verificación estrictos.	Operacional

3.5.2. Evaluación de riesgos

Una vez identificados los riesgos, se procedió a su evaluación utilizando un enfoque mixto cualitativo–cuantitativo, basado en dos criterios fundamentales: probabilidad de ocurrencia e impacto potencial. Para ello, se adoptó una escala ordinal de cinco niveles (Muy Bajo=1, Bajo=2, Moderado=3, Alto=4 y Muy Alto=5), asignando un valor numérico a cada categoría con el fin de permitir su tratamiento cuantitativo.

El nivel de criticidad de cada riesgo se determinó mediante el cálculo de la calificación P × I, de acuerdo con la matriz de riesgos utilizada en la industria para la gestión de incertidumbre en proyectos de software. Este método, ampliamente recomendado por marcos como el PMBOK del Project Management Institute (PMI, 2017) y alineado con los principios de la ISO 31000:2018 de Gestión del Riesgo (ISO, 2018), permite clasificar los riesgos según su severidad en niveles bajos, medios y altos, facilitando así la priorización de acciones de mitigación.

La Tabla 3.6 muestra los resultados obtenidos tras evaluar cada riesgo según su probabilidad e impacto.

Tabla 3.6: Evaluación de riesgos

ID	Probabilidad	Impacto	Calificación (P x I)	Nivel de riesgo
R1	Moderado(3)	Muy Alto(5)	15	Alto
R2	Bajo(2)	Muy Alto(5)	10	Medio
R3	Moderado(3)	Alto(4)	12	Medio
R4	Moderado(3)	Muy Alto(5)	15	Alto
R5	Alto(4)	Muy Alto(5)	20	Alto
R6	Alto(4)	Muy Alto(5)	20	Alto
R7	Moderado(3)	Alto(4)	12	Medio
R8	Moderado(3)	Alto(4)	12	Medio
R9	Moderado(3)	Alto(4)	12	Medio
R10	Moderado(3)	Muy Alto(5)	15	Alto
R11	Bajo(2)	Alto(4)	8	Medio
R12	Moderado(3)	Muy Alto(5)	15	Alto
R13	Bajo(2)	Muy Alto(5)	10	Medio
R14	Alto(4)	Alto(4)	16	Alto
R15	Moderado(3)	Muy Alto(5)	15	Alto
R16	Moderado(3)	Alto(4)	12	Medio
R17	Alto(4)	Alto(4)	16	Alto
R18	Moderado(3)	Muy Alto(5)	15	Alto
R19	Alto(4)	Moderado(3)	12	Medio
R20	Bajo(2)	Muy Alto(5)	10	Medio

3.5.3. Riesgos priorizados para el MVP

Con base en la evaluación realizada y considerando que el propósito del MVP es demostrar que la arquitectura propuesta es funcional, se seleccionaron aquellos riesgos cuyo nivel de criticidad y efecto directo podrían comprometer los objetivos propuestos en el proyecto. En la Tabla 3.7 se presentan las justificaciones correspondientes y las estrategias de mitigación planteadas para cada riesgo priorizado.

Tabla 3.7: Riesgos priorizados para el MVP

ID	Prioridad	Justificación	Estrategia de mitigación
R1	Alto	Este riesgo involucra datos personales sensibles cuya protección es una obligación legal y ética. Una filtración afecta la confianza de los usuarios, expone a la organización a sanciones y puede generar un daño reputacional irreversible. Por ello, la seguridad en el almacenamiento de la información debe ser una prioridad crítica.	Se recomienda implementar cifrado robusto en tránsito y en reposo, utilizando algoritmos y protocolos actuales que garantizan la confidencialidad de la información. Es fundamental establecer un control de acceso basado en roles (RBAC) junto con el principio de menor privilegio, asegurando que cada usuario acceda únicamente a la información estrictamente necesaria.
R4	Alto	Una mala configuración en servicios externos, como dejar un bucket S3 público, puede exponer información sensible y afectar la reputación de la plataforma, además de generar riesgos legales. También puede causar pérdida o corrupción de datos y comprometer la disponibilidad del sistema. Por ello, asegurar las configuraciones desde el inicio es esencial para evitar incidentes graves.	Para mitigar los riesgos asociados y parámetros defectuosos de artefactos que componen la arquitectura de la aplicación, se recomienda adoptar prácticas de infraestructura como código (IaC), que permitan gestionar la configuración de los recursos de forma trazable, automatizada y auditada. Estas plantillas deben estar sujetas a validaciones automáticas que aseguran configuraciones seguras desde el despliegue inicial.
R5	Alto	Este riesgo se prioriza porque los formularios son puntos vulnerables para los atacantes. Sin validaciones adecuadas, pueden inyectar código que comprometa la integridad y disponibilidad del servicio, exponiendo a los usuarios a fraudes. En una aplicación como SuricatosCare, esto tendría un impacto grave en la seguridad de los datos y la confianza de las familias. Reforzar la seguridad en los formularios es crucial para proteger la plataforma y sus usuarios.	Para mitigar el riesgo de validación deficiente en formularios, se recomienda implementar un sistema de validación robusto tanto en el cliente como en el servidor, siguiendo el principio de defensa en profundidad. Las entradas de usuario deben ser validadas con reglas estrictas de formato, longitud y tipo de datos, y posteriormente saneadas antes de ser procesadas o almacenadas.

3.5.4. Riesgos priorizados para futuros procesos de modernización y escalabilidad

Además de los riesgos críticos identificados para el MVP, se consideraron otros riesgos relevantes cuyo impacto tomará mayor importancia cuando la aplicación evolucione hacia una primera versión comercial. Estos riesgos se priorizan pensando en la futura operación del sistema en un entorno productivo, donde la estabilidad, la escalabilidad y el cumplimiento normativo son esenciales para

garantizar un funcionamiento confiable. La Tabla 3.8 presenta estos riesgos, junto con sus respectivas justificaciones y estrategias de mitigación.

Tabla 3.8: Riesgos priorizados para futuros procesos de modernización y escalabilidad

ID	Prioridad	Justificación	Estrategia de mitigación
R6	Alto	EEste riesgo se prioriza porque una gestión inadecuada de roles y permisos puede permitir accesos no autorizados a información sensible, comprometiendo la confidencialidad e integridad de los datos. Además, genera riesgos legales y daña la confianza de los usuarios. Por ello, es crucial establecer un control riguroso en los niveles de acceso para garantizar la seguridad y credibilidad de la plataforma.	Para mitigar este riesgo, se recomienda implementar un modelo RBAC bien definido y aplicado de forma consistente en toda la arquitectura. Deben documentarse los privilegios de cada rol, aplicar el principio de menor privilegio y deshabilitar permisos no necesarios. Además, es clave validar acciones sensibles con controles adicionales y realizar pruebas automatizadas de autorización para detectar brechas.
R9	Medio	Este riesgo se prioriza porque la pérdida de integridad en la comunicación entre sistemas afecta directamente la veracidad de los datos. Transacciones incompletas o alteradas generan inconsistencias, impactan procesos internos y aumentan los costos operativos al requerir correcciones. Por ello, aplicar validaciones de integridad robustas en la interoperabilidad es esencial para mantener la coherencia y estabilidad del sistema.	La mitigación consiste en aplicar validaciones de integridad en la comunicación entre sistemas, usando mecanismos como checksums, firmas o hashes antes de almacenar datos. También deben verificarse campos obligatorios y formatos esperados para evitar transacciones incompletas. Es clave incluir manejo robusto de errores, reintentos e idempotencia, junto con monitoreo y alertas que detecten anomalías. Auditorías y pruebas de resiliencia ayudan a mantener la consistencia y veracidad de la información entre sistemas.
R10	Alto	Este riesgo se prioriza porque la plataforma manejará información financiera sensible y debe cumplir estrictamente la normativa y los estándares de seguridad. No aplicar medidas como el cifrado expone a sanciones legales y afecta la confianza de los usuarios. En un servicio que depende de la credibilidad, garantizar la conformidad regulatoria es esencial para la sostenibilidad del proyecto.	Para mitigar este riesgo, se debe cumplir la normativa de protección de datos aplicando cifrado en tránsito y en reposo, usando protocolos seguros y gestionando claves adecuadamente. También es necesario restringir accesos, realizar auditorías periódicas y mantener políticas de privacidad actualizadas. La capacitación del equipo y las evaluaciones DPIA ayudan a detectar vulnerabilidades a tiempo.

Continúa en la siguiente página

Tabla 3.8 – *continuación de la página anterior*

ID	Prioridad	Justificación	Estrategia de mitigación
R13	Medio	Este riesgo se prioriza porque involucra directamente la seguridad y el bienestar de los menores, quienes son el eje central del servicio ofrecido por la plataforma. Además, es importante tener en alta consideración que cualquier conducta inapropiada por parte de un cuidador no solo puede representar un daño irreparable para el niño y su familia, sino que también compromete seriamente la credibilidad y la continuidad del servicio ofrecido por la plataforma.	Implementar sistema de evaluación post-servicio obligatoria, botón de emergencia en la app, geolocalización en tiempo real durante el servicio, y protocolo de respuesta inmediata con contacto a autoridades. Capacitación obligatoria en ética y protección infantil para cuidadores.
R14	Alto	Este riesgo se prioriza porque el sistema de reputación basado en calificaciones es un elemento esencial para que padres y madres tomen decisiones informadas y confíen en los cuidadores. La manipulación de reseñas afecta directamente la transparencia de las interacciones y distorsiona la percepción real de la calidad del servicio. Por ello, Si las calificaciones pueden alterarse, la plataforma pierde credibilidad, compromete la experiencia de los usuarios y debilita uno de los mecanismos centrales que garantizan seguridad y confianza dentro del sistema.	Implementar algoritmos de detección de patrones anómalos en calificaciones, requerir verificación de servicio completado antes de calificar, limitar una calificación por servicio real y usar CAPTCHA o verificación humana para nuevas cuentas.
R17	Alto	Este riesgo se prioriza porque, aunque la plataforma no tiene control directo sobre el estado de los dispositivos móviles de los usuarios, sí debe garantizar que, incluso en escenarios adversos, la información personal se mantenga protegida. Por lo tanto, si un atacante logra robar datos desde un dispositivo comprometido, el incidente terminaría afectando la reputación de la plataforma, ya que para el usuario promedio la diferencia entre un fallo de su dispositivo y un fallo del servicio no es evidente.	Implementar autenticación en dos factores (2FA) obligatoria, detección de dispositivos sospechosos, cifrado de datos en tránsito y en reposo, y campañas educativas para usuarios sobre seguridad en dispositivos móviles.

Capítulo 4

Evaluación

4.1. Evaluación y validación

La evaluación de la arquitectura propuesta tiene como propósito determinar si el diseño planteado basado en principios serverless, modularidad por dominios, arquitectura hexagonal y servicios administrados en AWS satisface los requisitos funcionales, no funcionales y operativos del sistema. Para ello, se aplicaron estrategias de validación estructuradas que combinan análisis cualitativo, revisión sistemática de decisiones arquitectónicas y pruebas controladas sobre el prototipo desarrollado.

4.1.1. Metodología de validación

Para garantizar una evaluación rigurosa, se aplicó una metodología basada en tres enfoques complementarios:

4.1.1.1. Evaluación basada en escenarios

Se construyeron escenarios derivados de los requisitos no funcionales del sistema escalabilidad, disponibilidad, mantenibilidad y costo; siguiendo lineamientos propuestos por Bass, Clements y Kazman en Software Architecture in Practice (Len Bass, 2012). Cada escenario fue analizado respecto a cómo la arquitectura responde ante cambios de carga, fallos, evolución funcional y restricciones de operación.

Figura 4.1: Consulta reserva. Fuente: Evidencia escenarios.

Figura 4.2: Creación reserva. Fuente: Evidencia escenarios..

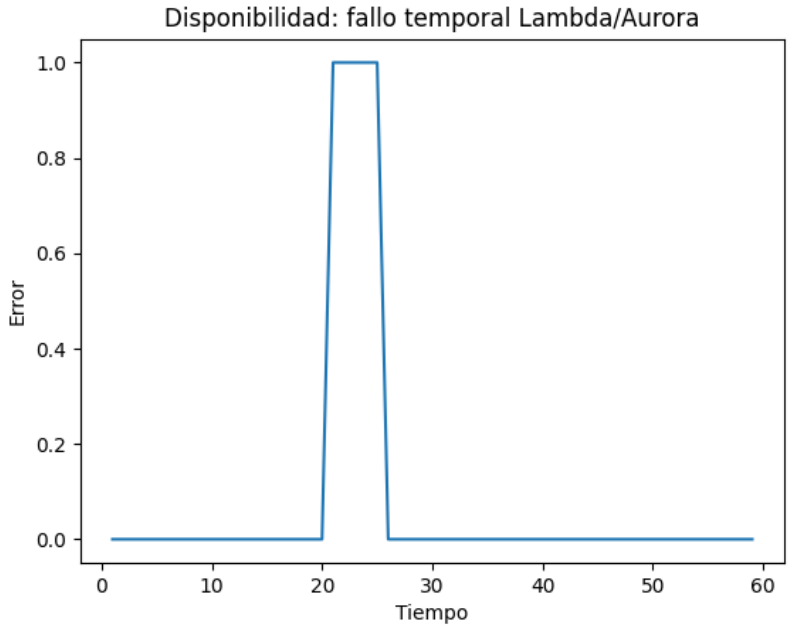


Figura 4.3: Disponibilidad sistema. Fuente: Evidencia escenarios.

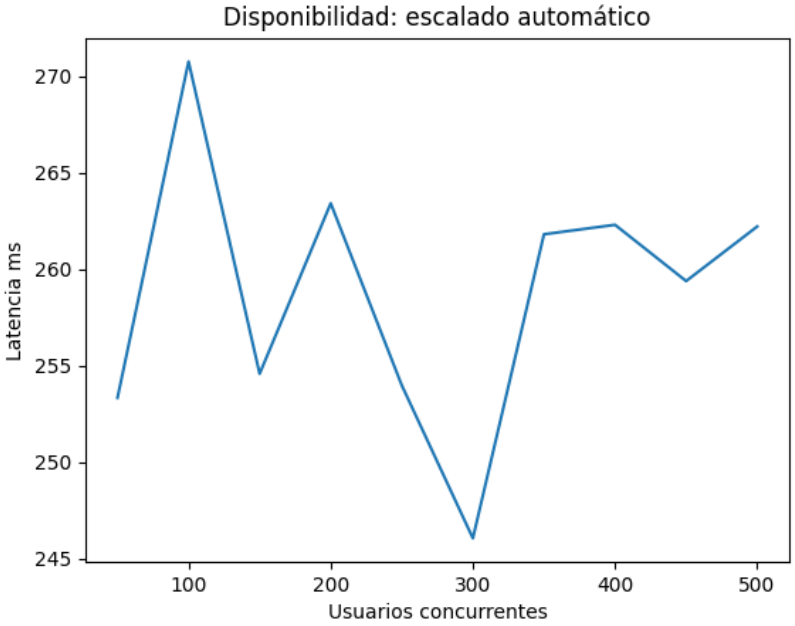


Figura 4.4: Escalabilidad. Fuente: Evidencia escenarios.

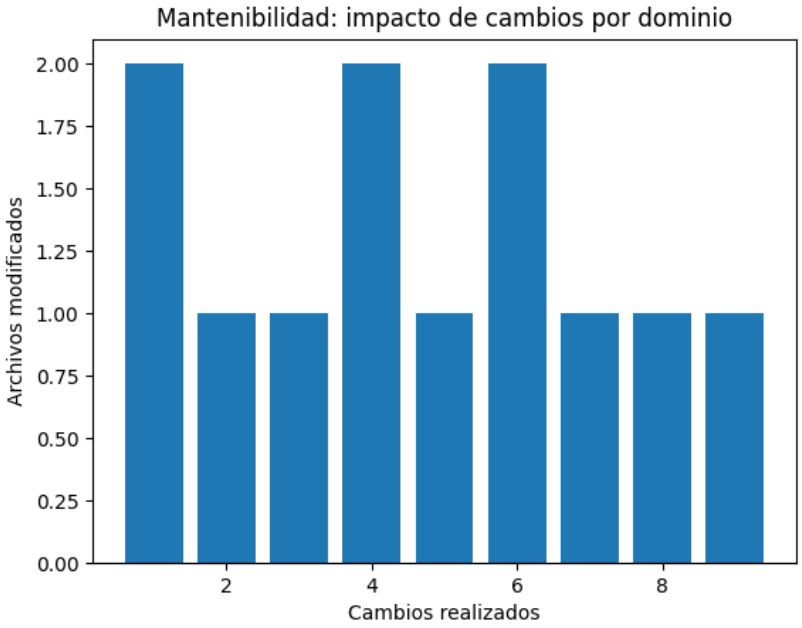


Figura 4.5: Mantenibilidad. Fuente: Evidencia escenarios.

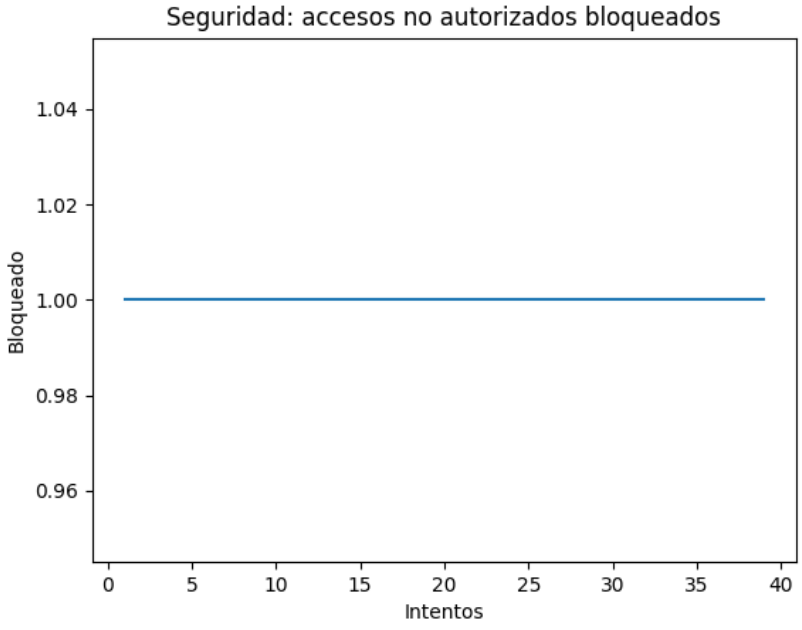


Figura 4.6: Seguridad. Fuente: Evidencia escenarios.

En la siguiente tabla se reflejan los detalles de cada escenarios de evaluación.

Tabla 4.1: Escenarios de evaluación arquitectónica

Atributo	**Fuente**	**Estímulo**	**Artefacto afectado**	**Respuesta esperada**	**Medida de respuesta**
Rendimiento	Usuario final	Solicita usuarios filtrados por rol	Lambda get-users-by-role, DynamoDB	Lista filtrada devuelta sin errores	Tiempo observado < 300 ms (pruebas manuales)
Rendimiento	Usuario final	Crea una reserva	Lambda reservation-create, Aurora MySQL	Inserción exitosa y respuesta 201	Tiempo observado < 400 ms; latencia Aurora < 50 ms
Disponibilidad	AWS (infraestructura)	Falla temporal en la Lambda o timeout en Aurora MySQL	API Gateway, Lambdas	Degradación controlada sin afectar otros dominios	Error 500 controlado; log en CloudWatch en <1 s
Disponibilidad	Sistema	Aumento moderado de solicitudes consecutivas	API Gateway, Lambdas	Lambda escala automáticamente manteniendo tiempos estables	Sin errores 429; latencia estable observada
Mantenibilidad	Equipo técnico	Cambios en lógica de un dominio	Capa de dominio, adaptadores hexagonales	Se modifica solo el dominio afectado	Máx. 2–3 archivos cambiados; sin impacto en otras Lambdas
Seguridad	Atacante externo	Intento de acceder a un endpoint sin permisos	API Gateway, IAM	Solicitud bloqueada	0 % éxito; evento registrado en CloudTrail

4.1.1.2. Validación mediante prototipo funcional

El prototipo se ejecutó en un entorno real de AWS con:

- Lambdas desplegadas para casos de uso críticos (creación de reservas, creación y consulta de usuarios, actualización de estados).
- API Gateway configurado con seguridad básica y CORS.
- DynamoDB y Aurora MYSQL para validar tiempos de respuesta, concurrencia y acceso a datos.

- CloudFront + S3 como capa cliente.

Esto permitió validar la arquitectura bajo condiciones reales de latencia, escalamiento automático y consumo de recursos.

4.1.2. Evaluación mediante AWS Well-Architected Framework

Además de la validación mediante escenarios, la arquitectura propuesta fue evaluada utilizando el *AWS Well-Architected Framework*, una guía de mejores prácticas ampliamente adoptada en la industria para el diseño, despliegue y operación de cargas de trabajo en la nube (Amazon Web Services, 2025). Este marco se compone de seis pilares fundamentales que permiten analizar la solidez técnica, operacional y económica de una arquitectura.

A continuación se describe cómo cada pilar fue aplicado al sistema desarrollado y cuáles fueron los hallazgos relevantes durante el proceso de validación.

4.1.2.1. Pilar 1: Excelencia Operacional

Este pilar evalúa la capacidad del sistema para operar correctamente, monitorear su comportamiento y evolucionar mediante procedimientos automatizados. En SuricatosCare se verificó que:

- Las funciones Lambda generan métricas de ejecución, latencia y errores mediante Amazon CloudWatch.
- La arquitectura permite incorporar rápidamente ajustes en lógica de negocio debido al desacoplamiento proporcionado por la arquitectura hexagonal.
- Los despliegues son independientes por función, lo que reduce el riesgo de fallo y fomenta la entrega continua.

La validación confirmó que el sistema facilita operaciones seguras, cumpliendo los criterios establecidos en este pilar (Bass et al., 2021).

4.1.2.2. Pilar 2: Seguridad

El pilar de seguridad evalúa mecanismos de protección de datos, gestión de identidades y aplicación del principio de mínimo privilegio. La evaluación reveló los siguientes elementos correctos:

- Control de acceso granular mediante IAM Roles específicos por Lambda.
- Acceso a la base de datos canalizado exclusivamente a través de secretos almacenados en AWS Secrets Manager.
- API Gateway configura controles CORS y actúa como primera línea de defensa ante solicitudes maliciosas.

La arquitectura aplica mecanismos modernos de seguridad en concordancia con buenas prácticas de AWS.

4.1.2.3. Pilar 3: Fiabilidad

El pilar de fiabilidad evalúa la capacidad de un sistema para recuperarse ante fallos y adaptarse a picos de demanda. La evaluación mostró que:

- AWS Lambda y API Gateway ofrecen tolerancia a fallos nativa y un modelo de ejecución distribuido.
- La base de datos utiliza Amazon Aurora MySQL en modo provisionado con un solo nodo, lo que proporciona replicación interna del almacenamiento y respaldos automáticos.
- DynamoDB replica sus datos automáticamente entre múltiples zonas de disponibilidad y ofrece alta durabilidad.

En conjunto, la arquitectura garantiza fiabilidad adecuada para un MVP, con una ruta clara de mejora hacia configuraciones Multi-AZ o Aurora Serverless v2.

4.1.2.4. Pilar 4: Eficiencia del Rendimiento

Este pilar examina la eficiencia en el uso de recursos computacionales y la capacidad de la arquitectura para ajustarse dinámicamente. Durante la validación se identificó que:

- Lambda escala automáticamente para adaptarse a picos de demanda sin necesidad de aprovisionamiento manual.
- DynamoDB ajusta su capacidad en función de la carga.
- Aurora provisionado ofrece un rendimiento estable y adecuado para el volumen transaccional del MVP.

La observación de métricas en CloudWatch durante invocaciones reales mostró tiempos de respuesta consistentes y sin saturación de recursos.

4.1.2.5. Pilar 5: Optimización de Costos

Siguiendo los lineamientos FinOps, se evaluó el uso eficiente de recursos y la reducción de costos innecesarios:

- El modelo *serverless* evita costos de servidores en estado inactivo.
- DynamoDB adopta un modelo de pago por solicitud.
- Aurora provisionado mantiene costos controlados durante la etapa de MVP.

La evaluación determinó que esta configuración es adecuada para un prototipo funcional, con posibilidad de optimizar costos migrando a un modelo auto escalable en fases posteriores.

4.1.2.6. Pilar 6: Sostenibilidad

Este pilar examina el impacto ambiental y la eficiencia energética del sistema. La arquitectura serverless cumple con este pilar al:

- Reducir el desperdicio computacional al utilizar recursos bajo demanda.
- Minimizar tiempos de ejecución y capacidad ociosa.
- Aprovechar centros de datos optimizados energéticamente por AWS.

4.1.2.7. Conclusión de la evaluación Well-Architected

La evaluación detallada mediante los seis pilares confirma que la arquitectura propuesta:

- Es técnicamente sólida.
- Escala adecuadamente dentro de los requisitos del MVP.
- Gestiona eficientemente costos en el contexto actual.
- Mantiene un alto nivel de seguridad y adecuadas garantías de fiabilidad.

Los resultados validan que la solución es viable para un despliegue inicial y cuenta con una ruta clara de evolución hacia entornos de mayor carga o criticidad.

4.1.3. Resultados de la Evaluación

Los resultados obtenidos muestran que la arquitectura es adecuada para los objetivos del proyecto y se ajusta a los requisitos técnicos identificados. La validación se basó en la observación de métricas en tiempo real, pruebas funcionales manuales y la revisión de los pilares del AWS Well-Architected Framework.

4.1.3.1. 2.1 Rendimiento y escalabilidad

Las funciones Lambda presentaron tiempos de ejecución observados entre 80 ms y 250 ms durante las pruebas manuales, dependiendo del caso de uso y del volumen de datos procesado.

La base de datos utiliza Amazon Aurora MySQL en modo provisionado (un solo nodo), ofreciendo tiempos de respuesta consistentes y respaldos automáticos, aunque sin capacidades de autoscalado propias de Aurora Serverless v2.

DynamoDB respondió con latencias inferiores a 20 ms según métricas de CloudWatch, confirmando su idoneidad para operaciones de lectura ligera y consultas directas desde la arquitectura.

La arquitectura basada en Lambda y API Gateway soporta incrementos moderados de demanda sin necesidad de reconfiguración manual, conforme al comportamiento esperado de los servicios serverless.

4.1.3.2. 2.2 Mantenibilidad

La separación en dominios y el uso de arquitectura hexagonal permitieron aislar la lógica de negocio, facilitando modificaciones sin impacto en infraestructura.

La reutilización de componentes como la fábrica de conexiones RDS y el adaptador DynamoDB redujo la duplicación de código.

Las Lambdas se mantuvieron pequeñas, específicas y fácilmente verificables mediante pruebas manuales.

4.1.3.3. 2.3 Robustez y resiliencia

El desacoplamiento entre componentes evita fallos en cascada y facilita la recuperación ante errores.

La combinación de API Gateway + Lambda ofrece alta disponibilidad nativa al operar sobre múltiples zonas de disponibilidad.

Secrets Manager asegura el manejo seguro de credenciales de acceso a los servicios.

Aunque el clúster de Aurora es de un solo nodo, ofrece almacenamiento replicado y respaldos automáticos, brindando un nivel adecuado de resiliencia para un MVP.

4.1.3.4. 2.4 Costos operativos

El modelo de pago por uso de los servicios serverless permitió costos muy bajos.

Aurora en modo provisionado mantiene un costo estable y adecuado para la etapa de MVP, evitando complejidades de configuración.

No se requiere administrar servidores ni infraestructura subyacente, reduciendo la carga operativa.

Estos resultados verifican que la arquitectura cumple con los lineamientos FinOps definidos y opera de forma eficiente dentro del alcance del prototipo.

4.1.4. Instrumentos Aplicados

- Logs en CloudWatch para trazabilidad de ejecución.
- Dashboards nativos de métricas Lambda.
- Amazon RDS Performance Insights para consultas SQL.
- Métricas de DynamoDB para throughput, latencia y consumo.
- Checklist del AWS Well-Architected Framework (Serverless Lens).
- Verificación de principios hexagonales mediante revisión cruzada de módulos.
- Evaluación del desacoplamiento usando diagramas C4 y análisis de interfaces.
- Pruebas manuales con Postman para rutas críticas.

- Pruebas de integración desde el frontend (React + CloudFront).
- Validación de flujos completos: registro → autenticación → creación de reserva → consulta.

4.1.5. Conclusión sobre Viabilidad Técnica

Los resultados de la evaluación demuestran que la arquitectura propuesta es técnicamente viable, escalable y alineada con buenas prácticas. La combinación de componentes serverless con una estructura modular y hexagonal soporta adecuadamente los requerimientos de un sistema de intermediación de servicios.

La solución:

- Cumple con los requisitos funcionales y no funcionales definidos.
- Escala automáticamente en sus componentes serverless, evitando sobre aprovisionamiento.
- Es mantenible y adaptable gracias a su diseño por dominios y arquitectura hexagonal.
- Mantiene mecanismos sólidos de seguridad mediante IAM, Secrets Manager y aislamiento de componentes.
- Ofrece resiliencia adecuada para un MVP mediante servicios gestionados.

En conjunto, los resultados validan que la arquitectura diseñada es adecuada para una primera versión (MVP) y constituye una base sólida para futuras extensiones y funcionalidades de mayor complejidad.

Capítulo 5

Conclusiones

5.1. Conclusiones

En el proyecto se logró diseñar e implementar una arquitectura modular, escalable y segura que facilita el acceso de familias a profesionales capacitados en atención a la primera infancia. La validación del MVP demostró que la solución permite la integración efectiva entre componentes, responde adecuadamente a escenarios de carga básica y protege datos sensibles mediante servicios especializados como Amazon Cognito. Esto confirma que la arquitectura propuesta no solo es técnicamente viable, sino que también está alineada con los atributos de calidad definidos en el proyecto.

El alcance geográfico limitado a familias de Bogotá representó una restricción importante para la generalización de los hallazgos, dado que no se incluyeron otras regiones con dinámicas sociales y tecnológicas distintas.

La validación del MVP se enfocó en flujos funcionales esenciales, sin incorporar escenarios críticos como concurrencia intensiva, interrupciones de conectividad o ciberataques, lo que restringe la evaluación completa de atributos como tolerancia a fallos y resiliencia operativa.

El análisis realizado evidenció una brecha significativa entre la demanda de servicios de cuidado infantil y la oferta actual, tanto pública como privada. En respuesta, se diseñó una solución técnica orientada a cubrir esta necesidad concreta, basada en principios arquitectónicos contemporáneos como el modelo C4, y una implementación sustentada en servicios administrados en la nube, que garantizan sostenibilidad tecnológica, escalabilidad y reducción de complejidad operativa. Esta aproximación evidencia cómo la ingeniería de software puede responder a desafíos sociales complejos desde una perspectiva técnica rigurosa y estratégicamente alineada con arquitecturas modernas orientadas a calidad.

La adopción del enfoque Serverless First permitió reducir la carga operativa y garantizar una escalabilidad automática ajustada al comportamiento real del sistema, validando que este paradigma es adecuado para soluciones de cuidado infantil con alta variabilidad en la demanda.

La organización del backend en funciones Lambda independientes demostró que una arquitectura modular facilita la mantenibilidad, la evolución del sistema y la incorporación de nuevos casos de uso sin afectar los componentes existentes.

El uso de servicios administrados de AWS como DynamoDB, RDS, IAM y CloudFront evidenció que es posible construir una plataforma robusta con un esfuerzo reducido en infraestructura, permitiendo concentrar los recursos del proyecto en lógica de negocio y experiencia del usuario.

La arquitectura propuesta permite incorporar en el futuro funcionalidades adicionales como pagos en línea, analítica de uso o matching inteligente, sin modificar la base del sistema, lo que evidencia su capacidad de crecimiento sostenible.

5.2. Trabajos futuros

El desarrollo de esta arquitectura y su validación inicial mediante el MVP abre múltiples líneas de trabajo para fortalecer y ampliar la solución propuesta. A continuación, se describen algunas de las proyecciones más relevantes:

- Integrar la plataforma con fuentes de información oficiales, como las bases de datos del ICBF, el SISBEN, u otras entidades territoriales, permitiría automatizar y fortalecer procesos clave como la validación de antecedentes de los cuidadores o la priorización de servicios según criterios sociales. A nivel arquitectónico implica diseñar mecanismos seguros para el consumo de APIs externas, definir políticas de acceso basadas en OAuth 2.0 o esquemas similares, y estructurar los datos de una forma que sea compatible con los estándares gubernamentales. Incorporar esta capacidad de interoperabilidad no solo mejora la eficiencia del sistema, también refuerza la confianza institucional y abre la posibilidad de implementar esquemas de cofinanciación o subsidios que aumenten el impacto social de la plataforma.
- La aplicación de inteligencia artificial en la plataforma constituye una oportunidad estratégica para enriquecer su funcionalidad y capacidad adaptativa. A partir de técnicas de aprendizaje automático, tanto supervisado como no supervisado, es factible desarrollar mecanismos de recomendación que asignen cuidadores en función del comportamiento histórico de los usuarios, sus preferencias declaradas y patrones de uso detectados. Este enfoque también permitiría incorporar esquemas de incentivos dinámicos, como descuentos basados en recurrencia, fidelidad o volumen de servicios solicitados. Adicionalmente, al considerar variables de ubicación, es viable optimizar la asignación de cuidadores según criterios de cercanía, reduciendo así tiempos de desplazamiento y costos operativos. En conjunto, estas capacidades no solo mejoran la eficiencia del sistema, sino que enriquecen la experiencia del usuario, fortaleciendo su retención y aumentando la confianza en la plataforma.

5.3. Lecciones aprendidas

A lo largo del desarrollo de este proyecto surgieron aprendizajes que van más allá de lo técnico. Comprendimos que en el diseño arquitectónico, no hay decisiones triviales: elegir entre servicios como Amazon DynamoDB, Aurora o RDS puede depender de una diferencia mínima, como la latencia de escritura, el formato de consistencia eventual o el soporte transaccional, que solo emergen cuando los requisitos están bien definidos y alineados con la naturaleza del problema. Descubrimos también que los atributos de calidad, como la escalabilidad o la seguridad, no son conceptos abstractos, sino resultados de decisiones concretas trazadas desde las primeras iteraciones del proyecto. La necesidad de enfrentar dilemas técnicos reales nos llevó a reconocer el valor de documentar no solo el ”qué”, sino también el ”por qué”detrás de cada trade-off. Finalmente, validar una solución técnica sin involucrar usuarios reales deja vacíos difíciles de llenar. Aunque se logró demostrar la viabilidad técnica del sistema, es necesario reconocer que la verdadera validación ocurre cuando el software se enfrenta a la diversidad, la imprevisibilidad y las prioridades de quienes lo van a usar.

Bibliografía

Amazon Web Services (2024). Overview of amazon web services. https://docs.aws.amazon.com/whitepapers/latest/aws-overview/. Accessed: 2025-10-14.

Amazon Web Services, I. (2024). aws.amazon.com. Accessed: 2025-10-14.

Amazon Web Services, I. (2025). aws.amazon.com. Accessed: 2025-10-14.

Araujo, María Caridad, . L.-B. (2015). Los servicios de cuidado infantil en américa latina y el caribe. *Revista de Estudios Sociales*, 82(326):249–275.

Bass, L., Clements, P., and Kazman, R. (2021). *Software Architecture in Practice*. Addison-Wesley Professional, 4th edition.

Blanchard, O. (2024). Plataformas digitales de cuidados y de servicio doméstico en américa latina y el caribe: un análisis inicial de sus modelos de negocio y su rol en la formalización del sector. *Revista de Estudios Sociales*, 89:143–157.

Boehm, B., Lane, J. A., Koolmanojwong, S., and Turner, R. (2014). *The Incremental Commitment Spiral Model: Principles and Practices for Successful Systems and Software*. Addison-Wesley Professional, Boston.

CEPAL (2016). Políticas de cuidado en américa latina: forjando la igualdad. Accedido: 2025-08-27.

Colombia, N. (2025). Agencia de cuidado infantil y acompañamiento especializado. Accessed: 2025-10-14.

DANE (2022). Análisis de accesibilidad a centros educativos. Accedido: 2025-08-27.

Funnana (2025). Servicios profesionales de niñeras y cuidado infantil. Accessed: 2025-10-14.

Glen, S. (s.f.). Finite population correction factor (fpc): Formula, examples. https://www.statisticshowto.com/finite-population-correction-factor/. Consultado noviembre de 2025.

ICBF (2018). PolÍtica nacional de infancia y adolescencia. Accessed: 2025-03-11.

ICBF (2021). Modalidades de atención. Accedido: 2025-03-06.

ISO (2018). Iso 31000:2018 risk management — guidelines.

ISO25000 (s.f.). Normas iso 25000. Accessed: 2025-03-31.

Karl Wiegers, J. B. (2013). *Software Requirements, 3rd Edition*. Microsoft Press.

Lanman, J., Darbin, R., Rivera, J., Clements, P., and Krueger, C. W. (2013). The challenges of applying service orientation to the U.S. Army's live training software product line. In *Proceedings of the 17th International Software Product Line Conference on - SPLC '13*, page 244, New York, New York, USA. ACM Press.

Len Bass, Paul Clements, R. K. (2012). *Software Architecture in Practice, Third Edition*. Addison-Wesley Professional.

Len Bass, Paul Clements, R. K. (2021). *Software Architecture in Practice, 4th Edition*. Addison-Wesley Professional.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):1–55.

Mark Richards, N. F. (2025). *Fundamentals of Software Architecture, 2nd Edition*. O'Reilly Media, Inc.

Medium (2019). Software development process models. Accessed: 2025-03-31.

Mell, P. and Grance, T. (2011). The nist definition of cloud computing. Technical Report Special Publication 800-145, National Institute of Standards and Technology (NIST).

Model, C. (s.f.). c4model.com. Accessed: 2025-03-17.

Niñeras, A. (2025). Niñeras a domicilio y estimulación temprana. Accessed: 2025-10-14.

ONUMujeres (2024). Cuidados y sector empresarial oportunidades para el desarrollo con igualdad en américa latina. Accedido: 2025-08-27.

PMI (2017). *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. Project Management Institute, Newtown Square, PA, 6th edition edition.

Richard N. Taylor, Nenad Medvidovic, E. M. D. (2009). *Software Architecture: Foundations, Theory, and Practice*. Wiley.

Salant, P. and Dillman, D. A. (1994). *How to Conduct Your Own Survey*. John Wiley & Sons, New York.

Sampieri, R. H., Collado, C. F., and Lucio, M. d. P. B. (2014). *Metodología de la investigación*. McGraw-Hill Education / Interamerica Editores, S.A DE C.V., México, D.F., 6a edition.

Sitly (2025). Plataforma de conexión entre padres y niñeras. Accessed: 2025-10-14.

Sommerville, I. (2016). *Software Engineering*. Pearson Education Limited, Harlow, England, 10th edition.

UNESCO (2022). Por qué es importante la atención y educación de la primera infancia. Accedido: 2025-03-06.

UNICEF (s.f.). Primera infancia. Accedido: 2025-03-06.

Varma, V. (2009a). *Software Architecture: A Case Based Approach*. Pearson India.

Varma, V. (2009b). *Software Architecture: A Case Based Approach*. Pearson India.

Wiegers, K. and Beatty, J. (2013). *Software Requirements*. Microsoft Press, Redmond, WA, 3rd edition.

Apéndice A

Anexos

A.0.1. Encuesta aplicada

El informe completo de la encuesta realizada a padres y cuidadores está disponible en el siguiente enlace:

Enlace: https://goo.su/60MMYRx

Descripción: Informe en PDF exportado desde Google Forms con resultados, métricas y visualizaciones de la encuesta, asi como el archivo Excel con los datos completos.

A.0.2. Matriz de riesgos

La matriz de riesgos utilizada en el proyecto se encuentra disponible en:

Enlace: https://goo.su/9aOfue

Descripción: Archivo en Excel con la identificación, valoración y priorización de los riesgos del sistema.

A.0.3. Diseño de interfaz en Figma

El prototipo de interfaz elaborado en Figma puede consultarse en:

Enlace: https://goo.su/1V5WnY

Descripción: Enlace al prototipo navegable y al diseño de alta fidelidad de la aplicación.

A.0.4. Diagramas arquitectónicos

Los diagramas de arquitectura, C4 y flujos utilizados en el proyecto se encuentran disponibles en:

Enlace: https://goo.su/nUpwVO6

Descripción: Archivos en formato svg utilizados para modelar el sistema.

A.0.5. Registros de decisiones arquitectónicas (ADR)

Los registros de decisiones arquitectónicas utilizados durante el desarrollo del proyecto se encuentran disponibles en el siguiente enlace:

Enlace: https://goo.su/5BWt9h

Descripción: Documento ADR que recopila las decisiones clave de arquitectura tomadas a lo largo del proyecto, incluyendo su contexto, las alternativas evaluadas, la decisión seleccionada y sus consecuencias técnicas.

A.0.6. Tablero de seguimiento del proyecto

El tablero Trello con la planificación del proyecto se encuentra en:

Enlace: https://goo.su/zMyk

Descripción: Vista del tablero Kanban empleado para organizar tareas, avances y entregas.

A.0.7. Colección de Postman

La colección de endpoints probados desde Postman está disponible en:

Enlace: https://n9.cl/ykpjw

Descripción: Archivo JSON exportado desde Postman con todas las rutas del backend.

A.0.8. Video demostrativo del MVP

El video demostrativo del funcionamiento del MVP puede verse en:

Enlace: https://n9.cl/skkcx

Descripción: Video demostrativo del uso del sistema, flujos principales y funcionalidades implementadas.

A.0.9. Acceso a la aplicación desplegada

La versión desplegada del MVP se encuentra disponible en el siguiente enlace:

Enlace: https://n9.cl/4uh5w

Descripción: Dirección pública del frontend desplegado para pruebas funcionales.