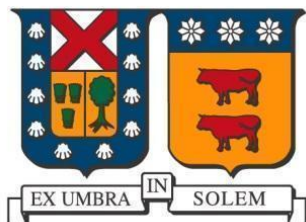

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA E INFORMÁTICA
CONCEPCIÓN - CHILE



**DESARROLLO DE
UN SISTEMA MÓVIL
DE REPORTES GEOLOCALIZADOS
Y REDES COMUNITARIAS CON FLUTTER,
API PHP Y AWS RDS**

IGNACIO CÁCERES VÁSQUEZ

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

Profesor Guía: Javier Maldonado Carmona
Septiembre - 2026



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: DESARROLLO DE UN SISTEMA MÓVIL DE REPORTES GEOLOCALIZADOS Y REDES COMUNITARIAS CON FLUTTER, API PHP Y AWS RDS.

Nombre del candidato(a): Ignacio Israel Cáceres Vásquez

Carrera / Grado: Ingeniería en Informática

Campus: Concepción **Departamento:** Electrónica e Informática.

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Javier Maldonado Carmona, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- ☒ He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- ☒ El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

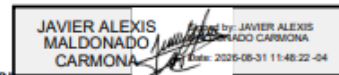
☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 31/08/2026 **Firma:**



Estudiante o Candidato(a):

Fecha: 14/08/2026 **Firma:**

DEDICATORIA

Dedico esta memoria a mi familia y mi pareja, quienes fueron parte fundamental para poder desarrollarme como persona a lo largo de mi vida y por estar en este proceso tan importante de mi vida, muchas gracias.

AGRADECIMIENTOS

Debo extender mis agradecimientos a personas muy importantes, quienes, sin su apoyo, no sería quien soy ahora. Comenzando por mi padre, quien me motivó a estudiar esta carrera que desconocía, ayudándome junto a mi madre a tomar esta trascendental decisión. Quiero destacar a mi madre por estar siempre a mi lado en los momentos en que más la necesité, y expresar mi gratitud a mi abuela, quien fue un apoyo constante durante mis años de estudio, junto a mis hermanas, que también forman una parte esencial de mi familia.

Además, me gustaría mencionar a quien se convirtió en una de las personas más significativas en mi vida: mi pareja. Ella estuvo presente tanto en los momentos felices como en los difíciles, siendo alguien fundamental en todo este proceso, gracias por quererme. Por último, no puedo dejar de reconocer a los compañeros que conocí durante esta etapa universitaria. Más que compañeros, se transformaron en grandes amigos, personas que tomaron un rol clave en este camino, tanto en lo académico como en lo personal, haciendo de esta experiencia un recorrido lleno de risas y buenos momentos.

Gracias a todas estas personas por ser parte de mi vida y por acompañarme en este tramo tan importante para mí.

RESUMEN

VisionSecurity es una plataforma móvil orientada a apoyar el registro y consulta de reportes geolocalizados, junto con funciones de interacción entre usuarios mediante solicitudes de amistad y comunidades. El propósito general de la aplicación es facilitar la organización de información ciudadana asociada a incidentes de seguridad y permitir su visualización desde un entorno móvil.

Esta memoria se centra en la capa backend e integración cliente-servidor del sistema. El trabajo comprende la implementación de endpoints PHP, la conexión con una base de datos MySQL alojada en AWS RDS, el intercambio de información mediante solicitudes HTTP y respuestas JSON, y la persistencia de datos asociados a usuarios, reportes, comunidades, solicitudes de amistad y ubicaciones.

La construcción de la solución se organizó mediante etapas de configuración de infraestructura, diseño de base de datos, desarrollo de endpoints, integración con la aplicación Flutter y verificación funcional de los módulos principales. La aplicación móvil se considera en esta memoria como cliente del backend, ya que origina las solicitudes y consume las respuestas devueltas por los servicios PHP.

La validación se realizó mediante trazabilidad técnica entre pantallas, endpoints, tablas y respuestas JSON, además de pruebas funcionales por módulos y pruebas de carga documentadas con Apache JMeter sobre flujos críticos del backend. Los resultados permitieron comprobar el funcionamiento del prototipo bajo condiciones controladas y observar limitaciones cuando aumenta la concurrencia de usuarios.

Como resultado del trabajo realizado, se obtuvo un prototipo funcional capaz de comunicar la aplicación móvil con el backend desarrollado en PHP y la base de datos MySQL. Sin embargo, al tratarse de una solución construida en un contexto académico, todavía existen aspectos que deben fortalecerse antes de pensar en una implementación productiva, especialmente en monitoreo, seguridad, optimización de consultas y estabilidad de la infraestructura.

GLOSARIO

AlertDialog: Componente de Flutter utilizado para mostrar ventanas emergentes de confirmación, advertencia o ingreso de datos.

Anonimato en reportes: Función que permite ocultar públicamente la identidad del autor de un reporte, manteniendo internamente su asociación para fines de trazabilidad.

API: Conjunto de reglas y operaciones que permite la comunicación entre componentes de software.

API REST / RESTful API: Interfaz de servicios web basada en recursos y métodos HTTP, utilizada para comunicar la aplicación móvil con el backend.

Autenticación: Proceso de comprobación de la identidad de un usuario mediante credenciales registradas.

AWS EC2: Servicio de Amazon Web Services utilizado para ejecutar el servidor virtual que alojó el backend PHP.

AWS RDS: Servicio administrado de Amazon Web Services utilizado para alojar la base de datos relacional MySQL.

Backend: Capa interna del sistema encargada de recibir solicitudes, ejecutar lógica de servidor e interactuar con la base de datos.

Base de datos relacional: Sistema que organiza la información en tablas relacionadas mediante claves e identificadores.

Comunidades: Módulo que permite crear grupos de usuarios y asociar integrantes mediante relaciones almacenadas en la base de datos.

Consultas parametrizadas: Técnica que separa la sentencia SQL de los datos de entrada mediante parámetros vinculados, reduciendo el riesgo de inyección SQL.

Deserialización: Conversión de una respuesta JSON en estructuras que pueden ser utilizadas por la aplicación Flutter.

Endpoints: Rutas del backend que reciben solicitudes HTTP y ejecutan operaciones específicas del sistema.

Flutter: Framework utilizado para desarrollar la aplicación móvil que actúa como cliente del backend.

Flutter Widgets: Componentes visuales y de interacción que conforman las pantallas de una aplicación Flutter.

Gestión de amigos: Módulo de backend que administra el envío, consulta, aceptación o eliminación de solicitudes de amistad.

Gestión de ubicación: Proceso de registro y consulta de coordenadas geográficas asociadas a usuarios o reportes.

Hash de contraseña (bcrypt): Transformación unidireccional aplicada a las contraseñas antes de su almacenamiento, evitando conservarlas en texto plano.

HTTP GET: Método HTTP utilizado principalmente para consultar información desde el servidor.

HTTP POST: Método HTTP utilizado para enviar datos desde la aplicación hacia el servidor.

JSON: Formato liviano de intercambio de datos utilizado en las respuestas entre el backend y la aplicación móvil.

Mapbox API: Servicio utilizado por la aplicación móvil para representar mapas y datos geográficos.

Multipart Request: Tipo de solicitud HTTP que permite enviar campos de datos junto con archivos, como imágenes.

MySQL: Sistema de gestión de bases de datos relacionales utilizado para persistir la información del sistema.

Paquete HTTP (Flutter): Biblioteca utilizada para ejecutar solicitudes GET y POST desde la aplicación móvil.

PHP: Lenguaje empleado para implementar los endpoints y la lógica de servidor del backend.

Pruebas de backend: Comprobaciones funcionales, de integración, recuperación y carga aplicadas a los servicios del servidor.

Reportes geolocalizados: Registros de incidentes asociados a coordenadas geográficas y datos descriptivos.

Seguridad: Conjunto de prácticas orientadas a proteger credenciales, entradas, comunicaciones y operaciones sobre la base de datos.

SQL: Lenguaje utilizado para crear, consultar y modificar datos en una base de datos relacional.

TileLayer (Mapbox): Capa utilizada por la biblioteca de mapas para representar mosaicos cartográficos dentro de Flutter.

Validación de credenciales: Comprobación de las credenciales ingresadas frente al hash almacenado para autorizar o rechazar el inicio de sesión.

ÍNDICE DE CONTENIDOS

RESUMEN	4
GLOSARIO	6
ÍNDICE DE FIGURAS	10
ÍNDICE DE TABLAS	11
INTRODUCCIÓN	12
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	13
1.1 Situación actual	13
1.2 Problemática	13
1.3 Objetivo general	14
1.4 Objetivos específicos	14
CAPÍTULO 2: MARCO CONCEPTUAL	15
2.1 Conceptos técnicos	15
2.1.1 Introducción a la solución técnica	15
2.1.2 Arquitectura del sistema	15
2.1.3 Base de datos (MySQL)	17
2.1.4 Desarrollo de la aplicación móvil	17
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	18
3.1 Enfoque técnico de la solución	18
3.2 Metodología de construcción aplicada	19
3.3 Configuración de infraestructura backend y base de datos	20
3.4 Diseño de base de datos y uso del id_Usuario	21
3.5 Construcción del backend PHP	23
3.6 Integración cliente-servidor mediante HTTP/JSON	24
3.7 Inventario de endpoints utilizados por los módulos principales	24
3.8 Relación entre pantallas, endpoints y base de datos	26
3.9 Validación funcional por módulos dentro de la metodología	27
3.10 Sistema de reportes geolocalizados	28
3.11 Comunidades y solicitudes de amistad	28
3.12 Síntesis de la propuesta de solución	29
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN	30
4.1 Enfoque de validación aplicado	30
4.2 Evidencia visual del prototipo y componentes utilizados en la validación	30
4.3 Validación funcional de los módulos principales	31

4.4 Validación de registro e inicio de sesión	32
4.4.1 Comprobación de consultas parametrizadas y hash de contraseñas	34
4.5 Validación del sistema de reportes geolocalizados	35
4.6 Validación de consulta y visualización de reportes	37
4.7 Validación de solicitudes de amistad	38
4.8 Validación de comunidades y asociación de miembros	40
4.9 Comprobación de integración entre aplicación, backend y base de datos	42
4.10 Validación de respuesta ante errores y recuperación del servicio	43
4.11 Pruebas de carga del backend	44
4.12 Síntesis de la validación	46
CAPÍTULO 5: CONCLUSIONES	47
REFERENCIAS BIBLIOGRÁFICAS	49

ÍNDICE DE FIGURAS

Figura 2.1. Arquitectura general de la aplicación.	16
Figura 3.1. Preparación y configuración de la infraestructura en AWS: EC2, RDS, VPC y grupos de seguridad.	19
Figura 3.2. Modelo relacional de base de datos utilizado en VisionSecurity.	22
Figura 3.3. Diagrama de uso del id_Usuario.	23
Figura 3.4. Flujo de comunicación del backend.	24
Figura 4.1. Criterio general de validación utilizado en los módulos principales.	30
Figura 4.2. Evidencia visual de pantallas vinculadas a los flujos validados: login, registro, mapa y reportes.	31
Figura 4.3. Flujo validado para registro e inicio de sesión.	32
Figura 4.4. Recorrido técnico validado para la creación y consulta de reportes.	35
Figura 4.5. Flujo de envío de reporte desde formulario hacia backend.	36
Figura 4.6. Flujo general de comunicación cliente-servidor utilizado para consultar datos.	38
Figura 4.7. Relación entre solicitudes de amistad, comunidades y tablas asociadas.	39
Figura 4.8. Flujo detallado de consulta, aceptación y eliminación de solicitudes.	39
Figura 4.9. Flujo de creación, consulta y asociación de miembros a comunidades.	41
Figura 4.10. Tiempo de respuesta en función del número de usuarios simultáneos.	44
Figura 4.11. Tasa de error observada durante la prueba de carga.	45
Figura 4.12. Uso de CPU y RAM del servidor durante la prueba de carga.	45

ÍNDICE DE TABLAS

Tabla 2.1. Componentes técnicos utilizados en la solución.	15
Tabla 3.1. Metodología de construcción aplicada.	19
Tabla 3.2. Endpoints de autenticación, reportes y consulta de datos.	24
Tabla 3.3. Endpoints de solicitudes de amistad y comunidades.	25
Tabla 3.4. Endpoints de ubicación de usuarios y miembros.	25
Tabla 3.5. Relación entre pantallas, endpoints y base de datos.	26
Tabla 3.6. Validación funcional por módulos dentro de la metodología.	27
Tabla 4.1. Validación funcional de los módulos principales.	31
Tabla 4.2. Validación de registro e inicio de sesión.	34
Tabla 4.3. Comprobación de consultas parametrizadas y hash de contraseñas.	35
Tabla 4.4. Validación del sistema de reportes geolocalizados.	37
Tabla 4.5. Validación de consulta y visualización de reportes.	38
Tabla 4.6. Validación de solicitudes de amistad.	39
Tabla 4.7. Validación de comunidades y asociación de miembros.	42
Tabla 4.8. Comprobación de integración entre aplicación, backend y base de datos.	42
Tabla 4.9. Validación de respuesta ante errores y recuperación del servicio.	43
Tabla 4.10. Síntesis de métricas de pruebas de carga.	46

INTRODUCCIÓN

La solución técnica desarrollada en esta memoria surge frente a la necesidad de contar con una plataforma que permita registrar y consultar reportes geolocalizados, además de habilitar la interacción entre usuarios mediante comunidades y solicitudes de amistad. En este contexto, VisionSecurity se plantea como un prototipo móvil orientado a organizar información ciudadana asociada a incidentes de seguridad en la Región del Biobío.

Esta memoria aborda la capa backend e integración cliente-servidor del proyecto VisionSecurity. Las funcionalidades de interfaz móvil desarrolladas en Flutter se consideran como cliente de consumo para validar los flujos, mientras que otros componentes visuales o de experiencia de usuario del proyecto grupal quedan fuera del foco principal de este documento.

El proyecto contempla una aplicación móvil construida en Flutter, utilizada como cliente del sistema. Desde esta aplicación se originan acciones como registro, inicio de sesión, creación de reportes, consulta de información y gestión de vínculos entre usuarios. En esta memoria, Flutter se aborda principalmente como el componente que consume los servicios del backend y permite validar la integración cliente-servidor mediante solicitudes HTTP y respuestas JSON.

La lógica de negocio se implementó mediante un backend PHP encargado de recibir solicitudes, procesar parámetros, ejecutar consultas sobre la base de datos y devolver respuestas estructuradas. Este esquema se relaciona con el modelo cliente-servidor utilizado en servicios web basados en HTTP, descrito por Richardson, Amundsen y Ruby (2013), donde el cliente solicita recursos y el servidor responde con representaciones de datos interpretables por la aplicación.

La información del sistema se almacenó en una base de datos relacional MySQL alojada en AWS RDS. Esta capa permitió organizar entidades como usuarios, reportes, comunidades, solicitudes de amistad, miembros de comunidad y ubicaciones. La infraestructura se complementó con una instancia EC2 para alojar los scripts PHP, separando la capa de presentación, la lógica de servidor y la persistencia de datos.

La seguridad se abordó mediante mecanismos aplicados al backend y a la base de datos, principalmente el almacenamiento de contraseñas mediante bcrypt y el uso de consultas SQL parametrizadas. Estas decisiones buscan reducir riesgos frecuentes en aplicaciones web, como el almacenamiento inseguro de credenciales y la inyección SQL, considerando las recomendaciones generales de OWASP (2023). Su incorporación se documenta posteriormente mediante evidencia de código en el flujo de registro e inicio de sesión, sin interpretar dicha comprobación como una auditoría formal de seguridad.

Finalmente, el comportamiento del backend fue evaluado mediante validaciones funcionales por módulo, comprobaciones de integración, recuperación ante indisponibilidad temporal del servicio y pruebas de carga con Apache JMeter. Estas pruebas no buscan afirmar preparación para producción masiva, sino documentar el funcionamiento del prototipo, sus límites bajo concurrencia y las mejoras necesarias para una evolución futura.

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA

1.1 1.1 Situación actual

En la actualidad, los ciudadanos y ciudadanas enfrentan diversas limitaciones relacionadas con la falta de una plataforma centralizada y accesible para reportar incidentes de seguridad. Uno de los principales desafíos radica en la dispersión de la información, los datos sobre situaciones de riesgo o emergencias suelen circular de forma dispersa y, en muchos casos, no se reportan a tiempo. Esta carencia repercute directamente en la capacidad de respuesta de las autoridades y debilita la confianza de la comunidad en los mecanismos de seguridad disponibles. Según Mingaco Informativo (2024) en *Delincuencia y migración irregular: una opinión desde las estadísticas*, la relación entre delincuencia y migración irregular ha aumentado debido a la falta de plataformas efectivas para gestionar esta información, lo que contribuye a una mayor incertidumbre y desconfianza en los sistemas existentes.

A ello se suma la ausencia de herramientas digitales diseñadas para una participación ciudadana amplia e inclusiva. El uso extendido de redes sociales como medio de reporte ha excluido a sectores de la población que no cuentan con el acceso, la conectividad o las habilidades tecnológicas necesarias para utilizarlas de forma efectiva. En este contexto, el Ministerio del Interior (2024) en su *Encuesta Nacional de Seguridad Pública*, subraya la importancia de contar con una infraestructura tecnológica adecuada que permita mejorar tanto la percepción de seguridad como la implicación de la comunidad en la vigilancia de su entorno.

De mantenerse este escenario, es probable que la sensación de inseguridad continúe creciendo, afectando negativamente la calidad de vida en la región del Biobío. Frente a esta situación, se plantea el desarrollo de una herramienta digital que permita concretar los reportes ciudadanos en una plataforma de fácil acceso, promoviendo una activa y eficaz participación.

1.1 1.2 Problemática

En Chile ya existen distintas herramientas orientadas a la seguridad ciudadana. Entre ellas se encuentra SOSAFE, utilizada para reportar delitos y emergencias y vinculada con municipios y organismos de seguridad (SOSAFE, 2025; Municipalidad de Colina, 2019, 2023). También existen soluciones municipales como Vigiapp, que permite enviar reportes georreferenciados y facilita la gestión de esta información desde los municipios (Vigiapp, 2025).

Por otra parte, Denuncia Seguro ofrece un canal formal y anónimo para realizar denuncias por teléfono o por internet. Sin embargo, este servicio no incorpora funciones de interacción entre vecinos ni herramientas de visualización comunitaria de los incidentes reportados (Subsecretaría de Prevención del Delito, 2024).

A partir de este contexto, VisionSecurity se propone como una alternativa enfocada en la organización entre usuarios mediante comunidades, solicitudes de amistad y reportes geolocalizados. El desafío técnico fue integrar estas funciones en una arquitectura compuesta por Flutter, un backend PHP y una base de datos MySQL alojada en AWS. Las pruebas realizadas

mostraron que el prototipo responde de manera adecuada con cargas moderadas, pero comienza a presentar mayores tiempos de respuesta y errores cuando aumenta la cantidad de usuarios simultáneos. Esto permitió identificar que todavía es necesario mejorar el rendimiento y fortalecer la infraestructura antes de considerar un uso a mayor escala.

1.2 1.3 Objetivo general

Desarrollar y validar, desde el enfoque de backend e integración cliente-servidor, la plataforma VisionSecurity: una solución móvil de reportes geolocalizados con Flutter como cliente, endpoints PHP, base de datos MySQL en AWS RDS y comunicación HTTP/JSON, orientada a apoyar la organización de reportes ciudadanos y la coordinación comunitaria en la Región del Biobío.

1.3 1.4 Objetivos específicos

Identificar y documentar los requerimientos funcionales y no funcionales del sistema, considerando los módulos de usuarios, reportes geolocalizados, comunidades, solicitudes de amistad, seguridad y rendimiento.

Definir la arquitectura cliente-servidor de tres capas, compuesta por aplicación Flutter, backend PHP y base de datos MySQL en AWS RDS, especificando responsabilidades, interfaces y flujos de comunicación entre componentes.

Integrar la aplicación móvil Flutter con el backend PHP mediante solicitudes HTTP de tipo GET/POST y respuestas JSON, permitiendo que las pantallas del cliente consuman los endpoints desarrollados.

Implementar endpoints PHP para la gestión de usuarios, autenticación, reportes, solicitudes de amistad, comunidades, ubicaciones y consulta de información asociada a los módulos principales.

Diseñar e implementar un modelo de base de datos relacional en MySQL sobre AWS RDS, incorporando tablas para usuarios, reportes, comunidades, amistades, ubicaciones y miembros de comunidad, con relaciones que permitan mantener la consistencia de los datos.

Aplicar y documentar mecanismos básicos de seguridad en el backend, incluyendo almacenamiento de contraseñas con bcrypt y consultas SQL parametrizadas, con el fin de reducir riesgos asociados a credenciales e inyección SQL.

Validar funcionalmente los módulos principales mediante trazabilidad entre pantallas Flutter, endpoints PHP, tablas MySQL, respuestas JSON y resultados visibles en la aplicación.

Evaluar el comportamiento del backend mediante pruebas de carga con Apache JMeter sobre flujos críticos, considerando tiempos de respuesta, tasa de errores, uso de recursos y limitaciones ante alta concurrencia.

CAPÍTULO 2: MARCO CONCEPTUAL

2.1 2.1 Conceptos técnicos

2.1.1 Introducción a la solución técnica

La solución técnica propone una *app* móvil para gestionar información geolocalizada y promover la interacción entre usuarios. Se apoya en los siguientes enfoques y tecnologías:

Tabla 2.1. Componentes técnicos utilizados en la solución. Fuente: Elaboración propia.

Componente	Rol en la solución	Fundamento técnico
<i>Flutter (framework multiplataforma)</i>	Permite desarrollar una única base de código que se compila para <i>Android</i> .	<i>Framework</i> multiplataforma que reduce tiempos y costos de desarrollo manteniendo buen rendimiento (<i>Prajapati, 2019; Richardson et al., 2013</i>).
<i>Backend en PHP y API RESTful</i>	Implementa la lógica de negocio y expone <i>endpoints REST</i> que se comunican mediante <i>JSON</i> .	Sigue el modelo cliente–servidor basado en recursos, simple de integrar, escalable y ampliamente usado (<i>Richardson et al., 2013; Miller y Rowe, 2022</i>).
<i>MySQL en Amazon RDS</i>	Almacena usuarios, reportes y comunidades en un esquema relacional.	Las bases de datos relacionales entregan integridad referencial y consistencia entre entidades relacionadas (<i>Silberschatz, Korth y Sudarshan, 2020; Longley et al., 2015</i>).
<i>Amazon EC2 (IaaS en la nube)</i>	Hospeda el <i>backend</i> y permite ajustar recursos de cómputo según la demanda.	La computación en la nube ofrece elasticidad y alta disponibilidad, adecuada para servicios móviles escalables (<i>Erl, Mahmood y Puttini, 2013; AWS RDS, s. f.</i>).

2.1.2 Arquitectura del sistema

La solución adopta una arquitectura de tres capas:

- **Capa de interfaz (aplicación móvil *Flutter*).** Gestiona la interacción con el usuario final y se comunica con el servidor mediante solicitudes *HTTP(S)* que intercambian datos en formato *JSON*.
- **Capa de servidor (*API REST en PHP sobre AWS EC2*).** Expone *endpoints RESTful* que reciben las solicitudes de la *app*, validan la información y ejecutan la lógica de negocio necesaria antes de acceder a la base de datos.

- **Capa de datos (MySQL en AWS RDS).** Almacena de forma relacional la información de usuarios, reportes y comunidades, garantizando la integridad y consistencia de los registros.

Complementariamente, la infraestructura incorpora servicios de **logs y respaldo**, que permiten registrar errores y eventos, así como realizar copias de seguridad automáticas de la base de datos.

En la Figura 2.1 se resume el flujo general: el usuario interactúa con la aplicación móvil, esta envía la solicitud al servidor, el servidor procesa y consulta la base de datos y, finalmente, la respuesta se devuelve en formato JSON para ser reflejada en la interfaz.

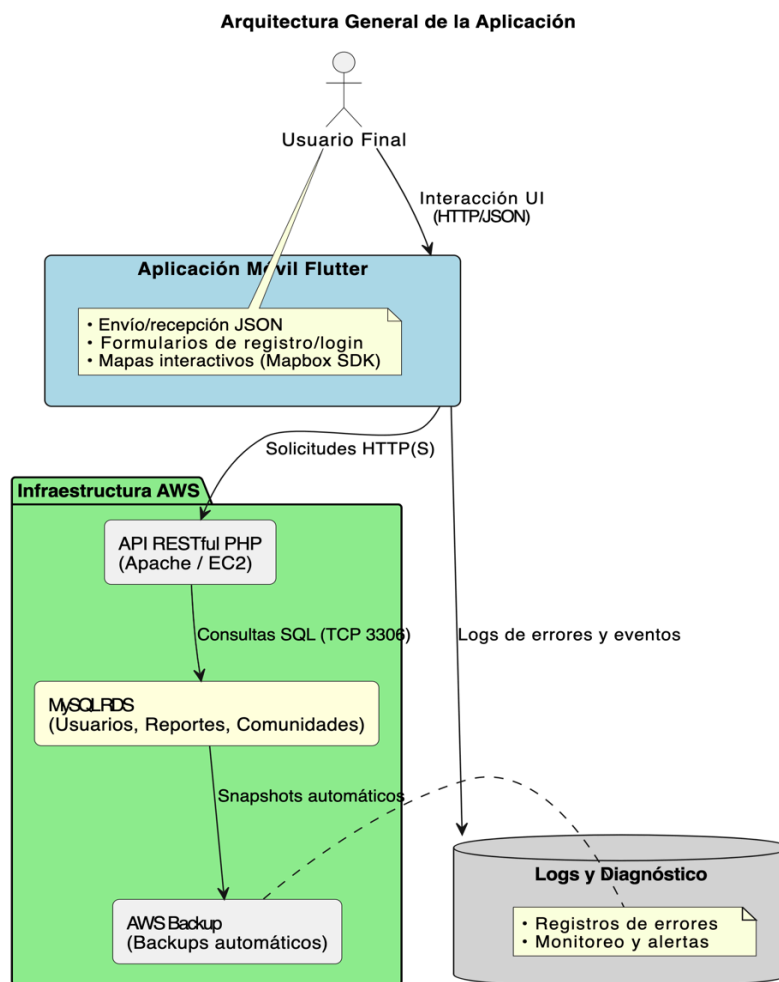


Figura 2.1. Arquitectura general de la aplicación. Fuente: Elaboración propia.

2.1.3 Base de datos (MySQL)

El sistema usa un modelo relacional en *AWS RDS* con tablas principales de usuarios, reportes, comunidades, amistades y ubicaciones. La estructura busca optimizar consultas y mantener la integridad referencial. La normalización y las claves foráneas son pilares de la coherencia en sistemas transaccionales (*Silberschatz et al., 2020*).

En aplicaciones de reporte geolocalizado, un diseño correcto permite analizar patrones espaciales y temporales, como muestran estudios de vigilancia ciudadana (*Goodchild y Li, 2012; Zhou et al., 2020*).

2.1.4 Desarrollo de la aplicación móvil

La aplicación móvil se organiza de forma modular mediante carpetas como *screens/*, *widgets/*, *services/* y *assets/*. En esta memoria, dicha estructura se considera principalmente como el punto de origen de las solicitudes hacia el backend, ya que las pantallas y servicios de Flutter permiten enviar datos, consumir endpoints PHP y procesar respuestas JSON devueltas por el servidor.

CAPÍTULO 3: PROPUESTA DE SOLUCIÓN

3.1 Enfoque técnico de la solución

La propuesta de solución de VisionSecurity se estructuró como una arquitectura cliente-servidor orientada a permitir el registro, consulta y visualización de reportes geolocalizados, junto con funciones de interacción entre usuarios mediante comunidades y solicitudes de amistad. El foco técnico de este capítulo se centra en la construcción del backend, la base de datos, la integración HTTP/JSON con la aplicación móvil y la forma en que cada módulo fue verificado durante el desarrollo.

La aplicación móvil desarrollada en Flutter actúa como cliente del sistema. Desde sus pantallas se generan solicitudes HTTP de tipo GET o POST hacia endpoints PHP alojados en servidor. Estos endpoints reciben parámetros, validan la información requerida, ejecutan operaciones sobre la base de datos MySQL alojada en AWS RDS y devuelven respuestas en formato JSON para que Flutter actualice la interfaz.

La infraestructura base se apoyó en servicios de AWS, principalmente EC2 para alojar el backend PHP sobre servidor Apache y RDS para almacenar la base de datos MySQL. Esta configuración permitió separar la capa de presentación, la lógica del servidor y la persistencia de datos, manteniendo un flujo de comunicación claro entre la aplicación móvil y el backend.

El flujo general de funcionamiento utilizado en los módulos principales fue el siguiente: Flutter genera la solicitud, el endpoint PHP procesa los parámetros, MySQL RDS almacena o recupera la información, el backend retorna una respuesta JSON y la aplicación actualiza el estado visual correspondiente. Este patrón se aplicó en módulos como registro, inicio de sesión, reportes, comunidades y solicitudes de amistad.

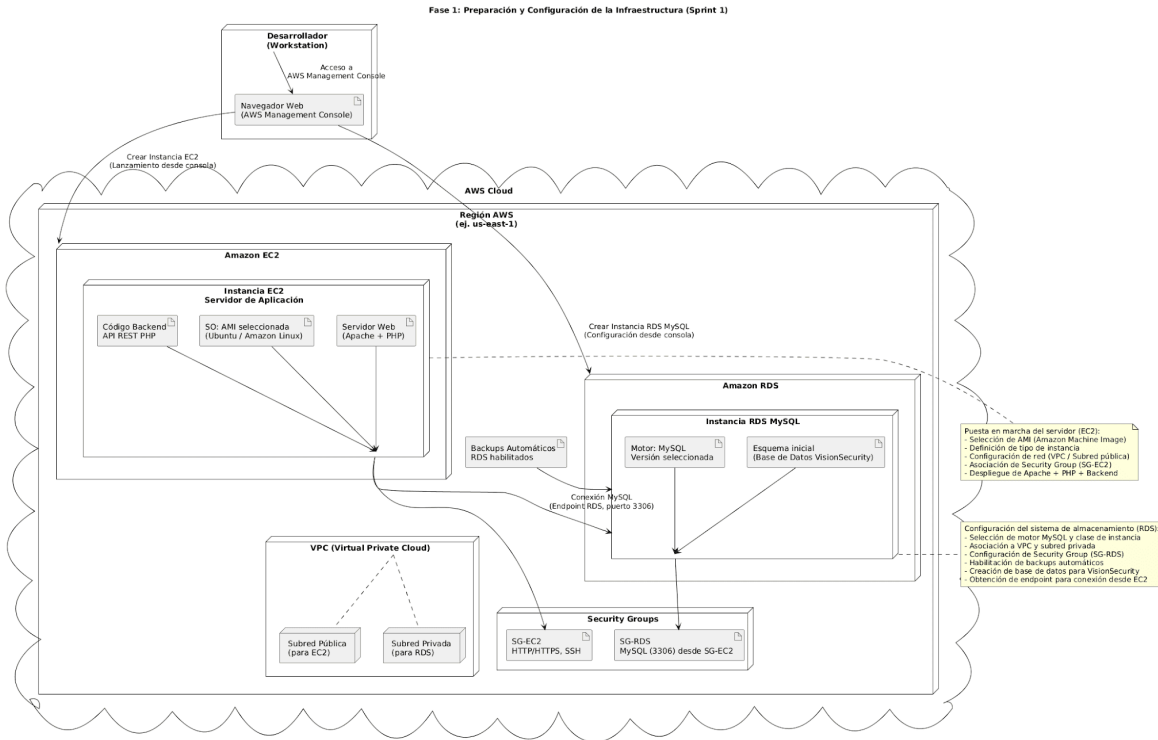


Figura 3.1. Preparación y configuración de la infraestructura en AWS: EC2, RDS, VPC y grupos de seguridad.
Fuente: Elaboración propia.

3.2 Metodología de construcción aplicada

La construcción del sistema se organizó en etapas iterativas, cada una con un objetivo técnico específico. En ellas se trabajó en la configuración de la infraestructura, el diseño de la base de datos, la implementación de los endpoints, la integración con Flutter y la validación de los módulos. Para cada etapa se definió el resultado esperado y la forma en que se comprobaría su funcionamiento.

Cada etapa tuvo un objetivo concreto, actividades técnicas asociadas, un producto obtenido y una forma de validación. Esta estructura permite vincular directamente la metodología con la evidencia técnica disponible: código fuente, endpoints, tablas de base de datos, capturas del prototipo y flujos de comunicación HTTP/JSON.

Tabla 3.1. Metodología de construcción aplicada. Fuente: Elaboración propia.

Etapas	Objetivo	Actividades realizadas	Producto obtenido	Forma de validación
1. Configuración de infraestructura	Habilitar el entorno de ejecución del backend y la base de datos.	Configuración inicial de EC2, servidor Apache/PHP, RDS MySQL y acceso entre	Servidor backend disponible y base de datos remota accesible.	Revisión de conexión entre backend y MySQL mediante endpoints PHP.

Etapas	Objetivo	Actividades realizadas	Producto obtenido	Forma de validación
		servidor y base de datos.		
2. Diseño de base de datos	Definir la persistencia de usuarios, reportes, comunidades, amistades y ubicaciones.	Definición de tablas, identificadores, relaciones y uso de id_Usuario como eje de asociación.	Modelo relacional usado por los módulos principales.	Revisión de relaciones y correspondencia entre tablas y funcionalidades.
3. Construcción de endpoints PHP	Implementar la lógica de servidor para recibir solicitudes de la app.	Creación de endpoints para registro, login, reportes, consulta de datos, comunidades y amistades.	API PHP basada en solicitudes GET/POST y respuestas JSON.	Pruebas funcionales desde Flutter y revisión de respuestas del backend.
4. Integración con Flutter	Conectar pantallas de la aplicación con endpoints del backend.	Implementación de solicitudes HTTP desde formularios, mapa, reportes, comunidades y solicitudes.	Flujo cliente-servidor funcional entre Flutter, PHP y MySQL.	Verificación de envío de parámetros, recepción JSON y actualización de interfaz.
5. Reportes geolocalizados	Permitir la creación y consulta de incidentes con ubicación.	Captura de coordenadas, formulario de reporte, envío al backend y almacenamiento en Reportes.	Módulo de reportes conectado a mapa y base de datos.	Validación por trazabilidad entre form_reporte.dart, endpoint PHP y tabla Reportes.
6. Módulos sociales	Permitir interacción entre usuarios mediante amistades y comunidades.	Gestión de solicitudes, estados pendiente/aceptado, creación de comunidades y miembros.	Módulos sociales asociados a Usuario, Amigos, Comunidades y Miembros_Comunidad.	Validación funcional por flujo: solicitud, consulta, aceptación y asociación de miembros.
7. Validación técnica inicial	Comprobar funcionamiento e integración de los módulos desarrollados.	Pruebas desde la app, revisión de respuestas JSON, consulta de datos y pruebas de carga documentadas.	Evidencia funcional y técnica para Capítulo 4.	Revisión por módulo, trazabilidad técnica y resultados de carga disponibles en memoria.

3.3 Configuración de infraestructura backend y base de datos

La primera etapa consistió en habilitar la infraestructura necesaria para que la aplicación móvil pudiera comunicarse con un servidor externo y almacenar información persistente. Para ello, se utilizó AWS como entorno cloud, principalmente mediante una instancia EC2 para alojar el backend y una base de datos MySQL administrada en AWS RDS.

En la instancia EC2 se preparó el entorno de servidor necesario para ejecutar scripts PHP mediante Apache. Esta capa recibió las solicitudes generadas desde Flutter y actuó como

intermediaria entre la aplicación móvil y la base de datos. En paralelo, RDS permitió disponer de una base de datos remota para almacenar usuarios, reportes, comunidades, solicitudes de amistad y ubicaciones.

La decisión de separar servidor y base de datos permitió organizar el sistema en capas: la app Flutter como cliente, PHP como capa de procesamiento y MySQL como capa de persistencia. La verificación de esta etapa se realizó mediante la conexión de endpoints PHP con la base de datos y la revisión de respuestas devueltas hacia la aplicación.

3.4 Diseño de base de datos y uso del id_Usuario

El diseño de la base de datos se realizó considerando las funcionalidades principales del sistema. Las tablas utilizadas se orientaron a representar usuarios, reportes, comunidades, amistades, ubicaciones y miembros de comunidad. Estas entidades permitieron que cada acción ejecutada desde la app quedara asociada a un usuario o a una relación específica dentro del sistema.

El campo id_Usuario cumplió un rol central dentro de la solución, ya que permitió vincular el inicio de sesión con acciones posteriores como crear reportes, consultar reportes propios, enviar solicitudes de amistad, crear comunidades o registrar ubicaciones. Una vez autenticado el usuario, este identificador era utilizado por la aplicación para enviar solicitudes al backend con la referencia necesaria para ejecutar operaciones sobre la base de datos.

En el módulo de reportes se decidió almacenar coordenadas geográficas, en lugar de depender únicamente de direcciones textuales. Esta decisión permitió ubicar los marcadores con mayor precisión dentro del mapa, ya que las coordenadas entregan una referencia directa para representar el punto seleccionado por el usuario.

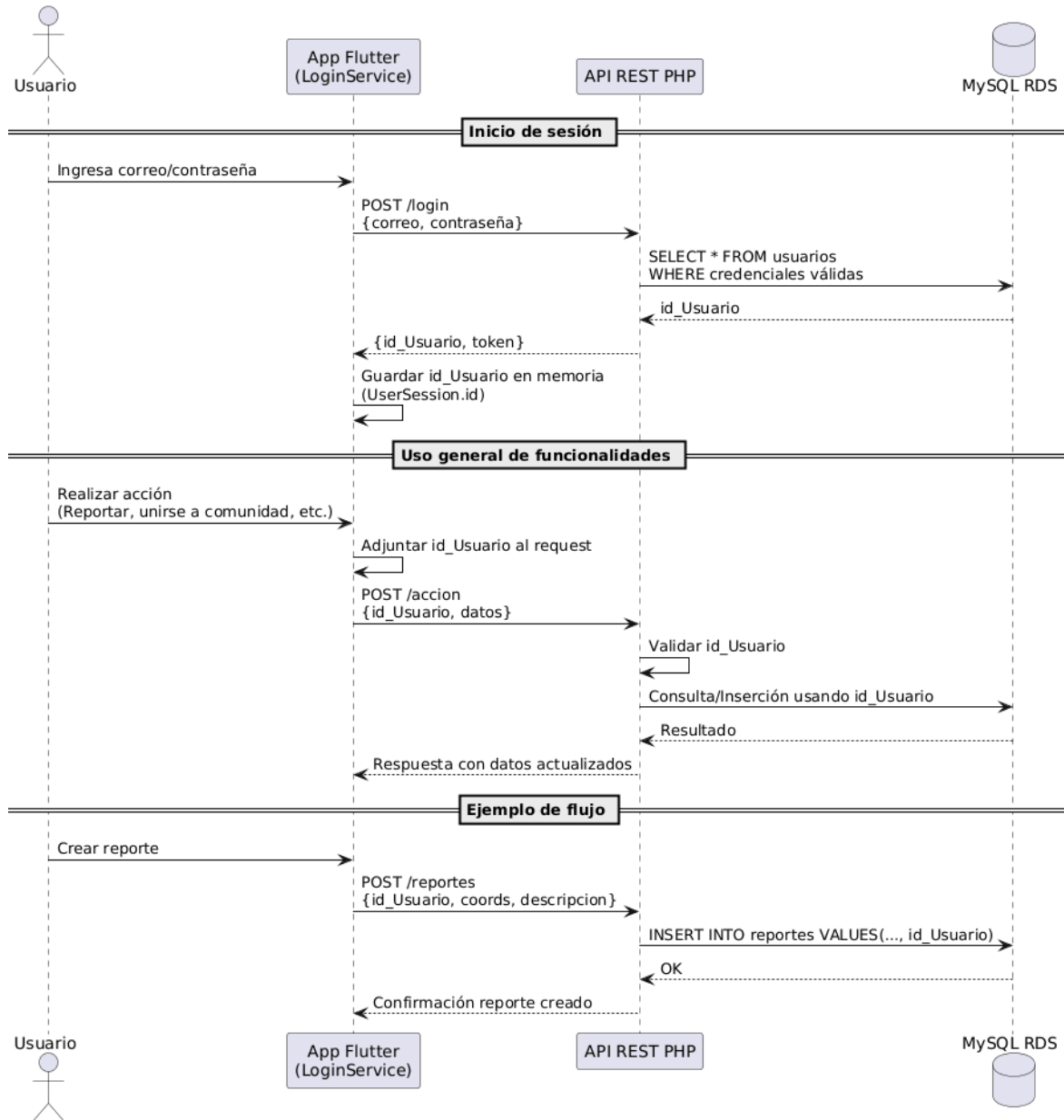


Figura 3.3. Diagrama de uso del id_usuario. Fuente: Elaboración propia.

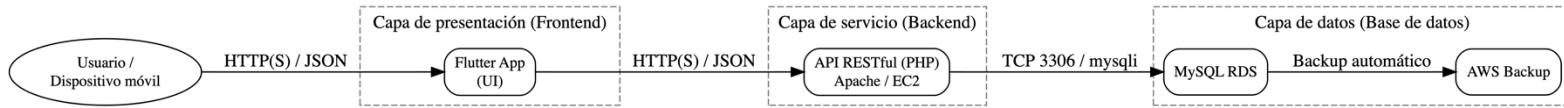
3.5 Construcción del backend PHP

El backend se implementó mediante scripts PHP encargados de recibir solicitudes HTTP, procesar parámetros, ejecutar consultas SQL y devolver respuestas a la aplicación móvil. Cada archivo PHP funcionó como un endpoint asociado a una acción específica del sistema.

Los endpoints principales se relacionaron con el registro de usuarios, inicio de sesión, creación y consulta de reportes, obtención de datos de usuario, solicitudes de amistad y comunidades. La

estructura permitió separar la lógica de cada módulo y facilitar la integración con las pantallas correspondientes de Flutter.

El backend operó como capa intermedia entre la interfaz móvil y MySQL. Esta decisión evitó que la aplicación accediera directamente a la base de datos y permitió centralizar la validación, el



procesamiento y la respuesta de cada operación en el servidor.

Figura 3.4. Flujo de comunicación del backend. Fuente: Elaboración propia.

3.6 Integración cliente-servidor mediante HTTP/JSON

La integración entre Flutter y PHP se realizó mediante solicitudes HTTP de tipo GET y POST. Las pantallas de la aplicación enviaban parámetros al servidor, tales como datos de usuario, credenciales, coordenadas, descripciones de reportes, identificadores de comunidad o estados de solicitudes.

El backend procesaba cada solicitud y respondía con datos estructurados en JSON o con mensajes de estado. Posteriormente, Flutter interpretaba la respuesta y actualizaba la interfaz, por ejemplo, mostrando reportes en el mapa, cargando listas, confirmando un registro o informando el resultado de una operación.

Este patrón de comunicación permitió construir una trazabilidad clara entre la acción del usuario, la pantalla Flutter, el endpoint PHP, la tabla MySQL utilizada y el resultado funcional esperado.

3.7 Inventario de endpoints utilizados por los módulos principales

Para facilitar su lectura en la memoria final, los endpoints se agrupan por tipo de funcionalidad. Esta organización mantiene la misma trazabilidad técnica entre pantalla Flutter, endpoint PHP, tabla MySQL y resultado funcional, pero evita una tabla única demasiado extensa.

Tabla 3.2. Endpoints de autenticación, reportes y consulta de datos. Fuente: Elaboración propia.

Endpoint	Módulo	Método / entrada principal	Tabla asociada	Resultado funcional
register.php	Registro de usuario	POST; nombre, apellido, correo/email, rut, contraseña y rol.	Usuario	Crear una cuenta de usuario.
login.php	Inicio de sesión	POST; nombre/correo y contraseña.	Usuario	Validar credenciales y retornar datos del usuario.

Endpoint	Módulo	Método / entrada principal	Tabla asociada	Resultado funcional
obtener_reportes.php	Consulta de reportes para mapa	GET/POST; sin parámetro o filtros de consulta.	Reportes	Obtener ubicaciones de reportes para marcadores.
card_reportes.php	Listado/detalle de reportes	GET; sin parámetro o id de reporte.	Reportes	Entregar información para tarjetas o listado de reportes.
obtener_reportes_usuario.php	Reportes por usuario	POST; id_Usuario.	Reportes	Consultar reportes asociados al usuario autenticado.
reporte.php	Creación de reporte	POST; id_Usuario, tipo, descripción, ubicación, anonimato e imagen/ruta.	Reportes	Registrar un nuevo reporte geolocalizado.

Tabla 3.3. Endpoints de solicitudes de amistad y comunidades. Fuente: Elaboración propia.

Endpoint	Módulo	Método / entrada principal	Tabla asociada	Resultado funcional
insertar_amigo.php	Solicitud de amistad	POST; id_Usuario e id_Usuario_solicitado.	Amigos	Crear una solicitud en estado pendiente.
obtener_solicitudes.php	Notificaciones de amistad	POST; id_Usuario.	Amigos	Consultar solicitudes recibidas.
aceptar_solicitud.php	Aceptación de amistad	POST; id_solicitud o usuarios relacionados.	Amigos	Actualizar estado de solicitud a aceptado.
eliminar_solicitud.php	Rechazo/eliminación de solicitud	POST; id_solicitud o usuarios relacionados.	Amigos	Eliminar o rechazar solicitud pendiente.
crear_comunidad.php	Creación de comunidad	POST; id_Usuario y nombre_comunidad.	Comunidades	Crear una comunidad asociada al usuario.
agregar_miembro_comunidad.php	Miembros de comunidad	POST; idComunidad e id_Usuario.	Miembros_Comunidad	Asociar usuarios a una comunidad.
obtener_comunidades.php	Consulta de comunidades	POST/GET; id_Usuario.	Comunidades / Miembros_Comunidad	Listar comunidades asociadas al usuario.

Tabla 3.4. Endpoints de ubicación de usuarios y miembros. Fuente: Elaboración propia.

Endpoint	Módulo	Método / entrada principal	Tabla asociada	Resultado funcional
registrar_ubicacion.php	Registro de ubicación	POST; id_Usuario, latitud y longitud.	Ubicaciones	Actualizar ubicación del usuario.
obtener_ubicaciones.php	Ubicación de miembros	POST/GET; idComunidad.	Ubicaciones / Miembros_Comunidad	Obtener ubicaciones para visualización en mapa.

3.8 Relación entre pantallas, endpoints y base de datos

La trazabilidad técnica relaciona cada módulo con la pantalla Flutter que origina la acción, el endpoint PHP que procesa la solicitud y la tabla MySQL involucrada. Esta matriz permite demostrar cómo se construyó el flujo de integración y servirá como base para la validación funcional del Capítulo 4.

Tabla 3.5. Relación entre pantallas, endpoints y base de datos. Fuente: Elaboración propia.

Módulo	Pantalla Flutter	Endpoint PHP	Tabla MySQL	Trazabilidad funcional
Registro	Pantalla de registro / main.dart	register.php	Usuario	Flutter envía datos del formulario por POST; PHP registra el usuario en MySQL.
Inicio de sesión	Pantalla de login / main.dart	login.php	Usuario	Flutter envía credenciales; PHP valida y retorna id_Usuario, nombre y rol.
Creación de reporte	form_reporte.dart	reporte.php	Reportes	Flutter envía tipo, descripción, ubicación, anonimato e imagen/ruta; PHP inserta el reporte.
Consulta de reportes en mapa	map_screen.dart	obtener_reportes.php	Reportes	Flutter solicita reportes; PHP retorna ubicaciones para marcadores.
Detalle/listado de reportes	report_screen.dart / tarjetas de mapa	card_reportes.php	Reportes	Flutter recibe datos descriptivos para mostrar tarjetas o lista de reportes.
Reportes del usuario	Vista de reportes propios	obtener_reportes_usuario.php	Reportes	Flutter envía id_Usuario y recibe reportes asociados a la cuenta.

Módulo	Pantalla Flutter	Endpoint PHP	Tabla MySQL	Trazabilidad funcional
Solicitud de amistad	communities.dart / pantalla social	insertar_amigo.php	Amigos	Flutter envía usuario solicitante y receptor; PHP crea solicitud pendiente.
Gestión de solicitudes	notifications.dart	obtener_solicitudes.php / aceptar_solicitud.php / eliminar_solicitud.php	Amigos	Flutter consulta solicitudes y actualiza estado según acción del usuario.
Creación de comunidad	crear_comunidad.dart	crear_comunidad.php	Comunidades	Flutter envía nombre e id_Usuario; PHP registra comunidad.
Miembros de comunidad	seleccionar_comunidad.dart / seleccionar_comunidad2.dart	agregar_miembro_comunidad.php	Miembros_Comunidad	Flutter envía usuarios seleccionados; PHP los asocia a la comunidad.
Ubicación de usuarios	map_screen.dart / map_screen2.dart	registrar_ubicacion.php / obtener_ubicaciones.php	Ubicaciones	Flutter actualiza y consulta coordenadas para mostrar ubicación de usuarios o miembros.

3.9 Validación funcional por módulos dentro de la metodología

Durante la construcción se definió una verificación funcional mínima por módulo. En este capítulo se presenta como criterio metodológico de revisión; los resultados específicos, evidencias e interpretación deben organizarse después en el Capítulo 4.

Tabla 3.6. Validación funcional por módulos dentro de la metodología. Fuente: Elaboración propia.

Módulo	Escenario funcional	Acción realizada	Resultado esperado	Forma de verificación
Registro	Crear una cuenta desde Flutter.	Completar formulario y enviar POST al backend.	Registro almacenado en Usuario y respuesta de confirmación.	Verificación del flujo formulario-endpoint-tabla.
Inicio de sesión	Acceder con credenciales registradas.	Enviar usuario/contraseña desde Flutter hacia login.php.	Respuesta con estado exitoso y datos mínimos del usuario.	Confirmación de id_Usuario para acciones posteriores.
Creación de reporte	Registrar incidente geolocalizado.	Seleccionar ubicación, completar formulario y enviar datos al backend.	Reporte almacenado con usuario, tipo, descripción, coordenadas y estado de anonimato.	Revisión de persistencia en tabla Reportes.

Módulo	Escenario funcional	Acción realizada	Resultado esperado	Forma de verificación
Consulta de reportes	Visualizar reportes en mapa o lista.	Solicitar reportes desde map_screen/report_screen.	Respuesta JSON con reportes y ubicaciones.	Actualización de marcadores o tarjetas en Flutter.
Solicitudes de amistad	Enviar y gestionar solicitud.	Enviar usuario solicitante/receptor y consultar solicitudes pendientes.	Solicitud registrada y actualizada según aceptación o eliminación.	Verificación de estados en tabla Amigos.
Comunidades	Crear comunidad y asociar miembros.	Enviar nombre de comunidad e integrantes seleccionados.	Comunidad creada y usuarios asociados.	Revisión de tablas Comunidades y Miembros_Comunidad.
Ubicaciones	Registrar y consultar posición de usuarios.	Enviar coordenadas desde la app y consultar ubicación de miembros.	Coordenadas disponibles para visualización en mapa.	Trazabilidad entre mapa, endpoint y tabla Ubicaciones.

3.10 Sistema de reportes geolocalizados

El sistema de reportes fue construido como uno de los módulos centrales de VisionSecurity. Desde la aplicación móvil, el usuario podía seleccionar una ubicación en el mapa, completar información del incidente y enviar los datos al backend. El servidor recibía la solicitud y registraba la información en la tabla Reportes.

La ubicación del reporte se almacenó mediante coordenadas, permitiendo que posteriormente los reportes fueran consultados y representados como marcadores dentro del mapa. Esta decisión fue relevante para mantener precisión en la visualización, ya que una dirección textual podía no coincidir exactamente con el punto seleccionado por el usuario.

El flujo técnico del módulo puede resumirse de la siguiente forma: form_reporte.dart captura los datos del formulario, el endpoint PHP recibe la solicitud, MySQL almacena el reporte y la aplicación consulta posteriormente los reportes para mostrarlos en el mapa o en tarjetas informativas.

3.11 Comunidades y solicitudes de amistad

Los módulos de comunidades y solicitudes de amistad fueron incorporados para permitir interacción entre usuarios. La lógica se apoyó en el id_Usuario como identificador principal para asociar solicitudes, amistades aceptadas, comunidades creadas y miembros vinculados a cada comunidad.

En el caso de solicitudes de amistad, el flujo considera el envío de una solicitud desde la aplicación, la validación del usuario receptor, el registro de la relación en la tabla Amigos y la actualización del estado de la solicitud. Este estado permite diferenciar solicitudes pendientes, aceptadas o eliminadas.

En el caso de comunidades, el usuario puede crear un grupo, seleccionar miembros y asociarlos mediante una tabla intermedia. Esta estructura permite mantener una relación entre usuarios y comunidades sin duplicar datos, facilitando la consulta posterior de miembros o ubicaciones asociadas.

3.12 Síntesis de la propuesta de solución

La propuesta técnica quedó organizada en una arquitectura de tres capas: aplicación Flutter, backend PHP y base de datos MySQL en AWS RDS. El valor principal de esta estructura fue permitir que cada funcionalidad de la aplicación tuviera un flujo de comunicación definido, desde la acción del usuario hasta la persistencia o consulta de datos.

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN

4.1 Enfoque de validación aplicado

La validación de VisionSecurity se enfocó en revisar el funcionamiento de los módulos principales a partir de su flujo completo. Para ello, se comprobó cómo una acción realizada desde la aplicación móvil era enviada al backend PHP, procesada junto con la base de datos MySQL y devuelta a Flutter mediante una respuesta JSON. Esta revisión permitió relacionar cada pantalla con el endpoint utilizado y con la tabla correspondiente en la base de datos.

Para cada módulo se relacionó la acción ejecutada desde Flutter con la solicitud HTTP generada, el endpoint PHP involucrado, la tabla MySQL utilizada, la respuesta JSON obtenida y el resultado visible en la interfaz. Además, se incorporan evidencias del código fuente, diagramas de flujo, estructura de base de datos, capturas del prototipo y resultados de pruebas de carga. Este esquema corresponde al modelo cliente-servidor utilizado habitualmente en servicios web basados en HTTP, descrito por Richardson, Amundsen y Ruby (2013).

La figura 4.1 resume el criterio usado para validar los flujos principales. La acción comienza en una pantalla Flutter, continúa mediante una solicitud HTTP hacia un archivo PHP, se procesa contra la base de datos MySQL y finaliza con una respuesta JSON que actualiza la interfaz del usuario.

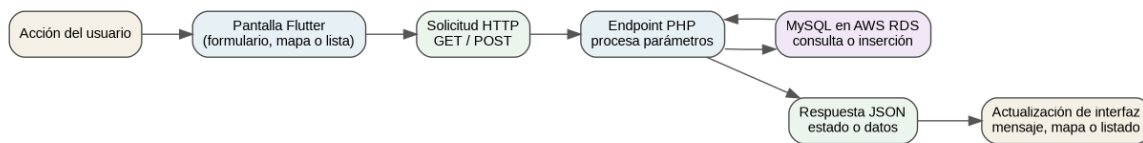


Figura 4.1. Criterio general de validación utilizado en los módulos principales. Fuente: Elaboración propia.

4.2 Evidencia visual del prototipo y componentes utilizados en la validación

La validación funcional consideró las pantallas principales del prototipo, ya que estas representan el punto de entrada de los flujos que posteriormente son procesados por el backend. En particular, se utilizaron como evidencia las pantallas de inicio de sesión, creación de cuenta, mapa interactivo y visualización de reportes, porque cada una inicia una operación que continúa hacia un endpoint PHP o consume información devuelta por el servidor. Estas pantallas permiten comprobar que la aplicación no funciona como una interfaz aislada, sino como cliente de los servicios PHP conectados a la base de datos.

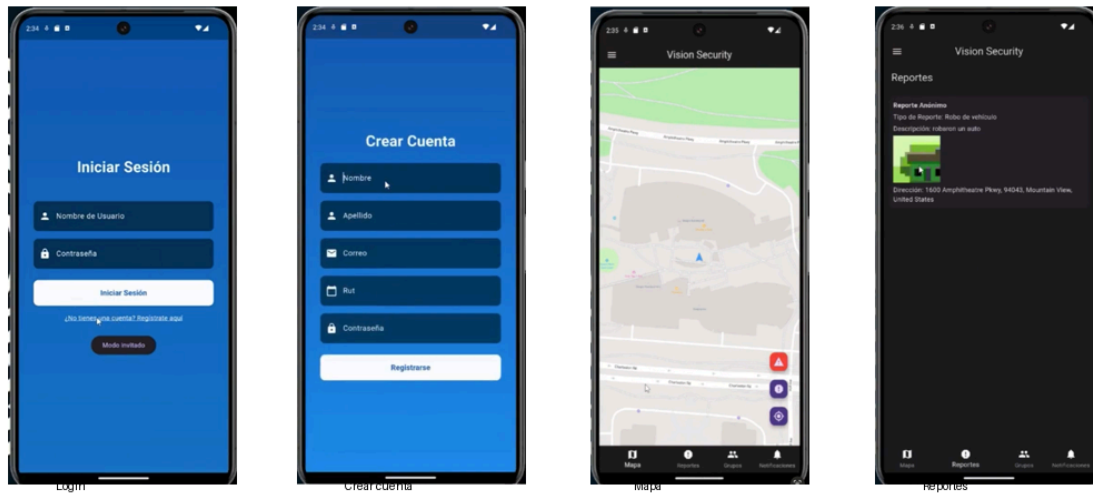


Figura 4.2. Evidencia visual de pantallas vinculadas a los flujos validados: login, registro, mapa y reportes.
Fuente: Elaboración propia.

Junto con las pantallas, el esquema relacional presentado en la Figura 3.2 permitió validar la persistencia de los módulos principales. La base de datos concentra usuarios, reportes, comunidades, solicitudes de amistad, ubicaciones y miembros de comunidad; por ello, cada prueba funcional se contrastó con la tabla involucrada y con el uso del identificador del usuario como vínculo entre autenticación y acciones posteriores. Esta organización se fundamenta en el uso de claves primarias y foráneas, criterio consistente con los principios de diseño relacional descritos por Silberschatz, Korth y Sudarshan (2020).

4.3 Validación funcional de los módulos principales

La validación funcional se organizó por módulos. En cada caso se consideró la acción realizada desde la aplicación, el componente técnico involucrado, el resultado obtenido y la interpretación del comportamiento. De esta forma, cada validación quedó asociada a una ruta completa entre la interfaz, el backend y la base de datos.

Tabla 4.1. Validación funcional de los módulos principales. Fuente: Elaboración propia.

Módulo	Escenario evaluado	Componente validado	Resultado obtenido	Interpretación
Registro de usuario	Creación de cuenta desde formulario Flutter.	register.php / Usuario	Cuenta registrada y respuesta de confirmación.	El flujo permite crear usuarios y dejar disponible el id_usuario para acciones posteriores.
Inicio de sesión	Ingreso con credenciales registradas.	login.php / Usuario	Respuesta JSON con estado, id_usuario, nombre y rol.	La autenticación habilita la sesión lógica que conecta al usuario con reportes y módulos sociales.
Creación de reportes	Selección de ubicación, ingreso de	reporte.php / Reportes	Reporte almacenado con usuario,	El módulo permite persistir incidentes

Módulo	Escenario evaluado	Componente validado	Resultado obtenido	Interpretación
	datos y envío del reporte.		coordenadas, descripción y anonimato.	geolocalizados y recuperarlos posteriormente.
Consulta de reportes	Carga de reportes desde mapa o vista de listado.	obtener_reportes.php, card_reportes.php / Reportes	Respuesta JSON con reportes para marcadores o tarjetas.	La información almacenada puede ser reutilizada por Flutter para actualizar la interfaz.
Solicitudes de amistad	Envío, consulta, aceptación o eliminación de solicitudes.	insertar_amigo.php, obtener_solicitudes.php, aceptar_solicitud.php, eliminar_solicitud.php / Amigos	Cambio de estado de la solicitud y actualización de notificaciones.	El módulo valida la relación social entre usuarios mediante estados controlados.
Comunidades	Creación de comunidad y asociación de integrantes.	crear_comunidad.php, agregar_miembro_comunidad.php, obtener_comunidades.php / Comunidades y Miembros_Comunidad	Comunidad registrada y usuarios asociados.	La estructura permite agrupar usuarios y consultar miembros vinculados a una comunidad.
Ubicaciones	Registro y consulta de coordenadas de usuarios o miembros.	registrar_ubicacion.php, obtener_ubicaciones.php / Ubicaciones	Coordenadas disponibles para visualización en mapa.	La información geográfica se mantiene separada, pero asociada al usuario mediante id_Usuario.

4.4 Validación de registro e inicio de sesión

El registro y el inicio de sesión fueron validados como flujo base del sistema, ya que ambos permiten crear una identidad de usuario y utilizarla en los demás módulos. La comprobación se realizó a partir del envío de datos desde el formulario Flutter hacia el backend, el procesamiento de la solicitud en el endpoint correspondiente y la respuesta utilizada por la aplicación para continuar o detener la navegación.

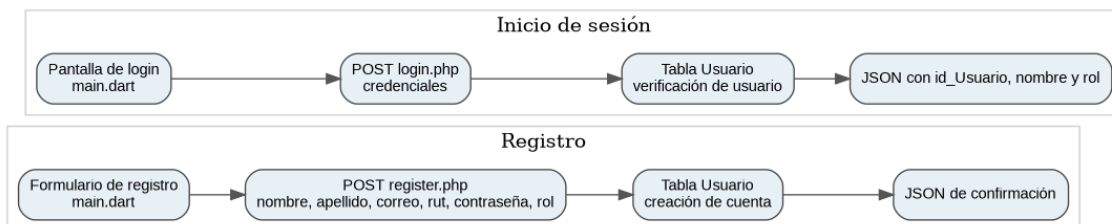


Figura 4.3. Flujo validado para registro e inicio de sesión. Fuente: Elaboración propia.

En el registro, el formulario construido en Flutter envía los datos del usuario mediante una solicitud POST. El siguiente fragmento muestra el envío de parámetros relevantes desde la

aplicación hacia el endpoint de registro. La evidencia permite confirmar que el flujo no se limita a la interfaz, sino que existe una comunicación explícita con el backend.

El fragmento de código se presenta con una ruta genérica para mostrar la lógica de comunicación sin exponer direcciones reales del servidor ni información sensible del entorno.

Código 4.1. Envío de datos de registro desde Flutter hacia el backend PHP.

```
final url = Uri.parse('http://servidor/consultas/register.php');
final response = await http.post(url, body: {
  'nombre': nombre,
  'apellido': apellido,
  'correo': correo,
  'rut': rut,
  'contrasena': contrasena,
  'rol': 'Comun',
});
```

La respuesta del backend fue utilizada para confirmar el resultado del registro. En caso correcto, el sistema habilitaba la continuidad del flujo de usuario; en caso contrario, la aplicación informa el error correspondiente. Este comportamiento permitió validar que los datos mínimos de registro fueran enviados y procesados desde la aplicación móvil hacia la capa de servidor.

En el inicio de sesión se validó que el backend devolviera una respuesta estructurada con los datos mínimos de sesión. El uso de una respuesta JSON permitió que Flutter obtuviera el identificador del usuario y lo reutilizara en reportes, solicitudes de amistad y comunidades. El uso de JSON como formato de intercambio favorece la interoperabilidad entre cliente y servidor, tal como se plantea en la arquitectura REST descrita por Richardson et al. (2013).

Código 4.2. Respuesta JSON esperada al validar credenciales de usuario.

```
if (password_verify($contrasena, $hash_almacenado)) {
  $response = array(
    'status' => 'success',
    'id_usuario' => $fila['id_usuario'],
    'nombre' => $fila['nombre'],
    'rol' => $fila['rol']
  );
  echo json_encode($response);
}
```

Tabla 4.2. Validación de registro e inicio de sesión. Fuente: Elaboración propia.

Prueba	Resultado obtenido	Interpretación
Registro de cuenta	La aplicación envía nombre, apellido, correo, RUT, contraseña y rol al endpoint de registro.	El flujo permite crear la identidad inicial del usuario en la base de datos.
Inicio de sesión	El backend retorna una estructura JSON con estado de operación e identificador de usuario.	El id_Usuario queda disponible para asociar reportes, comunidades y solicitudes.
Credenciales inválidas	El servidor puede responder con estado de error y mensaje asociado.	La aplicación informa el fallo sin continuar con acciones indicadas por el usuario.

4.4.1 Comprobación de consultas parametrizadas y hash de contraseñas

Como complemento al flujo de registro e inicio de sesión, se incorporó una comprobación de implementación sobre las prácticas básicas de seguridad declaradas para el backend. Esta comprobación no corresponde a una prueba de penetración ni a una auditoría de seguridad, sino a la verificación de que el código PHP utiliza consultas parametrizadas y funciones de hash de contraseña en operaciones sensibles del sistema.

En el caso del registro de usuario, la evidencia técnica se basa en la estructura del endpoint PHP, donde la sentencia SQL se define con marcadores de posición y los valores recibidos desde Flutter se vinculan posteriormente mediante *bind_param*. Este patrón evita concatenar directamente los datos ingresados por el usuario dentro de la consulta SQL, reduciendo el riesgo de inyección SQL en las operaciones de inserción.

El fragmento se presenta de forma parcial y genérica para evidenciar la lógica utilizada, omitiendo rutas reales, credenciales y detalles sensibles del entorno.

Código 4.3. Estructura genérica de consulta parametrizada aplicada al registro de usuario.

```
$hash = password_hash($contrasena, PASSWORD_BCRYPT);

$stmt = $conexion->prepare(
    "INSERT INTO Usuario (nombre, apellido, correo, rut, contrasena, rol) VALUES (?, ?, ?, ?, ?, ?)"
);

$stmt->bind_param("ssssss", $nombre, $apellido, $correo, $rut, $hash, $rol);
$stmt->execute();
```

A partir de esta evidencia, la comprobación se enfocó en confirmar tres aspectos: la separación entre sentencia SQL y datos de entrada, el almacenamiento de la contraseña como hash y la posterior validación de credenciales mediante comparación segura. El alcance de esta revisión se limita a demostrar la aplicación de estas prácticas dentro del flujo de autenticación, sin afirmar que el sistema haya sido sometido a una evaluación integral de seguridad.

Tabla 4.3. Comprobación de consultas parametrizadas y hash de contraseñas. Fuente: Elaboración propia.

Elemento comprobado	Evidencia observada	Resultado obtenido	Interpretación
Consulta parametrizada	Uso de prepare() con marcadores de posición (?) y asignación de valores mediante bind_param().	Los datos enviados desde Flutter se tratan como parámetros y no como parte directa de la sentencia SQL.	La implementación reduce el riesgo de inyección SQL en el flujo de registro documentado.
Hash de contraseña	Uso de password_hash(\$contraseña, PASSWORD_BCRYPT) antes de almacenar la credencial.	La contraseña no se registra en texto plano dentro de la base de datos.	La práctica disminuye la exposición de credenciales ante un acceso no autorizado a los registros.
Validación de credenciales	Uso de password_verify() en el flujo de inicio de sesión para comparar la contraseña ingresada con el hash almacenado.	La autenticación se realiza sin recuperar ni comparar la contraseña original en texto plano.	El flujo de autenticación queda alineado con una práctica básica de protección de credenciales.

Esta revisión permitió comprobar que el backend incorpora consultas parametrizadas y mecanismos de hash para el manejo de contraseñas. Sin embargo, su alcance se limitó a verificar la implementación presente en el código, por lo que no corresponde considerarla como una evaluación completa de seguridad.

4.5 Validación del sistema de reportes geolocalizados

El sistema de reportes fue validado a partir del recorrido completo que realiza un incidente desde la selección de ubicación hasta su almacenamiento y posterior visualización. Este módulo es crítico, porque conecta la interacción del mapa con la persistencia de datos en MySQL y la respuesta visual hacia el usuario.



Figura 4.4. Recorrido técnico validado para la creación y consulta de reportes. Fuente: Elaboración propia.

La primera parte de la validación consistió en comprobar que la ubicación seleccionada desde el mapa quedara disponible para el formulario del reporte. Posteriormente, el formulario enviaba los datos del incidente al backend mediante una solicitud HTTP POST. La validación consideró que el reporte quedara asociado a un usuario, una descripción, una ubicación y una condición de anonimato.

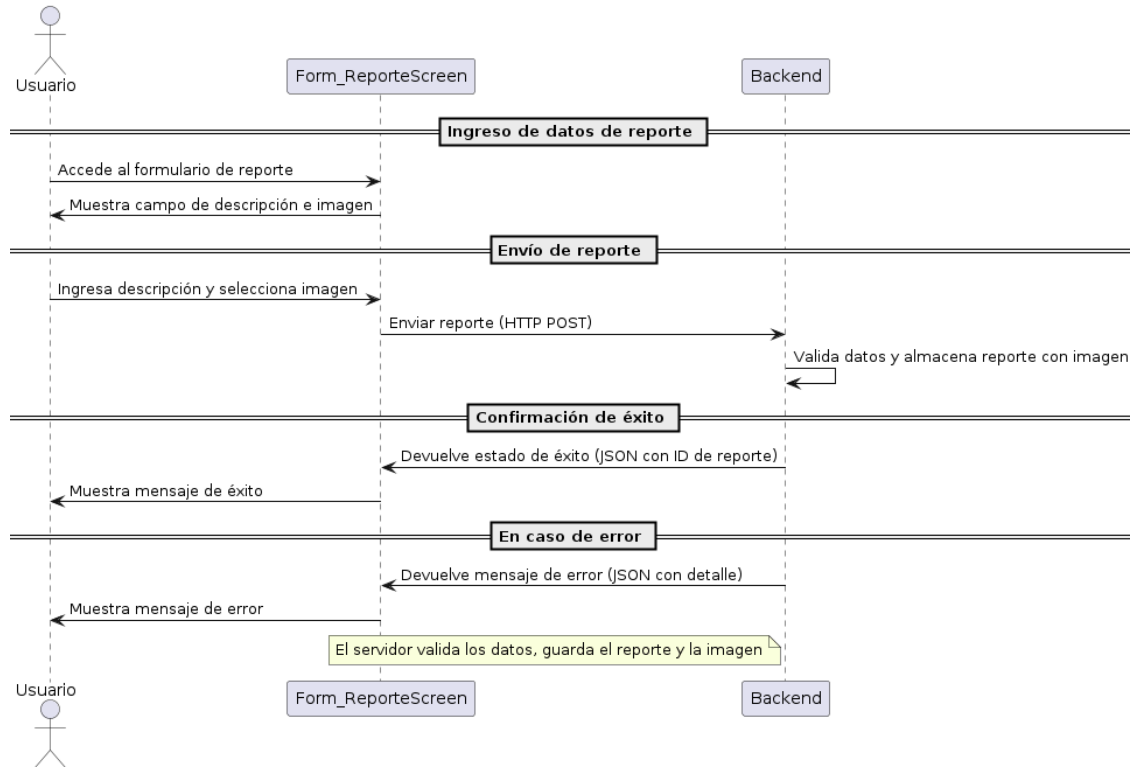


Figura 4.5. Flujo de envío de reporte desde formulario hacia backend. Fuente: Elaboración propia.

La estructura de la tabla Reportes confirma los campos necesarios para almacenar la información principal del incidente. Entre ellos se consideran el identificador del reporte, el identificador del usuario, el tipo de reporte, descripción, imagen, ubicación, anonimato y dirección. Esta estructura permite que el reporte pueda consultarse nuevamente desde el mapa o desde una vista de listado.

Código 4.4. Estructura lógica utilizada para respaldar la persistencia de reportes.

```

CREATE TABLE Reportes (
  id_Reporte INT NOT NULL AUTO_INCREMENT,
  id_Usuario INT,
  nombre VARCHAR(255) NOT NULL,
  tipo_reporte VARCHAR(50) NOT NULL,
  descripcion_emergencia TEXT NOT NULL,
  imagen VARCHAR(255),
  ubicacion_reporte VARCHAR(255),
  anonimo TINYINT(1),
  direccion VARCHAR(255),
  PRIMARY KEY (id_Reporte),
  FOREIGN KEY (id_Usuario) REFERENCES Usuario(id_Usuario)
);

```

Tabla 4.4. Validación del sistema de reportes geolocalizados. Fuente: Elaboración propia.

Etapa del flujo	Archivo o componente	Dato validado	Resultado
Selección de ubicación	map_screen.dart	Latitud y longitud seleccionadas	La ubicación queda preparada para generar el reporte.
Envío del reporte	form_reporte.dart + reporte.php	Descripción, tipo, coordenadas, usuario y anonimato	El reporte se inserta en la tabla Reportes.
Consulta posterior	obtener_reportes.php / card_reportes.php	Reportes almacenados	El backend devuelve datos en JSON para marcadores o tarjetas.
Visualización	map_screen.dart / report_screen.dart	Respuesta JSON procesada	Flutter actualiza mapa o listado con la información recibida.

La interpretación de esta prueba es que el módulo de reportes cumple el flujo esperado de extremo a extremo: la aplicación captura el evento, el backend lo procesa, la base de datos lo conserva y la interfaz puede recuperarlo. La precisión de la ubicación se apoya en el uso de coordenadas, lo que resulta adecuado para sistemas de información geográfica y reportes geolocalizados, en línea con lo señalado por Longley et al. (2015) y Zhou et al. (2020).

4.6 Validación de consulta y visualización de reportes

La consulta de reportes fue validada como flujo inverso al registro. En este caso, la aplicación solicita datos al backend, el servidor consulta la tabla Reportes y devuelve una respuesta JSON que contiene la información necesaria para construir marcadores, tarjetas o listados dentro de la app.

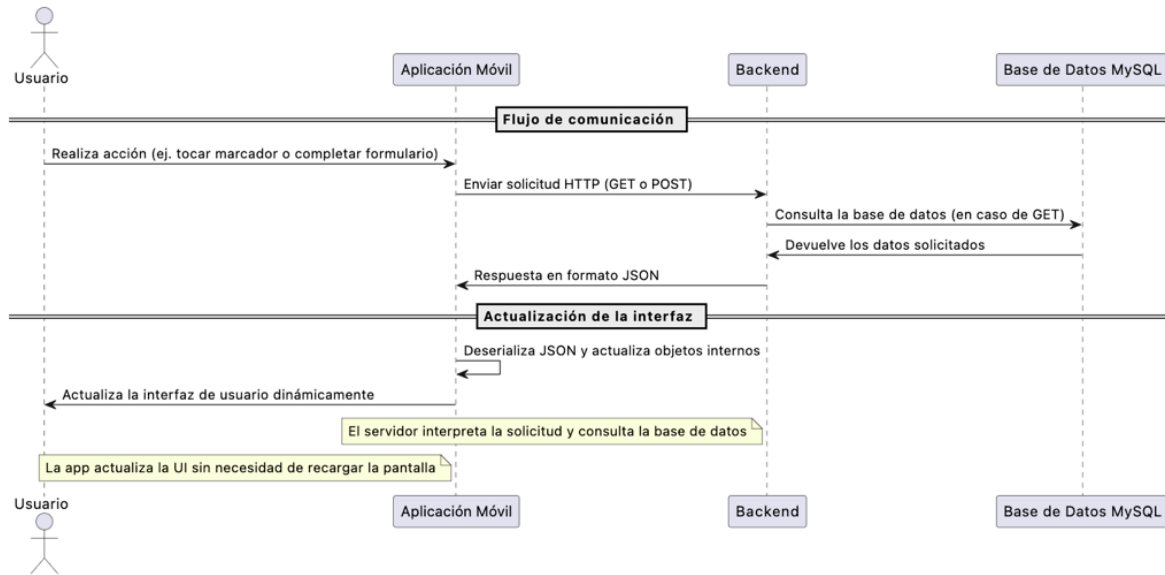


Figura 4.6. Flujo general de comunicación cliente-servidor utilizado para consultar datos. Fuente: Elaboración propia.

La validación se centró en comprobar que los reportes almacenados pudieran ser reutilizados por la aplicación sin acceso directo a la base de datos. Esta separación mantiene a PHP como capa intermedia de procesamiento y evita que la lógica de consulta quede alojada directamente en el cliente móvil.

Tabla 4.5. Validación de consulta y visualización de reportes. Fuente: Elaboración propia.

Escenario	Endpoint asociado	Respuesta esperada	Interpretación
Carga de reportes para mapa	obtener_reportes.php	JSON con coordenadas e información del reporte.	Permite construir marcadores dentro del mapa.
Carga de tarjetas o listado	card_reportes.php	JSON con título, descripción, ubicación o datos resumidos.	Permite presentar información legible al usuario.
Consulta de reportes propios	obtener_reportes_usuario.php	Listado filtrado por id_Usuario.	Permite verificar que los reportes quedan asociados a quien los generó.

4.7 Validación de solicitudes de amistad

El módulo de solicitudes de amistad fue validado revisando el ciclo de vida de una solicitud: envío, consulta, aceptación o eliminación. La lógica se apoya en la tabla Amigos, donde se relaciona el usuario que envía la solicitud con el usuario receptor y el estado de la relación.

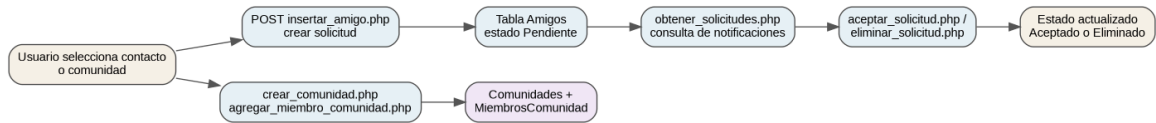


Figura 4.7. Relación entre solicitudes de amistad, comunidades y tablas asociadas. Fuente: Elaboración propia.

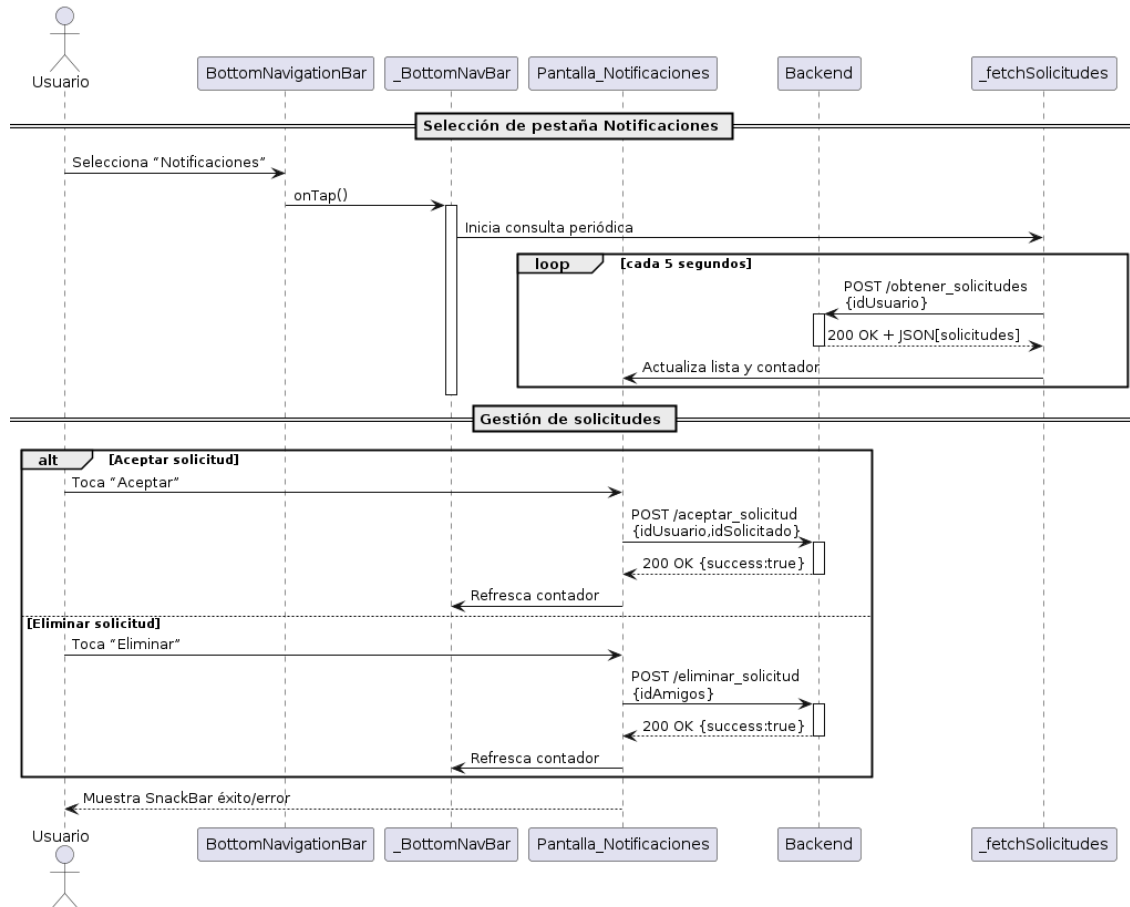


Figura 4.8. Flujo detallado de consulta, aceptación y eliminación de solicitudes. Fuente: Elaboración propia.

La validación consideró que la solicitud quedara inicialmente en estado pendiente y que, posteriormente, el backend permitiera actualizar su estado. Esta lógica permite diferenciar entre relaciones aún no confirmadas y relaciones aceptadas, evitando que todas las interacciones entre usuarios sean tratadas como una misma operación.

Tabla 4.6. Validación de solicitudes de amistad. Fuente: Elaboración propia.

Operación	Endpoint asociado	Tabla	Resultado validado
Enviar solicitud	insertar_amigo.php	Amigos	Se crea una relación entre usuario solicitante y usuario solicitado con estado pendiente.

Operación	Endpoint asociado	Tabla	Resultado validado
Consultar solicitudes	obtener_solicitudes.php	Amigos	Se devuelve al usuario receptor el listado de solicitudes recibidas.
Aceptar solicitud	aceptar_solicitud.php	Amigos	Se actualiza la relación para habilitar la conexión entre usuarios.
Eliminar o rechazar solicitud	eliminar_solicitud.php	Amigos	Se descarta la solicitud y se actualiza la vista de notificaciones.

4.8 Validación de comunidades y asociación de miembros

El módulo de comunidades fue validado como extensión de la relación entre usuarios. Su objetivo fue permitir que un usuario creara una comunidad y asociara otros integrantes a través de una tabla intermedia. Esta estructura evita duplicar información de usuarios y permite consultar qué miembros pertenecen a cada grupo.

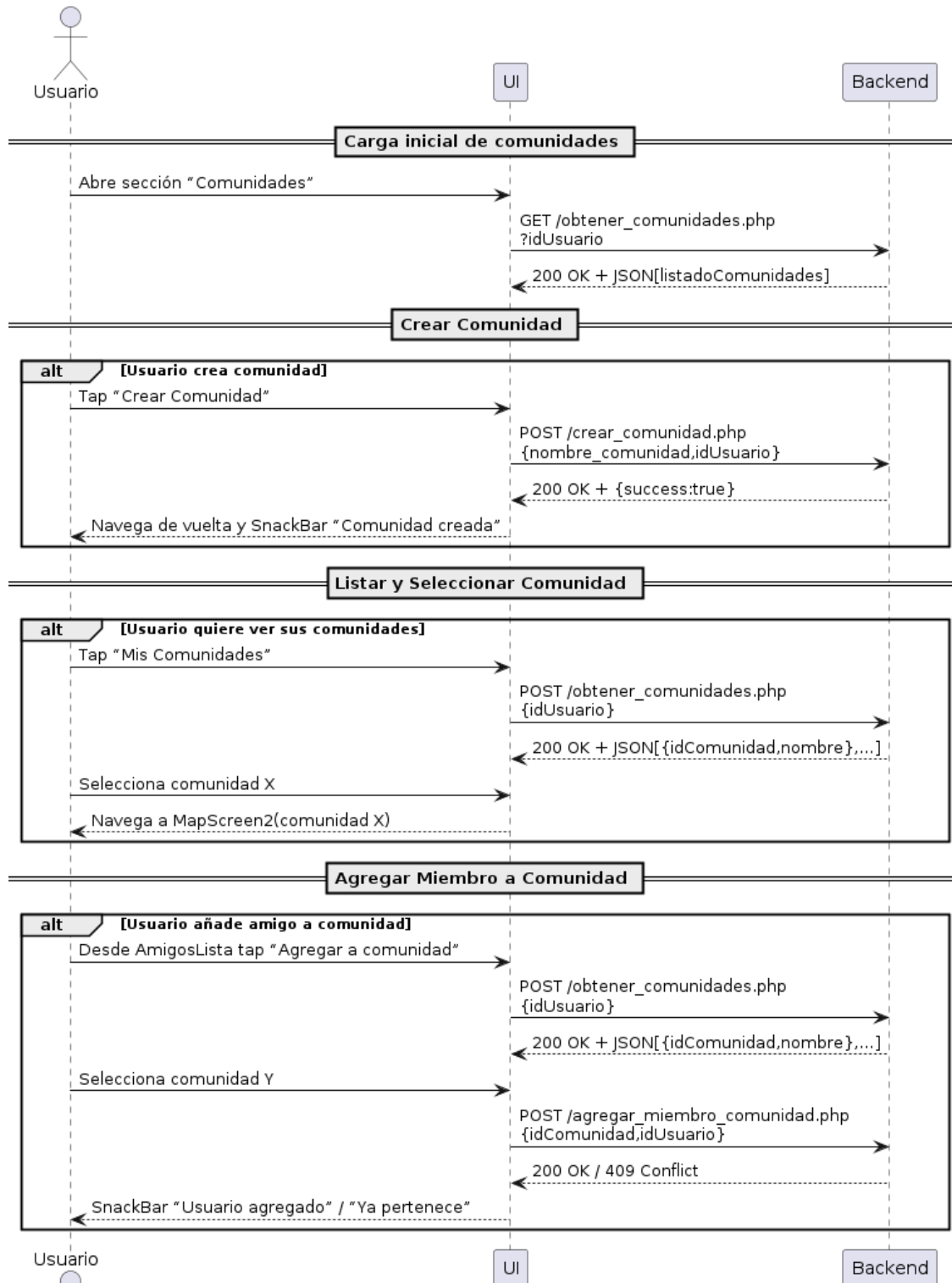


Figura 4.9. Flujo de creación, consulta y asociación de miembros a comunidades. Fuente: Elaboración propia.

La validación se apoyó en tres operaciones: creación de comunidad, consulta de comunidades asociadas al usuario y asociación de integrantes. Desde el punto de vista de base de datos, el funcionamiento se sostiene en las tablas Comunidades y Miembros_Comunidad, ambas

relacionadas con Usuario. Esta separación permite mantener la integridad de los datos y facilita futuras ampliaciones sin alterar la estructura central del sistema.

Tabla 4.7. Validación de comunidades y asociación de miembros. Fuente: Elaboración propia.

Escenario	Archivo o endpoint	Resultado obtenido	Interpretación
Crear comunidad	crear_comunidad.dart / crear_comunidad.php	Se registra una comunidad con nombre e id_Usuario creador.	El sistema permite crear grupos asociados a un usuario.
Listar comunidades	obtener_comunidades.php	Se devuelven comunidades vinculadas al usuario.	La app puede mostrar comunidades existentes sin consultar directamente MySQL.
Agregar miembro	agregar_miembro_comunid ad.php	Se inserta relación en Miembros_Comunidad.	La tabla intermedia permite representar pertenencia de usuarios a comunidades.

4.9 Comprobación de integración entre aplicación, backend y base de datos

Después de validar los módulos por separado, se comprobó la integración entre las capas principales del sistema. Esta comprobación se realizó relacionando cada pantalla Flutter con el endpoint PHP que recibía la solicitud y con la tabla MySQL donde se consultaba o persistía la información. Con ello se verificó que las pantallas no operaran de forma independiente, sino conectadas a servicios específicos del backend.

La integración se verificó observando tres elementos por cada flujo: el archivo de la aplicación que inicia la acción, el endpoint PHP que recibe la solicitud y la tabla involucrada en la persistencia o consulta. El resultado esperado era que cada operación retornara una respuesta interpretable por Flutter, principalmente mediante objetos o listas derivadas de JSON.

Tabla 4.8. Comprobación de integración entre aplicación, backend y base de datos. Fuente: Elaboración propia.

Flujo inicial	Aplicación móvil	Backend PHP	Base de datos	Resultado visible
Registro	Formulario en main.dart	register.php	Usuario	Cuenta creada o mensaje de error.
Inicio de sesión	Pantalla de login	login.php	Usuario	Ingreso a la aplicación con datos de usuario.
Crear reporte	form_reporte.dart	reporte.php	Reportes	Reporte disponible para consulta posterior.
Consultar reportes	map_screen.dart / report_screen.dart	obtener_reportes.php / card_reportes.php	Reportes	Marcadores o tarjetas actualizadas.
Solicitudes	notifications.dart / communities.dart	obtener_solicitudes.ph p / aceptar_solicitud.php	Amigos	Notificaciones y estados de solicitud actualizados.

Flujo inicial	Aplicación móvil	Backend PHP	Base de datos	Resultado visible
Comunidades	crear_comunidad.dart / seleccionar_comunida d.dart	crear_comunidad.php / obtener_comunidades .php	Comunidades / Miembros_Comunidad	Comunidades listadas o miembros asociados.

Durante la integración se comprobó que cada componente cumplía una función específica: Flutter enviaba y mostraba la información, PHP procesaba las solicitudes y MySQL almacenaba los datos. Esta organización facilitó la identificación de errores y permitió revisar cada parte del flujo por separado.

4.10 Validación de respuesta ante errores y recuperación del servicio

Además de los flujos exitosos, se validó el comportamiento esperado frente a errores de usuario o indisponibilidad temporal del servicio. En los módulos de autenticación y consulta, el backend podía retornar mensajes de error estructurados, permitiendo que la aplicación informara al usuario sin continuar el flujo como si la operación hubiese sido correcta.

Código 4.5. Ejemplos de respuestas estructuradas para manejo de errores.

```
{
  "status": "error",
  "message": "Credenciales incorrectas."
}

{
  "status": "error",
  "message": "Error interno del servidor. Intente más tarde."
}
```

Para la recuperación del servicio, la validación se basó en el reinicio controlado de la instancia del servidor y en la comprobación posterior de que la aplicación volviera a comunicarse con los endpoints. Durante la interrupción, la API quedaba temporalmente no disponible; luego de restablecer el servicio, la aplicación podía volver a realizar solicitudes. La persistencia de los datos dependía de MySQL en RDS, por lo que la información almacenada no quedaba ligada al estado visual de la aplicación móvil.

Tabla 4.9. Validación de respuesta ante errores y recuperación del servicio. Fuente: Elaboración propia.

Escenario	Resultado observado	Interpretación
Credenciales incorrectas	El backend responde con estado de error y mensaje.	La aplicación puede detener el ingreso y mostrar una advertencia.
Fallo temporal del servidor	La API queda momentáneamente no disponible.	El sistema depende de la disponibilidad del servidor para responder solicitudes.

Escenario	Resultado observado	Interpretación
Restablecimiento del servicio	La aplicación vuelve a comunicarse con los endpoints.	La persistencia en base de datos permite retomar operaciones sin reconstruir los datos.

La prueba permitió comprobar que, después de una interrupción temporal del servidor, la aplicación podía volver a comunicarse con los endpoints una vez restablecido el servicio. Como mejora futura, sería necesario incorporar monitoreo para detectar este tipo de fallas y reducir el tiempo de recuperación.

4.11 Pruebas de carga del backend

Las pruebas de carga permitieron evaluar el comportamiento del backend bajo distintos niveles de concurrencia. Para ello, se utilizó Apache JMeter sobre el flujo de inicio de sesión, considerando como métricas principales el tiempo de respuesta, la tasa de error y el uso de recursos del servidor. Esta prueba no buscó demostrar una escalabilidad ilimitada, sino identificar el rango en que el prototipo mantenía un comportamiento aceptable y el punto desde el cual comenzaba a degradarse.

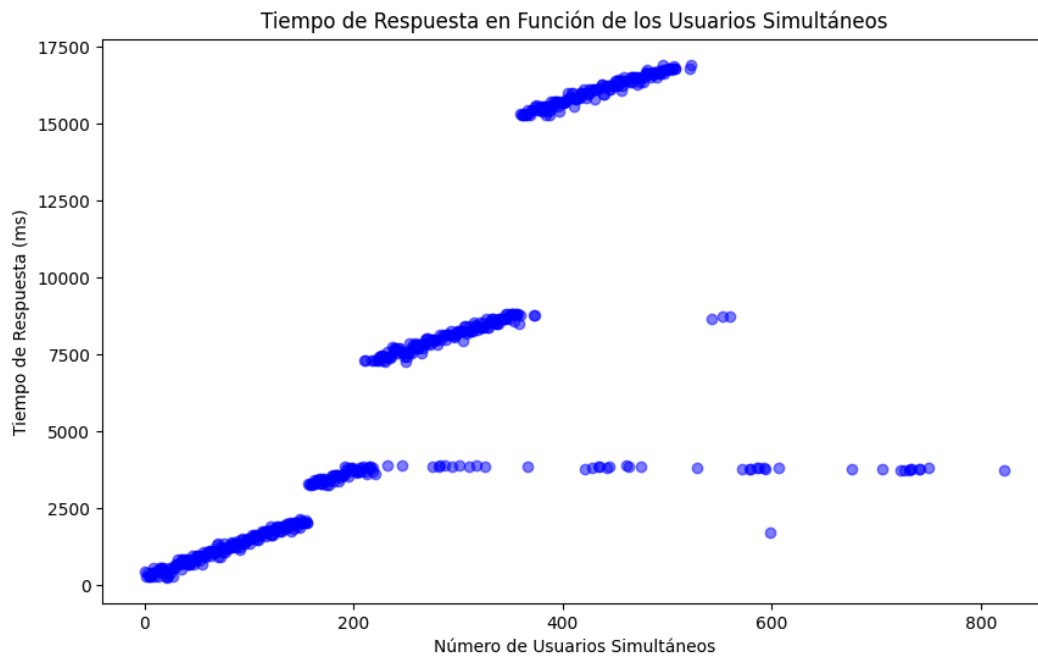


Figura 4.10. Tiempo de respuesta en función del número de usuarios simultáneos. Fuente: Elaboración propia.

El comportamiento observado muestra que, con cargas bajas y medias, el tiempo de respuesta se mantuvo dentro de un rango aceptable para el contexto del prototipo. Sin embargo, al aumentar la concurrencia, el tiempo de respuesta creció de manera progresiva. Esto indica que el backend podía responder adecuadamente bajo demanda moderada, pero requería optimización adicional para escenarios de uso masivo.

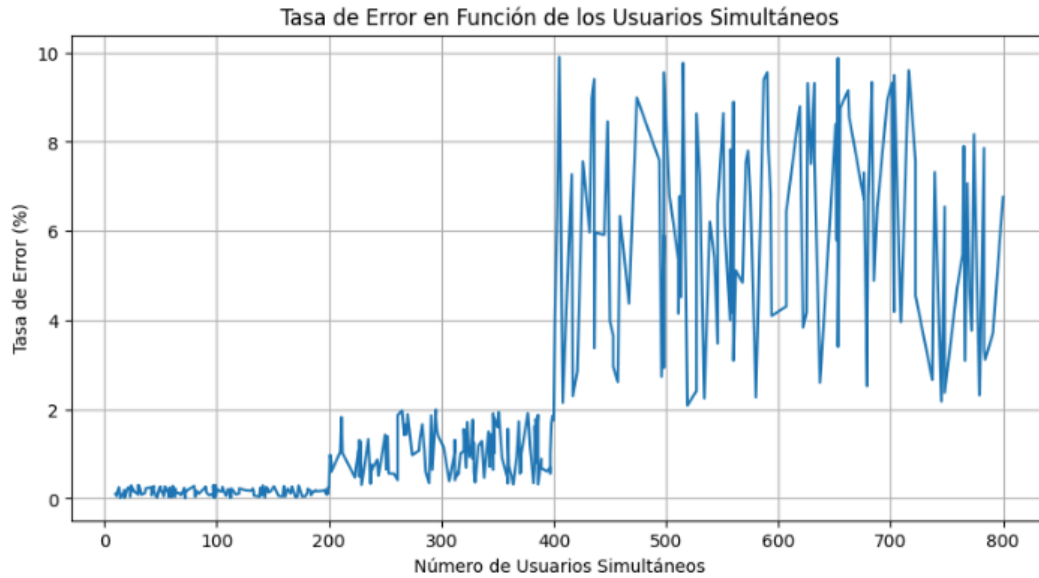


Figura 4.11. Tasa de error observada durante la prueba de carga. Fuente: Elaboración propia.

La tasa de error se mantuvo baja en los niveles iniciales de concurrencia y aumentó cuando el número de usuarios simultáneos superó la capacidad estable del entorno probado. Esta tendencia es coherente con el aumento del tiempo de respuesta, ya que la saturación del servidor incrementa la probabilidad de fallas o respuestas tardías.

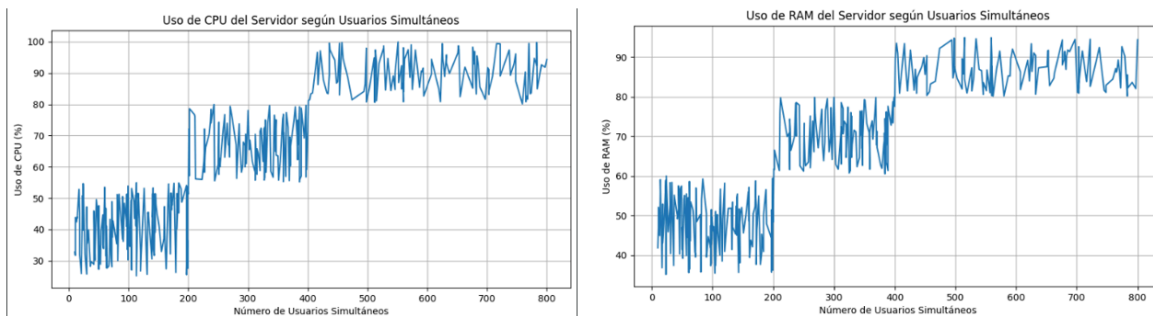


Figura 4.12. Uso de CPU y RAM del servidor durante la prueba de carga. Fuente: Elaboración propia.

El uso de CPU y RAM aumentó a medida que creció la cantidad de usuarios simultáneos. En los niveles de mayor carga, ambos recursos se aproximaron a zonas de saturación, lo que explica la degradación observada en los tiempos de respuesta y en la tasa de error. En consecuencia, el resultado de la prueba permite afirmar que el prototipo fue funcional bajo cargas moderadas, pero que requiere ajustes de infraestructura y optimización para escenarios de concurrencia elevada.

Tabla 4.10. Síntesis de métricas de pruebas de carga. Fuente: Elaboración propia.

Métrica	Comportamiento observado	Interpretación
Tiempo de respuesta	Se mantiene controlado en cargas bajas y medias, aumentando con concurrencias altas.	El backend responde adecuadamente en el rango esperado del prototipo, pero se degrada bajo alta demanda.
Tasa de error	Baja en cargas iniciales y creciente al acercarse a la saturación.	La estabilidad disminuye cuando el servidor supera su capacidad de procesamiento.
CPU	Aumenta progresivamente hasta niveles altos.	El procesamiento del servidor se convierte en un factor limitante.
RAM	Crece junto con el número de usuarios simultáneos.	La memoria disponible condiciona la capacidad de atender múltiples solicitudes.

4.12 Síntesis de la validación

La validación realizada permitió comprobar que los módulos principales de VisionSecurity mantienen una relación funcional entre la aplicación móvil, el backend PHP y la base de datos MySQL. Los flujos de registro, inicio de sesión, reportes, consulta de reportes, solicitudes de amistad y comunidades presentan una ruta técnica clara: una acción del usuario genera una solicitud HTTP, el servidor procesa parámetros, la base de datos almacena o devuelve información y Flutter actualiza la interfaz mediante la respuesta recibida.

La validación permitió comprobar que los componentes principales del sistema se integraron correctamente. Las evidencias de código, los endpoints, los diagramas y la estructura de la base de datos muestran que el prototipo contaba con comunicación real entre la aplicación móvil, el backend y MySQL.

Los resultados de carga muestran un comportamiento aceptable en rangos moderados de concurrencia, pero también evidencian limitaciones ante usuarios simultáneos elevados. Por ello, el prototipo valida la arquitectura propuesta sin afirmar preparación para producción masiva, dejando como mejoras futuras el rendimiento, el monitoreo, la robustez de infraestructura y la optimización de consultas.

CAPÍTULO 5: CONCLUSIONES

El desarrollo de VisionSecurity permitió construir y validar un prototipo basado en una arquitectura cliente-servidor compuesta por una aplicación móvil en Flutter, un backend PHP y una base de datos MySQL alojada en AWS RDS. Desde el enfoque de esta memoria, el principal resultado fue la implementación de la capa backend y su integración con la aplicación móvil mediante solicitudes HTTP y respuestas JSON.

Respecto a los objetivos definidos, se logró organizar los requerimientos principales del sistema, definir una arquitectura de tres capas y construir una base de datos relacional capaz de representar usuarios, reportes, comunidades, solicitudes de amistad, ubicaciones y miembros de comunidad. El uso del identificador `id_Usuario` permitió vincular la autenticación con acciones posteriores, como crear reportes, consultar información, gestionar solicitudes y asociar usuarios a comunidades.

El backend PHP permitió centralizar la lógica de servidor mediante endpoints asociados a funciones específicas, tales como registro, inicio de sesión, creación y consulta de reportes, solicitudes de amistad, comunidades y ubicación de usuarios. Esta estructura evitó que la aplicación móvil accediera directamente a la base de datos y permitió mantener una separación clara entre cliente, procesamiento y persistencia.

La integración entre Flutter, PHP y MySQL fue validada mediante trazabilidad técnica entre pantallas, endpoints, tablas y respuestas JSON. Esta validación permitió comprobar que los módulos principales no funcionaban como elementos aislados, sino como flujos completos donde una acción del usuario generaba una solicitud HTTP, el backend procesaba los parámetros, la base de datos almacenaba o recuperaba información y la aplicación actualizaba su interfaz con la respuesta obtenida.

Las pruebas funcionales permitieron validar los módulos de registro, inicio de sesión, reportes, consulta de reportes, solicitudes de amistad y comunidades. Además, las pruebas de recuperación mostraron que el sistema podía retomar la comunicación después de una interrupción temporal del servicio, aunque esta comprobación no equivale a una validación de alta disponibilidad empresarial.

Las pruebas de carga realizadas con Apache JMeter permitieron observar el comportamiento del backend bajo distintos niveles de concurrencia. Los resultados muestran que el prototipo mantiene un comportamiento aceptable en cargas bajas y medias, pero presenta degradación en tiempos de respuesta, tasa de error y uso de recursos cuando la concurrencia aumenta. Por ello, la arquitectura se considera válida para el contexto de prototipo académico, pero no debe interpretarse como una solución preparada para operación masiva sin mejoras adicionales.

Entre las principales limitaciones se encuentran la ausencia de evaluación prolongada con usuarios finales, la falta de monitoreo productivo, la necesidad de robustecer mecanismos de seguridad y la falta de optimizaciones avanzadas para alta concurrencia. Si bien se documentó evidencia de implementación de consultas parametrizadas y hash de contraseñas, no se realizaron pruebas formales de penetración ni una auditoría especializada de seguridad. En una versión futura sería recomendable incorporar monitoreo de recursos, respaldo formal de

configuración, optimización de consultas SQL, revisión de índices, gestión avanzada de sesiones, mecanismos de despliegue más robustos y estrategias de escalamiento controlado.

En síntesis, VisionSecurity permitió demostrar la viabilidad técnica de una solución móvil con backend PHP, base de datos MySQL en AWS RDS e integración HTTP/JSON para reportes geolocalizados y módulos de interacción entre usuarios. El aporte principal de esta memoria se concentra en la construcción, integración y validación técnica de la capa backend y sus flujos de comunicación con la aplicación móvil, dejando identificadas las mejoras necesarias para una evolución futura del sistema.

REFERENCIAS BIBLIOGRÁFICAS

- Amazon Web Services. (s. f.). Amazon EC2 documentation. <https://aws.amazon.com/ec2/>
- Amazon Web Services. (s. f.). Amazon RDS documentation. <https://aws.amazon.com/rds/documentation/>
- Burns, E., & Carter, S. (2019). Building scalable web applications with AWS. O'Reilly Media.
- Erl, T., Mahmood, Z., & Puttini, R. (2013). Cloud computing: Concepts, technology & architecture. Prentice Hall.
- Flutter. (s. f.). Flutter documentation. <https://flutter.dev/docs>
- Goodchild, M. F., & Li, L. (2012). Big data in geographic information systems. *Annals of GIS*, 18(2), 89–98.
- Hernández, J., Martínez, R., & Silva, T. (2024). Real-time geolocation and emergency response: A technological review. *International Journal of Public Safety*, 12(3), 89–102.
- Longley, P. A., Goodchild, M. F., Maguire, D. J., & Rhind, D. W. (2015). *Geographic information systems and science* (4.^a ed.). Wiley.
- Miller, T., & Rowe, C. (2022). Community Engagement through Mobile Reporting: An Analysis of Snap Send Solve. *Australian Journal of Public Administration*, 81(4), 601–617.
- Ministerio del Interior. (2024). Encuesta nacional de seguridad pública 2024. Gobierno de Chile. <http://www.ministeriodelinterior.cl>
- Mingaco Informativo. (2024). Delincuencia y migración irregular: Una opinión desde las estadísticas. <http://www.mingacoinformativo.cl>
- Municipalidad de Colina. (s. f.). Municipalidad de Colina. <https://www.colina.cl/>
- OWASP Foundation. (2023). OWASP Top 10: The 10 most critical web application security risks. <https://owasp.org>
- Prajapati, R. (2019). Flutter for beginners. Packt Publishing.
- Rao, A., & Bhalla, S. (2020). REST APIs and their impact on modern application development. *Journal of Software Engineering*.
- Richardson, L., Amundsen, M., & Ruby, S. (2013). *RESTful Web APIs*. O'Reilly Media.
- Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database system concepts* (7.^a ed.). McGraw-Hill.
- SOSAFE. (s. f.). SOSAFE. <https://www.sosafeapp.com/>
- Subsecretaría de Prevención del Delito. (s. f.). Denuncia Seguro. <https://www.denunciaseguro.cl/>
- Vigiapp. (s. f.). Aplicación de seguridad para tu municipalidad. <https://vigiapp.cl/>

Zhou, X., Wu, H., & Su, X. (2020). Geospatial technologies in disaster management: A case study. *Journal of Disaster Risk Studies*, 12(3), 23–34.