INGENIERIA DE SOFTWARE
Semestre: 2025-2
Equipo: BUMPER
Fecha: 06 de Abril de 2025

Especificación de Requerimientos

Índice

1. Introducción

Proyecto: **Bumper**

- Aplicación web para el registro y visualización de incidentes urbanos.
- Bumper planea ser una herramienta colaborativa para mejorar comunicación ciudadana.
- Permitirá a ciudadanos, comerciantes y organizaciones vecinales participar activamente en seguimiento de reportes.

Requisitos del cliente

- **Registro de incidentes**
  - Permitir a los usuarios marcar la ubicación del incidente en un mapa interactivo.
  - Adjuntar una o más fotografías que respalden el incidente reportado.
  - Registrar una breve descripción del problema.
- **Actualización de estado del incidente**

- Permitir que cualquier usuario (no solo el creador del reporte) actualice el estado del incidente a Resuelto", siempre que adjunte pruebas fotográficas que validen la solución.

- **Visualización de incidentes**
  - Mostrar todos los incidentes en un mapa interactivo, categorizados por tipo y estado (Reportado, En proceso, Resuelto).

2. Benchmark

Este paso lo hemos completado para conocer las **soluciones actuales** en el mundo, conocer las fortalezas, debilidades, las formas de trabajo, conocer de sus interfaces y procesos. Con esta información tendremos un panorama de lo que ya existe y podremos hacer un camino de usuario para lograr una experiencia más amigable de nuestra aplicación.

	FixMyStreet (Reino Unido)	**SeeClickFix (EE.UU.)**	**Mejora tu Ciudad**	**Colab.re (Brasil)**
Marcación en mapa	Sí (OpenStreetMap)	Sí (Google Maps)	Sí (Google Maps)	Sí (Leaflet/OpenStreetMap)
Adjuntar fotos	Sí (1+ imágenes)	Sí (hasta 5 imágenes)	Sí (fotos y videos)	Sí (3+ imágenes)
Descripción breve	Texto libre	Campos predefinidos + texto	Texto libre	Texto + etiquetas
Actualización de estado	No (solo creador/autoridades)	Sí (usuarios verificados)	Sí (usuarios y autoridades)	Sí (usuarios registrados)
Visualización de incidentes	Mapa con filtros	Mapa interactivo con capas	Mapa con íconos	Mapa con filtros y heatmaps
API del mapa	OpenLayers + OpenStreetMap	Google Maps API + Esri	Google Maps API	Leaflet API + OpenStreetMap
Tecnología	Perl, PostgreSQL, OpenLayers	React, Node.js, AWS	MySQL/Firebase	JavaScript, Python, PostgreSQL
Disponibilidad	Reino Unido, Australia	EE.UU., Canadá	España, México	Brasil, Argentina, Uruguay

Cuadro 1: Comparativa de plataformas de gestión de incidentes urbanos

Enlaces a las páginas

- FixMyStreet (Reino Unido)
- SeeClickFix (EE.UU.)
- Mejora tu Ciudad (España)
- Colab.re (Brasil)

	FixMyStreet (UK)	SeeClickFix (EEUU)	Mejora tu Ciudad (ES)	Colab.re (BR)
Fortalezas	Integración autoridades Open-source PostgreSQL robusto	Interfaz intuitiva Colaboración real-time Escalabilidad AWS	Multimedia completo Flexibilidad descripciones	Visualización avanzada Stack moderno Foco LATAM
Debilidades	Actualizaciones limitadas Alcance reducido	APIs pagas Límite imágenes	Tecnología oculta Cobertura indefinida	Baja visibilidad global Registro obligatorio
Base de datos	PostgreSQL	AWS RDS (MySQL/PG)	MySQL/Firebase	PostgreSQL
Propuesta de valor	Comunicación simple ciudadano-gobierno	Transparencia mediante reportes verificados	Empoderamiento ciudadano con evidencia visual	Soluciones basadas en datos LATAM

Cuadro 2: Comparativa de puntos específicos

Conclusiones

Basado en el análisis de la competencia, se identificaron prácticas clave que se aplicarán en *Bumper*:

- En FixMyStreet y Colab.re, los filtros en el mapa mejoran la navegación; implementaremos filtros por estatus en React con Leaflet.
- SeeClickFix permite actualizar el estatus con usuarios verificados; habilitaremos esta función para el creador del reporte vía Spring Boot.
- Mejora tu Ciudad destaca por subir multimedia; integraremos la carga de 1-3 fotos, almacenadas en Supabase o el servidor.
- Colab.re usa Leaflet y OpenStreetMap con éxito; adoptaremos esta API gratuita para el mapa interactivo.
- Identificamos en las vistas que componentes como barras laterales y dropmenus mejorar la experiencia de los usuarios al realizar reportes.

Estas decisiones aprovechan las fortalezas de la competencia, adaptándolas a nuestra pila tecnológica (Spring Boot, React, PostgreSQL) para una solución eficiente y diferenciada.

3. Requerimientos

Requerimientos Funcionales

1. **Autenticación de Usuarios (RF1)**
    - El usuario puede registrarse e iniciar sesión con un correo electrónico y contraseña.

- Solo los usuarios autenticados pueden crear y modificar reportes.

2. **Gestión de Incidentes (RF2)**
   - El usuario puede crear un reporte con ubicación, tipo de incidente, descripción y hasta 3 fotos.
   - El usuario creador puede actualizar el estatus del reporte (“No resuelto”, “En proceso”, “Resuelto”).
   - Los reportes incluyen un código único para seguimiento.
3. **Integración con Mapa Interactivo (RF3)**
   - El usuario visualiza un mapa con marcadores de incidentes existentes.
   - El usuario puede marcar una ubicación en el mapa al crear un reporte, con opción de escribir una dirección.
   - El mapa permite zoom y arrastre para explorar áreas.
4. **Filtrado de Incidentes (RF4)**
   - El usuario puede filtrar incidentes en el mapa por estatus (“Todos”, “No resuelto”, etc.) y tipo de incidente (e.g., Bache, Basura).
5. **Visualización de Información (RF5)**
   - El usuario ve una pantalla principal con el mapa y un botón para reportar.
   - Al hacer clic en un marcador, se muestra una ventana con fotos, estatus y tipo de incidente.
   - Una barra lateral ofrece el formulario para crear reportes y opciones de filtrado.

Requerimientos No Funcionales

1. **Seguridad (RNF1)**
   - Las contraseñas y correos de los usuarios están protegidos con cifrado.
   - Solo el creador puede modificar su reporte, garantizando control de acceso.
2. **Rendimiento (RNF2)**
   - El mapa carga rápidamente y responde con baja latencia al interactuar.
   - La aplicación soporta múltiples usuarios creando reportes al mismo tiempo.
3. **Disponibilidad (RNF3)**
   - La solución funciona inicialmente en un entorno local o en un servidor básico.
4. **Usabilidad (RNF4)**
   - La interfaz se adapta a diferentes dispositivos (móviles y escritorio).
   - El diseño es claro, con navegación sencilla y acciones intuitivas.
5. **Escalabilidad (RNF5)**
   - La aplicación permite añadir nuevas funcionalidades sin rediseños complejos.
   - El almacenamiento de datos y fotos puede crecer según la demanda.

4. Casos de Uso

CU1: Hacer Login

Actores: Usuario, Sistema
Flujo Principal:

1. El usuario accede a la página de inicio de sesión. (RF1)
2. Ingresa su correo electrónico y contraseña.
3. El sistema verifica las credenciales.
4. Si las credenciales son correctas, el usuario es redirigido al mapa principal. (RF5)

Flujo Alternativo:

- *Credenciales incorrectas:*
    1. El sistema muestra un mensaje de error indicando que las credenciales no son válidas.
    2. Ofrece un enlace para recuperar la contraseña por correo.

CU2: Reportar Incidente Urbano

Actores: Ciudadano, Sistema
Flujo Principal:

1. El usuario inicia sesión. (RF1)
2. Hace clic en “Reportar Incidente” desde el mapa principal. (RF3)
3. Marca la ubicación en el mapa o escribe una dirección. (RF3)
4. Completa un formulario con tipo de incidente, descripción y hasta 3 fotos. (RF2)
5. Confirma el reporte tras ver una vista previa.
6. El sistema muestra el reporte en el mapa con un código único. (RF5)

Flujo Alternativo:

- *Sin inicio de sesión:*
    1. El sistema redirige al usuario a la página de inicio de sesión antes de continuar.

CU3: Modificar Estatus de Reporte

Actores: Ciudadano (creador del reporte), Sistema
Flujo Principal:

1. El usuario inicia sesión y selecciona uno de sus reportes en el mapa. (RF2)
2. Abre la ventana del reporte y elige un nuevo estatus (“En proceso” o “Resuelto”). (RF2)

3. Confirma el cambio.
4. El sistema actualiza el estatus y muestra la versión modificada en el mapa. (RF5)

Flujo Alternativo:

- *Usuario no creador:*
    1. El sistema no muestra opciones de edición y limita la interacción a solo visualización.

CU4: Ver Detalles de un Incidente

Actores: Usuario (registrado o invitado), Sistema
Flujo Principal:

1. El usuario accede al mapa principal. (RF3)
2. Hace clic en un marcador de incidente en el mapa.
3. El sistema muestra una ventana emergente con los detalles del incidente, incluyendo:
    - Tipo de incidente.
    - Ubicación.
    - Estado actual (Pendiente, En proceso, Resuelto).
    - Fotos asociadas al incidente.
    - Fecha y hora del reporte.

Flujo Alternativo:

- *Incidente sin fotos:*
    1. El sistema muestra un mensaje indicando que no hay fotos disponibles para este incidente.

CU5: Editar Perfil de Usuario

Actores: Usuario, Sistema
Flujo Principal:

1. El usuario inicia sesión en el sistema. (RF1)
2. Accede a la sección de ”Perfil”desde el menú principal. (RF10)
3. Visualiza su información actual, como nombre, apellido, correo electrónico y contraseña.
4. Realiza cambios en los campos permitidos (por ejemplo, nombre o contraseña).
5. Confirma los cambios haciendo clic en ”Guardar”. (RF11)
6. El sistema valida los datos ingresados y actualiza la información en la base de datos.
7. El sistema muestra un mensaje de confirmación indicando que los cambios se han guardado correctamente.

Flujo Alternativo:

- *Datos inválidos:*

  1. El sistema muestra un mensaje de error indicando qué campos necesitan corrección.
  2. El usuario corrige los datos y vuelve a intentar guardar los cambios.

5. Tecnología Usada

Backend

- **Lenguaje de Programación: Kotlin**
  - *Razón de uso:* Kotlin es un lenguaje moderno, conciso y seguro que mejora la productividad del desarrollo al reducir el código repetitivo y minimizar errores comunes (como null pointer exceptions). Su interoperabilidad con Java permite aprovechar bibliotecas existentes, mientras que su sintaxis clara facilita el mantenimiento del código.
  - *Ventaja para Bumper:* Ideal para implementar una API RESTful robusta (RF2, RF3) que gestione autenticación (RF1) y modificaciones de reportes (CU3), asegurando un backend eficiente y fácil de escalar (RNF5).
- **Framework: Spring Boot**
  - *Razón de uso:* Spring Boot simplifica la creación de aplicaciones backend con configuraciones automáticas, integración nativa con bases de datos y soporte para seguridad. Su arquitectura basada en microservicios permite manejar solicitudes simultáneas con baja latencia (RNF2).
  - *Ventaja para Bumper:* Facilita la gestión de reportes (RF2) y la autenticación de usuarios (CU1), ofreciendo un entorno estable y modular que puede crecer con nuevas funcionalidades, como notificaciones o integración con autoridades (RNF5).

Frontend

- **React**
  - *Razón de uso:* React es una biblioteca de JavaScript que permite construir interfaces dinámicas y responsivas (RNF4) mediante componentes reutilizables. Su enfoque en el estado y la renderización eficiente soporta interacciones en tiempo real, como filtros en el mapa (RF4) y vistas previas de reportes (CU2).
  - *Ventaja para Bumper:* Perfecto para integrar un mapa interactivo (RF3) y una barra lateral dinámica (CU4), asegurando una experiencia de usuario intuitiva y adaptable a dispositivos móviles (RNF4). Además, su ecosistema (e.g., react-leaflet) simplifica la integración con mapas.

Base de Datos

- **PostgreSQL en Supabase**

- *Razón de uso:* PostgreSQL es un sistema de base de datos relacional robusto y de código abierto, ideal para almacenar datos estructurados como reportes con ubicaciones, estatus y fotos (RF2). Supabase añade una capa de facilidad con almacenamiento de archivos (fotos) y autenticación integrada, reduciendo la complejidad inicial.
- *Ventaja para Bumper:* Garantiza la persistencia de los reportes (CU2, CU3) y permite consultas rápidas para filtros (RF4), con escalabilidad para soportar más usuarios y datos (RNF5). El uso de Supabase ofrece un nivel gratuito inicial, optimizando costos para el prototipo y el que este activo siempre nos da cierta ventaja.

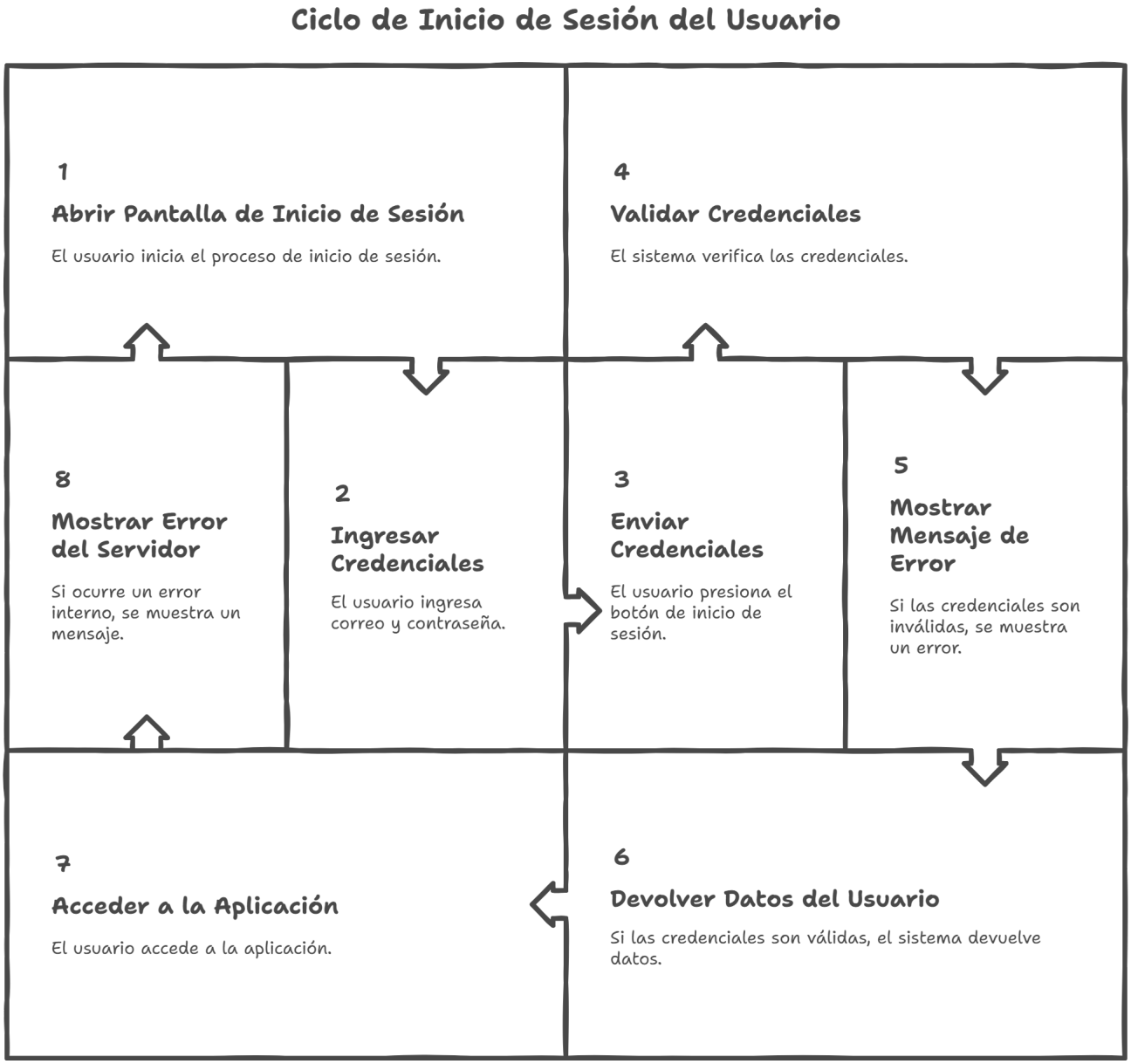
API para Mapas

- **Leaflet con OpenStreetMap (Opción Principal)**
  - *Razón de uso:* Leaflet es una biblioteca ligera y de código abierto para mapas interactivos, combinada con OpenStreetMap, una fuente de datos gratuita y global. Esto permite visualizar y marcar incidentes (RF3) sin costos asociados, a diferencia de APIs comerciales.
  - *Ventaja para Bumper:* Su integración con React (via react-leaflet) asegura un mapa fluido y personalizable (CU4), con filtros dinámicos (RF4) y baja latencia (RNF2). Es ideal para un MVP escalable y económico (RNF5), con soporte para marcadores y popups (CU2, CU3).

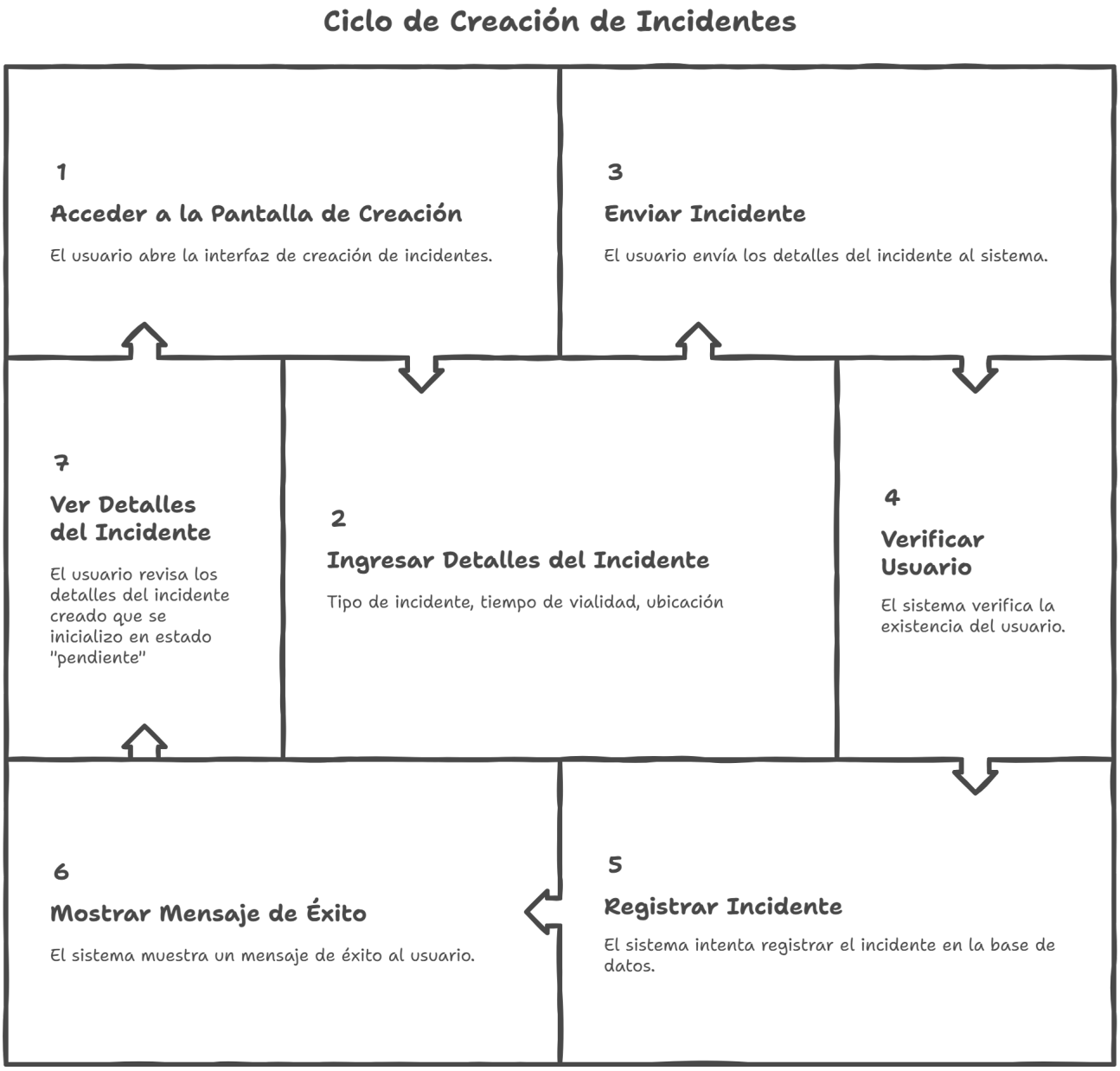
6. Diagramas

Registro Usuario

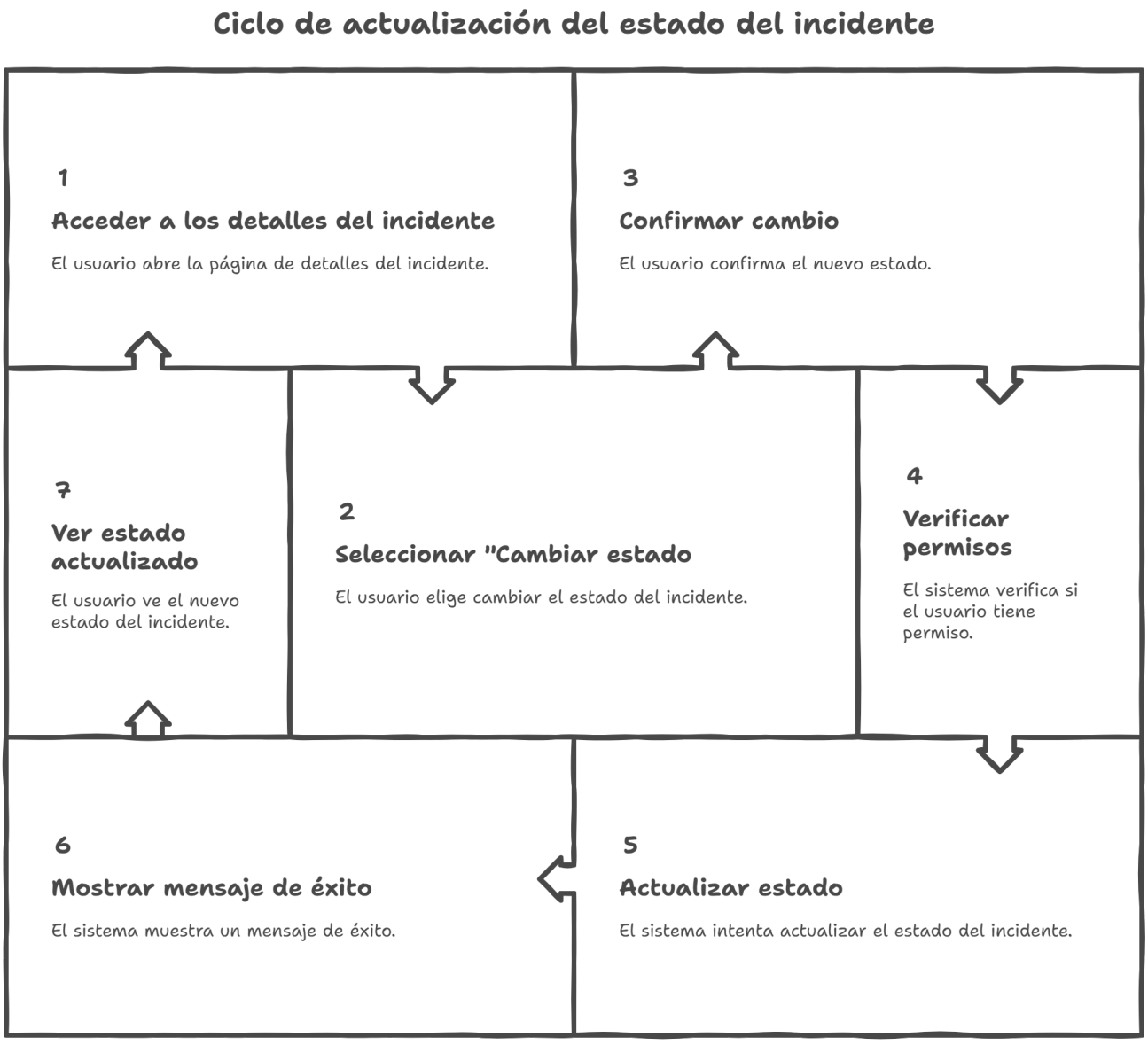
Iniciar Sesión



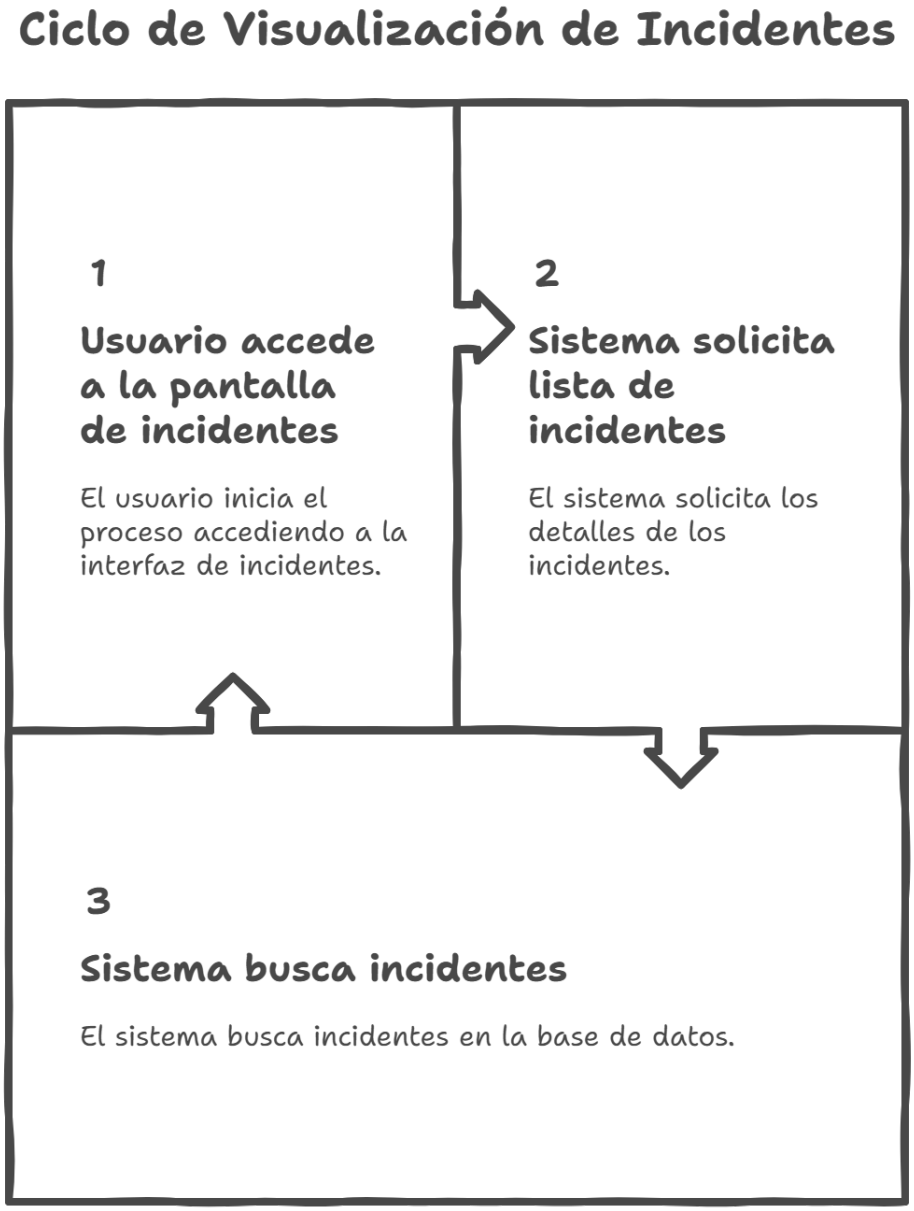
Crear Incidente



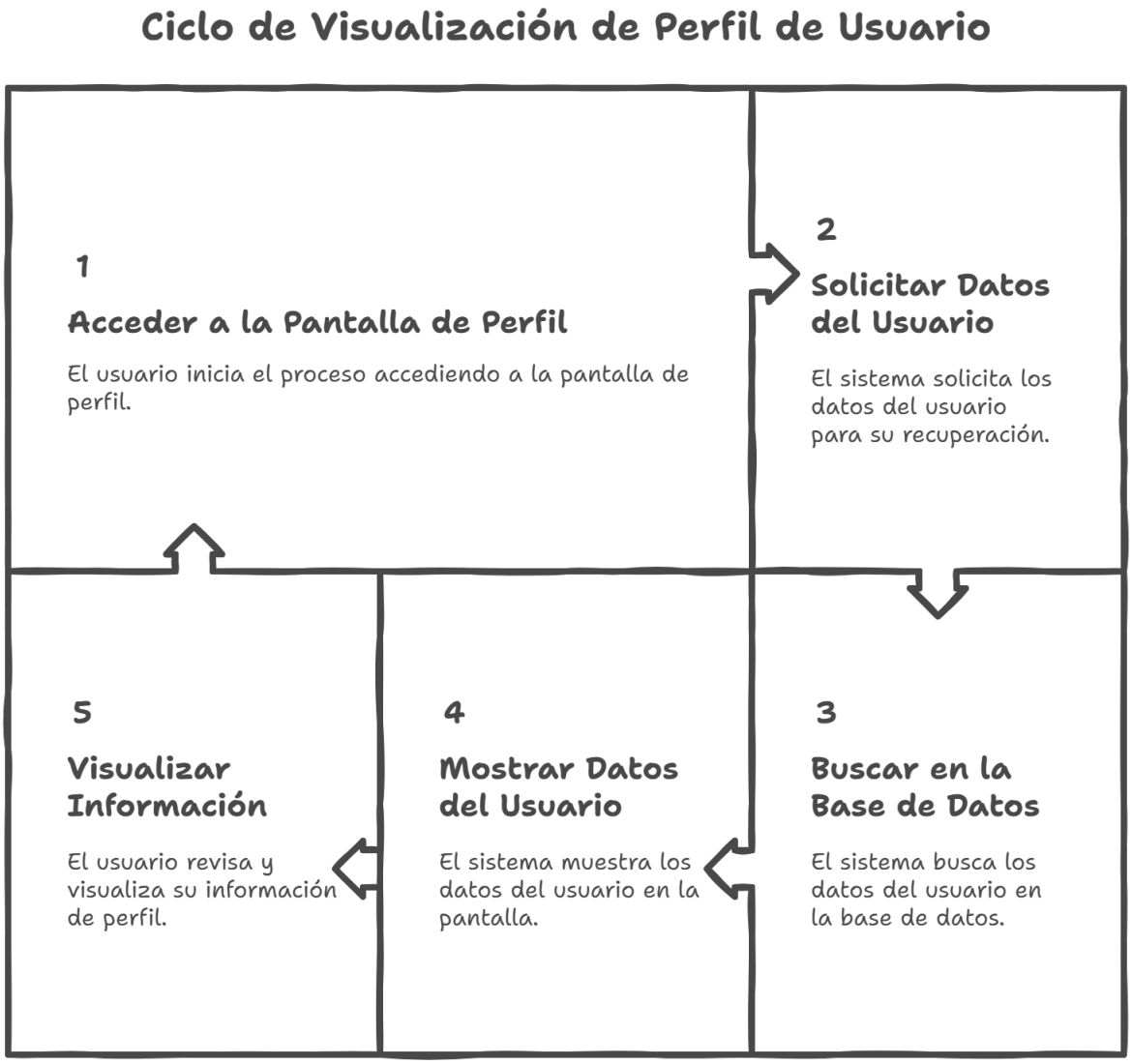
Cambiar estado incidente



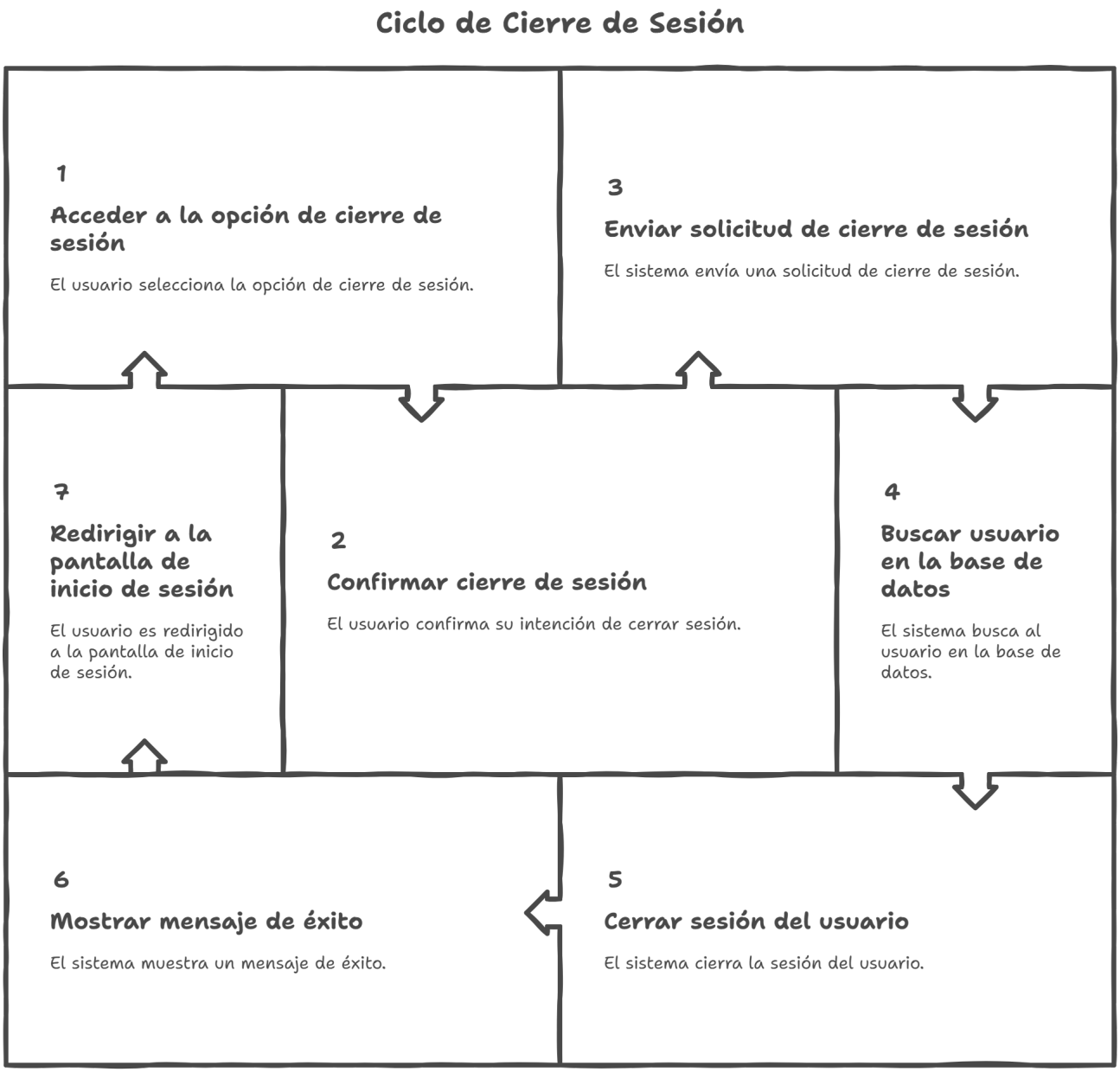
Mostar incidentes



Ver perfil


Ciclo de Visualización de Perfil de Usuario
1
Acceder a la Pantalla de Perfil
El usuario inicia el proceso accediendo a la pantalla de perfil.
2
Solicitar Datos del Usuario
El sistema solicita los datos del usuario para su recuperación.
3
Buscar en la Base de Datos
El sistema busca los datos del usuario en la base de datos.
4
Mostrar Datos del Usuario
El sistema muestra los datos del usuario en la pantalla.
5
Visualizar Información
El usuario revisa y visualiza su información de perfil.

Cerrar sesión



Documentación de la API REST

Endpoints de Usuarios

1. Registrar Usuario

Endpoint

POST `/v1/users/create`
Registra un nuevo usuario en el sistema.

Request

URL: `http://localhost:8080/v1/users/create`
Método: `POST`
Headers: `Content-Type: application/json`
Body:

```
{
    "nombre": "Juan",
    "apellido": "Pérez",
    "correo": "juan@example.com",
    "password": "secreto123"
}
```

Response

Respuesta Exitosa HTTP 201

```
{
    "mensaje": "Usuario registrado exitosamente",
    "usuario": {
        "id": 1,
        "nombre": "Juan",
        "apellido": "Pérez",
        "correo": "juan@example.com",
        "token": "inactivo",
        "numeroIncidentes": 0,
        "fechaRegistro": "2025-04-26T10:30:00"
    }
}
```

Errores

- HTTP 400 - Correo electrónico inválido
- HTTP 500 - Error interno del servidor

2. Iniciar Sesión

Endpoint

POST `/v1/users/login`
Inicia sesión de un usuario en el sistema.

Request

URL: `http://localhost:8080/v1/users/login`
Método: `POST`
Headers: `Content-Type: application/json`
Body:

```
{
    "correo": "juan@example.com",
    "password": "secreto123"
}
```

Response

Respuesta Exitosa ✔ HTTP 200

```
{
    "mensaje": "Login exitoso",
    "usuario": {
        "id": 1,
        "nombre": "Juan",
        "apellido": "Pérez",
        "correo": "juan@example.com",
        "token": "activo",
        "numeroIncidentes": 0,
        "fechaRegistro": "2025-04-26T10:30:00"
    }
}
```

Errores

- HTTP 401 - Credenciales inválidas
- HTTP 500 - Error en el servidor

3. Cerrar Sesión

Endpoint

POST `/v1/users/logout`
Cierra la sesión de un usuario.

Request

URL: `http://localhost:8080/v1/users/logout`
Método: `POST`
Headers: `Content-Type: application/json`
Body:

```
{
    "correo": "juan@example.com"
}
```

Response

Respuesta Exitosa ✔ HTTP 200

```
{
    "mensaje": "Sesión cerrada correctamente"
}
```

Errores

- HTTP 404 - Usuario no encontrado
- HTTP 500 - Error al cerrar sesión

4. Obtener Usuario por Correo

Endpoint

GET `/v1/users/correo/{correo}`
Obtiene los datos de un usuario por su correo electrónico.

Request

URL: `http://localhost:8080/v1/users/correo/juan@example.com`
Método: `GET`

Response

Respuesta Exitosa ✔ HTTP 200

```
{
    "mensaje": "Usuario encontrado",
    "usuario": {
        "id": 1,
        "nombre": "Juan",
        "apellido": "Pérez",
        "correo": "juan@example.com",
        "token": "activo",
        "numeroIncidentes": 0,
        "fechaRegistro": "2025-04-26T10:30:00"
    }
}
```

Errores

- HTTP 404 - Usuario no encontrado
- HTTP 500 - Error al buscar usuario

⚠ Endpoints de Incidentes

5. Crear Incidente

Endpoint

POST `/v1/incidentes/create`
Registra un nuevo incidente en el sistema.

Request

URL: `http://localhost:8080/v1/incidentes/create`
Método: `POST`
Headers: `Content-Type: application/json`
Body:

```
{
    "usuarioId": 1,
    "tipoIncidente": "BACHES",
    "ubicacion": "Av. Principal 123",
    "latitud": 19.4326,
    "longitud": -99.1332,
    "tipoVialidad": "AVENIDA"
}
```

Response

Respuesta Exitosa ✔ HTTP 201

```
{
    "mensaje": "Incidente creado exitosamente",
    "incidente": {
        "id": "20250426103000_BAC_AVE",
        "usuarioId": 1,
        "tipoIncidente": "BACHES",
        "ubicacion": "Av. Principal 123",
        "latitud": 19.4326,
        "longitud": -99.1332,
        "tipoVialidad": "AVENIDA",
        "estado": "PENDIENTE"
    }
}
```

Errores

- HTTP 404 - Usuario no encontrado
- HTTP 500 - Error al registrar el incidente

6. Obtener Todos los Incidentes

Endpoint

GET `/v1/incidentes/all`
Obtiene todos los incidentes registrados en el sistema.

Request

URL: `http://localhost:8080/v1/incidentes/all`
Método: `GET`

Response

Respuesta Exitosa ✔ HTTP 200


```
{
    "mensaje": "Incidentes encontrados",
    "total": 1,
    "incidentes": [
        {
            "id": "20250426103000_BAC_AVE",
            "usuarioId": 1,
            "tipoIncidente": "BACHES",
            "ubicacion": "Av. Principal 123",
            "latitud": 19.4326,
            "longitud": -99.1332,
            "tipoVialidad": "AVENIDA",
            "estado": "PENDIENTE"
        }
    ]
}
```


Errores

- HTTP 500 - Error al obtener los incidentes

7. Obtener Incidentes por Usuario

Endpoint

GET `/v1/incidentes/usuario/{usuarioId}`
Obtiene todos los incidentes asociados a un usuario específico.

Request

URL: `http://localhost:8080/v1/incidentes/usuario/1`
Método: `GET`

Response

Respuesta Exitosa ✔ HTTP 200


```
{
    "mensaje": "Incidentes encontrados para el usuario",
    "usuario": {
        "id": 1,
        "nombre": "Juan Pérez"
    },
    "total": 1,
    "incidentes": [
        {
            "id": "20250426103000_BAC_AVE",
            "tipoIncidente": "BACHES",
            "ubicacion": "Av. Principal 123",
            "estado": "PENDIENTE"
        }
    ]
}
```


Errores

- HTTP 404 - Usuario no encontrado
- HTTP 500 - Error al obtener los incidentes

8. Actualizar Estado de Incidente

Endpoint

PUT `/v1/incidentes/update-status/{id}?usuarioId={usuarioId}`
Actualiza el estado de un incidente específico.

Request

URL: `http://localhost:8080/v1/incidentes/update-status/20250426103000_BAC_AVE`
Método: `PUT`
Headers: `Content-Type: application/json`
Body:

```
{
    "estado": "RESUELTO"
}
```

Response

Respuesta Exitosa ✔ HTTP 200

```
{
    "mensaje": "Estado actualizado correctamente",
    "incidente": {
        "id": "20250426103000_BAC_AVE",
        "estado": "RESUELTO"
    }
}
```

Errores

- HTTP 403 - No tiene permiso para modificar este incidente
- HTTP 404 - Incidente no encontrado
- HTTP 500 - Error al actualizar el estado

9. Buscar Incidentes Cercanos

Endpoint

GET `/v1/incidentes/cercanos`
Busca incidentes cercanos a una ubicación geográfica.

Request

URL: `http://localhost:8080/v1/incidentes/cercanos?latitud=19.4326&longitud=-`
Método: `GET`
Parámetros:

- `latitud`: Latitud del punto central
- `longitud`: Longitud del punto central
- `radioKm`: Radio de búsqueda en kilómetros (default: 5.0)

Response

Respuesta Exitosa ✔ HTTP 200


```
{
    "mensaje": "Búsqueda completada",
    "parametros": {
        "latitud": 19.4326,
        "longitud": -99.1332,
        "radioKm": 5.0
    },
    "total": 1,
    "incidentes": [
        {
            "id": "20250426103000_BAC_AVE",
            "tipoIncidente": "BACHES",
            "ubicacion": "Av. Principal 123",
            "estado": "PENDIENTE",
            "distancia": 0.5
        }
    ]
}
```


Errores

- HTTP 500 - Error al buscar incidentes cercanos

7. Diagrama de Base de datos