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: Ingreso del Usuario

Actores: Ciudadano, 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. El usuario es redirigido al mapa principal. (RF5)

Flujo Alternativo:

- *Credenciales incorrectas:*
    1. El sistema muestra un mensaje de error.
    2. Ofrece un enlace para recuperar la contraseña por correo.

CU3: 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.

CU4: 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.

CU5: Explorar Incidentes en Mapa

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

1. El usuario accede al mapa principal y ve los marcadores de incidentes. (RF3)
2. Usa filtros en la barra lateral para mostrar incidentes por estatus o tipo. (RF4)
3. Hace clic en un marcador para ver detalles.
4. El sistema muestra una ventana con fotos, estatus y tipo de incidente. (RF5)

Flujo Alternativo:

- *Filtros sin resultados:*
    1. El sistema indica que no hay incidentes y sugiere ajustar los filtros.

CU6: Cerrar Sesión

Actores: Usuario, Sistema

Flujo Principal:

1. El usuario hace clic en el botón de Çerrar Sesión"desde cualquier página del sistema. (RF6)
2. El sistema cierra la sesión del usuario y lo redirige a la página de inicio de sesión. (RF1)

Flujo Alternativo:

- *Error en el cierre de sesión:*
  1. El sistema muestra un mensaje de error indicando que no se pudo cerrar la sesión.
  2. El usuario puede intentar nuevamente o contactar soporte técnico.

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

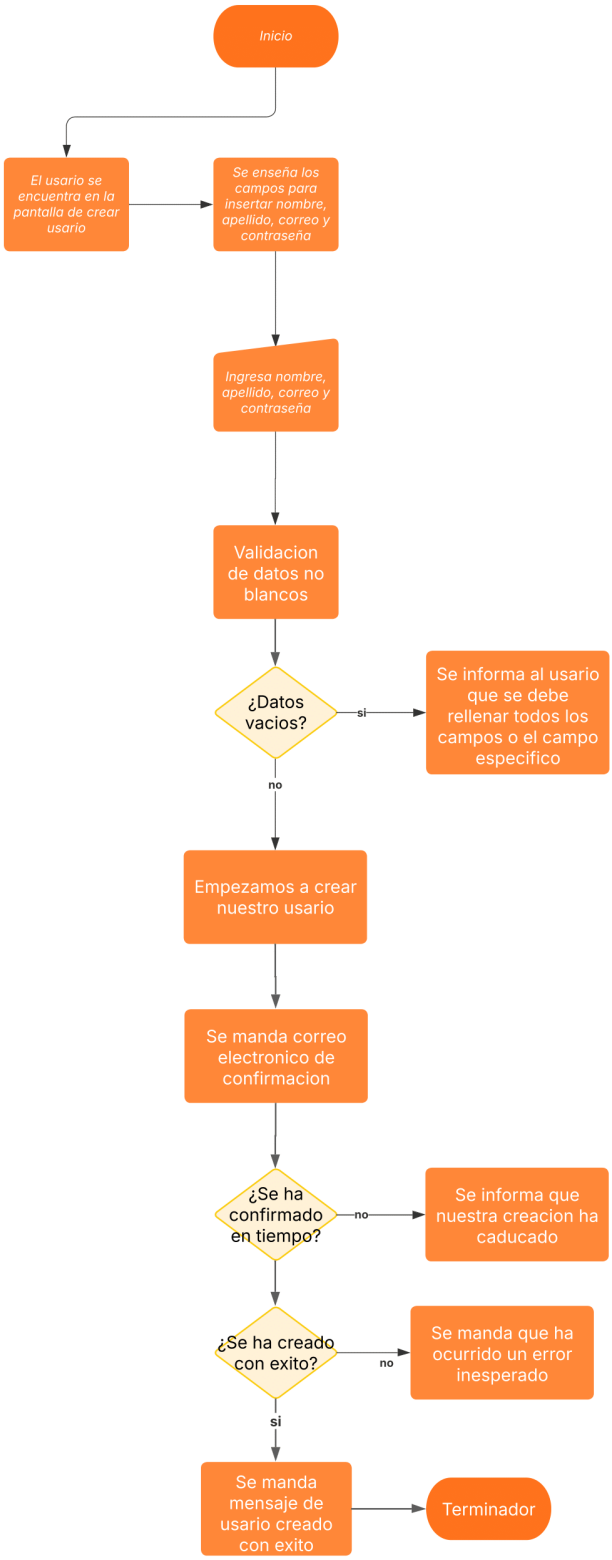
- **PostgreSQL en Supabase**
  - *Razón de uso:* PostgreSQL es un sistema de base de datos relacional robusto y de código abierto, ideal para almacenar datos estructurados como reportes con ubicaciones, estatus y fotos (RF2). Supabase añade una capa de facilidad con almacenamiento de archivos (fotos) 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

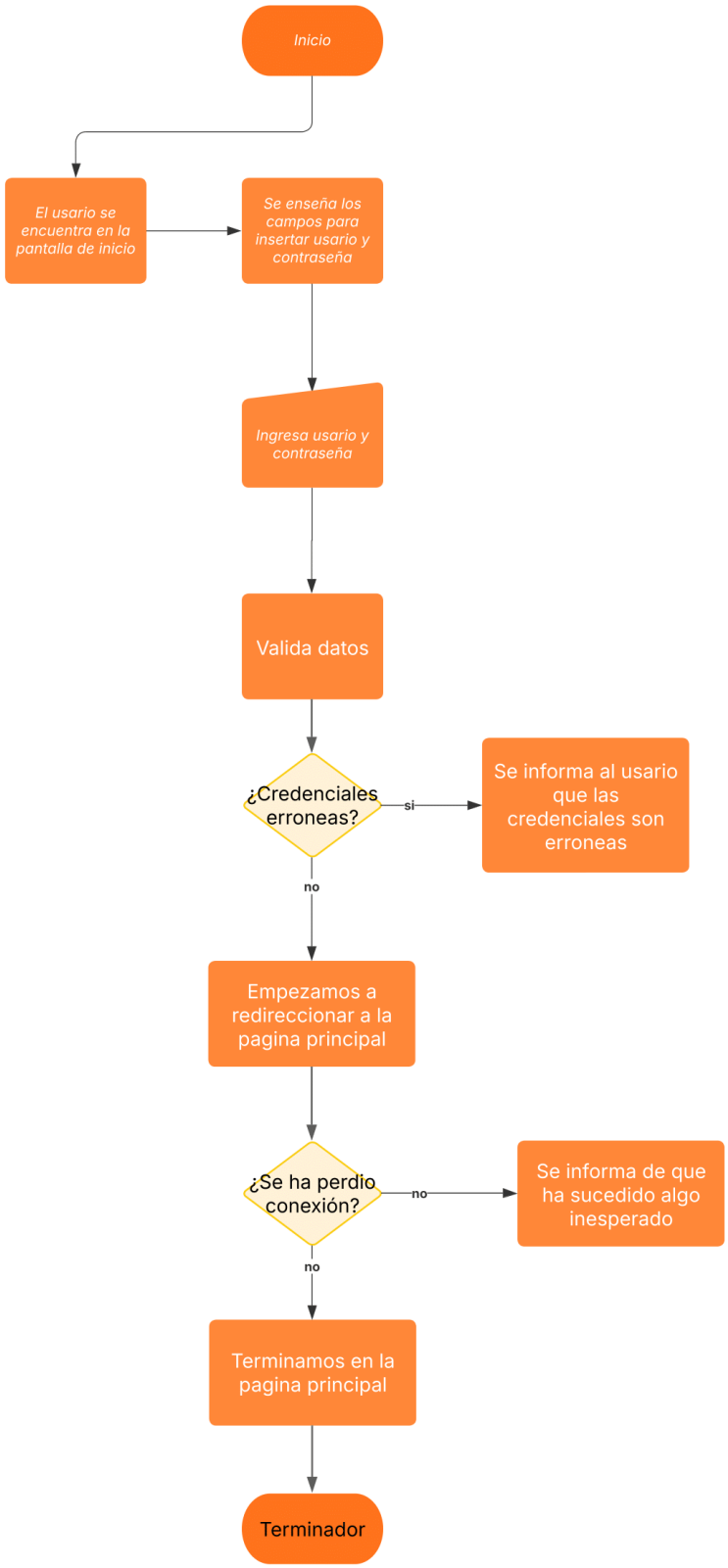
- **Leaflet con OpenStreetMap (Opción Principal)**
  - *Razón de uso:* Leaflet es una biblioteca ligera y de código abierto para mapas interactivos, combinada con OpenStreetMap, una fuente de datos gratuita y global. Esto permite visualizar y marcar incidentes (RF3) sin costos asociados, a diferencia 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).
- **Google Maps API (Alternativa)**
  - *Razón de uso:* Ofrece funcionalidades avanzadas como autocompletado de direcciones y alta calidad visual, útil si se prioriza una experiencia premium en el futuro.
  - *Ventaja para Bumper:* Podría mejorar la precisión de ubicaciones (CU2), pero su costo (crédito limitado de $200/mes) lo hace menos viable para la fase inicial frente a Leaflet.

6. Diagramas

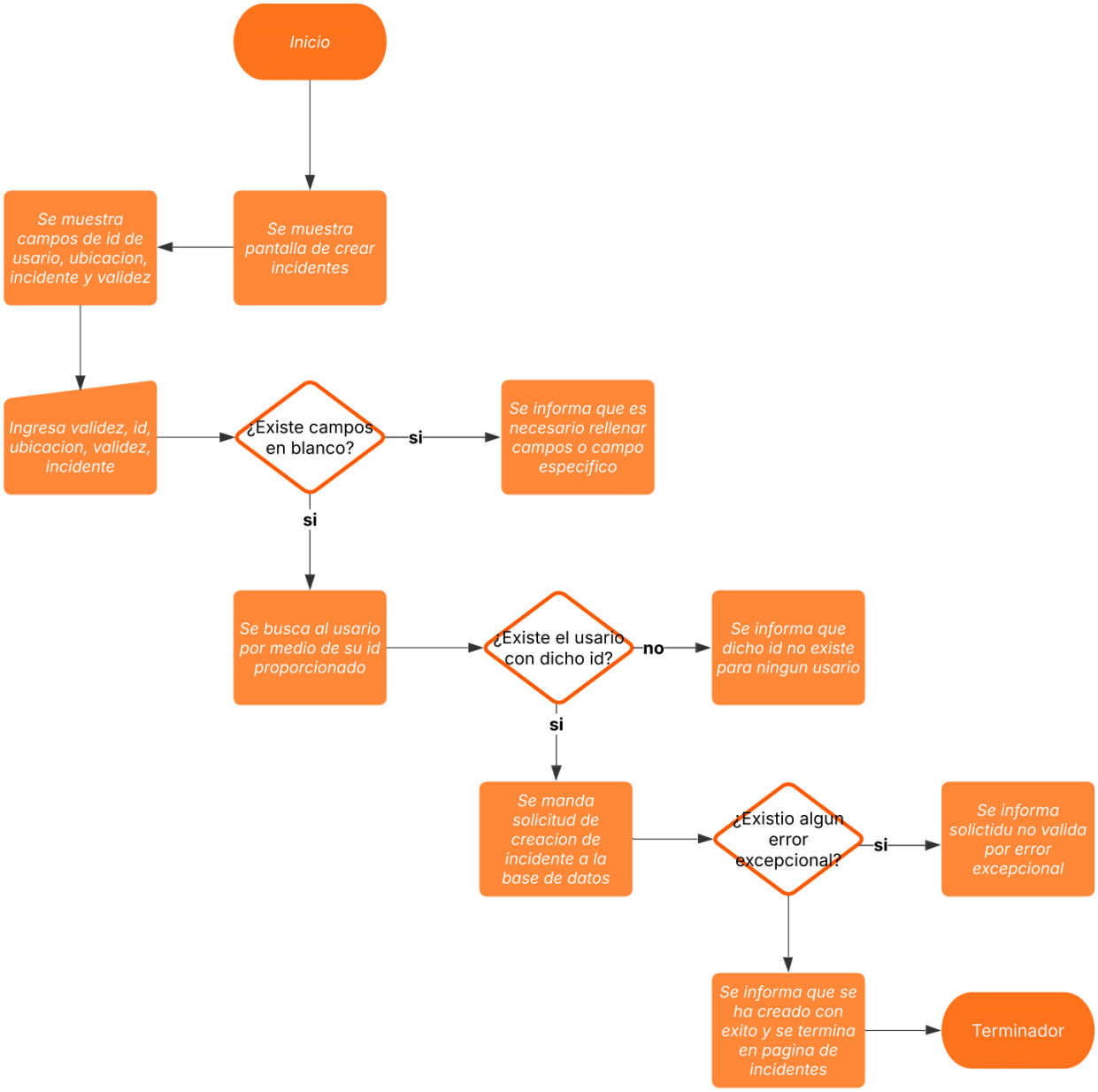
Crear Usuario



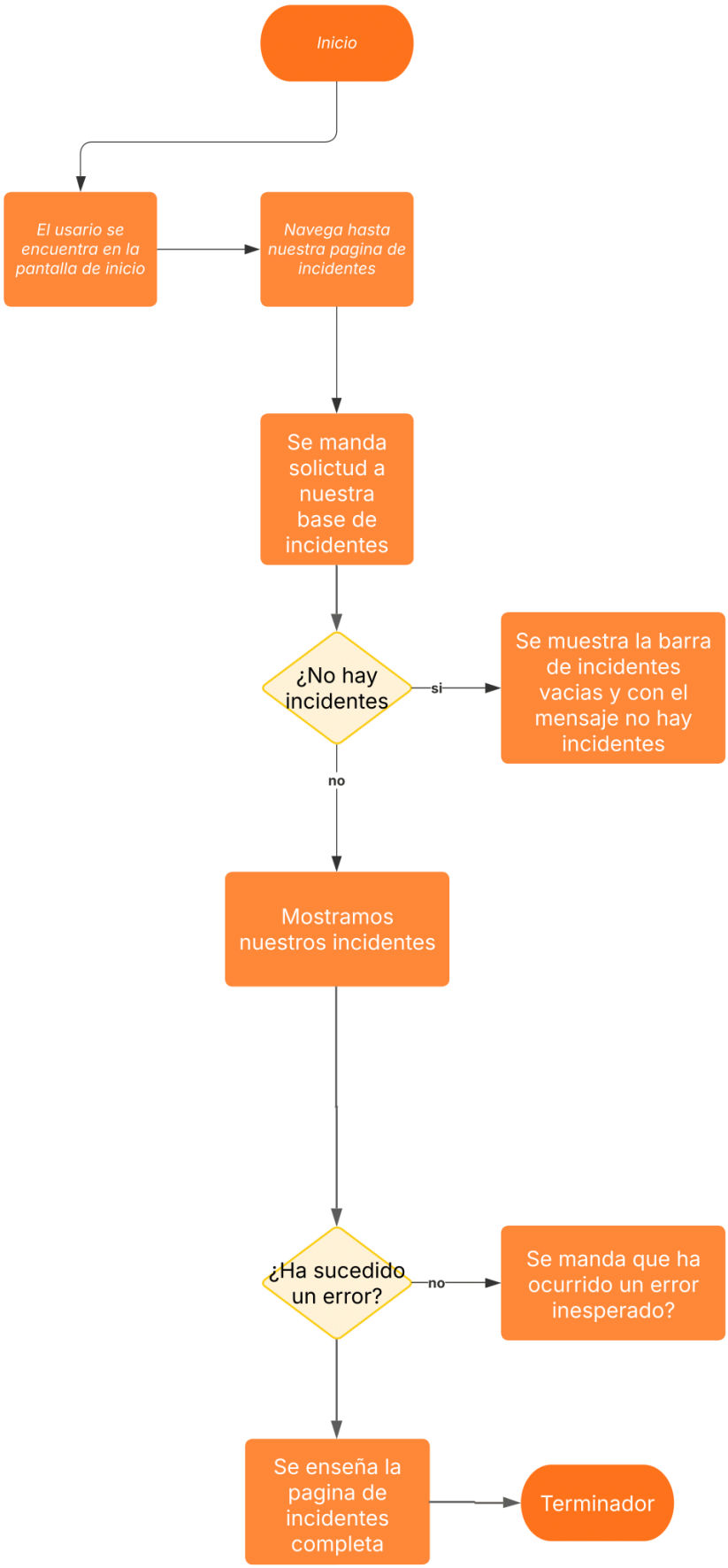
Iniciar Sesión



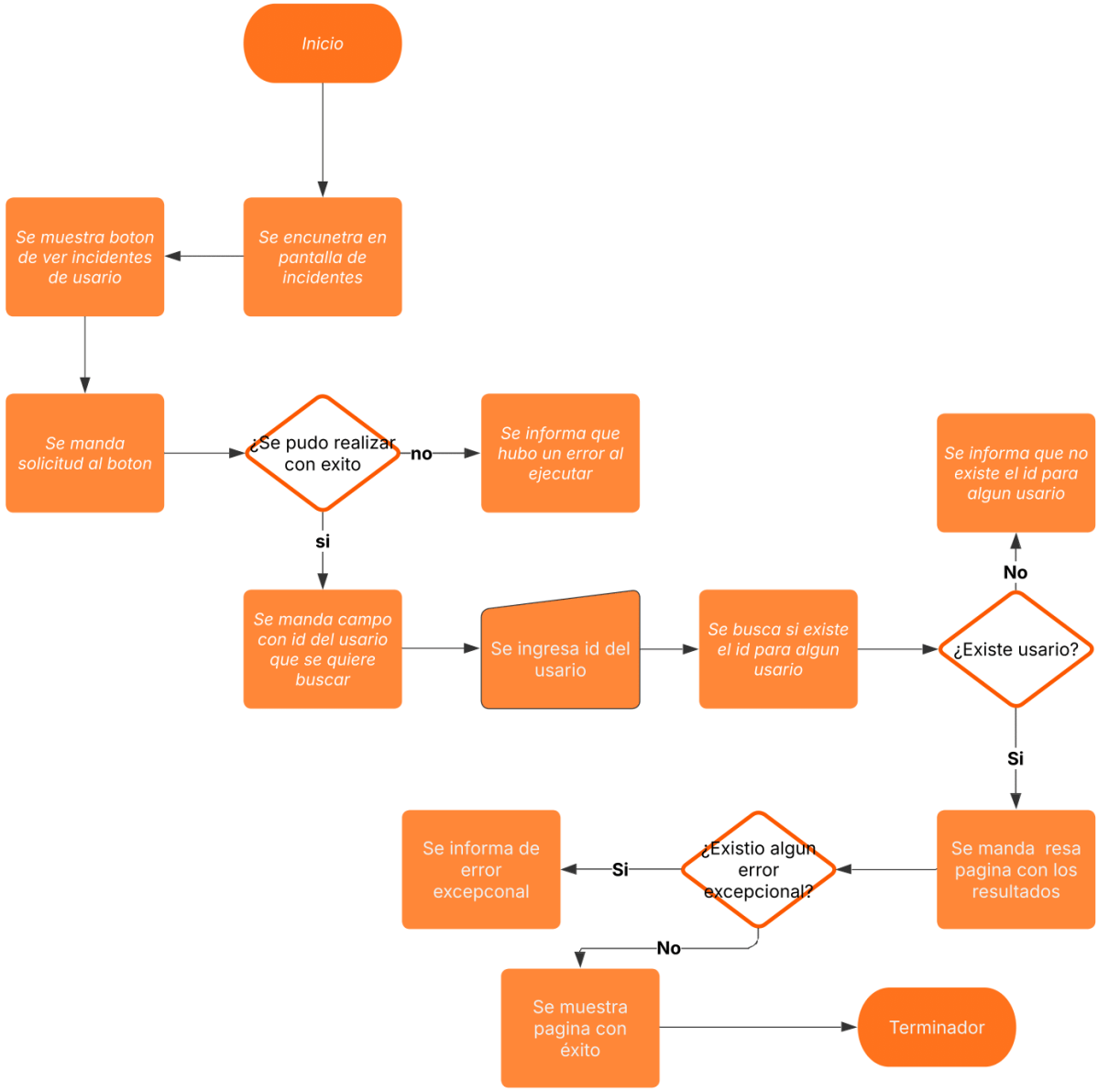
Crear Incidente



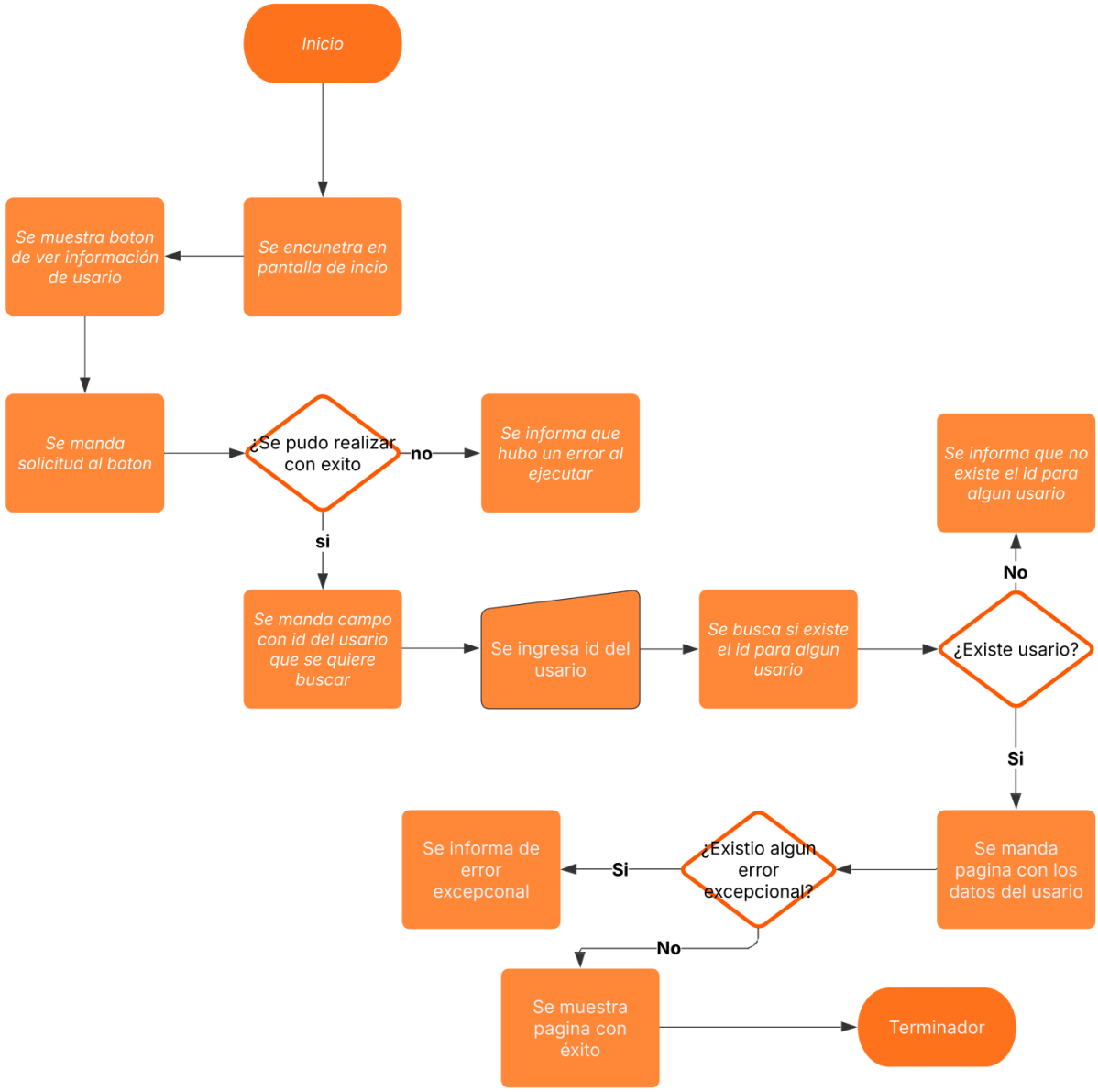
Mostar incidentes



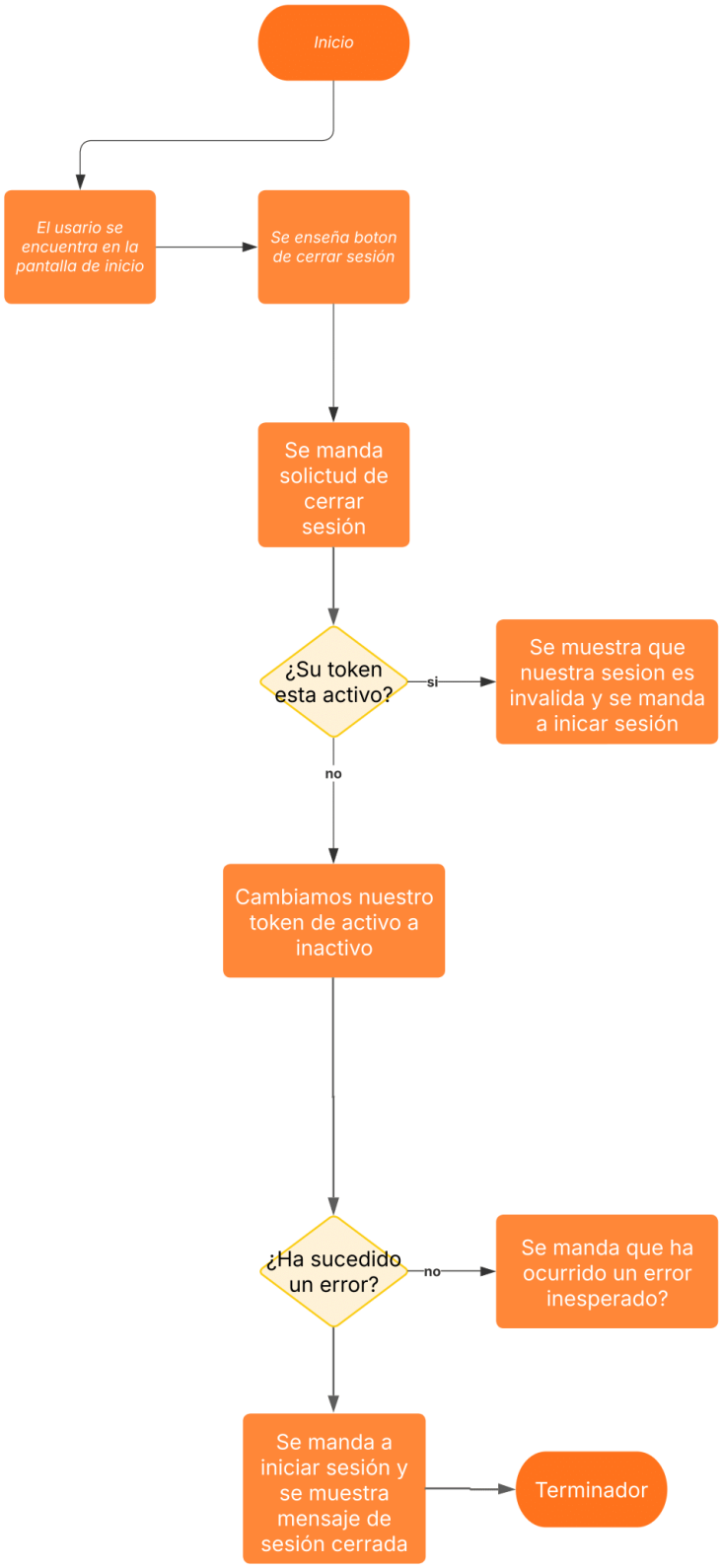
Mostar incidenes usuarios



Mostar información



Cerrar sesión



1. Crear un nuevo usuario

Endpoint

POST `/v1/users/create`
Crea un nuevo usuario en la base de datos.

Request

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

```
{
    "nombre": "Juan",
    "apellido": "Perez",
    "correo": "juan@example.com",
    "password": "1234",
    "token": "inactivo",
    "numeroIncidentes": 0
}
```

Response

Respuesta Exitosa HTTP 201

```
{
    "id": 1,
    "nombre": "Juan",
    "apellido": "Perez",
    "correo": "juan@example.com",
    "password": "1234",
    "token": "inactivo",
    "numeroIncidentes": 0,
    "incidentes": []
}
```

Posibles Errores

- HTTP 400 - Datos inválidos
- HTTP 409 - Correo ya existe

Notas

Notas importantes:

- El `id` será generado automáticamente
- `token` inicia como ïnactivo”
- `numeroIncidentes` inicia en 0

2. Iniciar sesión (Login)

Endpoint

POST `/v1/users/login`
Inicia sesión y cambia el token a estado .ªctivo”.

Request

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

```
{
    "correo": "juan@example.com",
    "password": "1234"
}
```

Response

Respuesta Exitosa HTTP 200

```
{
    "id": 1,
    "nombre": "Juan",
    "apellido": "Perez",
    "correo": "juan@example.com",
    "password": "1234",
    "token": "activo",
    "numeroIncidentes": 0,
    "incidentes": []
}
```

Posibles Errores

- HTTP 401 - Credenciales incorrectas
- HTTP 404 - Usuario no encontrado

Notas

Notas importantes:

- El token cambiará automáticamente a .ªctivo"tras un login exitoso
- La sesión permanecerá activa hasta que se realice logout
- Se recomienda almacenar el token para futuras peticiones

3. Crear un nuevo incidente

Endpoint

POST `/v1/incidentes`
Crea un incidente asociado a un usuario específico.

Request

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


```
{
    "usuarioId": 1,
    "tipoIncidente": "BACHES",
    "ubicacion": "Av. Principal 123",
    "tipoVialidad": "AVENIDA"
}
```


Response

Respuesta Exitosa ✔ HTTP 201


```
{
    "id": 1,
    "usuario": {
        "id": 1,
        "nombre": "Juan",
        "apellido": "Perez",
        "correo": "juan@example.com",
        "password": "1234",
        "token": "activo",
        "numeroIncidentes": 1,
        "incidentes": []
    },
    "tipoIncidente": "BACHES",
    "ubicacion": "Av. Principal 123",
    "horaIncidente": "2025-03-23T10:00:00",
    "tipoVialidad": "AVENIDA"
}
```


Posibles Errores ⚠

- HTTP 400 - Datos inválidos en el cuerpo de la solicitud
- HTTP 404 - Usuario no encontrado

Notas

Notas importantes:

- El campo `usuarioId` debe corresponder a un usuario existente en la base de datos.
- El campo `horaIncidente` será generado automáticamente por el sistema.
- El tipo de vialidad (`tipoVialidad`) debe ser una categoría válida como `AVENIDA`, `CALLE`, etc.

4. Recuperar todos los incidentes

Endpoint

GET `/v1/incidentes`
Lista todos los incidentes registrados en la base de datos.

Request

URL: `http://localhost:8080/v1/incidentes`
Método: `GET`
Headers: (ninguno requerido)
Body: (ninguno)

Response

Respuesta Exitosa HTTP 200


```
[
    {
        "id": 1,
        "usuario": {
            "id": 1,
            "nombre": "Juan",
            "apellido": "Perez",
            "correo": "juan@example.com",
            "password": "1234",
            "token": "activo",
            "numeroIncidentes": 1,
            "incidentes": []
        },
        "tipoIncidente": "BACHES",
        "ubicacion": "Av. Principal 123",
        "horaIncidente": "2025-03-23T10:00:00",
        "tipoVialidad": "AVENIDA"
    }
]
```


Posibles Errores

- HTTP 500 - Error interno del servidor

Notas

Notas importantes:

- Este endpoint devuelve todos los incidentes registrados en el sistema, sin filtros.
- Si no hay incidentes registrados, devuelve una lista vacía (`[]`).
- Se recomienda implementar paginación en caso de grandes volúmenes de datos.

5. Recuperar incidentes por usuario

Endpoint

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

Request

URL: `http://localhost:8080/v1/incidentes/usuario/1`
Método: `GET`
Headers: (ninguno requerido)
Body: (ninguno)

Response

Respuesta Exitosa ✔ HTTP 200


```
[
    {
        "id": 1,
        "usuario": {
            "id": 1,
            "nombre": "Juan",
            "apellido": "Perez",
            "correo": "juan@example.com",
            "password": "1234",
            "token": "activo",
            "numeroIncidentes": 1,
            "incidentes": []
        },
        "tipoIncidente": "BACHES",
        "ubicacion": "Av. Principal 123",
        "horaIncidente": "2025-03-23T10:00:00",
        "tipoVialidad": "AVENIDA"
    }
]
```


Posibles Errores ⚠

- HTTP 400 - El parámetro {`usuarioId`} no es válido.
- HTTP 404 - Usuario no encontrado o sin incidentes asociados.

Notas

ℹ Notas importantes:

- El parámetro {`usuarioId`} debe coincidir con un ID existente en la base de datos.
- Si el usuario no tiene incidentes registrados, el sistema devuelve una lista vacía (`[]`).
- Este endpoint es útil para filtrar incidentes por usuario específico.

6. Obtener información del usuario (verificación)

Endpoint

GET `/v1/users/me`
Obtiene los detalles del usuario, incluyendo el contador actualizado de incidentes.

Request

URL: `http://localhost:8080/v1/users/me`
Método: `GET`
Headers:
`correo: juan@example.com`
Body: (ninguno)

Response

Respuesta Exitosa HTTP 200

```
{
    "id": 1,
    "nombre": "Juan",
    "apellido": "Perez",
    "correo": "juan@example.com",
    "password": "1234",
    "token": "activo",
    "numeroIncidentes": 1,
    "incidentes": []
}
```

Posibles Errores

- HTTP 400 - Header `correo` no proporcionado o inválido.
- HTTP 404 - Usuario no encontrado.

Notas

Notas importantes:

- Es obligatorio incluir un header `correo` válido para identificar al usuario.
- El campo `numeroIncidentes` refleja el número actual de incidentes asociados al usuario.
- Este endpoint muestra información sensible como `password` solo para fines demostrativos; en un entorno real, se recomienda omitir este campo.

7. Cerrar sesión (Logout)

Endpoint

POST `/v1/users/logout`
Cierra la sesión del usuario cambiando el estado del token a `inactivo"`.

Request

URL: `http://localhost:8080/v1/users/logout`
Método: `POST`
Headers:
`correo: juan@example.com`
Body: (ninguno)

Response

Respuesta Exitosa HTTP 200

```
"Sesion cerrada correctamente"
```

Posibles Errores

- HTTP 400 - Header `correo` no proporcionado o inválido.
- HTTP 404 - Usuario no encontrado.
- HTTP 500 - Error interno del servidor.

Notas

Notas importantes:

- El token del usuario se actualiza automáticamente a `inactivo"` tras un logout exitoso.
- Es obligatorio incluir el mismo header `correo` utilizado durante el login.
- Se recomienda verificar el estado del token con el endpoint `GET /v1/users/me` después de cerrar sesión.

7. Diagrama de Base de datos