INGENIERIA DE SOFTWARE
Semestre: 2025-2
Equipo: BUMPER
Fecha: 17 de Febrero de 2025

Especificación de Requerimientos

Índice

1. Introducción

Proyecto: **Bumper**
Aplicación web para el registro y gestión de incidentes urbanos en la Ciudad de México.

1.1. Propósito

Esta aplicación web está pensada para permitir a los usuarios **registrar**, **visualizar** y **gestionar incidentes urbanos**, tales como baches, luminarias descompuestas, obstáculos en la vía pública, basura en banquetas, entre otros. El objetivo principal es brindar una **herramienta colaborativa** para mejorar la comunicación y participación entre los ciudadanos sobre problemas en la ciudad.

1.2. Alcance

Esta plataforma web permite a ciudadanos, comerciantes y organizaciones vecinales registrar y gestionar incidentes en las calles de sus colonias, utilizando una aplicación móvil sencilla de usar para mejorar la participación comunitaria, optimizar recursos y fomentar una ciudad más segura y limpia.

1.3. El cliente pidió

- **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

3. Viaje del usuario

Primera iteración sobre lo que pensaría tanto el cliente como el equipo de desarrollo sobre la forma de trabajar del sistema y la forma en como los usuarios pueden hacer uso de la aplicación.

1. Acceso Inicial

Paso 1: Ingresa a la web → *Landing page* con:

- Mapa interactivo central (iconos de incidentes existentes).
- Botón para **Reportar Incidente**.
- Menú superior: Registro/Ingresar o acceso como invitado (solo visualización).

2. Exploración del Mapa

Paso 2: Interactúa con el mapa:

- Zoom/Arrastre: Explora zonas geográficas.
- Clic en incidente: Pop-up muestra:
  - Mini-galería de fotos (vistas previas).
  - Estado (Reportado/En Proceso/Resuelto).
  - Comentarios.
- Filtros Laterales:
  - Categorías (Baches, Alumbrado, Basura, etc.).
  - Rango de fechas.

3. Creación de Reporte

Paso 3: Clic en Reportar Incidente → Formulario de 4 pasos:

Inicio de sesión previo

Paso 3.1 - Ubicación

- Mapa para marcar ubicación exacta:
  - Auto-detección de geolocalización (opcional).
  - Escribir la dirección
  - Ajuste manual con arrastre del marcador (opcional)

Paso 3.2 - Detalles del Incidente

- Campos obligatorios:
  - Tipo: Dropdown con categorías predefinidas.
  - Título: 60 caracteres máx. (ej: Bache peligroso en Av. Principal).
  - Descripción: Texto libre (500 caracteres) con placeholder de instrucciones
  - Fotos: Subida de hasta 4 archivos (formatos: JPG/PNG).
- Campos opcionales:
  - Etiquetas: sobre tendencias de la ciudad o características del incidente.

Paso 3.3 - Previsualización

- Resumen del reporte con:
  - Miniatura del mapa + dirección aproximada.
  - Vista previa de fotos subidas.
- Advertencia de datos públicos: Este reporte será visible para todos los usuarios.

Paso 3.4 - Confirmación

- Opciones:
  - Publicar ahora
  - Descartar cambios

4. Post-Publicación

Paso 4: Reporte publicado → Redirección a página del incidente con:

- Código Único: para seguimiento.
- Sección de Actualizaciones:
  - Timeline vacío (se llenará con interacciones).
  - lista de incidentes por las fechas
- Acciones Habilitadas: (EXTRAS)
  - Apoyar → Más comentarios
  - Compartir → Genera enlace para posteos.

5. Interacción de Otros Usuarios (EXTRA)

Escenario A - Actualizar Estado:

- Usuario secundario clic en Marcar como Resuelto.
- Modal solicita:
  - Subida de 1+ fotos que evidencien la solución.
  - Breve descripción (ej: Bache rellenado con asfalto).
- Sistema verifica:
  - Geolocalización del usuario vs ubicación del reporte (radio de 50m).
  - Consenso comunitario: Si 3+ usuarios validan en 24h → Estado cambia a Resuelto.

Escenario B - Aportar Información:

- Clic en Añadir Pruebas → Sube fotos/videos adicionales.
- Comentario contextual: Texto libre (280 caracteres).
- Nuevos archivos aparecen en galería con tag Contribución comunitaria.

Escenario C - Discrepar:

- Botón Esto no está resuelto → Abre formulario de disputa.
- Usuario debe:
  - Seleccionar motivo (opciones predefinidas: Solución parcial, Daño recurrente).
  - Opcional: Adjuntar foto actualizada.
- Disputa reinicia el estado a En Proceso.

6. Seguimiento Personalizado (EXTRA)

Perfil de Usuario (requiere registro):

- Mis Reportes: Listado con filtros por estado.
- Contribuciones: Historial de incidentes, fotos añadidas y validaciones.

Notificaciones:

- Alertas cuando otros interactúan con sus reportes.
- Recordatorios si un incidente sigue activo tras 15 días.

4. Requerimientos

Funcionales

1. **Autenticación de Usuarios (RF1)**
   - Registro y acceso con email y contraseña.
   - Niveles de acceso: Ciudadano y Administrador (E)
2. **Gestión de Incidentes (RF2)**
   - Usuario puede crear, editar y eliminar reportes.
   - Verificar datos (las imágenes se suben correctamente)
   - Sistema de etiquetas para hacer filtros más sencillos.
3. **Integración con Mapa en Tiempo Real (RF3)**
   - Importar API para visualizar e interactuar con el mapa.
   - Manipulación visual para añadir items dentro del mapa.
   - Geolocalización automática. (E)
4. **Filtro (RF4)**
   - Búsqueda por palabras clave, categoría y rango de fechas.
5. **Visualización de Datos (RF5)**
   - Pantalla principal.
   - Pantalla login.
   - Pantalla crear incidente.
   - Pantalla editar incidente.
   - Pantalla filtrar incidentes.

Requerimientos No Funcionales

1. **Seguridad (RNF1)**
   - Seguridad en el registro de contraseñas y correos.
   - Proteger los datos de los usuarios. (E)
2. **Rendimiento (RNF2)**
   - Buscar optimización del mapa para cargar y manipularlo con baja latencia
   - Implementación para soportar registros simultáneos.
3. **Disponibilidad (RNF3)**
   - Solución local.
   - Buscar un host sencillo. (E)
4. **Usabilidad (RNF4)**
   - Interfaz responsiva.
   - Diseño intuitivo.
5. **Escalabilidad (RNF5)**
   - Arquitectura modular.
   - Base de datos escalable.

5. Casos de Uso

CU1: Autenticación de Usuario

Actores: Ciudadano, Sistema
Flujo Principal:

1. Usuario accede a la página de login (RF1)
2. Ingresa email y contraseña registrados
3. Sistema verifica credenciales en base de datos (RNF1)
4. Redirección al dashboard principal (RNF4)

Flujo Alternativo:

- Credenciales incorrectas:
  1. Sistema muestra error específico
  2. Ofrece recuperación de contraseña vía email
- Cuenta no verificada:
  1. Sistema bloquea acceso
  2. Reenvía enlace de verificación

CU2: Reportar Incidente Urbano

Actores: Ciudadano, Sistema
Flujo Principal:

1. El usuario inicia sesión (RF1) o se registra
2. Selecciona Reportar Incidente desde el mapa (RF3)
3. Marca ubicación en el mapa con geolocalización asistida (RNF2)
4. Completa formulario con: categoría, título, descripción y fotos (RF2)
5. Sistema valida y almacena el reporte en la base de datos (RNF5)

6. Muestra página de confirmación con código único (RF5)

CU3: Modificar Reporte Existente

Actores: Ciudadano, Sistema
Flujo Principal:

1. Usuario accede a Mis Reportes en su perfil (RF2)
2. Selecciona incidente no publicado para editar
3. Modifica campos permitidos: descripción, fotos, etiquetas (RF2)
4. Sistema valida cambios y actualiza timestamp (RNF5)
5. Muestra versión actualizada con historial de modificaciones

Flujo Alternativo:

- Intento de editar reporte publicado:
    1. Sistema bloquea cambios directos
    2. Ofrece crear nueva versión como actualización

CU4: Explorar Incidentes en Mapa

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

1. Usuario visualiza mapa principal (RF3)
2. Aplica filtros por categoría/fecha (RF4)
3. Haz clic en marcador para ver detalle
4. Sistema carga pop-up con:
    - Galería de fotos (RNF2)
    - Estado actualizado
    - Botones de interacción

Flujo Alternativo:

- Filtros sin resultados:
    - Sistema sugiere ajustar parámetros
    - Muestra heatmap de zonas problemáticas

CU5: Añadir Comentario Público

Actores: Usuario Registrado, Sistema
Flujo Principal:

1. Usuario selecciona incidente en mapa/listado (RF4)
2. Clic en Añadir Comentario
3. Escribe mensaje
4. Sistema analiza contenido (RNF1):
    - Filtra lenguaje ofensivo
    - Detecta spam
5. Publica comentario

Flujo Alternativo:

- Comentario rechazado:
    1. Sistema muestra políticas de comunidad
    2. Ofrece editar el contenido

6. Tecnológica usada

Backend

- Lenguaje de programación: Kotlin
- Framework: Spring Boot

Frontend

- React

Base de datos

- PostgreSQL en Supabase

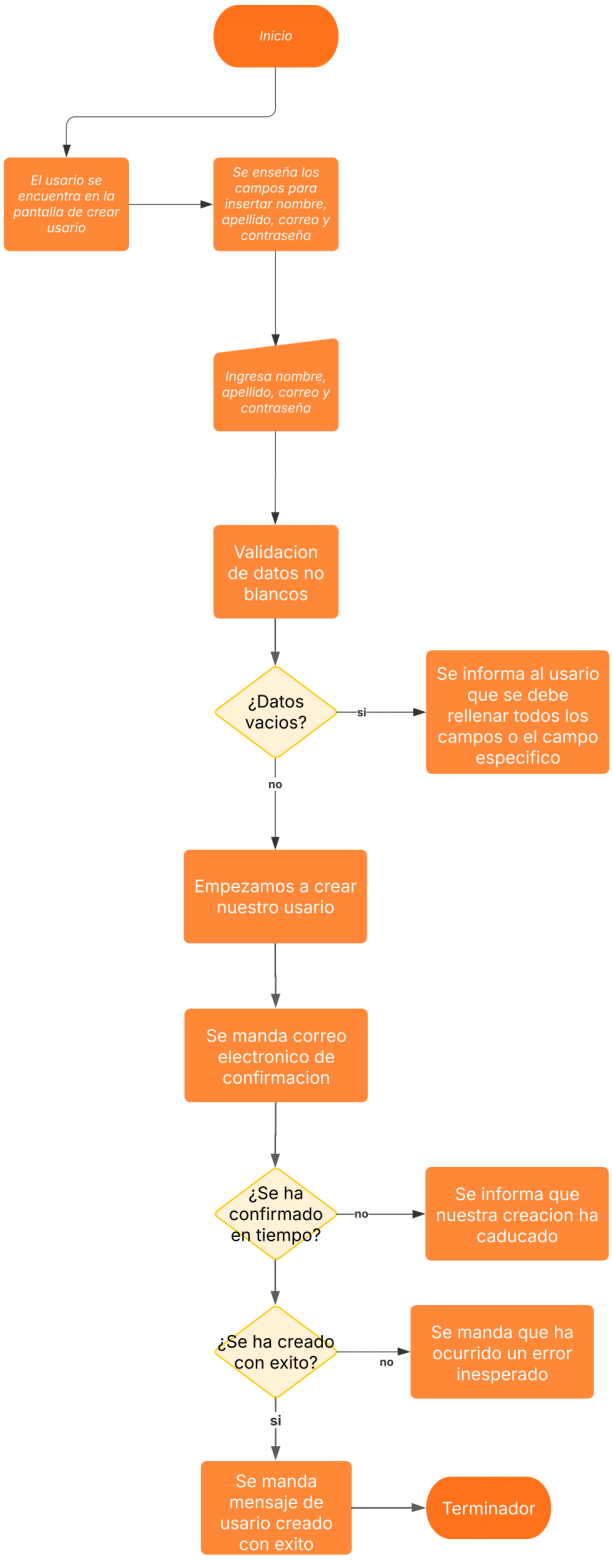
API para Mapas

- OpenLayers (primera opción) / Google Maps API (alternativa)

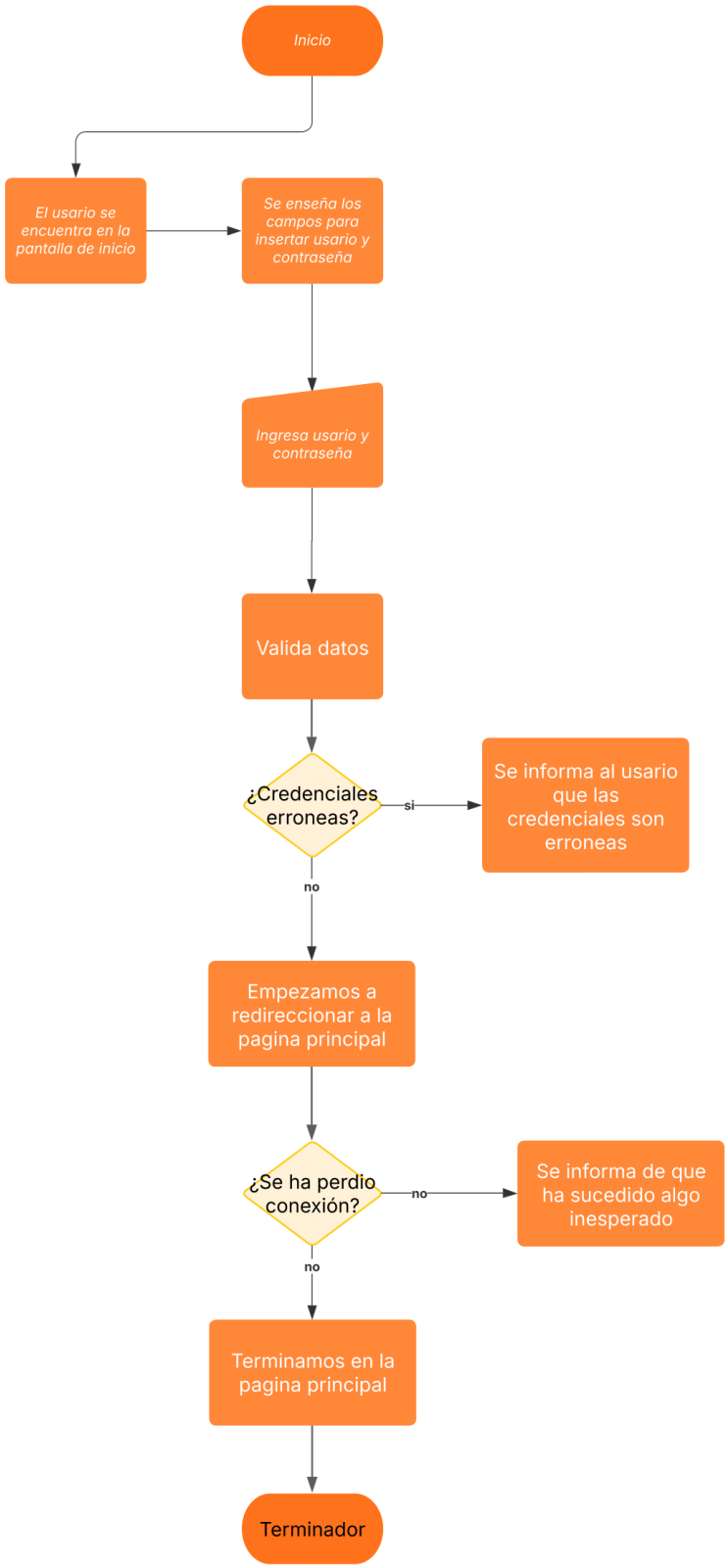
7. Iteración Dos

7.1. Diagramas

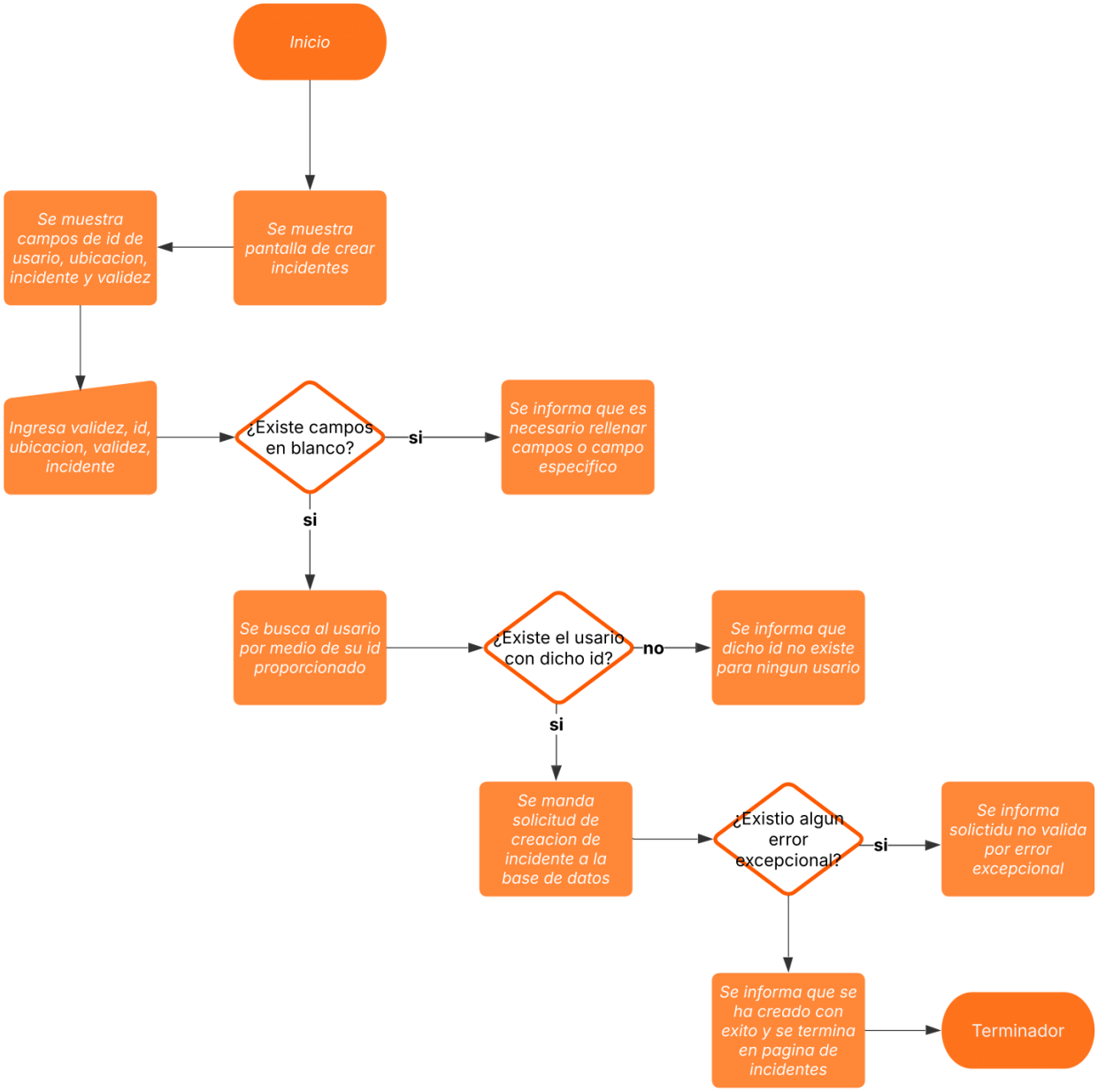
Crear Usuario



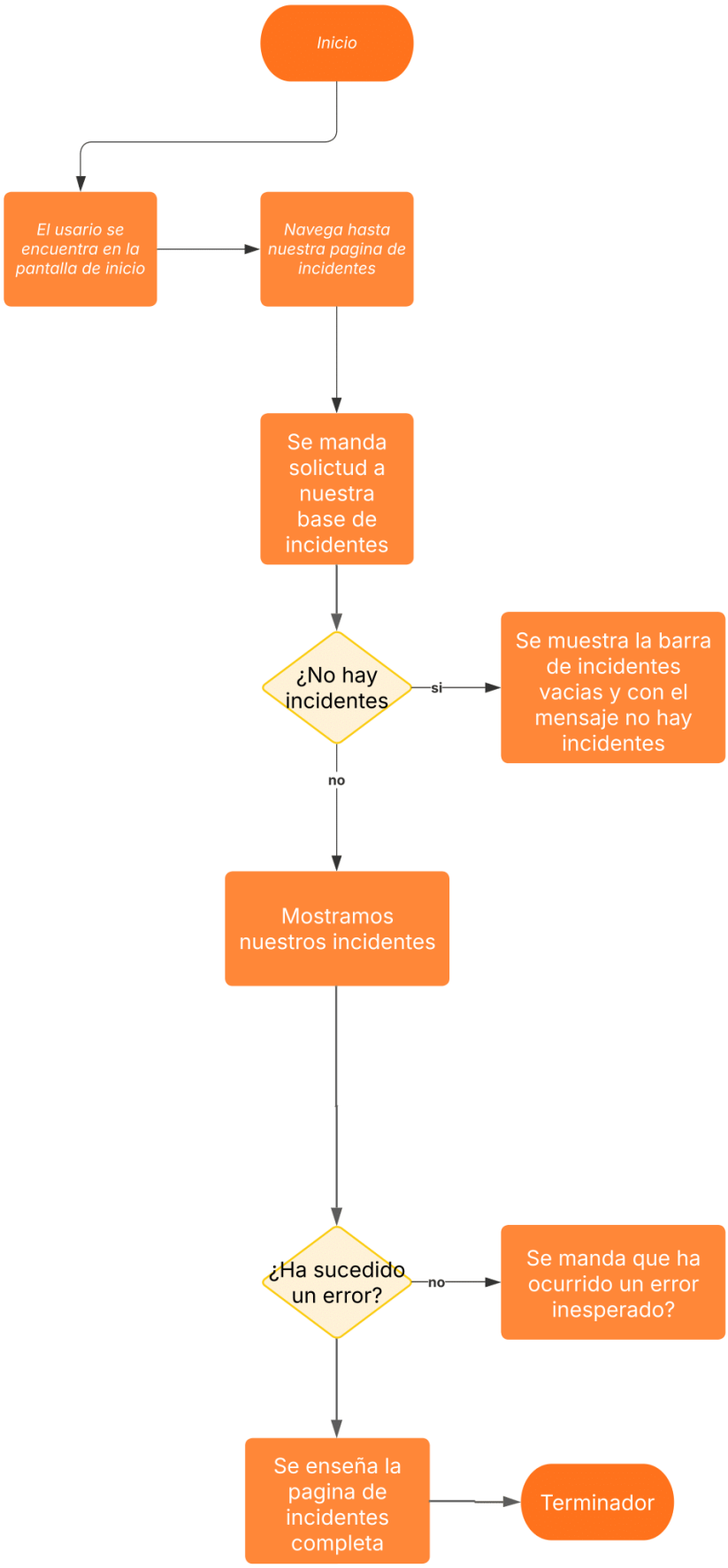
Iniciar Sesión



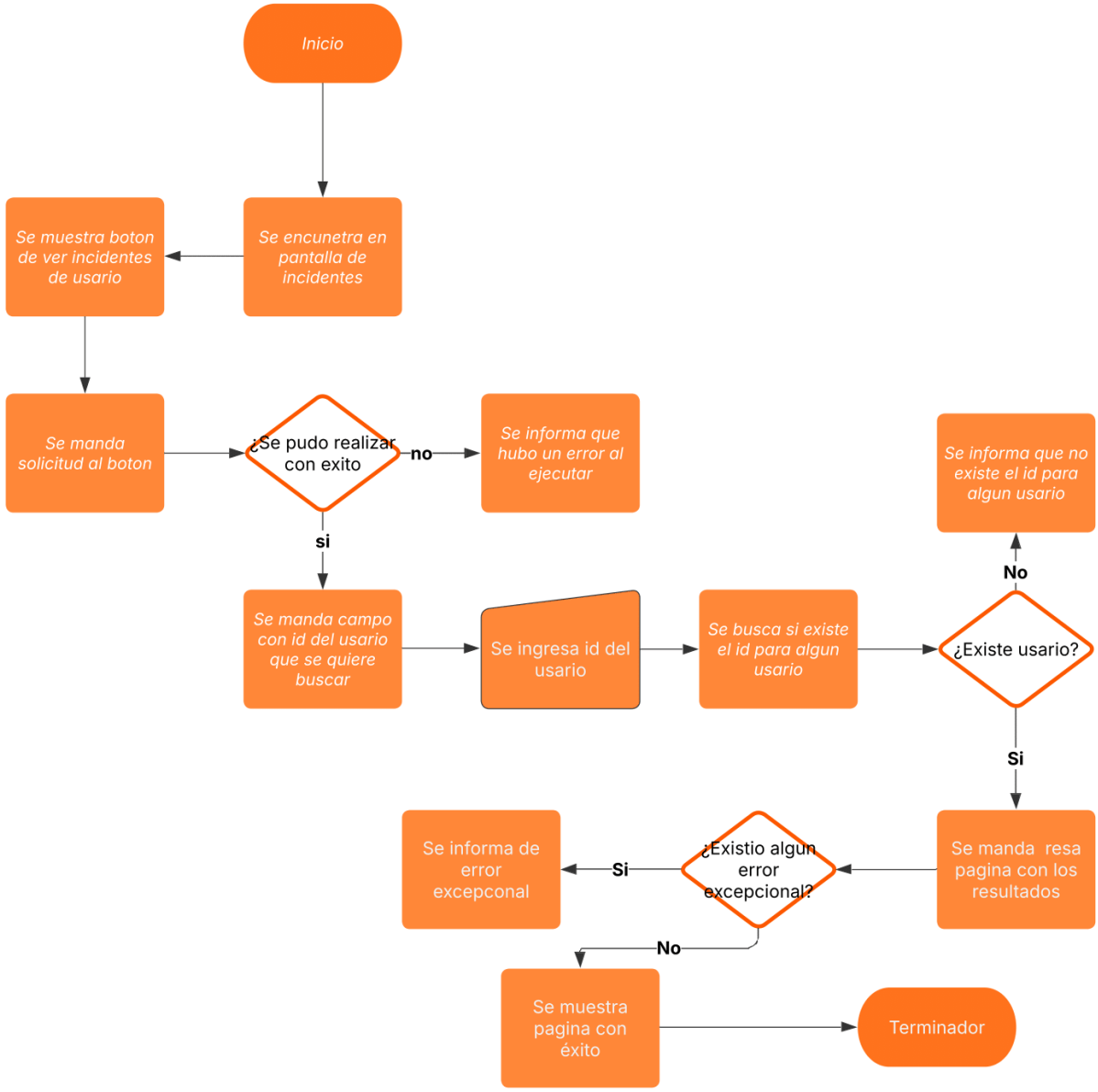
Crear Incidente



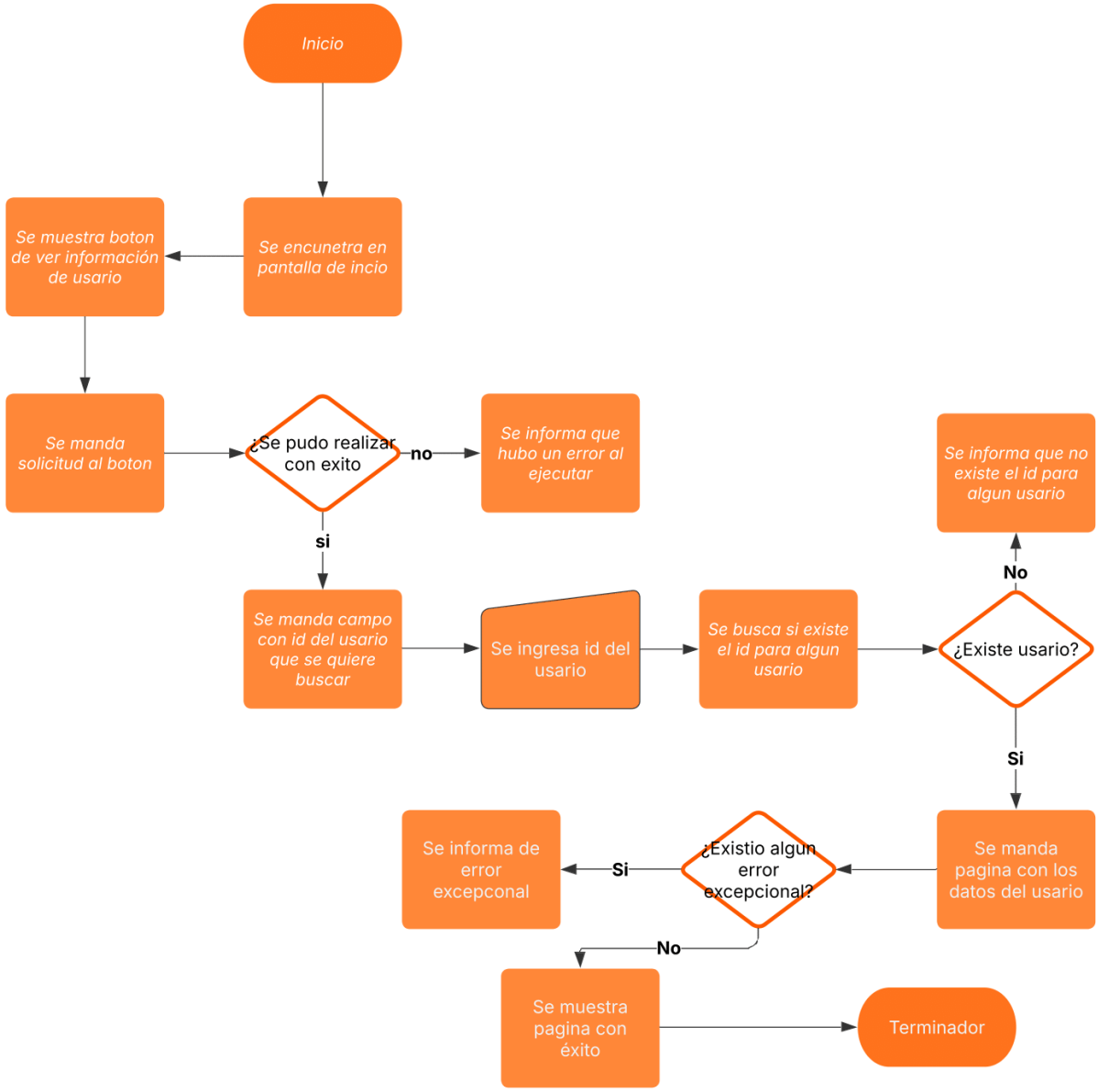
Mostar incidentes



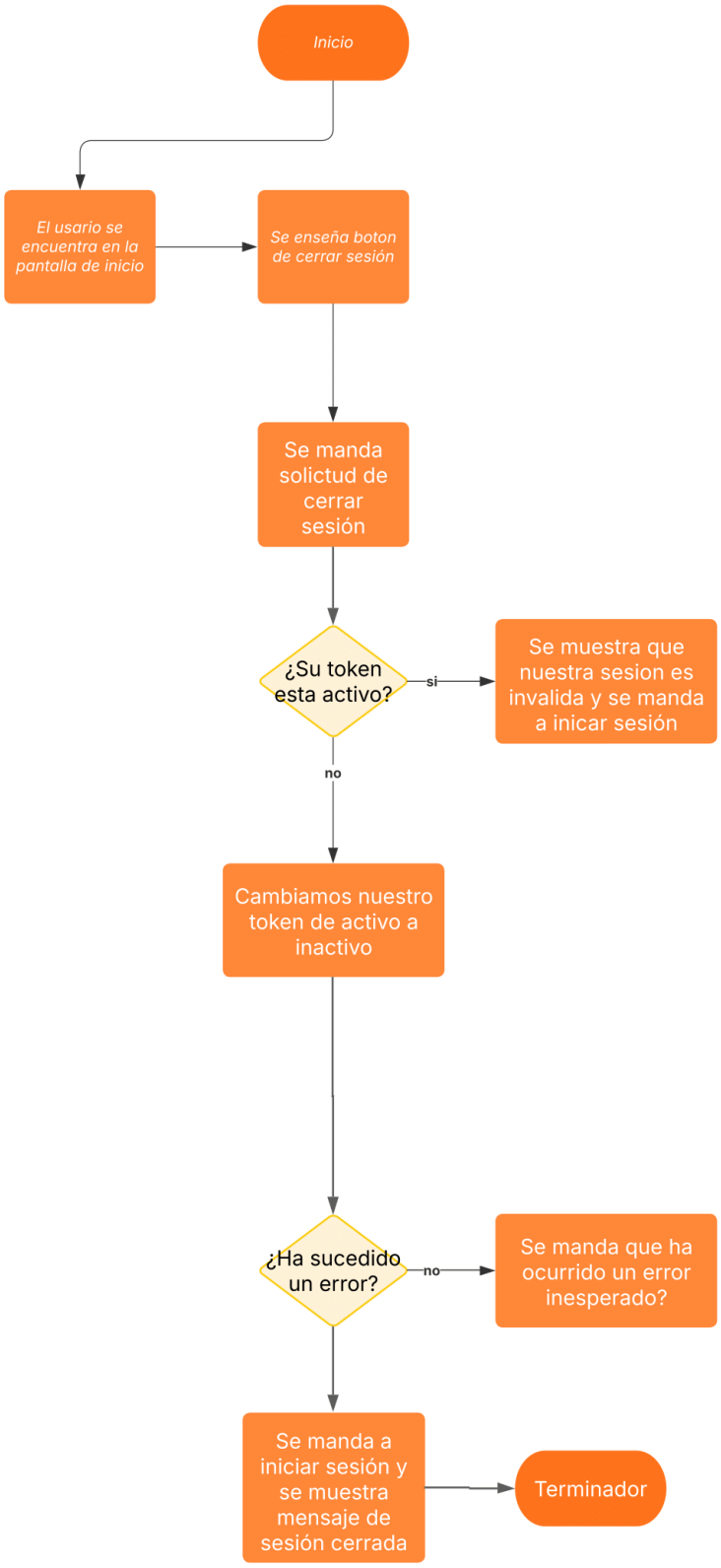
Mostar incidenes usuarios



Mostar información



Cerrar sesión



8. Documentación API

1. Crear un nuevo usuario
Endpoint: `POST /v1/users/create`
Descripción: Crea un nuevo usuario en la base de datos.
Solicitud:

- **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
}
```

Respuesta esperada (éxito, HTTP 201):

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

Notas:

- El `id` será generado automáticamente por la base de datos
- `token` y `numeroIncidentes` tienen valores por defecto

2. Iniciar sesión (Login)
Endpoint: `POST /v1/users/login`
Descripción: Inicia sesión y cambia el token a `.ªctivo"`.
Solicitud:

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

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

Respuesta esperada (éxito, HTTP 200):

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

Notas:

- Credenciales incorrectas devuelven HTTP 401

3. Crear un nuevo incidente
Endpoint: `POST /v1/incidentes`
Descripción: Crea incidente asociado al usuario.
Solicitud:

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

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

Respuesta esperada (éxito, 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"
}
```

4. Recuperar todos los incidentes
Endpoint: `GET /v1/incidentes`
Descripción: Lista todos los incidentes en la base de datos.
Solicitud:

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

Respuesta esperada (éxito, 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"
    }
]
```


Notas:

- Muestra todos los incidentes creados en el sistema

5. Recuperar incidentes por usuario
Endpoint: `GET /v1/incidentes/usuario/{usuarioId}`
Descripción: Lista incidentes asociados a un usuario específico.
Solicitud:

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

Respuesta esperada (éxito, 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"
    }
]
```


Notas:

- El parámetro {`usuarioId`} debe coincidir con un ID existente
- Devuelve lista vacía si no hay incidentes asociados

6. Obtener información del usuario (verificación)
Endpoint: `GET /v1/users/me`
Descripción: Obtiene detalles del usuario con contador actualizado.
Solicitud:

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

Respuesta esperada (éxito, HTTP 200):


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


Notas:

- Requiere header `correo` válido
- Verifica estado actual de `numeroIncidentes`

- Muestra información sensible como `password` (solo para fines demostrativos)

7. Cerrar sesión (Logout)
Endpoint: `POST /v1/users/logout`
Descripción: Cierra sesión cambiando token a `inactivo"`.
Solicitud:

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

Respuesta esperada (éxito, HTTP 200):

```
"Sesion cerrada correctamente"
```

Notas:

- Cambia estado del token a inactivo
- Verificación posterior con `GET /v1/users/me` muestra nuevo estado
- Requiere mismo header `correo` usado en login

9. Diagrama de Base de datos