Universidad Nacional Autónoma de México

Facultad de Ciencias

Documentación del Proyecto (v. 1.0.0)

Equipo:
Skops & Company

Integrantes:
Flores Gutierrez José Luis
García López Francisco Daniel
Gómez López Erik Eduardo
Luna Campos Emiliano
Vázquez Reyes Jesús Elías

Asignatura:
Ingeniería de Software

Historial de Cambios

Versión	Fecha	Descripción	Autor(es)
1.0.0	05/04/2025	Se agregaron diagramas de flujo de los casos de uso relacionados al usuario, así como ocho casos de uso relacionados al sistema de incidentes junto su diagrama. También agregamos los diagramas E-R y Relacional de la BDD.	> Flores Gutierrez José Luis > García López Francisco Daniel > Gómez López Erik Eduardo > Luna Campos Emiliano > Vázquez Reyes Jesús Elías

1. Introducción

1.1. Propósito del Sistema

Este documento especifica los requerimientos funcionales y no funcionales del Sistema de Incidentes Urbanos (SIU), que permitirá registrar, visualizar y gestionar incidentes registrados en vías urbanas.

1.2. Alcance

El SIU permitirá a los usuarios dar a conocer la existencia de incidentes urbanos en algunas zonas de interés en particular. Además, podrán actualizar el estado de los incidentes reportados, así como poder ver incidentes reportados por otros usuarios.

1.3. Definiciones y Abreviaciones

(a) SIU: Sistema de Incidentes Urbanos
(b) Usuario: Persona que puede acceder al sistema
(c) Incidente: Registro de afectación en una ubicación especı´fica

2. Requerimientos del Sistema

2.1. Requerimientos Funcionales

(a) Registro de Usuario
El sistema permitirá el acceso de un usuario mediante un nombre único y contraseña.

(b) Gestión de Usuario
El sistema permite registro, edición y eliminación de algún usuario registrado.

(c) Gestión de Incidentes
El sistema permitirá registrar incidentes con título, fotografías, ubicación y descripción, así como editar el estado de algún incidente subido por los demás usuarios (con pruebas fotográficas).

(d) Gestión de Búsquedas
El sistema permite la búsqueda de incidentes por medio de filtros para la obtención de información de los mismos, así como visualizarlos a través de un mapa interactivo.

(e) Gestión de Búsquedas
El sistema permite la búsqueda de incidentes por medio de filtros para la obtención de información de los mismos, así como visualizarlos a través de un mapa interactivo.

(f) Pruebas para incidentes
No estará habilitada la modificación de incidentes sin pruebas adjuntas.

2.2. Requerimientos No Funcionales

(a) Seguridad

i. Sistema

- Se habilitará un protocolo HTTPS con el fin de mantener la seguridad en el SUI.
- La rama principal se encontrará protegida en todo momento.
- No se permite el acceso de la base de datos a cada uno de los administradores, ni a los usuarios.

ii. Usuario

- Solo los usuarios con una sesión iniciada pueden registrar incidentes urbanos.
- El software será capaz de cifrar y proteger la información de los usuarios, garantizando seguridad para sus cuentas.
- Solo los usuarios administradores pueden eliminar y/o editar cualquier incidente, o en su defecto, el dueño de éste.

(b) Usabilidad

i. Sistema intuitivo.
ii. Contenido adecuado y regulado para mostrar.
iii. Interfaz amigable, accesible y agradable a la vista.

(c) Rendimiento

i. El filtraje de incidentes por categorías tardará menos de 3 segundos.
ii. El proceso de dar de alta un incidente tomará menos de 5 segundos en promedio.
iii. El software soportará hasta 99 usuarios simultáneos.
iv. La actualización de incidentes y el tiempo de respuesta en la interfaz son imperceptibles.

(d) Disponibilidad

i. El sistema deberá estar disponible al menos el 99% del tiempo anual.

3. Casos de Uso

(a) Registrar usuario

Actor: Usuario

> Flujo Principal:

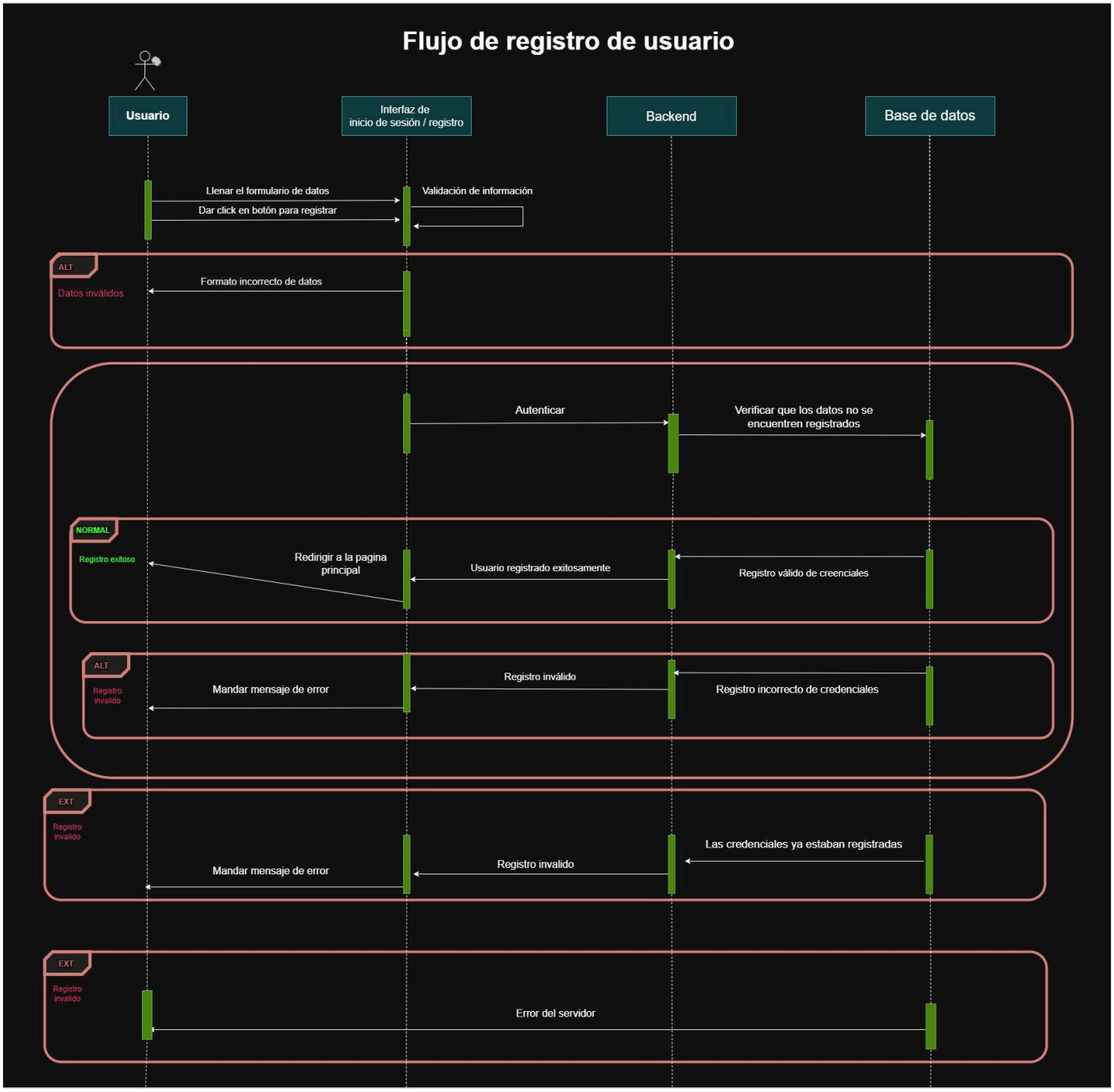
i. El usuario rellena el formulario de datos y da click en iniciar sesión.
ii. Se verifica que los datos no se encuentren registrados.
iii. El registro de las credenciales es exitoso, y se redirige a la página principal.

>> Flujo Alternativo 1:
iv. El usuario ingresa un formato incorrecto en los datos.
v. Se envía un mensaje de error.

>> Flujo Alternativo 2:
iv. Las credenciales ya estaban registradas.
v. Se envía un mensaje de error.

>> Flujo Alternativo 3:
iv. Se produce una falla en el servidor.
v. Mandamos un mensaje de error.

> Diagrama de flujo del caso de uso:



(b) Iniciar sesión

> Flujo Principal:

i. El usuario rellena el formulario de datos y da click en registrar datos.
ii. Se verifica que los datos ingresados (correo y contraseña) se encuentren registrados.
iii. La autenticación de las credenciales es exitosa, y se redirige a la página principal.

>> Flujo Alternativo 1:

iv. El usuario ingresa un formato incorrecto en los datos.
v. Se envía un mensaje de error.

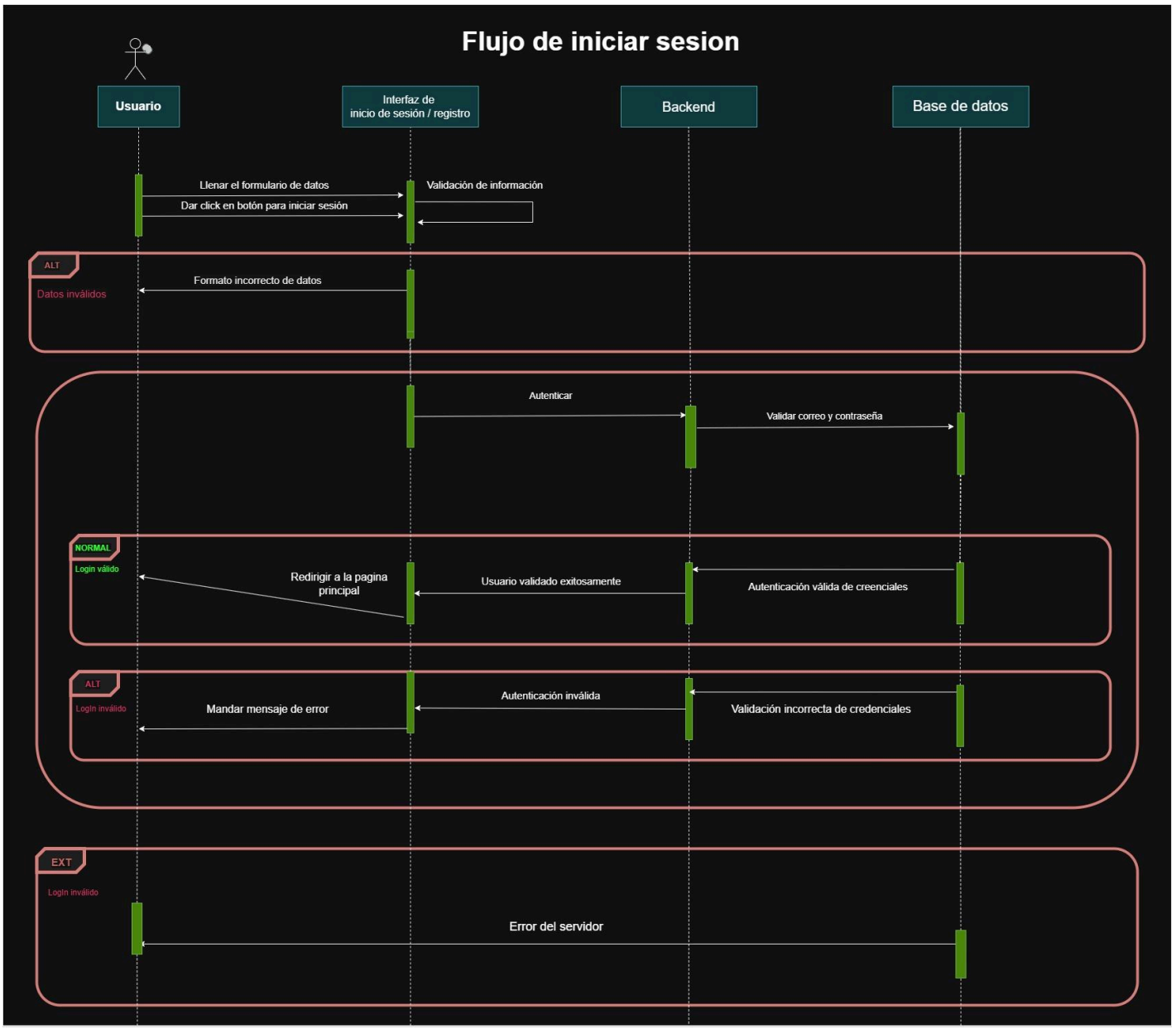
>> Flujo Alternativo 2:

iv. Las credenciales no existen en la base de datos.
v. Se envía un mensaje de error.

>> Flujos Alternativo 3:

iv. Se produce una falla en el servidor.
v. Mandamos un mensaje de error.

> Diagrama de flujo del caso de uso:



(c) Actualización de datos personales

> Flujo Principal:

i. El usuario inicia sesión, da click en el apartado de **actualización de datos**.
ii. Se verifica la existencia de un token que garantiza el inicio de sesión.
iii. Registra los nuevos datos a actualizar.
iv. Los datos se actualizan correctamente.

>> Flujo Alternativo 1:

v. El usuario ingresa un formato incorrecto en los datos.
vi. Se envía un mensaje de error.

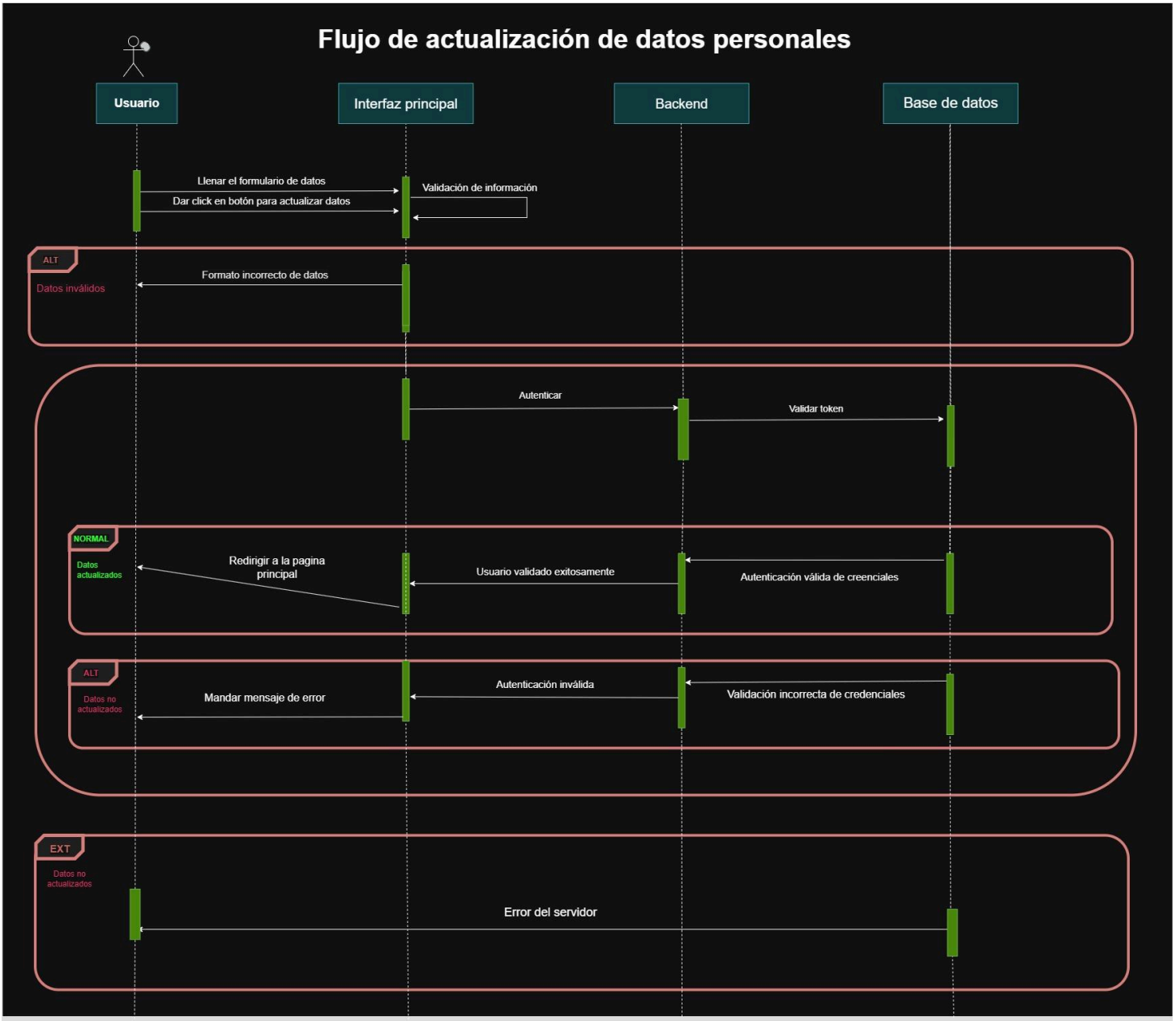
>> Flujo Alternativo 2:

v. Se produce un error en la autenticación del token.
vi. Se envía un mensaje de error.

>> Flujos Alternativo 3:

v. Se produce una falla en el servidor.
vi. Mandamos un mensaje de error.

> Diagrama de flujo del caso de uso:



(d) Consulta de datos del usuario

> Flujo Principal:

i. El usuario inicia sesión, da click en la sección '**Mi cuenta**'.
ii. Se verifica la existencia de un token que garantiza el inicio de sesión.
iii. Los datos se despliegan correctamente.

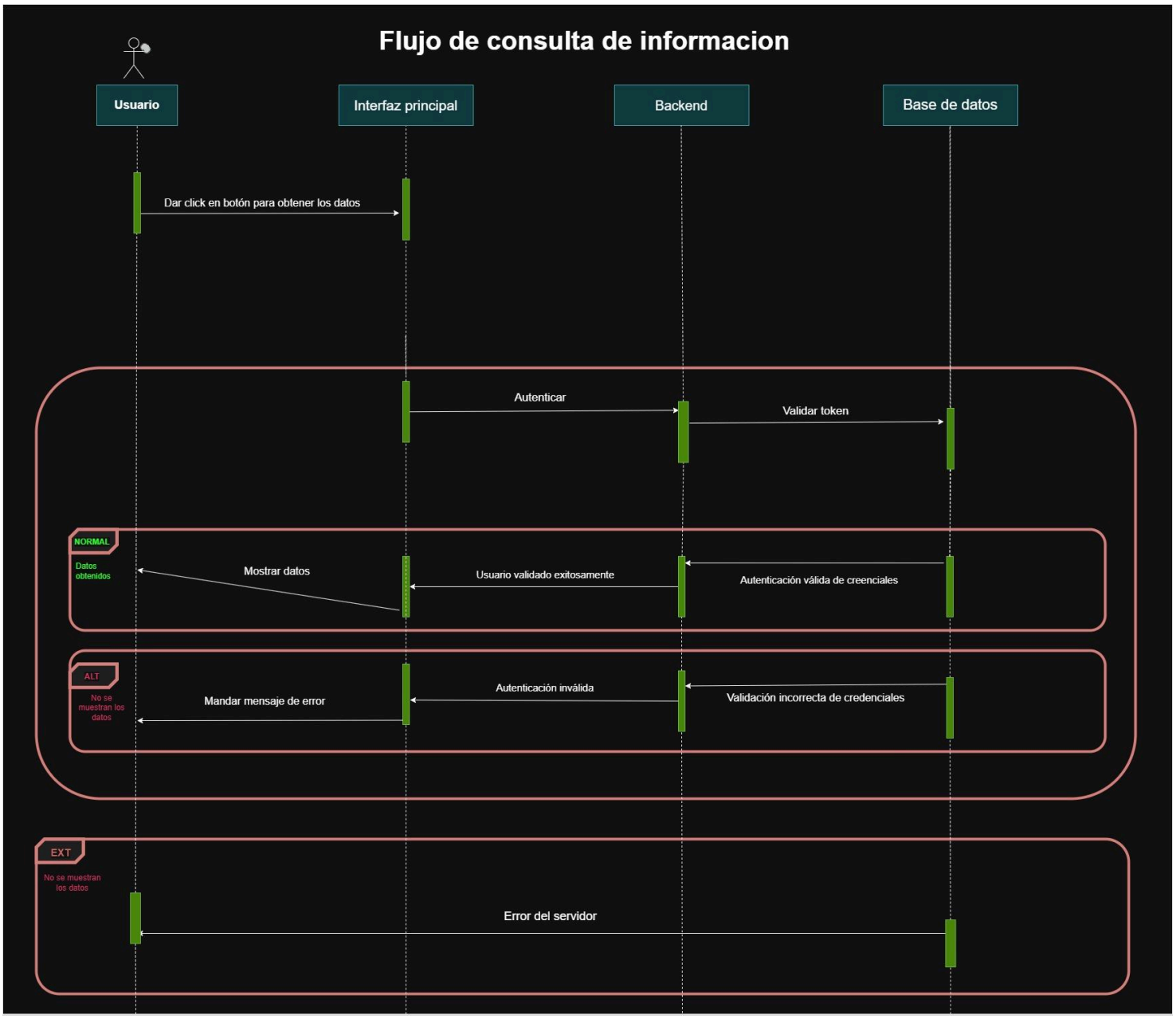
>> Flujo Alternativo 1:

v. Se produce un error en la autenticación del token.
v. Se envía un mensaje de error.

>> Flujos Alternativo 2:

v. Se produce una falla en el servidor.
v. Mandamos un mensaje de error.

> Diagrama de flujo del caso de uso:



(e) Registrar un incidente

Actor: Usuario

> Flujo Principal:

i. El usuario inicia sesión y selecciona la opción para registrar un incidente.
ii. Llena los campos solicitados para registrar el incidente.
iii. El sistema se actualiza con el nuevo incidente.
iv. Selecciona la opción de registrar incidente (actualiza el estado del incidente a ”no solucionado”).

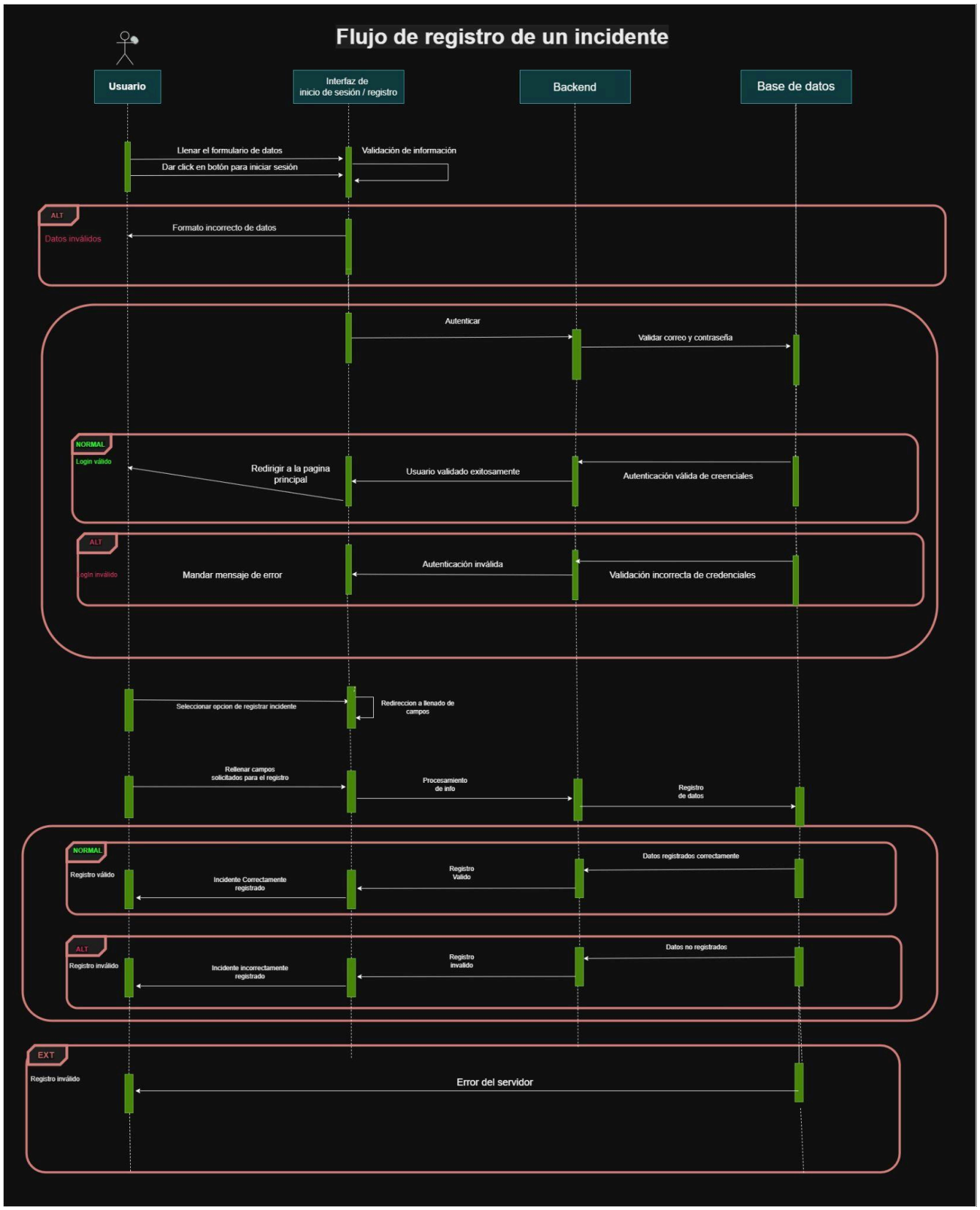
>> Flujo Alternativo 1:

v. Se produce un error en el registro de la información.
vi. Se envía un mensaje de error.

>> Flujos Alternativo 2:

v. Se produce una falla en el servidor.
vi. Mandamos un mensaje de error.

> Diagrama de flujo del caso de uso:



(f) Actualizar estado del incidente

Actor: Usuario

> Flujo Principal:

i. Selecciona la opción de actualizar incidente.
ii. Llena los campos necesarios y solicitados para poder actualizar el estado del incidente.
iii. El sistema se actualiza con la nueva información recibida del usuario (actualiza el estado del incidente a ”solucionado”, con una fotografía como prueba).

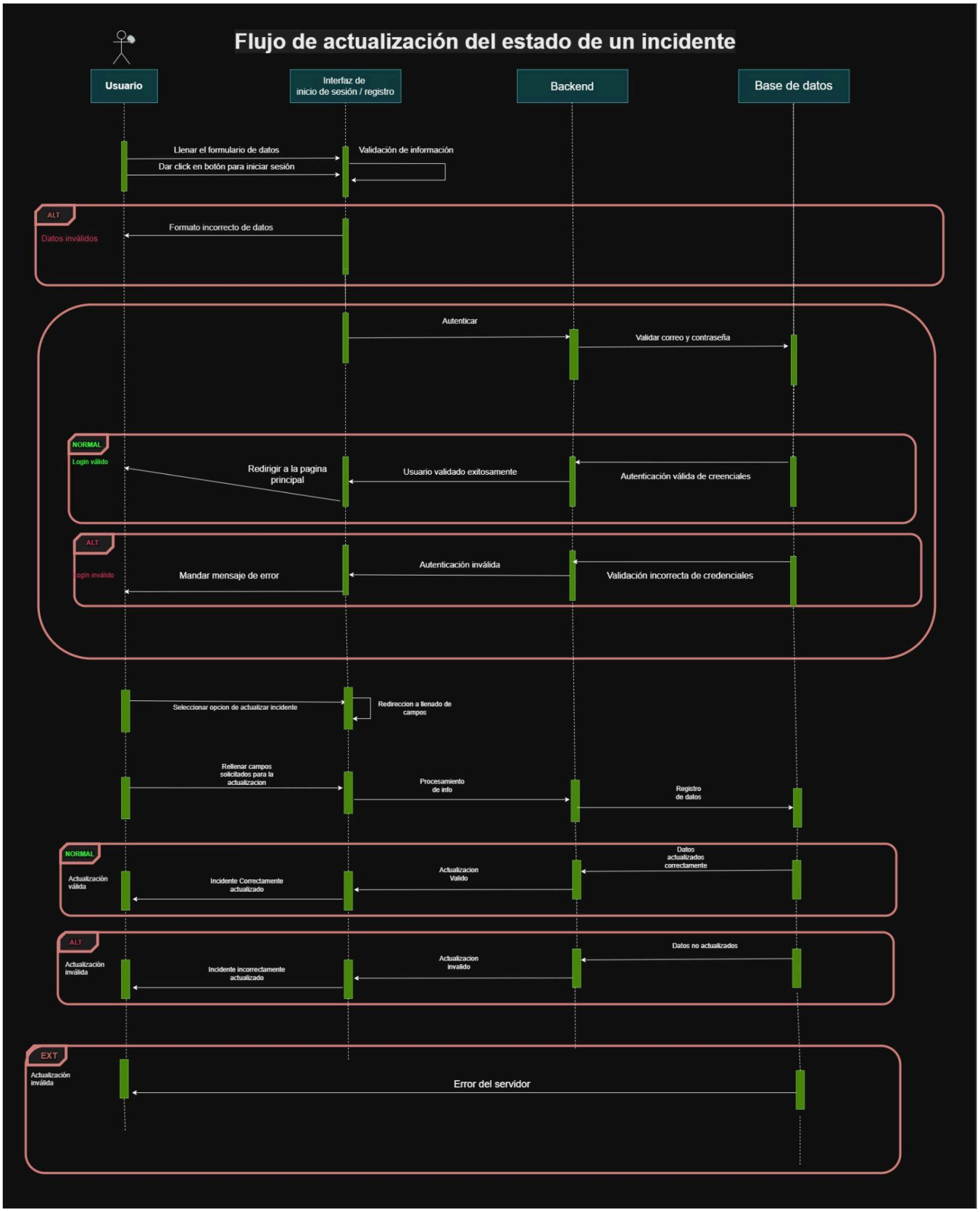
>> Flujo Alternativo 1:

iv. Se produce un error en la actualización de la información (estado o fotografía).
v. Se envía un mensaje de error.

>> Flujos Alternativo 2:

iv. Se produce una falla en el servidor.
v. Mandamos un mensaje de error.

> Diagrama de flujo del caso de uso



(g) Búsqueda de incidentes

Actor: Usuario

> Flujo Principal:

i. El usuario selecciona la opción de buscar incidente
ii. El usuario realiza una búsqueda, la cuál puede ser reducida a través de filtros que proporciona el sistema
iii. El sistema muestra los resultados encontrados en un mapa

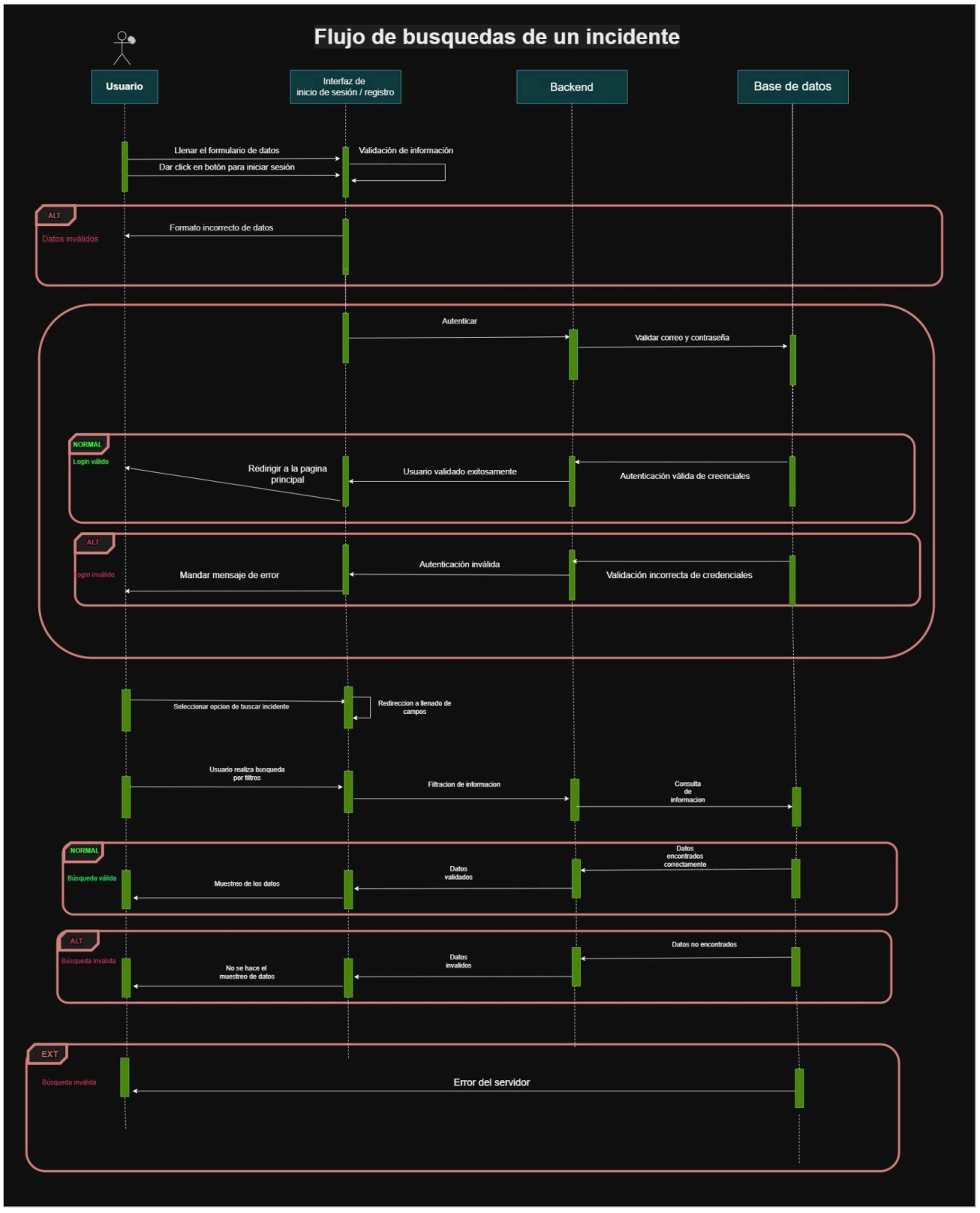
>> Flujo Alternativo 1:

iv. Si no se encuentra algún incidente con relación a la búsqueda solicitada, el sistema notifica al usuario.

>> Flujo Alternativo 2:

iv. Se produce una falla en el servidor.
v. Mandamos un mensaje de error.

> Diagrama de flujo del caso de uso:



(h) Notificar incidente cercano

Actor: Usuario

> Flujo Principal:

i. El usuario revisa su bandeja de notificaciones en su perfil.
ii. El sistema muestra los resultados de incidentes que se produjeron cerca de su área seleccionada.

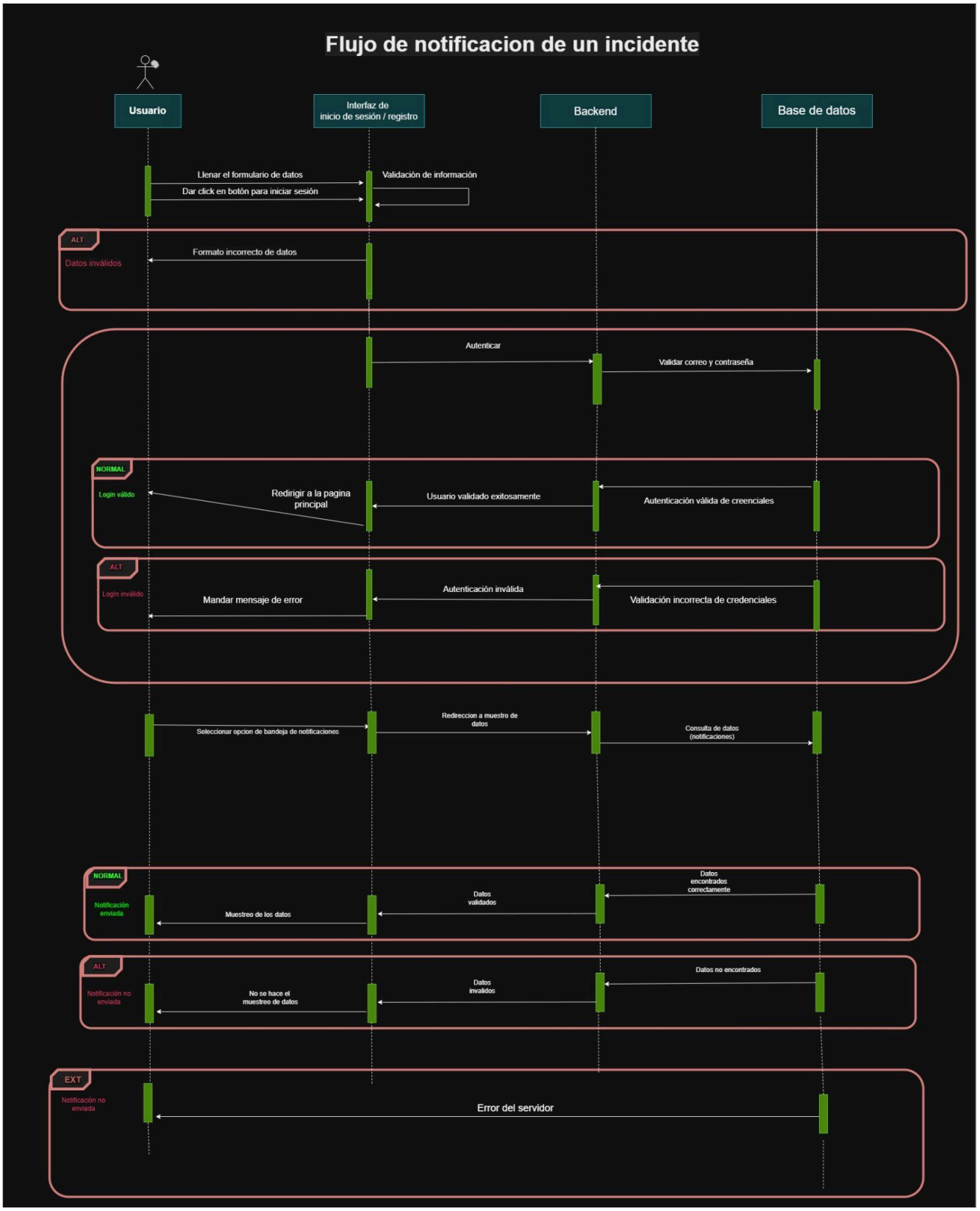
>> Flujo Alternativo 1:

iii. Si no se encuentra algún incidente que se haya reportado cerca del área, el sistema no notifica al usuario.

>> Flujo Alternativo 2:

iii. Se produce una falla en el servidor.
iv. Mandamos un mensaje de error.

> Diagrama de flujo del caso de uso:



(i) Mostrar incidente

Actor: Usuario

> Flujo Principal:

i. El usuario selecciona uno de los incidentes registrados dentro de una lista de incidentes.
ii. El sistema muestra la información correspondiente a los atributos del incidente.

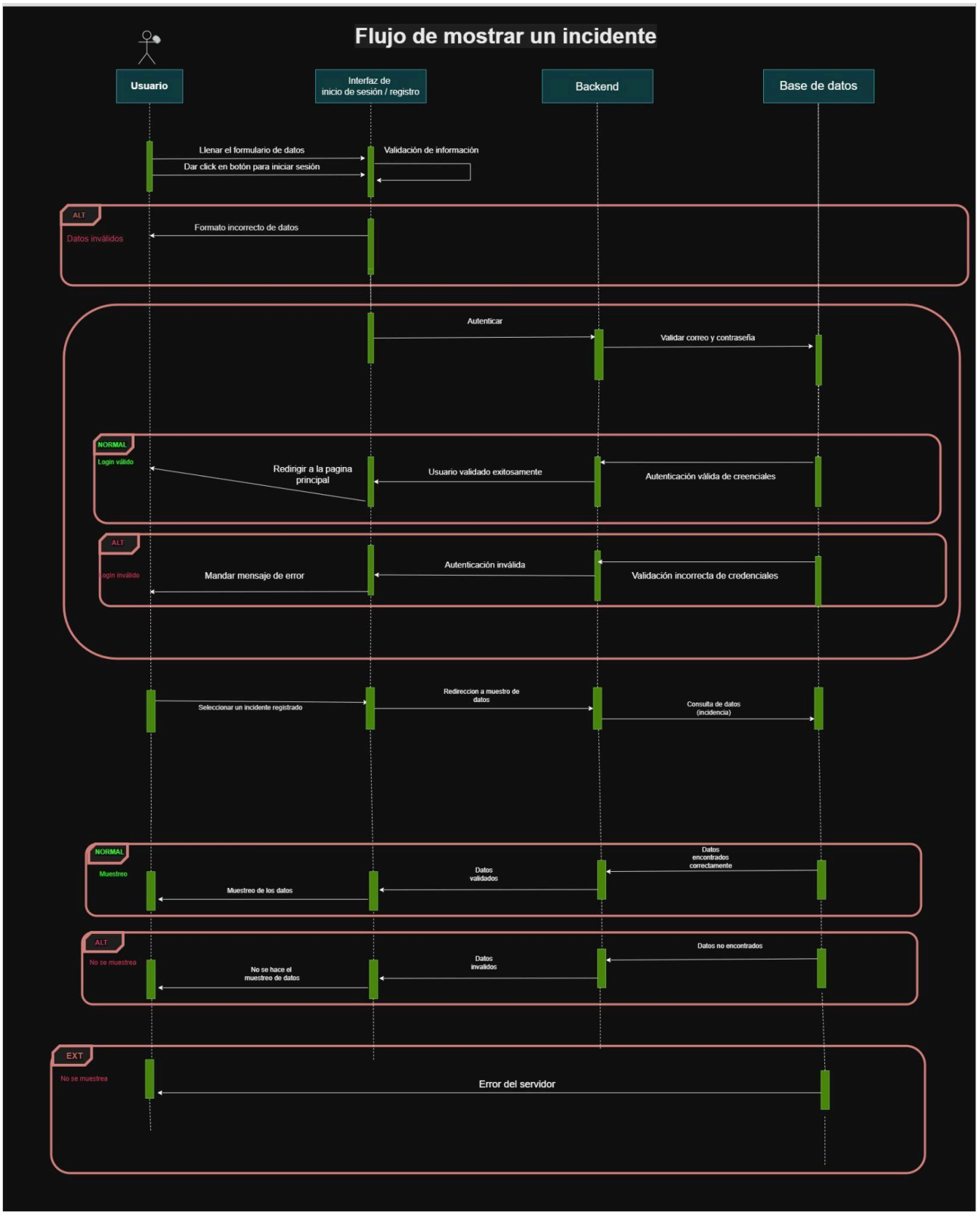
>> Flujo Alternativo 1:

iii. Si el incidente ha sido eliminado o ya no existe, el sistema notifica un error.

>> Flujo Alternativo 2:

iii. Se produce una falla en el servidor.
iv. Mandamos un mensaje de error.

> Diagrama de flujo del caso de uso:



(j) Comentar publicación de incidente

Actor: Usuario

> Flujo Principal:

i. El usuario selecciona mostrar incidente.
ii. El usuario realiza un comentario dentro del incidente.
iii. El sistema registra el comentario dentro de la sección de comentarios del incidente.

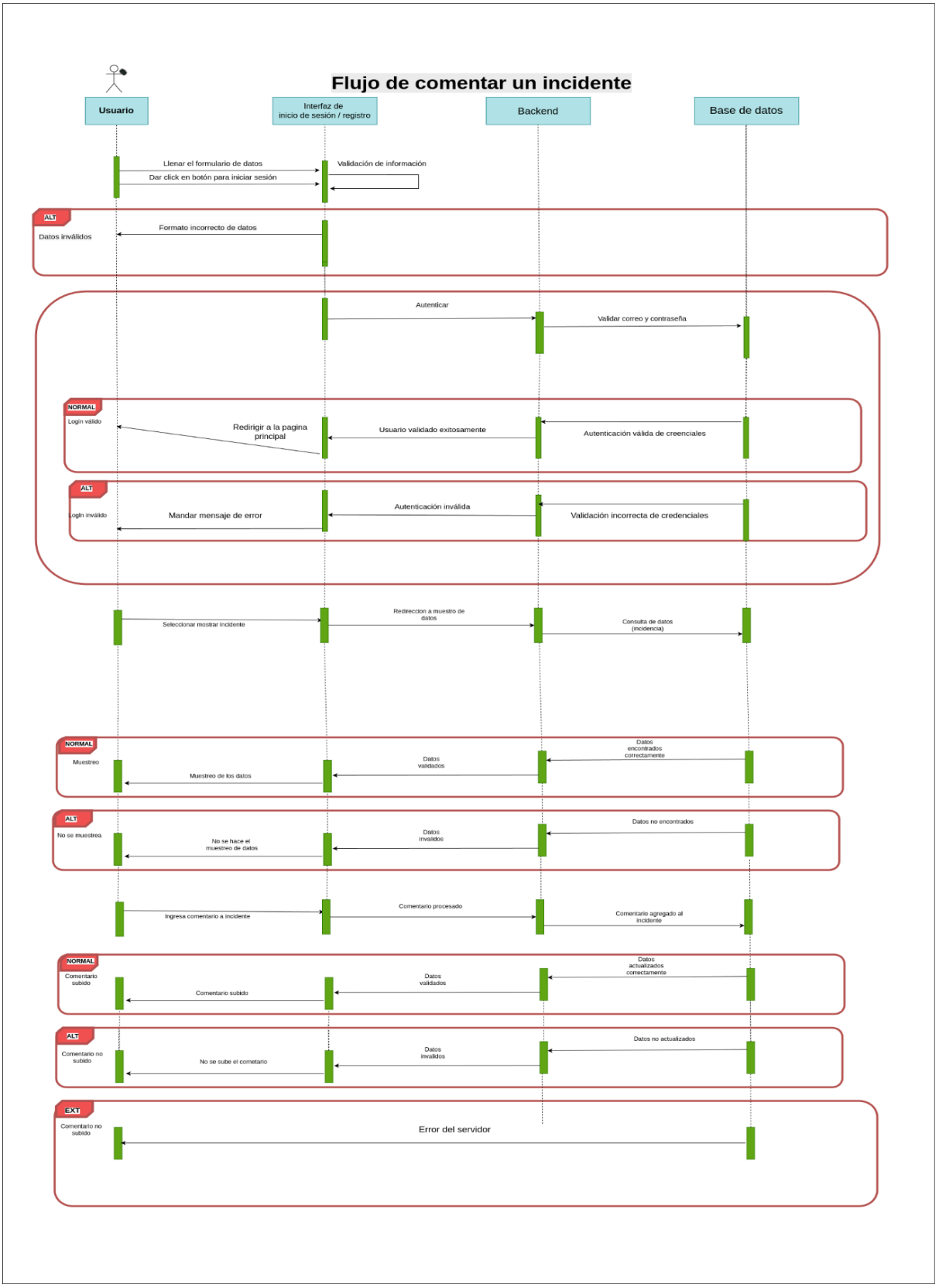
>> Flujo Alternativo 1:

iv. Se produce un error al realizar un comentario.

>> Flujo Alternativo 2:

iv. Se produce una falla en el servidor.
v. Mandamos un mensaje de error.

> Diagrama de flujo del caso de uso:



(k) Eliminación y/o edición de incidente

Actor: Usuario

> Flujo Principal:

i. El usuario selecciona la opción de “mis incidentes”.
ii. El usuario selecciona la opción de editar, en el incidente que desee editar.
iii. El sistema muestra los campos que se pueden editar.
iv. El usuario modifica los campos que necesita editar.

>> Flujo Alternativo 1:

v. Si no se encuentra algún incidente que el usuario haya subido el sistema muestra un mensaje “no hay incidentes”.

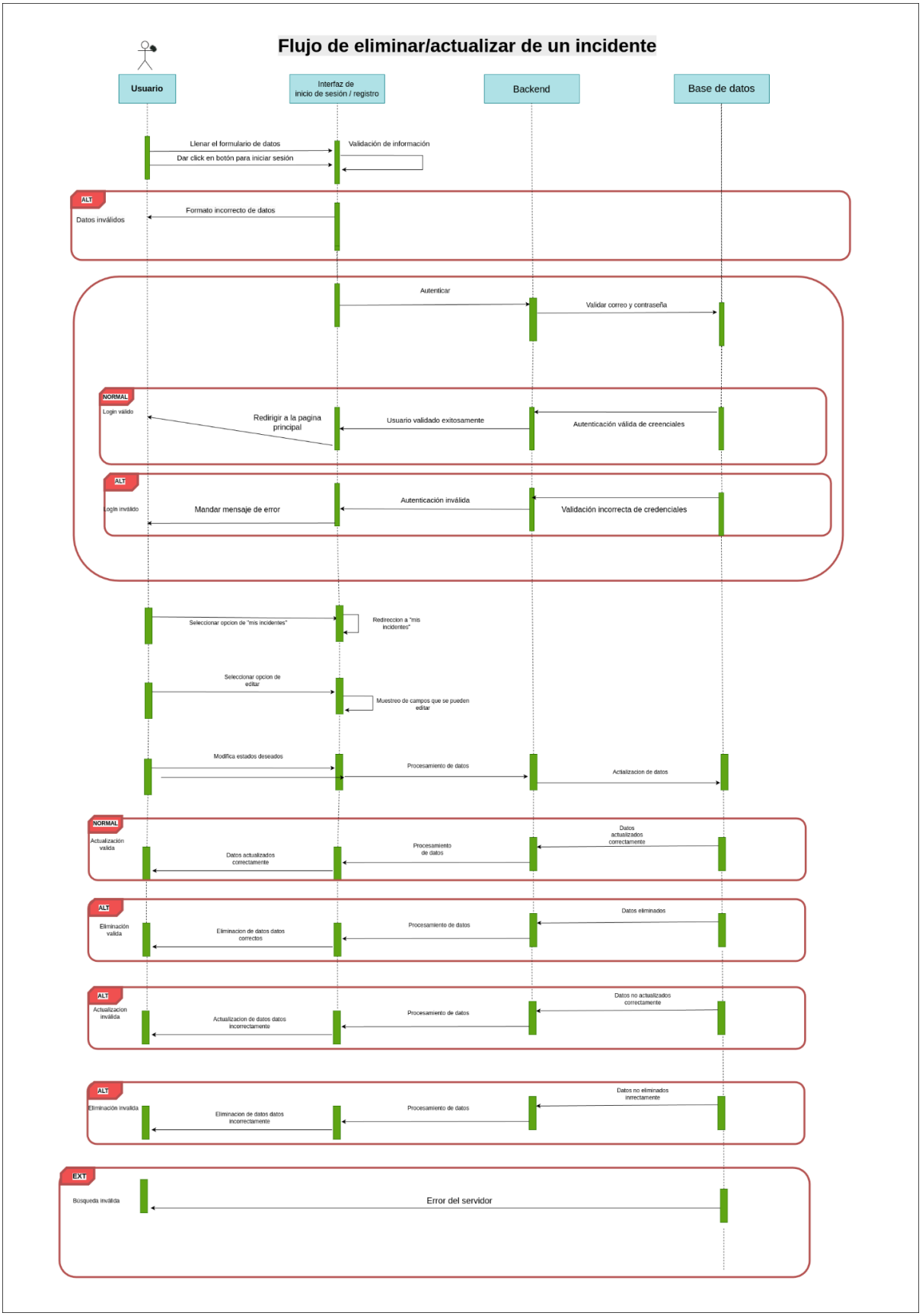
>> Flujo Alternativo 2:

v. Si desea eliminar el incidente, le da al botón de eliminar incidente.
vi. El sistema elimina el incidente de la base de datos.

>> Flujo Alternativo 3:

v. Se produce una falla en el servidor.
vI. Mandamos un mensaje de error.

> Diagrama de flujo del caso de uso:



(l) Reportar incidente

Actor: Usuario

> Flujo Principal:

i. El usuario selecciona mostrar incidente.
ii. El usuario selecciona la opción de reportar incidente.
iii. El usuario elige el motivo de reporte, los cuales pueden ser por falsedad de información, datos erróneos, contenido inapropiado, y la opción de comentar una descripción del reporte.
iv. El sistema registra el reporte a una base de registro de reportes.

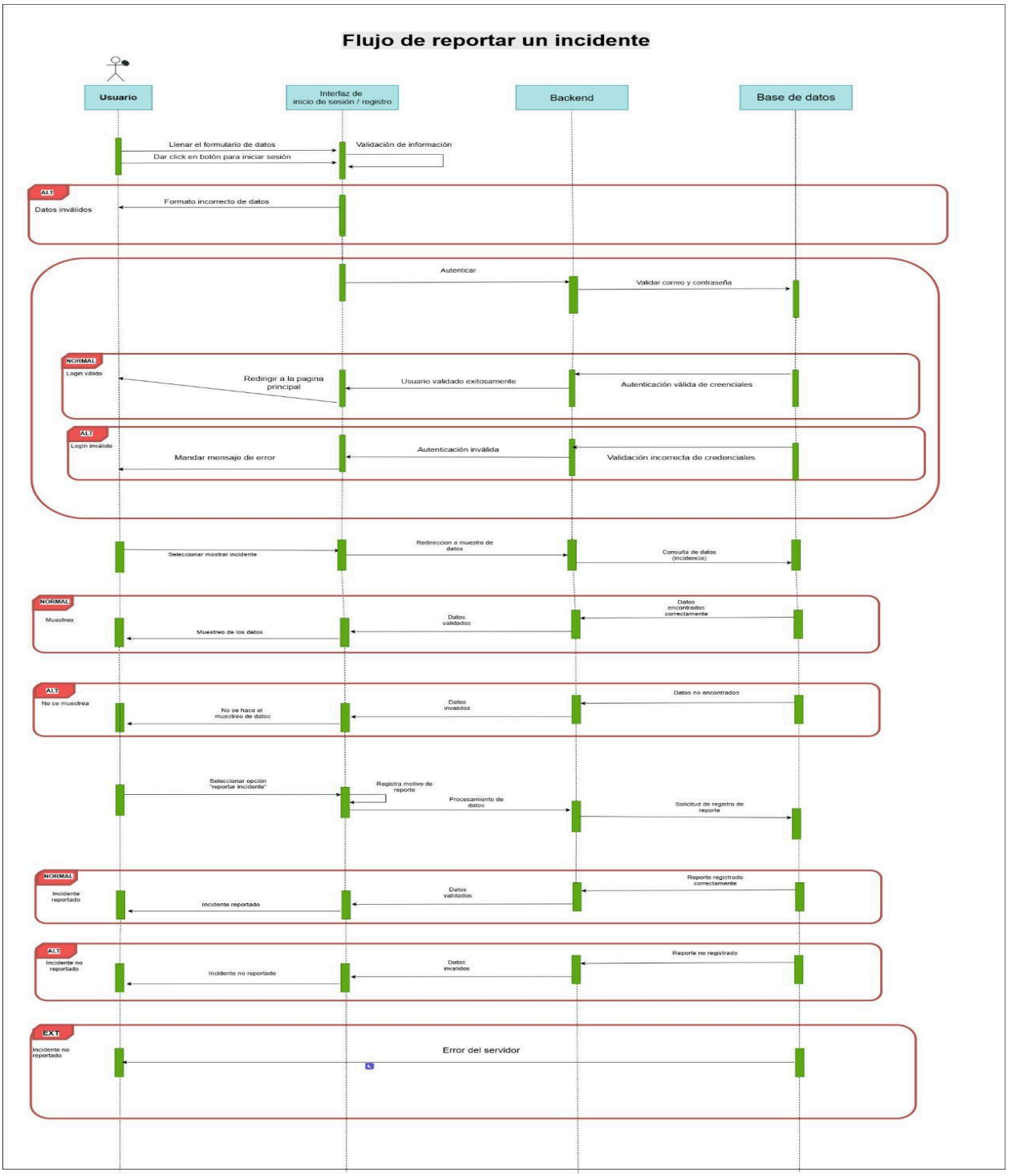
>> Flujo Alternativo:

v. Se produce un error al realizar el reporte.

>> Flujo Alternativo 2:

v. Se produce una falla en el servidor.
vI. Mandamos un mensaje de error.

> Diagrama de flujo del caso de uso



4. Requisitos de Hardware y Software

(a) Hardware

i. Servidor con al menos 8Gb de RAM y procesador AMD Rayzen 3 3200G

ii. Bases de datos alojadas en un sistema con almacenamiento SSD

(b) Software

i. Lenguaje de Programación: Kotlin / HTML / CSS / Javascript

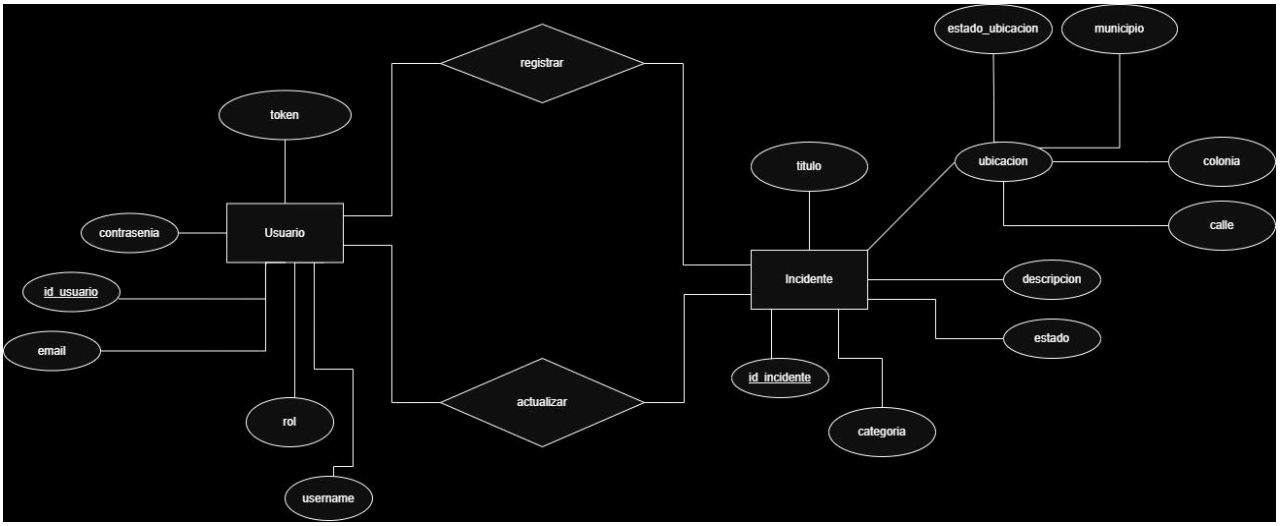
ii. Base de datos: PostgreSQL

iii. Framework web: React / Spring

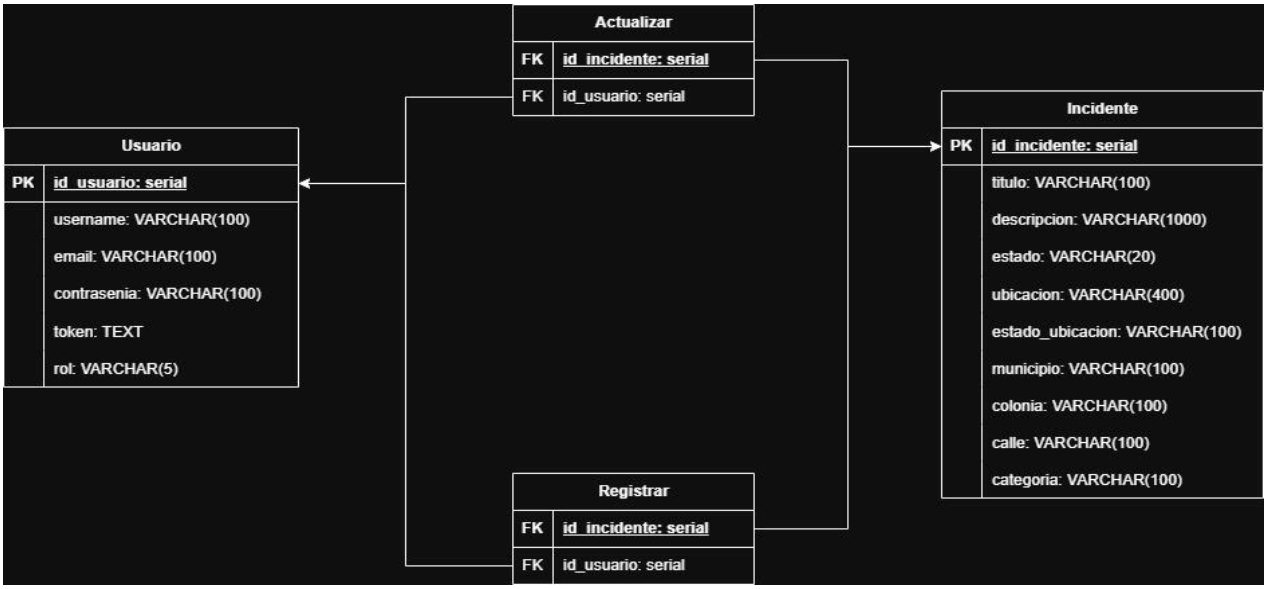
5. Base de Datos (SQL)

En nuestro proyecto, utilizamos el sistema de gestión de bases de datos relacional PostgreSQL, dado que estamos trabajando con una aplicación web y también porque estamos acostumbrados a trabajar con este sistema.
Por ahora, contamos con dos entidades (Usuario e Incidente) y dos relaciones (Actualizar y Registrar) para nuestra base. Cabe mencionar que esto es muy propenso a cambiar; conforme se comience a implementar la lógica detrás de los incidentes muy seguramente se agreguen nuevas entidades y relaciones.

Se cuenta con el diagrama entidad-relación para observar la organización de la base de datos:



Así mismo, se transformó a un diagrama relacional para una representación más clara respecto a la estructura de la BDD.



6. Documentación sobre Backend y Frontend

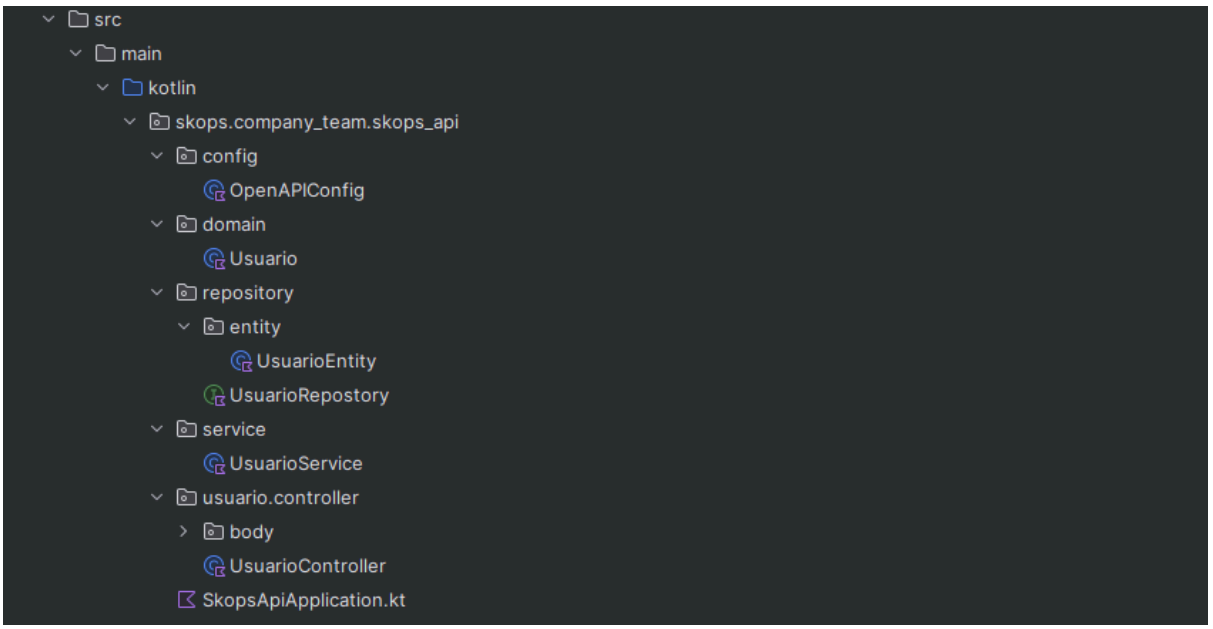
El proyecto Skops-Incidencias consta principalmente de dos módulos que sobresalen: el Backend y el Frontend, ambos estrechamente relacionados con las funciones que los usuarios pueden realizar en la aplicación de incidencias.

6.1. Backend (v1.0.0)

El *Backend* del proyecto está desarrollado en el lenguaje de programación **Kotlin**, esto debido fuertemente al framework que se utiliza para desarrollarlo, qué es *Spring*, que nos da varias herramientas para implementar un *Backend* completo en dicho lenguaje.

Este funciona a través de *Maven,* cuyo principal fin es la creación de proyectos completos, simplificando el compilar los distintos módulos que forman el componente en general.

Hacemos uso de las convenciones de la ingeniería de software para el *backend*. Primeramente, al organizar el componente en distintos paquetes que se relacionan entre sí. Nuestra estructura **actual** es la siguiente:



Se usa el paquete *entity* para definir los atributos que tendrá el usuario en la aplicación. El paquete *repository* funciona como conexión con la base de datos, ésta realiza las inserciones y consultas para actualizar la base de datos cada vez que el usuario hace uso de un recurso de la aplicación.

El paquete *service* recibe entradas en **JSON** para copiarlas en la base de datos, así como devolver salidas en formato **JSON** para que las funcionalidades devuelvan datos que serán guardados en la BD.
Finalmente, tenemos al paquete *controller* el cual define los *endpoints* que se usan en el *frontend* y a los cuales acude el usuario para realizar sus operaciones particulares. Este hace uso del paquete *service* y atrapa las posibles excepciones, además de realizar la conexión directa con la dirección del *frontend*.

Los *endpoints* definidos en *controller* son los siguientes:

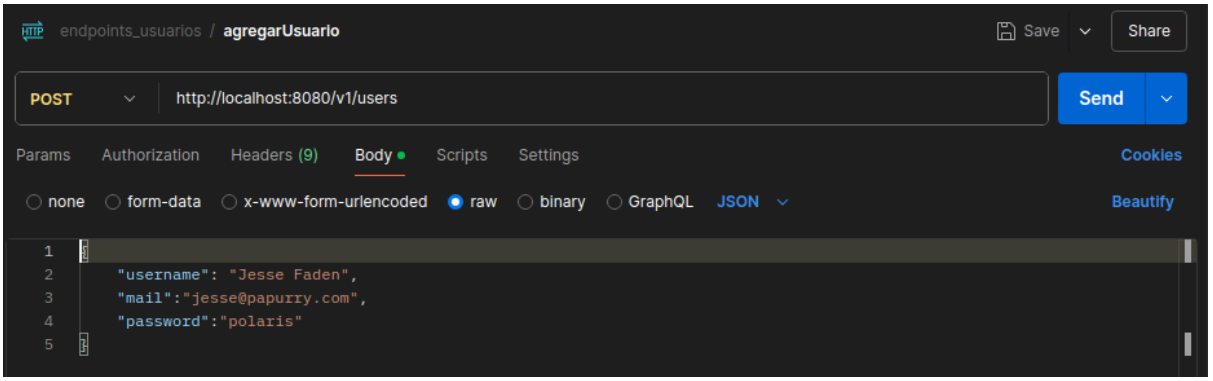
1.- POST: /v1/users/

Este *endpoint* sirve para registrar nuevos usuarios, a través de la función '*agregarUsuario*'.

```
import org.springframework.web.bind.annotation.CrossOrigin;

@CrossOrigin(origins = ["http://localhost:3000"])  Francisco Daniel García López +2
@RestController
@RequestMapping("/v1/users")

class UsuarioController (var usuarioService: UsuarioService){

    @PostMapping  Francisco Daniel García López +1
    fun agregarUsuario(@RequestBody usuarioBody: UsuarioBody): ResponseEntity<Any> {
        val miUsuario = Usuario(username = usuarioBody.username,
                                mail = usuarioBody.mail,
                                password = usuarioBody.password)
        val response = usuarioService.addUser(miUsuario)
        return ResponseEntity.ok(response)
    }
```

Recibe un cuerpo **JSON** con los parámetros: *username, mail, password.* en ese mismo orden para ser mapeados correctamente.

Ejemplo:

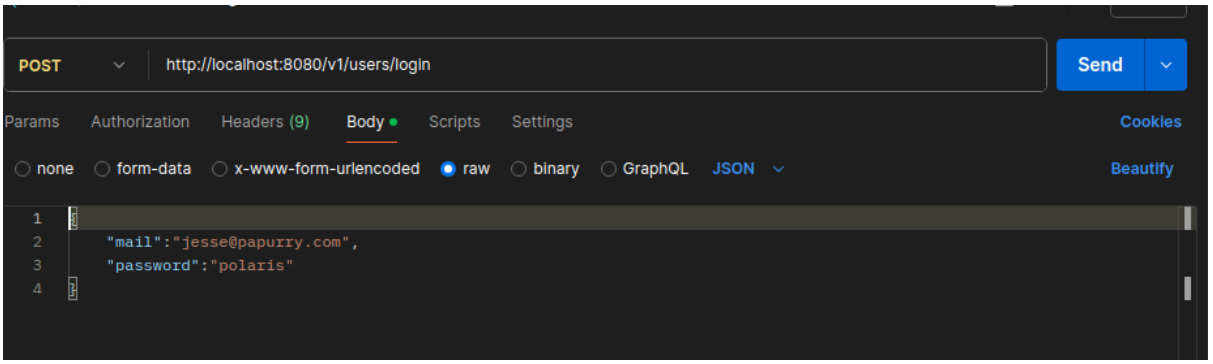


2.- POST: /v1/users/login

Este *endpoint* se encarga de ingresar a la aplicación una vez que se tiene una cuenta registrada. Recibe un **JSON** con un correo y contraseña que hayan sido registrados anteriormente, a través de la función *createLogin*.

```
@PostMapping("/login")  Francisco Daniel García López
fun createLogIn(@RequestBody loginUserBody: LoginUserBody): ResponseEntity<Usuario> {
    val result = usuarioService.login(loginUserBody.mail,loginUserBody.password)
    return if (result == null){
        ResponseEntity.notFound().build()
    } else {
        ResponseEntity.ok(result)
    }
}
```

Ejemplo:



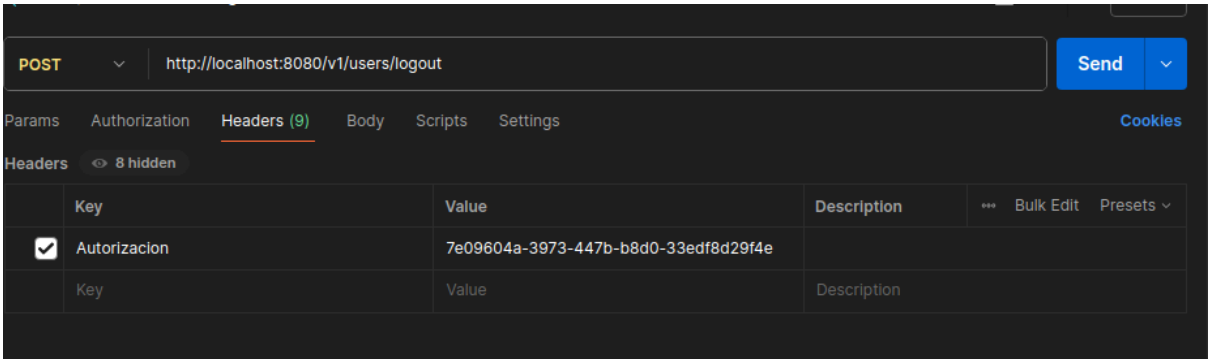
3.- POST: /v1/users/logout

Este *endpoint* sirve para cerrar sesión y que el usuario salga con éxito de la aplicación, a través de la función *createLogOut*.

```
@PostMapping("/logout")  Francisco Daniel García López
fun createLogOut(@RequestHeader("Autorizacion") token:String): ResponseEntity<String>{
    val logout = usuarioService.logout(token)
    return if (!logout){
        ResponseEntity.badRequest().build()
    } else {
        ResponseEntity.ok( body: "Sesión Cerrada")
    }
}
```

Cada vez que se ingresa en la aplicación, se genera un token único para dicho usuario, el cual se pide como autorización para poder cerrar sesión.

Ejemplo:

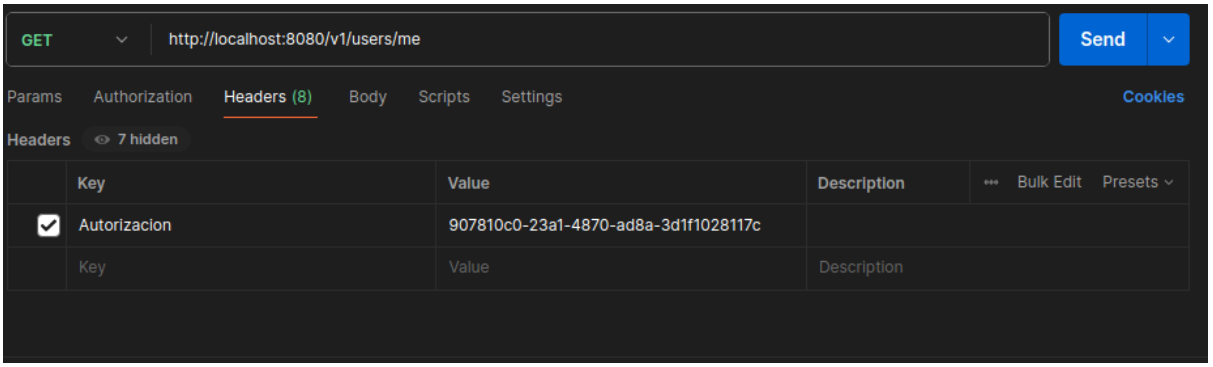


4.- GET: v1/users/me

Este *endpoint* sirve para poder ver la información de un usuario, a través de la función *meUsuario*. Al igual que *logout*, el *endpoint* funciona con el token único para pedirlo como autorización.

```
@GetMapping("/me")  Francisco Daniel García López
fun meUsuario(@RequestHeader("Autorizacion") token:String): ResponseEntity<Usuario>{
    val response = usuarioService.getInfoAboutMe(token)
    return if (response != null){
        ResponseEntity.ok(response)
    } else {
        ResponseEntity.status(401).build()
    }
}
```

Ejemplo:



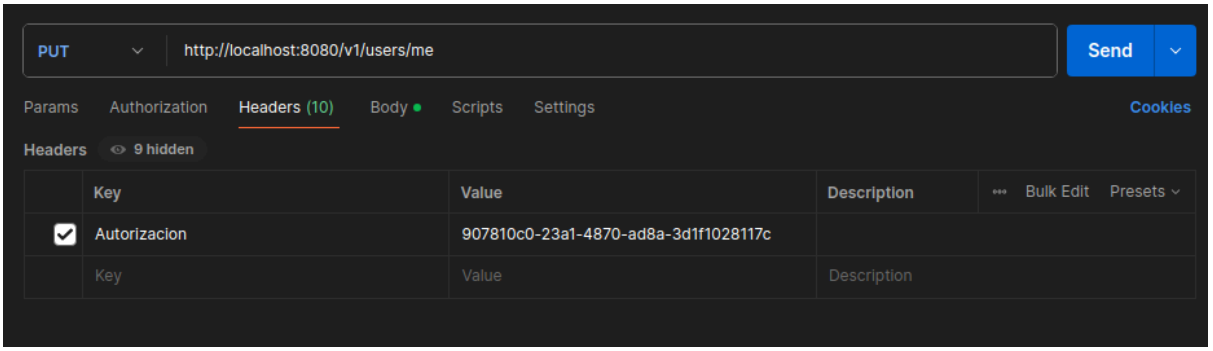
5.- PUT: GET: v1/users/me

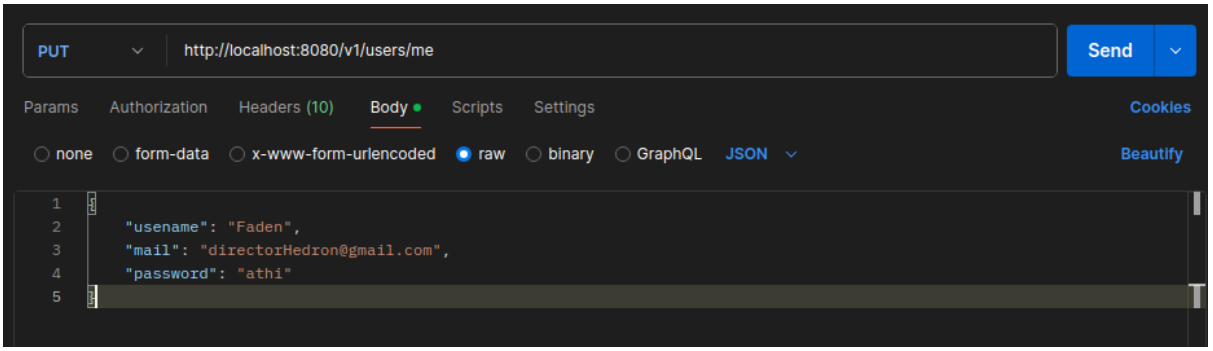
Este *endpoint* también sirve para actualizar la información del usuario.

```
@PutMapping("/me")  Francisco Daniel García López
fun actualizarUsuario(@RequestHeader("Autorizacion") token: String, @RequestBody usuarioUpdateBody
    if (token == "") {
        return ResponseEntity.status(401).build()
    }
    val usuarioActualizado = usuarioService.updateUser(token, usuarioUpdateBody)
    return if (usuarioActualizado != null) {
        ResponseEntity.ok(usuarioActualizado)
    } else {
        ResponseEntity.status(401).build()
    }
}

}
```

Recibe un ***JSON*** con los nuevos nombre de usuario, correo y contraseña. También se pedirá el token único como autorización.

Ejemplo:





6.2. Swagger

En la estructura del backend, contamos con una carpeta llamada *config* que posee dentro una clase de nombre ***OpenAPIConfig***, la cual se encarga de personalizar y configurar la documentación del *backend* utilizando la herramienta *Swagger*.

Asignamos un título propio, descripción, versión y agrupamiento a la API, mientras que, en la clase *UsuarioController*, colocamos ciertos comentarios para poder configurar esto de manera correcta.

El uso de Swagger nos permite, además de visualizar cada endpoint junto con una breve descripción y ejemplos, hacer uso de ellos y realizar pruebas sin necesidad de usar *Postman*.

En la sección de '**Ejecución**' se detalla de mejor manera cómo hacer uso de esto último.

6.3. Frontend (v1.0.0)

El *frontend* está desarrollado en el lenguaje React, ya que resultó ser el más sencillo de usar para nosotros, y así lograr facilitarnos la implementación de la interfaz que utilizará el usuario para poder reportar sus incidencias.

Nuestra interfaz tiene cinco funcionalidades (hasta el momento), las cuáles son **Registrarse, Iniciar Sesión**, **Mostrar Datos**, **Actualizar Datos** y **Cerrar Sesión**.

Comenzando con **Registrar**, en esta función el usuario debe ingresar un nombre de usuario, su email y una contraseña de manera obligatoria, para poder darlos de alta en nuestra base de datos. Esta acción es realizada por ***handleRegistro*** dentro de nuestros componentes, el cual hace un *Post* con los datos necesarios para registrar al usuario.

Similar a *Registrar*, en **Iniciar Sesión** el usuario ingresa su correo y contraseña con la que se registró anteriormente, y ahora ***handleLogin*** se encarga de buscarlo en la base de datos y asignarle un token temporal.

Luego tenemos dos funciones más que sirven para que el usuario pueda interactuar con su cuenta, las cuales son **Mostrar Datos** y **Actualizar Datos**.

Mostrar Datos funciona una vez que el usuario inició sesión. Se da click en la pestaña de 'Mi Cuenta', y se despliega la información del usuario. La clase *Cuenta* maneja y busca la información del usuario mediante el token y le regresa dos datos de éste. Cabe aclarar que, **por ahora**, solo se le muestra al usuario su usuario y

correo electrónico, dado que el token y contraseña son datos con un mayor grado de privacidad. Así mismo, no se le muestra su rol porque, en esta iteración, el rol de todos los usuarios es '*user*'.

Actualizar Datos se realiza mediante el componente homónimo, asegurándose que en la barra de navegación se pueda acceder a la actualización de los datos del usuario. El campo de la contraseña no es obligatorio, por lo que puede dejarse en blanco.

Por último tenemos **Cerrar Sesión**, que en la interfaz consiste en dos botones que, en el momento cuando se presione alguno de los dos, la clase Cuenta con ayuda de su método ***handleLogout*** hace un *Post* para buscar al usuario por token y después procede a eliminar dicho token de la base de datos. Finalmente, regresa al usuario a la página para iniciar sesión o registrarse.

7. Ejecución

A continuación se detalla cómo ejecutar ambos módulos: backend y frontend.

7.1. Backend

1. Clonar el repositorio **skops-api**.
2. Correr un contenedor para la base de datos.
3. Ingresar en el directorio

   skops-api/src/main/resources/application.properties

   y cambiar los campos correspondientes:

   spring.datasource.username=nombreDeTuContenedor
   spring.datasource.password=contraseñaDeTuContenedor

4. Abrir un sistema manejador de bases de datos, y correr los scripts del directorio

   skops-api/documentos/base-de-datos

   Primero se debe ejecutar **BDD.sql**, para luego abrir y ejecutar el script **DDLv1.sql** en la base de datos recién creada por el script anterior: "Incidencias-DataBase".
5. Realizar la configuración del repositorio en IntelliJ (JDK, maven, Sync Project, Build Project, etc.)
6. Ejecutar la clase del directorio donde se encuentra la clase ejecutable, es decir:

src/main/kotlin/skops/company_team/skops_api/SkopsApiApplication.kt

7. Para acceder a la documentación de Swagger y probar los endpoints sin necesidad de Postman, se ejecuta el backend y luego se ingresa a la dirección:

http://localhost:8080/swagger-ui.html

7.2. Frontend

1. Clonar el repositorio **skops-frontend.**
2. Para *Fedora*, se ejecutan los siguientes comandos en terminal:

   curl -fsSL https://rpm.nodesource.com/setup_20.x | sudo bash -
   sudo dnf install -y nodejs

3. Una vez terminado el proceso anterior, se ejecutan los siguientes comandos en la carpeta **skops-frontend**:

   rm -rf node_modules package-lock.json
   npm install
   npm start

4. Para *Ubuntu*, se ejecutan los siguientes comandos:

   sudo apt update
   sudo apt install -y nodejs npm

5. Comprobamos la instalación:

   node -v
   npm -v

6. Una vez terminado el proceso anterior, se ejecutan los siguientes comandos en la carpeta **skops-frontend**:

   rm -rf node_modules package-lock.json
   npm install
   npm start

7. La página se abre por sí sola en el navegador.