**Documentación proyecto Cosmos – Página Web del Tiempo**

**1. Objetivos del proyecto**

* El propósito del proyecto Cosmos (página web del tiempo) es poder visualizar la información climática por municipio en Colombia.
* Mostrando datos importantes como:
  + Temperatura
  + Humedad
  + Lluvias
  + Tormentas
  + Sensación térmica
  + Etc.
* Se busca ofrecer una experiencia moderna con una interfaz bonita y completa para usuario habituales.

**2. Alcance preliminar**

**2.1 Funcionalidades clave**

* Login con API de Google Sign-In.
* Panel de administración de usuario.
* Visualización del tiempo por municipio Colombia.
* Visualización histórica con gráficas del tiempo.
* Visualización del pronóstico.
* Filtro básico para pronóstico del tiempo.
* Notificación vía Gmail sobre posibles tormentas en X municipio.
* Notificación Push en el navegador como pop-ups del sistema operativo incluso si la pestaña del navegador no está activa.

**2.2 Restricciones conocidas (presupuesto, tiempo, tecnología)**

* Tiempo “Indefinido”
* Presupuesto nulo; todas las tecnologías a utilizar deben ser free o fremium.
* Serán tecnologías rápidas que faciliten el desarrollo con un enfoque moderno.

**3. Stakeholders y roles**

**3.1 Usuarios principales**

* Persona encargada de revisar el proyecto para pasar la entrevista técnica.
* Usuario habitual que necesite conocer el tiempo con un enfoque moderno y completo.

**3.2 Quién toma decisiones en el cliente**

* La aprobación del proyecto recaerá sobre el ingeniero de software revisor/evaluador (David Esteban Jiménez), quien definirá si el producto cumple con los objetivos planteados.
* El desarrollador a cargo es libre de tomar las decisiones que considere pertinentes cumpliendo con el plazo y objetivo principal.

**3.3 Product Owner**

* El Product Owner del proyecto es David Felipe Soto Gallego encargado de definir la visión, priorizar funcionalidades y gestionar el alcance del proyecto.

**4. Entregables esperados**

**4.1 Producto mínimo viable (MVP)**

* Página del tiempo (Cosmos) con selección por municipio.

**4.2 Entregas parciales o demos intermedias**

* El proyecto se entregará completo sin entregables parciales o demos intermedias.

**5. Tiempos y presupuesto estimados**

**5.1 Fechas clave**

* Plazo indefinido a partir de 26/08/2025.

**6. Criterios de éxito**

**6.1 Cómo se medirá si el proyecto es exitoso**

* El ingeniero de software David Esteban Jiménez revisará, evaluará y dará seguimiento al proyecto realizado.

**7. Riesgos iniciales**

**7.1 Dependencias externas**

* Poco probable pero posible pago o consumo de los “créditos” hacia api fremium.
* Posible error de envío de correo por la dependencia externa.

**8. Ingeniería de Requerimientos**

**8.1 Descripción General del Sistema**

* Contexto: Sistema construido para visualizar el estado del clima, pronóstico e histórico por municipio en Colombia. Tiene la posibilidad de mandar notificación de posible tormenta con una interfaz moderna y personalizable.
* Usuarios: usuario evaluador y usuario habitual.
* Ámbito del producto: escenarios de uso:
  + consulta clima
  + consultar pronóstico
  + recibir alertas
  + graficar históricos
  + personalización de la UI

**8.2 Requerimientos Funcionales (RF)**

Ejemplo:

* RF1: El sistema permitirá al usuario autenticarse con Google Sign-in
* RF2: El Sistema mostrará el estado actual del clima por municipio en Colombia.
* RF3: El sistema mostrará el pronóstico del clima por municipio en Colombia.
* RF4: El sistema generará gráficas históricas a partir de los datos climáticos registrados.
* RF5: El sistema enviará notificaciones de posible tormenta vía correo electrónico.
* RF6: El sistema enviará notificaciones push (Tipo pop-ups) en el navegador, aunque la pestaña no esté activa.
* RF7: El sistema permitirá al usuario poner como favorito los municipios que quiera.
* RF8: El sistema hará un tour si es la primera vez que el usuario ingresa a la aplicación.

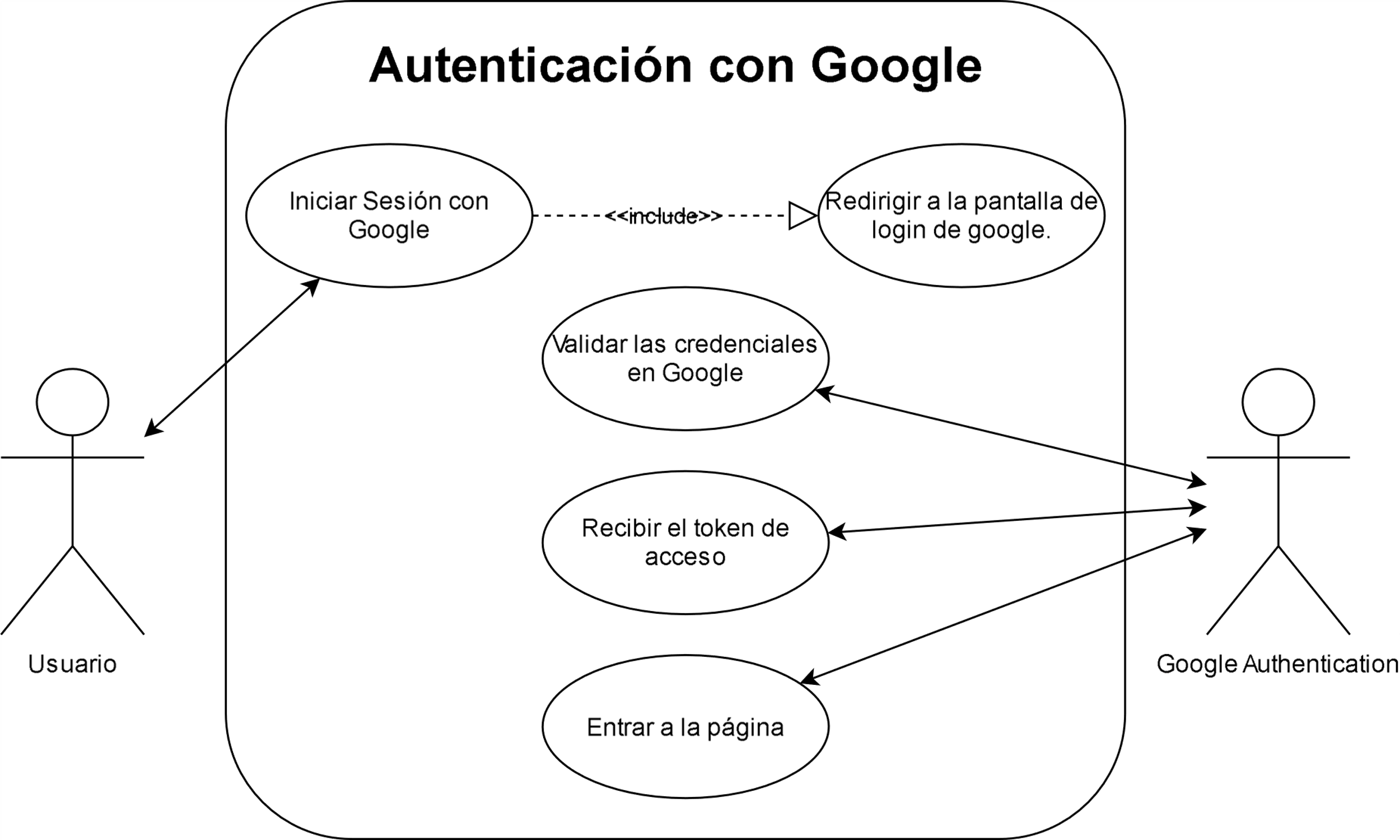
**8.3 Requerimientos No Funcionales (RNF)**

Ejemplo:

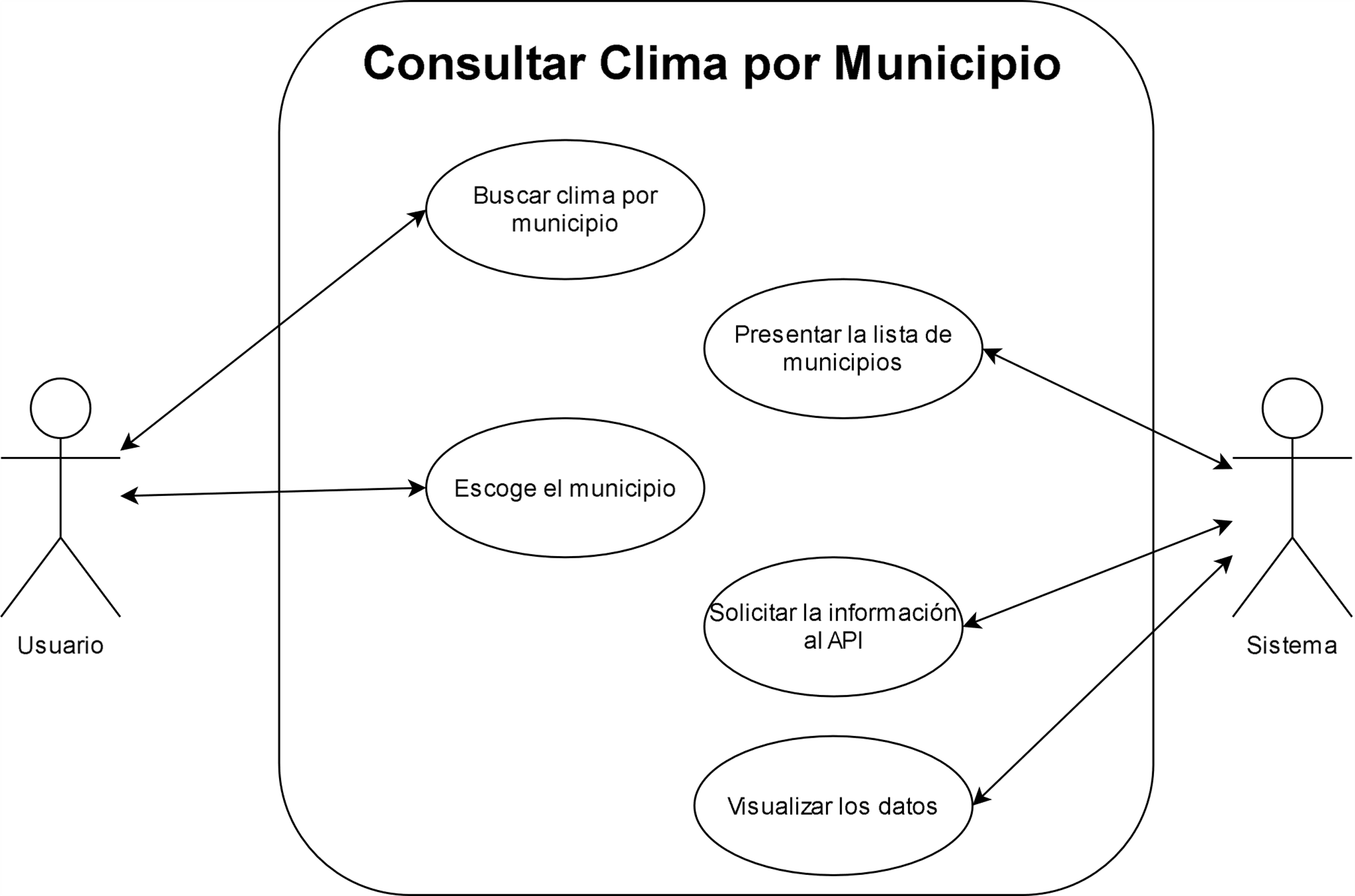
* RNF2: El sistema tendrá un diseño responsive para dispositivos móviles.
* RNF3: El sistema deberá implementar protocolos seguros (HTTPS, OAuth2).
* RNF4: El tiempo de carga no superará los 3 segundos con una conexión de 10Mbps.

**8.4 Casos de Uso (mínimo diagramas y texto breve)**

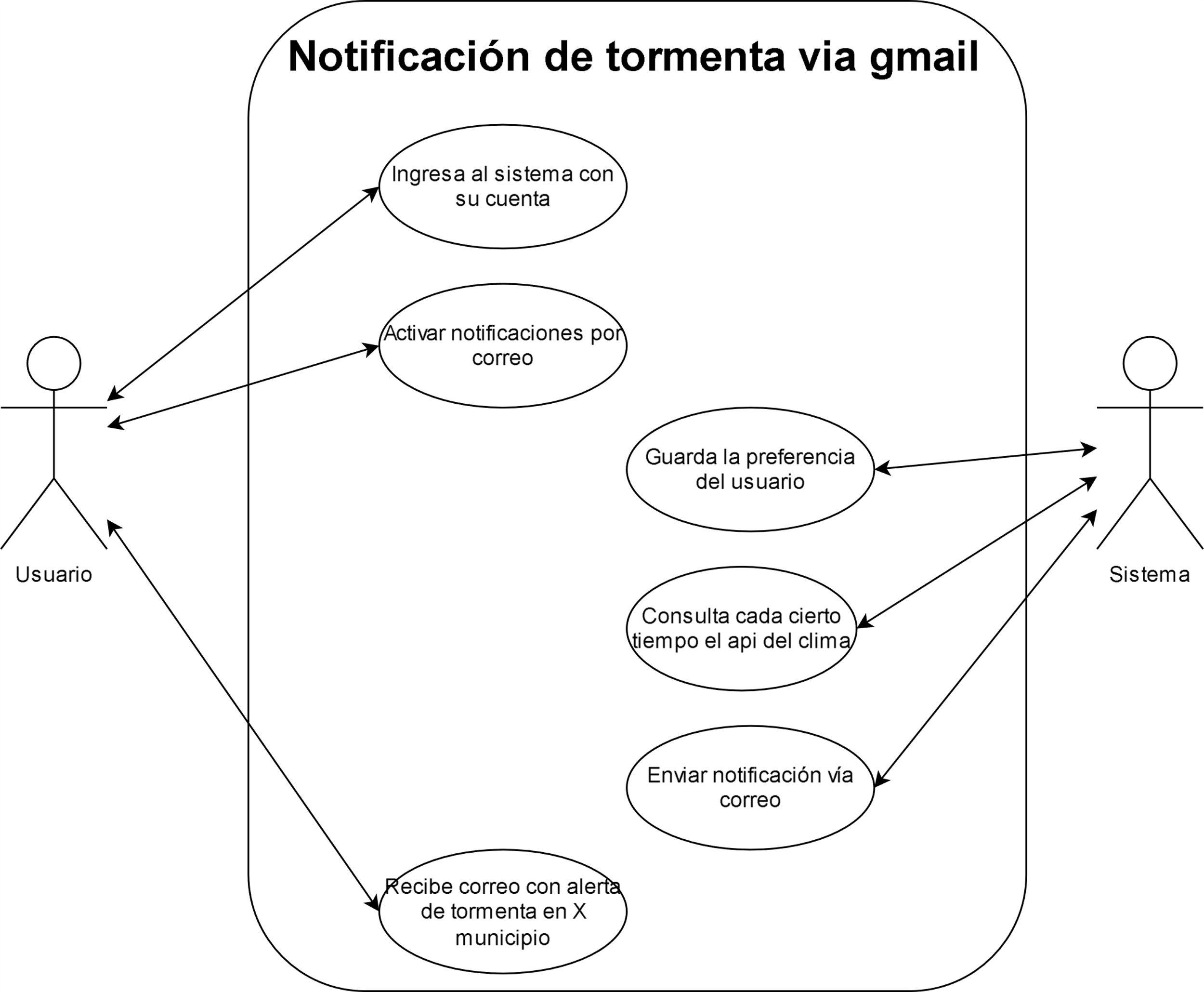
* Caso de uso 1: Autenticarse con Google.



* **Caso de uso 2: Consultar clima por municipio**



* **Caso de uso 3: Recibir alerta de tormenta vía Gmail**



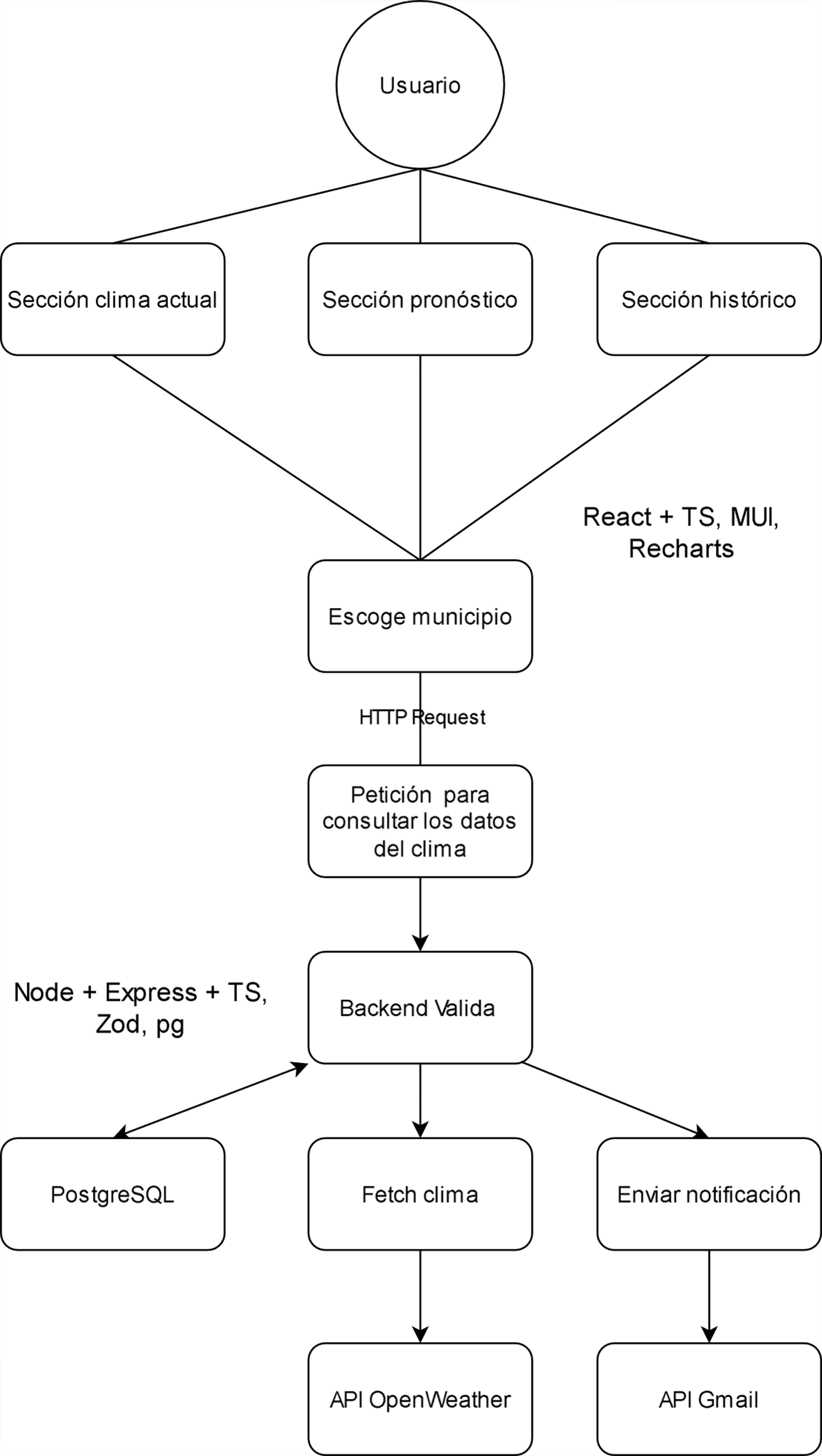
**9. Diseño Técnico**

**9.1 Arquitectura del Sistema**

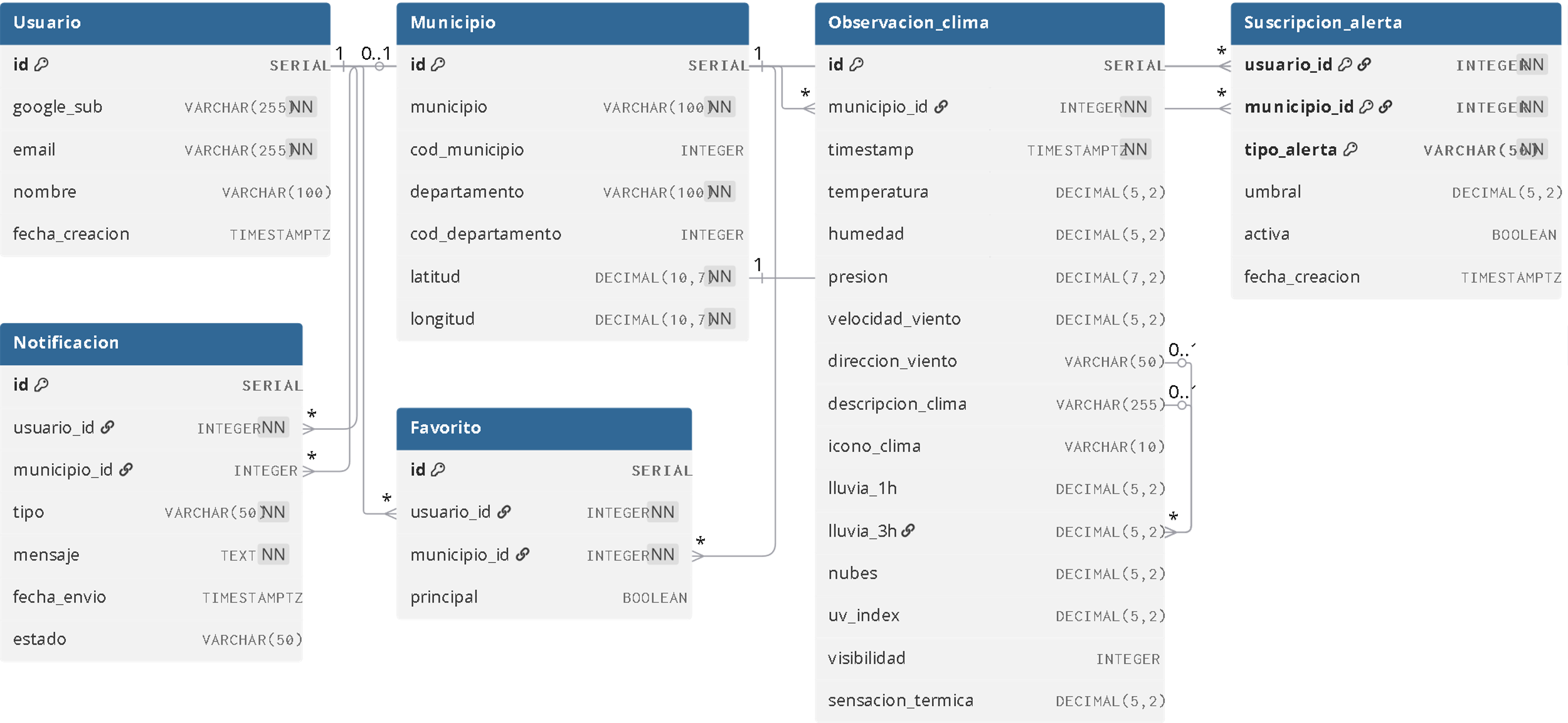
**Componentes Principales**

* **Frontend**: React + Vite + TypeScript
  + @mui/material @emotion/react @emotion/styled
  + react-router-dom
  + chart.js
  + @types/chart.js
  + driver.js
  + swapy
* **Backend**: Node.js + Express + TypeScript
  + typescript
  + @types/node
  + ts-node
  + nodemon
  + ESLint
  + Prettier
  + @typescript-eslint/eslint-plugin
  + @typescript-eslint/parser
  + eslint-config-prettier
  + eslint-plugin-prettier
  + Express
  + @types/express
  + Pg
  + @types/pg
  + dotenv
  + google-auth-library
  + jsonwebtoken
  + @types/jsonwebtoken
  + Nodemailer
* **Base de datos:** PostgreSQL
* **APIs externas:**
  + OpenWeather (datos clima actual, pronóstico).
  + Google Outh2.0

**Diagrama de Arquitectura (General)**



**9.2 Diseño de la Base de Datos**



**10. Patrón de diseño**

**Frontend: MVVM + Smart/Dumb Components**

* Model–View–ViewModel
* Smart/Dumb Components

**Backend: MVC + Services + Repositories**

* Model
* Controller
* Service
* Repository (DAO)
* View: no se utiliza

**Frontend MVVM + Smart/Dumb Components:**

El frontend será construido en React + TypeScript utilizando el patrón MVVM debido a la arquitectura basada en componentes y hooks. Además, será reforzada la separación entre componentes inteligentes y tontos (Smart/Dumb Components) para mantener la interfaz modularizada y reutilizable.

Model–View–ViewModel:

* Model: Representa los datos que vienen de las peticiones.
* ViewModel: Representa los hooks y contextos, además, contiene la lógica de negocio que no pertenece a la interfaz.
* View: Representa los componentes visuales sin lógica de negocio.

Smart/Dumb Components:

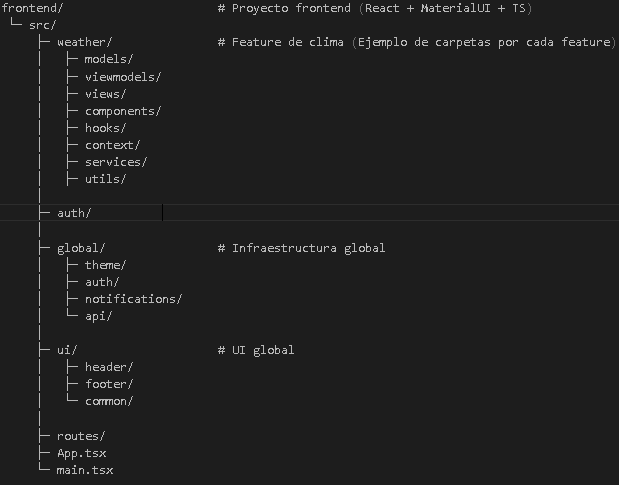
* Smart (Containers): Representa la conexión con ViewModel y realiza llamadas al api decidiendo cómo ver los datos en la vista.
* Dumb (Presentacionales): Representa el renderizado de la UI con las props.

**Backend MVC:**

El Backend será desarrollado con Node.js + Express + TypeScript como una API REST; el patrón MVC es uno de los mejores por la naturaleza del proyecto, su pequeñez y su adaptabilidad en arquitecturas de servidores HTTP. También se agrega la capa de services y Repositories para separar los puntos de entrada, lógica de negocio, persistencia. Etc.

* Model: Representa las entidades de la base de datos.
* Controller: Representa las validaciones con los middlewares recibiendo las peticiones y respondiendo con los datos.
* Service: Representa la lógica de negocio centralizando las reglas.
* Repository (DAO): Representa la comunicación con la base de datos, la persistencia de la información y en este caso el guardado en cache de los datos del clima.
* View: No se utilizará, no será necesaria.

**Estructura de carpetas:**



Texto

El contenido generado por IA puede ser incorrecto.