DAS – Documento Arquitectura del Sistema

Proyecto: Barrio Activo

Revisión*: [1.0]*
[16-10-2024]

Tabla de contenido

1. Introducción

1.1 Propósito del Documento

El propósito del Documento de Arquitectura de Software (DAS) es describir la arquitectura del sistema "Barrio Activo", proporcionando una guía detallada sobre su estructura y componentes.

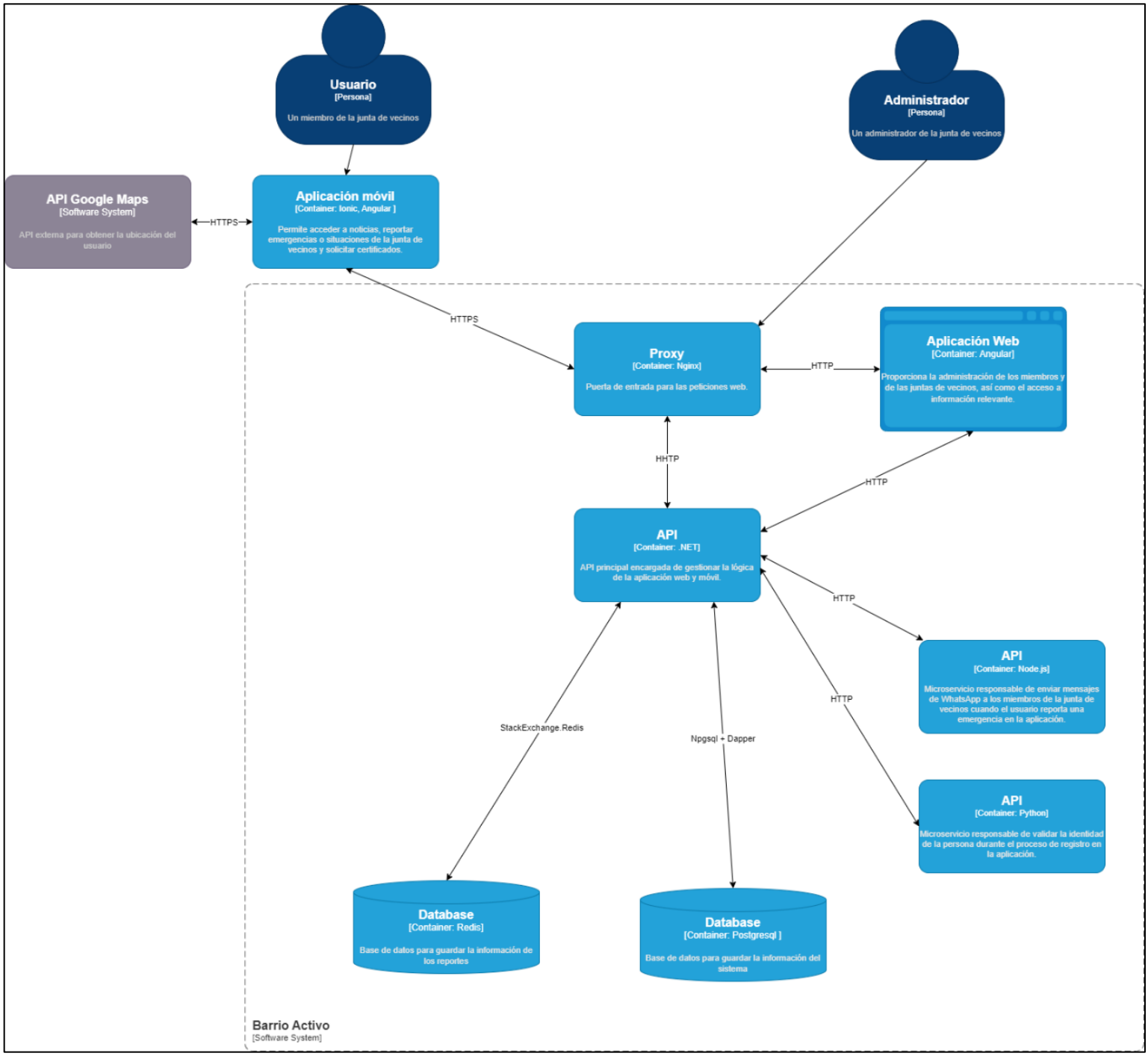
El DAS servirá como referencia tanto para el equipo de desarrollo como para los interesados en el proyecto, facilitando la comprensión del funcionamiento del sistema y proporcionando la información necesaria para futuras extensiones, mantenimientos y evaluaciones técnicas. Además, permitirá alinear las expectativas de todos los involucrados, garantizando que el desarrollo siga los objetivos técnicos y funcionales definidos previamente, asegurando así la calidad del producto final.

1.2 Alcance del Proyecto

El alcance del proyecto "Barrio Activo" incluye el desarrollo de una solución tecnológica para mejorar la gestión de las juntas de vecinos en la Región Metropolitana, en su primera etapa, para una junta de vecinos de la comuna de Independencia. El proyecto abarca la creación de una aplicación web para los administradores de juntas de vecinos y una aplicación móvil para los miembros de la junta de vecinos. Las funcionalidades principales incluyen la emisión de certificados, la gestión de emergencias y reportes, las reservas de espacios comunes, entre otros.

2. Visión General de la Arquitectura

2.1 Diagrama de Arquitectura:



3. Descripción General de la Arquitectura

El sistema presentado en el diagrama sigue una **arquitectura híbrida** que combina **microservicios** y el patrón **cliente-servidor**. A continuación, se explica cada estilo arquitectónico y cómo se implementa en el proyecto.

3.1 Arquitectura de Microservicios

La arquitectura de "Barrio Activo" está basada en microservicios, lo cual significa que el sistema se ha dividido en servicios independientes, cada uno con una responsabilidad específica y bien definida. Esta elección arquitectónica permite escalabilidad, flexibilidad y mantenibilidad. Los puntos clave de la arquitectura de microservicios en este sistema incluyen:

- **Servicios Desacoplados**:

  El sistema está compuesto por varias APIs independientes, cada una encargada de un conjunto de funcionalidades específicas. Por ejemplo:

  a. La **API Principal** desarrollada en .NET se encarga de la lógica central de la aplicación tanto para la interfaz web como móvil.

  b. La **API de Mensajes**, desarrollada en Node.js, se encarga de enviar notificaciones por WhatsApp en caso de emergencia.

  c. La **API de Validación de Identidad**, desarrollada en Python, valida la identidad de los usuarios al registrarse.

- **Bases de Datos Especializadas**:

  La API principal tiene acceso a bases de datos específicas según su funcionalidad:

  a. **Redis**: Utilizado para almacenar datos que necesitan acceso rápido, como información de reportes.

  b. **PostgreSQL**: Utilizado para el almacenamiento de datos relacionales y estructurados.

- **Balanceo de Carga y Gestión de Tráfico**:

  El sistema utiliza **NGINX** como **Proxy Manager**, lo cual permite balancear la carga entre los microservicios y distribuir eficientemente las solicitudes. NGINX actúa como el punto de entrada al sistema, mejorando la disponibilidad y la seguridad.

3.2 Patrón Cliente-Servidor

Adicionalmente, el sistema implementa el patrón **cliente-servidor** para gestionar la interacción entre los usuarios y el back-end. En este patrón se detalla lo siguiente:

- **Clientes**:

  El sistema cuenta con dos tipos de aplicaciones cliente:

  a. **Aplicación Móvil**: Desarrollada con Ionic y Angular, es utilizada por los miembros de la junta de vecinos para acceder a funcionalidades como la solicitud de certificados, generar reportes y emergencia, reservar áreas comunes, entre otros.
  b. **Aplicación Web**: Desarrollada con Angular, está destinada a los administradores de las juntas de vecinos para la gestión de miembros, junta de vecinos y obtención de información.

- **Servidor Backend**:

  a. Los clientes se conectan al back-end mediante el Proxy Manager (NGINX), que redirige las solicitudes hacia las APIs correspondientes.
  b. La API Principal se comunica con otras APIs especializadas para proporcionar una experiencia de usuario coherente, lo que sigue los principios del patrón cliente-servidor, en el que los clientes (interfaces de usuario) realizan peticiones a servidores (back-end) para interactuar con la lógica y los datos del sistema.

3.3 Vista lógica

En el sistema "Barrio Activo", los módulos principales incluyen:

I. **Aplicación Móvil (Ionic, Angular)**
   a. Esta aplicación es utilizada por los miembros de la junta de vecinos. Su principal función es permitir a los usuarios reportar emergencias, generar certificados y hacer reservas de áreas comunes.
   b. La aplicación móvil se comunica con la **API Principal** y consume sus servicios mediante HTTPS a través del proxy NGINX.

II. **Aplicación Web (Angular)**
   a. Desarrollada para los administradores de las juntas de vecinos. Proporciona funcionalidades administrativas como la gestión de usuarios, juntas y reservas.
   b. Al igual que la aplicación móvil, se comunica con la **API Principal** a través del proxy NGINX.

III. **API Principal (.NET Core)**
   a. Es el servicio central del sistema y gestiona la lógica de negocio. Provee servicios tanto para la aplicación móvil como la web.
   b. Esta API también se encarga de comunicarse con otros microservicios, como la API de mensajes y la API de validación de identidad.

IV. **API de Mensajes (Node.js)**
   a. Este microservicio es responsable de enviar mensajes de emergencia a través de WhatsApp. Recibe las solicitudes desde la API Principal cuando se genera una emergencia y se requiere notificación inmediata.

V. **API de Validación de Identidad (Python)**:
   a. Este servicio gestiona la validación de la identidad de los usuarios cuando se registran en el sistema. Realiza verificaciones y autenticación basadas en las imágenes proporcionadas.

VI. **Base de Datos Redis**:
    a. Utilizada para almacenar información temporal y de acceso rápido, como los reportes.

VII. **Base de Datos PostgreSQL**:
    a. Almacena toda la información estructurada y relacional, como datos de usuarios, juntas, certificados, y reservas.

VIII. **NGINX (Proxy Manager)**:
    a. Actúa como puerta de entrada al sistema, distribuyendo el tráfico entre las aplicaciones front-end y los microservicios backend. Implementa balanceo de carga y seguridad.

IX. **API de Google Maps**:
    a. Se utiliza para obtener la geolocalización de los usuarios en tiempo real, permitiendo una mejor gestión de las emergencias en la aplicación móvil.

3.4 Vista física

El sistema Barrio Activo está desplegado en un único servidor en la nube, y utiliza contenedores Docker para aislar y gestionar cada uno de sus componentes. Esto permite mayor flexibilidad y portabilidad, ya que cada contenedor tiene su propio entorno independiente, lo que facilita el despliegue y el mantenimiento.

- **Servidor en la Nube**:
    - Todo el sistema está alojado en un servidor en la nube, que alberga los diferentes módulos del sistema en contenedores Docker separados.
    - El servidor está configurado para ejecutar todos los servicios necesarios, como Docker, NGINX y los sistemas de bases de datos.

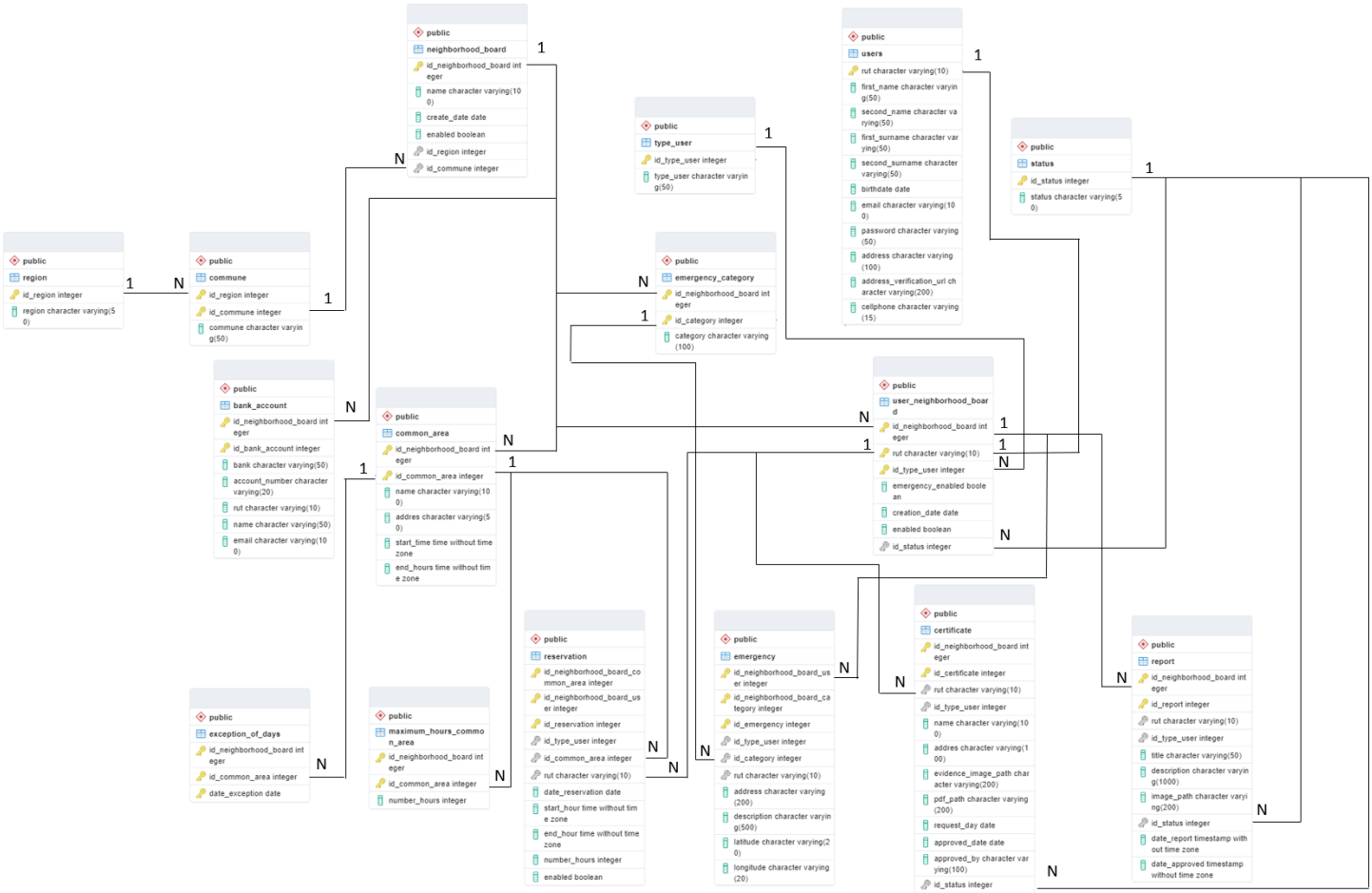
4. Diseño Detallado

4.1 Vista de Datos

La **gestión y almacenamiento de datos** en el sistema "Barrio Activo" se realiza utilizando dos tecnologías de bases de datos principales: **PostgreSQL** para el almacenamiento relacional y **Redis** para el almacenamiento temporal y de caché. Cada una de estas tecnologías se despliega en contenedores Docker separados para asegurar una correcta aislación y flexibilidad.

4.2 Modelo de Base de Datos Relacional (PostgreSQL)

La base de datos principal del sistema es **PostgreSQL**, que gestiona los datos estructurados del sistema. A continuación, el modelo de base de datos utilizado:



4.3 Almacenamiento Temporal y Caché (Redis)

Redis es utilizado como base de datos **en memoria** para gestionar datos temporales y mejorar el rendimiento del sistema. La función clave de Redis es:

Caché de Reportes:

a. Los datos de los reportes generados por los usuarios se almacenan temporalmente en Redis para facilitar un acceso rápido. Dado que los reportes son críticos para el sistema, Redis permite que estos datos sean procesados y recuperados de forma eficiente.

5. Decisiones Técnicas y Justificación

5.1 Decisiones de Diseño

- **Elección de una Arquitectura basada en Microservicios**

  Se optó por una arquitectura de microservicios debido a la **modularidad** y **escalabilidad** que ofrece. Cada módulo o servicio maneja una responsabilidad específica dentro del sistema, lo que permite el desarrollo y despliegue independiente de las funcionalidades. Esta elección facilita la evolución del sistema, permitiendo actualizaciones o mejoras en módulos específicos sin afectar a todo el sistema.

- **Uso de PostgreSQL (Base de Datos Relacional)**

  PostgreSQL fue seleccionada como la base de datos relacional debido a su **consistencia transaccional** y **soporte avanzado para consultas SQL**. La naturaleza de los datos manejados, como usuarios, reservas, juntas de vecinos, y certificados, requiere estructuras relacionales bien definidas y la capacidad de realizar consultas complejas con integridad referencial. PostgreSQL también soporta características avanzadas como índices eficientes, manejo de grandes volúmenes de datos y transacciones, lo cual es necesario para el sistema.

- **Uso de Redis (Base de Datos NoSQL en Memoria)**

  Redis fue seleccionado para gestionar datos temporales y en memoria debido a su **bajo tiempo de latencia** y **alto rendimiento**. Se decidió usar Redis en lugar de una base de datos relacional o permanente para gestionar los datos temporales, como reportes.

- **Separación de Aplicaciones Móvil y Web**:

  Se decidió desarrollar una aplicación móvil separada de la aplicación web debido a las diferentes necesidades de los usuarios finales. Los miembros de las juntas de vecinos (usuarios de la aplicación móvil) necesitan una interfaz optimizada para móviles y accesible en cualquier momento, mientras que los administradores requieren una interfaz web más robusta para gestionar el sistema. Esta separación garantiza que cada grupo de usuarios tenga una experiencia de usuario adecuada y eficiente según el dispositivo que utilicen.

- **Elección de NGINX como Proxy Manager**:

  NGINX fue elegido para gestionar la distribución de tráfico entre los microservicios debido a su capacidad para **balancear carga** y **gestionar certificados SSL**. NGINX es reconocido por su alto rendimiento y eficiencia en la gestión de solicitudes concurrentes. Actúa como puerta de entrada única para todo el tráfico del sistema, distribuyendo las peticiones hacia las APIs y garantizando que el sistema pueda manejar múltiples usuarios sin comprometer la seguridad o el rendimiento.

5.2 Tecnologías Utilizadas

- **.NET Core (API Principal)**:

  .NET Core fue seleccionado para la API principal debido a su **versatilidad**, **rendimiento** y **soporte multiplataforma**. Permite crear servicios backend robustos, escalables y de alto rendimiento. Además, tiene un soporte sólido para la integración con PostgreSQL (utilizando Npgsql) y Redis (con StackExchange.Redis), lo que facilita la gestión de los datos dentro del sistema.
- **Node.js (API de Mensajes)**:
  - Node.js fue elegido para la API de mensajería debido a su **eficiencia en operaciones de entrada/salida** y su capacidad de manejar **conexiones concurrentes**. Como esta API se encarga de enviar notificaciones en tiempo real, la naturaleza asincrónica de Node.js es ideal para gestionar eventos en segundo plano, como el envío de mensajes de WhatsApp.
- **Python (API de Validación de Identidad)**:
  - Python fue seleccionado para la API de validación de identidad debido a su **facilidad de desarrollo rápido** y **amplia gama de bibliotecas** para el manejo de autenticación y seguridad. Python es una excelente opción para servicios que requieren procesamiento intensivo de datos y validaciones complejas.
- **Ionic + Angular (Aplicación Móvil)**:
  - Ionic y Angular fueron seleccionados para el desarrollo de la aplicación móvil por su capacidad de crear **aplicaciones multiplataforma** (iOS y Android) con una única base de código. Además, Angular proporciona un framework robusto para la gestión del estado y la interacción con las APIs del sistema. La elección de Ionic permite un desarrollo ágil y accesible a través de tecnologías web.

- **Angular (Aplicación Web)**:
    - Angular fue elegido para la aplicación web debido a su capacidad de crear **interfaces de usuario dinámicas y ricas en funcionalidades**. Angular también ofrece una excelente integración con APIs RESTful, lo que facilita la comunicación eficiente con el backend. Además, su fuerte sistema de tipado y modularidad permite un desarrollo mantenible y escalable.
- **Docker (Contenedores)**:
    - Docker fue elegido para gestionar la infraestructura del sistema debido a su capacidad de **aislar servicios** y **facilitar el despliegue**. Al encapsular cada componente del sistema en un contenedor, se asegura que las dependencias estén controladas y que los servicios puedan ejecutarse de manera consistente en cualquier entorno. Además, Docker permite una fácil escalabilidad horizontal si el sistema necesita manejar más tráfico.
- **NGINX (Proxy Manager)**:
    - NGINX fue seleccionado para actuar como proxy debido a su capacidad para manejar una gran cantidad de conexiones concurrentes y por su eficiencia como balanceador de carga. NGINX también ofrece una excelente integración con certificados SSL, lo que permite asegurar la comunicación entre los clientes (móvil y web) y los servicios backend.

6. Integración de Terceros

6.1 APIs Externas

En el sistema **Barrio Activo**, se ha integrado una API externa para proporcionar funcionalidades adicionales que no requieren ser desarrolladas desde cero, lo que ahorra tiempo de desarrollo y asegura un alto nivel de eficiencia y calidad. A continuación, se describe la integración de terceros:

- **API de Google Maps**:

  La API de Google Maps se utiliza en la aplicación móvil para obtener la ubicación precisa de los usuarios al reportar emergencias. Permite visualizar la ubicación en un mapa y proporciona coordenadas geográficas que pueden ser utilizadas en la lógica de la aplicación.

  Se eligió la API de Google Maps por su precisión, facilidad de uso y porque es una de las soluciones más confiables para gestionar geolocalización. Google Maps también ofrece una amplia documentación y un ecosistema robusto que facilita la integración con aplicaciones móviles y web.

7. Rendimiento y Capacidad

El sistema "Barrio Activo" está optimizado para ofrecer un alto rendimiento y manejar grandes volúmenes de tráfico mediante varias estrategias clave. Utiliza Redis para el almacenamiento temporal y caché, permitiendo acceso rápido a datos críticos como reportes de emergencia, mientras que PostgreSQL gestiona eficientemente los datos relacionales. NGINX actúa como proxy inverso y balanceador de carga, distribuyendo solicitudes de manera eficiente entre los microservicios.

8. Disponibilidad y Tolerancia a Fallos

8.1 Estrategias de Alta Disponibilidad

El sistema "Barrio Activo" está diseñado para asegurar **alta disponibilidad** y minimizar el tiempo de inactividad, garantizando que el servicio esté disponible para los usuarios en todo momento. Esto debido a que es importante que los usuarios de la aplicación puedan tener acceso a realizar reportes de emergencia y obtener información en cualquier momento del día, por lo que influye la disponibilidad del servicio. Debido a esto, se diseñó la arquitectura mencionada anteriormente para poder cumplir con la alta disponibilidad del servicio.

8.2 Mecanismos de Recuperación ante Fallos

Para garantizar que el sistema pueda recuperarse de desastres o fallos, se han implementado planes de contingencia específicos:

- **Backups Automáticos y Programados**:
  Se realizan **copias de seguridad** automáticas y periódicas de la base de datos PostgreSQL y los datos clave almacenados en Redis. Estos backups se almacenan en ubicaciones seguras, tanto localmente como en la nube, permitiendo la restauración rápida de datos en caso de un fallo o pérdida de información.
- **Despliegue en Contenedores con Orquestación**:
  En caso de que un contenedor falle, la orquestación de contenedores con **Docker** permite que un nuevo contenedor sea lanzado automáticamente, garantizando la continuidad del servicio. Esto proporciona un mecanismo de auto recuperación que minimiza la interrupción del sistema.