**Servidor**

La propuesta de servidor está enfocada a ofrecer servicios a los clientes móviles y web, para ello se encargará de:

**Procesamiento de Levantamientos**: El servidor será responsable de manejar las solicitudes entrantes de los clientes. Esto incluye operaciones como cargar, leer, y eliminar Levantamientos. La carga de estos levantamientos será realizada desde el cliente móvil que subirá al servidor el levantamiento una vez finalizado, mientras que la lectura y las eliminaciones de levantamientos será realizada desde el cliente web, ambas dos realizaran estas solicitudes vía HTTPS.

**Gestión de Configuraciones**: Además de las operaciones de Levantamientos, el servidor gestionará las configuraciones del sistema. Esto implica mantener un sistema de versiones para las configuraciones existentes. Los usuarios podrán acceder a versiones anteriores si es necesario. Además, el servidor permitirá replicar configuraciones ya existentes para facilitar la creación de nuevas configuraciones, para lograr esto fue desarrollado un algoritmo de replicación de configuraciones el cual creará una copia exacta de una configuración seleccionada por el usuario. Además, se permite la carga de un json de configuración completo, ajeno a los mecanismos de inserción del servidor, si el json presenta la estructura adecuada validada en el servidor será insertado sin problemas como una nueva configuración del sistema.

**Cálculos Complejos:** Realizará operaciones de procesamiento de datos complejas destinadas a hallar el estado de los levantamientos y sistemas componentes del mismo a través de las directrices configuradas para ese proceso, de esta forma tanto cliente móvil como cliente web recibirán los datos procesados listos para ser mostrados.

**Arquitectura Propuesta**

Para la realización de estas tareas se propone la siguiente arquitectura:

Componentes Clave

**Arquitectura REST** para manejar grandes volúmenes de solicitudes sin problemas. La API proporcionará *endpoints* específicos para las operaciones de Levantamientos y para la gestión de configuraciones.

**Base de Datos relacional PostgreSQL** para almacenar los datos de forma consistente garantizando su integridad referencial. Debido a la complejidad de las relaciones entre las entidades que intervienen en la problemática y las frecuentes inserciones de datos se optó por su elección, eso debido a que **PostgreSQL** ofrece:

* Soporte para gran parte del estándar SQL y se puede extender con nuevos tipos de datos, funciones, operadores y lenguajes de programación.
* Transacciones con propiedades ACID, vistas actualizables, disparadores, integridad referencial, control de concurrencia multiversión y funciones de ventana.
* Opciones de indexación sofisticadas, como índices B-tree, GIN, GiST, BRIN y de expresión.
* Permite realizar consultas complejas, como consultas recursivas, subconsultas, uniones y agregaciones.

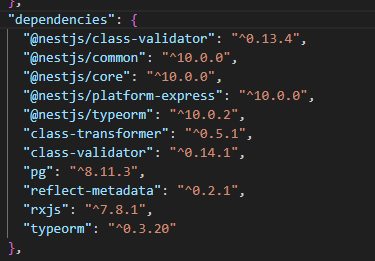
Además de todo lo anterior, incluye soporte para datos geoespaciales, temporales, JSON, XML y documentos, por lo que es la idónea para extender las funcionalidades de la aplicación a un sistema SIG en un futuro.

**TypeScript** como lenguaje principal por su robustez, facilidad de uso y perfecta integración con demás tecnologías como **Nest.js** y **TypeORM**

**Node.js** como entorno de ejecución por su excelente manejo de la asincronía y potente rendimiento.

**Nest.js** como frameWork backend construido en TypeScript por su excelente estructura de proyecto, documentación y arquitectura modular que facilita la organización del código en módulos reutilizables; además Nest.js es ideal para construir microservicios, punto este, a desarrollar en futuras versiones de la propuesta.

**TypeORM** como ORM (Mapeador de Objetos Relacionales) por permitir escribir aplicaciones de alta calidad, acopladas de forma flexible, escalables y mantenibles de la manera más productiva, excelente integración con TypeScript ya que TypeORM está diseñado específicamente para ser utilizada con TypeScript. Además, añade soporte para base de datos no relacionales como MongoDB lo que haría factible en un futuro realizar una migración de base de datos de forma más limpia y segura.



Dependencias utilizadas desde archivo package.json

**Endpoints API RESTful**

Para atender las solicitudes de los clientes, el servidor dispone de varios **endpoints**, cada uno diseñado para procesar una solicitud específica. A continuación, algunos de ellos agrupados por controladores:

|  |  |
| --- | --- |
| **/sistemas-config** | |
| **Descripción:** Desde esta ruta es posible acceder a varios caminos para procesar todas las solicitudes referentes a los sistemas de configuración. | |
| **Método de Ruta** | **Descripción** |
| /createSistemaConfig (Método POST) | Este método de ruta es el encargado de atender las solicitudes de inserción de nuevos sistemas de configuración a la base de datos. Recibe un **json** en el cuerpo de la solicitud el cuál es una representación del sistema configuración a insertar. |
| /getAllSistemasConfig (Método GET) | Este método de ruta es el encargado de atender las solicitudes de obtención de todos los sistemas de configuración registrados en la base de datos. |
| /getAllBelongConfig/:versión (Método GET) | Este método de ruta es el encargado de atender las solicitudes de obtención de todos los sistemas de configuración que pertenezcan a una configuración en específico. Recibe un **parámetro de ruta** que representa la versión de la configuración. |
| /deleteSistemaConfig/:id(Método DELETE) | Este método de ruta es el encargado de atender las solicitudes de eliminación de un sistema de configuración en específico. Recibe un **parámetro de ruta** que representa el id del sistema de configuración a eliminar. |

Tabla # : Controlador de Ruta sistemas-config

|  |  |
| --- | --- |
| **/configs** | |
| **Descripción:** Desde esta ruta es posible acceder a varios caminos para procesar todas las solicitudes referentes a las configuraciones. | |
| **Método de Ruta** | **Descripción** |
| /createNewConfig (Método POST) | Este método de ruta es el encargado de atender las solicitudes de inserción de nuevas configuraciones a la base de datos. Recibe un **json** en el cuerpo de la solicitud el cuál es una representación de la configuración a insertar. |
| /getAllConfigs(Método GET) | Este método de ruta es el encargado de atender las solicitudes de obtención de todas las configuraciones registradas en la base de datos. |
| /getLastConfig(Método GET) | Este método de ruta es el encargado de atender las solicitudes de obtención de la configuración insertada de forma más reciente en la base de datos. Esta es una operación solicitada de forma recurrente, es por ello que para la obtención de esta herramienta se optó por almacenar una referencia en la memoria caché del servidor. |
| /createConfigByOtherConfig/:versionOtherConfig(Método POST) | Este método de ruta es el encargado de atender las solicitudes de **replicación de una configuración** existente; proceso de replicación de una configuración implica la creación de una nueva configuración basada en la elegida por el usuario. Recibe un **parámetro de ruta** que representa la versión de la configuración a replicar |

Tabla # : Controlador de Ruta configs

|  |  |
| --- | --- |
| **/herramienta-analisis-criticidad** | |
| **Descripción:** Desde esta ruta es posible acceder a varios caminos para procesar todas las solicitudes referentes a las herramientas de análisis de criticidad. | |
| **Método de Ruta** | **Descripción** |
| /createHerramientaAnalisisCriticidad (Método POST) | Este método de ruta es el encargado de atender las solicitudes de inserción de nuevas herramientas de análisis de criticidad a la base de datos. Recibe un **json** en el cuerpo de la solicitud, el cuál es una representación de la herramienta análisis de criticidad a insertar. |
| /getAllHerramientasAnalisisCritcidad(Método GET) | Este método de ruta es el encargado de atender las solicitudes de obtención de todas las herramientas de análisis de criticidad registradas en la base de datos. |
| /deleteHerramientaAnalisisCritcidad/:id(Método DELETE) | Este método de ruta es el encargado de atender las solicitudes de eliminación de una herramienta de análisis de criticidad en específico. Recibe un **parámetro de ruta** que representa el id de la herramienta análisis de criticidad a eliminar. |

Tabla # : Controlador de Ruta herramienta-analisis-criticidad

|  |  |
| --- | --- |
| **/levantamiento** | |
| **Descripción:** Desde esta ruta es posible acceder a varios caminos para procesar todas las solicitudes referentes a los levantamientos de las edificaciones. | |
| **Método de Ruta** | **Descripción** |
| /createLevantamiento(Método POST) | Este método de ruta es el encargado de atender las solicitudes de inserción de nuevos levantamientos a la base de datos. Recibe un **json** en el cuerpo de la solicitud, el cuál es una representación del levantamiento a insertar. Esta es una operación realizada de forma recurrente por el cliente móvil que es el encargado de la creación de un levantamiento, es por ello que, en este caso, la carga de una configuración en la base de datos no se hace de forma paulatina si no de forma gradual una vez el cliente móvil haya reunido todos los datos del levantamiento. |
| /getAllLevantamientos(Método GET) | Este método de ruta es el encargado de atender las solicitudes de obtención de todos los levantamientos registrados en la base de datos. |
| /deleteLevantamiento/:id(Método DELETE) | Este método de ruta es el encargado de atender las solicitudes de eliminación de un levantamiento en específico. Recibe un **parámetro de ruta** que representa el id del levantamiento a eliminar. |

Tabla # : Controlador de Ruta levantamiento

Estas rutas fueron seleccionadas porque representan acciones clave en el flujo del programa a la hora de establecer la comunicación entre el servidor y los clientes web y móvil.