Tecnologias utilizadas

Las tecnologías utilizadas son: Spring para la estructura y configuración del proyecto, y para las anotaciones utilizadas; Spring Boot para la creación, configuración, y manejo de dependencias y estructura del proyecto; Spring data para el acceso a la base de datos y mapeo de objetos; ElasticSearch cómo motor de base de datos; Logstash para el parseo, conversión y carga de datos desde el formato csv a la base de datos de elastic; y JSON schemas para la validación de los parámetros recibidos por la API.

Spring boot

Spring boot es parte de Spring Framework y ayuda a generar una aplicación con base Spring, resolviendo cuestiones de configuración e infraestructura básicas y generales, dejando por configurar cuestiones puntuales cómo los datos de coneccion a la base de datos, y permitiendo modificar la configuración para cuestiones más puntuales (al menos que exista un requerimiento muy preciso que requiera la creación de una aplicación de Spring desde cero).

Utilizamos esta herramienta para la creación del proyecto, dándonos una estructura básica de un proyecto Spring y debiendo sólo configurar los datos de acceso a la base de datos para poder correr la aplicación y comenzar a desarrollar.

Spring data (Spring Data Elasticsearch)

Spring Data es un proyecto de SpringSource cuyo propósito es unificar y facilitar el acceso a distintos tipos de tecnologías de persistencia, tanto a bases de datos relacionales como a las del tipo NoSQL.
Con cada tipo de tecnología de persistencia los DAOs (Data Access Objects) ofrecen las funcionalidades típicas de un CRUD (Create-Read-Update-Delete) para objetos de dominio propios, métodos de búsqueda, ordenación y paginación. Spring Data proporciona interfaces genéricas para estos aspectos (CrudRepository, PagingAndSortingRepository) e implementaciones específicas para cada tipo de tecnología de persistencia.

Spring Data para Elasticsearch es parte del proyecto general Spring Data que tiene como objetivo proporcionar un modelo de programación familiar y consistente basado en Spring para nuevos bancos de datos, al mismo tiempo que conserva las características y capacidades específicas del almacenamiento.
El proyecto Spring Data Elasticsearch proporciona integración con el motor de búsqueda Elasticsearch. Las áreas funcionales clave de Spring Data Elasticsearch son un modelo centrado en POJO para interactuar con documentos de Elasticsearch y escribir fácilmente una capa de acceso a datos de estilo Repository.

Fue utilizada para la persistencia de datos y realizar las consultas de los objetos almacenados, haciendo transparente el acceso a la base de datos y mapeo de objetos,

además de que, a traves del patrón repository, permite tener el resto de codigo de la aplicación limpio con apenas detalles de la base de datos en el modelo.

ElasticSearch

Elasticsearch es un motor de analítica, análisis distribuido, y servidor de búsqueda basado en Lucene, open source. Provee un motor de búsqueda de texto completo, distribuido y con capacidad de multitenencia con una interfaz web RESTful y con documentos JSON.
En sí mismo podríamos definir a Elasticsearch como una base de datos NoSQL orientada a documentos JSON, los cuales pueden ser consultados, creados, actualizados o borrados mediante una sencilla API Rest.

Los datos sin procesar fluyen hacia Elasticsearch desde una variedad de fuentes, incluidos logs, metricas de sistema y aplicaciones web. La ingesta de datos es el proceso mediante el cual estos datos son parseados, normalizados y enriquecidos antes de su indexación en Elasticsearch. Una vez indexados en Elasticsearch, los usuarios pueden ejecutar consultas complejas sobre sus datos y usar agregaciones para recuperar resúmenes complejos de sus datos.
Elasticsearch utiliza una estructura de datos llamada índice invertido, que está diseñado para permitir búsquedas de texto completo muy rápidas. Un índice invertido hace una lista de cada palabra única que aparece en cualquier documento e identifica todos los documentos en que ocurre cada palabra.
Durante el proceso de indexación, Elasticsearch almacena documentos y construye un índice invertido para poder buscar datos en el documento casi en tiempo real. La indexación comienza con la API de índice, a través de la cual puedes agregar o actualizar un documento JSON en un índice específico.
Como Elasticsearch está desarrollado sobre Lucene, es excelente en la búsqueda de texto completo. Elasticsearch también es una plataforma de búsqueda en casi tiempo real, lo que implica que la latencia entre el momento en que se indexa un documento hasta el momento en que se puede buscar en él es muy breve.

Fue utilizada cómo base de datos, siendo accedida a través de Spring data mediante el módulo correspondiente, y se utilizaron consultas de agregación para el cálculo de los factores más comunes en los accidentes, y consultas de búsqueda (search query) con las opciones para realizar consultas geoespaciales utilizando los campos de tipo geo_point para buscar los accidentes dentro de un polígono o un círculo.

Logstash

Logstash, uno de los productos principales del Elastic Stack, se usa para agregar y procesar datos y enviarlos a Elasticsearch. Logstash es un pipeline de procesamiento de datos open source y del lado del servidor que permite ingestar datos de múltiples fuentes simultáneamente y enriquecerlos y transformarlos antes de que se indexen en Elasticsearch.

Fue utilizado para cargar la base de datos de elasticsearch a partir de un archivo csv, convirtiendo cada línea en un nuevo documento, transformando cada columna en el tipo correspondiente y pasándolo a elasticsearch para qué lo indexe.

JSON Schemas

JSON Schema es una especificación para el formato basado en JSON para definir la estructura de datos JSON.

- Describe su formato de datos existente.
- Documentación clara, legible por humanos y máquinas.
- Validación estructural completa, útil para pruebas automatizadas.
- Validación estructural completa, validando los datos enviados por el cliente.

Fue usado en los métodos del controlador de buscar los accidentes dentro de un polígono o un círculo para validar los parámetros(más de 2 puntos geoespaciales o un punto geoespacial y una distancia respectivamente) mediante la librería 'com.github.everit-org:json-schema', y utilizando los esquemas definidos en resources/schemas.

Nuestra solución

La solución presentada consta de 4 capas: controlador, servicio, repositorio y modelo. El controlador pública los endpoints, recibe las solicitudes hechas por REST a la API (donde procesa y valida los parámetros, y llama al servicio) y devuelve la respuesta al cliente. El servicio se encarga de la lógica de negocio, interactuando con el repositorio, tomando los datos y formateandolos de manera que tengan la forma que necesita en la consulta que se debe resolver. El repositorio es básicamente una abstracción del acceso a la base de datos. El modelo es la representación de la información de los accidentes cargados en el sistema.

El repositorio utiliza 2 tipos de consulta:

- El primero es extendiendo ElasticsearchRepository, generando las consultas mediante la opción @Query (con el mismo formato que se envían a la API de elasticsearch), para las consultas de tipo geoespaciales.

- El segundo es haciendo que el repositorio extienda una interfaz personalizada, la cual se implementa por separado y donde se construyen las consultas a ejecutar mediante el constructor de consultas SearchSourceBuilder, debido a que por la complejidad y necesidad de utilizar consultas de agregación no se puede realizar una consulta tradicional @Query (ya que en estas solo se permiten consultas de búsqueda, no de agregación).

En src/main/resources/schemas, definimos los esquemas de los JSON que son utilizados para validar que los parámetros recibidos sean válidos.

Capas

Controller (@RestController)

Funcionan como una puerta de enlace entre la entrada y la lógica del dominio, decide qué hacer con la entrada y cómo generar la respuesta. Declara los puntos de entrada de la API a través de @GetMapping, valida que los parámetros tengan un formato válido y esté presente toda la información necesaria para llevar a cabo la solicitud, y en caso de que lo tengan, realiza llamadas al servicio para resolver la petición.

Service (@Service)

El servicio debe proporcionar una API a la lógica de negocio. Los servicios deberían ser los únicos con acceso a los repositorios, porque la capa empresarial es una abstracción de su capa de acceso a datos. Toma los parámetros pasados desde el controller, formatea el parámetro cómo se lo necesite en el repository, realiza las llamadas a este para obtener los datos y genera una respuesta con los datos recibidos de cada llamada al repository.

Model (@Document)

Representa el objeto de dominio que queremos persistir, es una representación de los datos que maneja el sistema (accidentes).

Repository (@Repository)

El repositorio es una puerta de enlace entre el dominio/capa empresarial y una capa de mapeo de datos(capa que accede a la base de datos y realiza las operaciones). Básicamente, el repositorio es una abstracción del acceso a la base de datos. Un repositorio es un mecanismo para encapsular el comportamiento de almacenamiento, recuperación y búsqueda que emula una colección de objetos.