Introducción

Han pasado varios meses desde que hice la última actualización de la serie “How to Build a Conversational Assistant with Java Spring Boot, Langchain4j and OpenAI”. Y es que he estado inmerso en un proyecto de desarrollo e implementación de una nueva plataforma digital para Bancaseguros en una compañía de seguros de Panamá; un importante cliente nuestro con gente muy buena.

Sin embargo, ahora que ya pusimos en producción la nueva plataforma y ésta se encuentra estable, he vuelto para retomar esta serie.

A modo de un breve repaso, en las partes anteriores de esta serie logramos construir un agente recomendador de seguros, bastante simple, conectado a una base de datos vectorial de embeddings implementada en PostgreSQL.

Ahora, tal como lo prometí, seguiremos profundizando nuestro agente Recomendador, agregándole nuevas capacidades como búsqueda semántica, filtros basados en metadata e invocación de herramientas para cortizar seguros (cálculo de prima).

Esta nueva entrega quizá tenga un nivel de complejidad técnica mayor a las entregas anteriores sin embargo, al igual que en éstas, iremos paso a paso y les mostraré cada fragmento de código fuente para que puedan replicar este agente Recomendador en sus entornos locales.

Actualización de librerías

Como pasó bastante tiempo desde la publicación de la parte 3 de esta serie, comenzaremos presentando una nueva versión del archivo pom.xml, con una actualización de las versiones de las librerías:

Archivo pom.xml:

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>3.5.6</version>  <relativePath/> <!-- lookup parent from repository -->  </parent>  <groupId>com.superchat</groupId>  <artifactId>chatia</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>chatia</name>  <description>Mi primer Chat IA con LangChain4j y OpenIA</description>   <properties>  <java.version>24</java.version>  <langchain4j.version>1.7.1</langchain4j.version>  <langchain4j-pgvector.version>1.3.0-beta9</langchain4j-pgvector.version>  <langchain4j-spring-boot-starter.version>1.7.1-beta14</langchain4j-spring-boot-starter.version>  </properties>   <dependencies>   <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>   <dependency>  <groupId>dev.langchain4j</groupId>  <artifactId>langchain4j-spring-boot-starter</artifactId>  <version>${langchain4j-spring-boot-starter.version}</version>  </dependency>   <dependency>  <groupId>dev.langchain4j</groupId>  <artifactId>langchain4j</artifactId>  <version>${langchain4j.version}</version>  </dependency>   <dependency>  <groupId>dev.langchain4j</groupId>  <artifactId>langchain4j-open-ai</artifactId>  <version>${langchain4j.version}</version>  </dependency>   <dependency>  <groupId>dev.langchain4j</groupId>  <artifactId>langchain4j-pgvector</artifactId>  <version>${langchain4j-pgvector.version}</version>  </dependency>   <dependency>  <groupId>org.postgresql</groupId>  <artifactId>postgresql</artifactId>  <scope>runtime</scope>  </dependency>   <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <optional>true</optional>  </dependency>   <!-- ADDED: Dependency for Testing -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>   <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

Repositorio de productos

En aras de la simplicidad simularemos un repositorio de productos, el cual cargaremos en memoria en un List:

|  |
| --- |
| package com.superchat.repositories;  import com.superchat.model.Product;  import java.util.ArrayList; import java.util.List;  public final class ProductRepository {   private ProductRepository(){}   public static List<Product> findAllProducts() {  List<Product> items = new ArrayList<>();   items.add(new Product(  "PROD\_01",  "Individual Life Insurance",  "Insurance designed to provide financial protection to your loved ones in case of death.",  """  - Natural death: Provides a benefit for death due to natural causes.  - Accidental death: Covers death by accidents, offering an additional benefit.  """,  """  Adults aged 25-65, of any gender, who are primary income earners or have financial dependents  (such as spouses, children, or elderly parents), seeking to ensure the financial security and  well-being of their families in the event of unforeseen circumstances.  """,  25, 65, "life"  ));   items.add(new Product(  "PROD\_02",  "Personal Accident Insurance",  "Insurance that offers protection in case of accidents resulting in injuries or death.",  """  - Accidental death: Provides a benefit for death due to accidents.  - Permanent disability: Covers permanent disability resulting from an accident, offering financial  benefits.  """,  """  Adults aged 25-65, of any gender, who are exposed to risks of accidents in their daily activities,  such as workers, students, athletes, or people who frequently travel, and who wish to protect themselves  and their families from the financial consequences of accidental injuries or death.  """,  25, 65, "accident"  ));   items.add(new Product(  "PROD\_03",  "Health Insurance",  "Insurance that covers medical expenses for illnesses or accidents.",  """  - Hospitalization: Covers costs of hospitalization due to illness or accident.  - Surgical procedures: Covers expenses for surgeries required due to health issues.  - Medical consultations: Provides coverage for medical consultations with specialists.  """,  """  Individuals and families of all ages (18-120) who are concerned about potential medical expenses due to  illness or accidents, including those with pre-existing health conditions, self-employed professionals,  parents seeking coverage for their children, elderly individuals, and anyone who wants to ensure access  to quality healthcare and financial protection against unexpected medical costs.  """,  18, 120, "health"  ));   items.add(new Product(  "PROD\_04",  "Young Adult Travel Insurance",  """  A comprehensive travel insurance plan designed for young adults who seek adventure, exploration, and  peace of mind while traveling. It offers essential protection against unexpected events that may  occur during domestic or international trips, allowing you to focus on enjoying your journey without  worries.  """,  """  - Medical emergencies abroad: Covers medical expenses resulting from illness or accidents during your trip.  - Trip cancellation or interruption: Provides reimbursement for non-refundable expenses if your trip is  canceled or cut short due to covered reasons.  - Lost or delayed baggage: Compensates for lost, stolen, or significantly delayed luggage.  - Travel assistance services: Offers 24/7 support for emergencies, including medical evacuation, legal  assistance, and travel advice.  """,  """  Young single adults aged 18–35, of any gender, who travel for leisure, study, or work and seek reliable  protection against travel-related risks. Ideal for frequent travelers, digital nomads, students studying  abroad, or professionals on business trips who value safety, flexibility, and peace of mind while  exploring the world..  """,  18, 35, "travel"  ));   items.add(new Product(  "PROD\_05",  "Pets Insurance",  "Insurance that covers medical expenses for illnesses or accidents of your loved pet.",  """  - Hospitalization: Covers costs of hospitalization due to illness or accident.  - Surgical procedures: Covers expenses for surgeries required due to health issues.  - Medical consultations: Provides coverage for medical consultations with specialists.  """,  """  Oriented to people of all ages (18-120), owners of pets such as dogs and cats, who want to provide them  with protection against diseases.  """,  18, 120, "pet"  ));   items.add(new Product(  "PROD\_06",  "Home Insurance",  "Insurance that protects your home’s structure and contents against covered events and includes personal liability coverage.",  """  - Fire and smoke: Covers damage to the dwelling and contents caused by fire or smoke.  - Theft and vandalism: Covers stolen belongings and damage from forced entry or malicious acts.  - Water damage (sudden/accidental): Covers damage from burst pipes or appliance leaks (non-gradual).  - Natural events: Windstorm and hail; earthquake/flood available via optional endorsements.  - Glass breakage and fixtures: Covers windows, sanitary ware, and fixed installations.  - Temporary accommodation (loss of use): Pays for lodging if the home becomes uninhabitable due to a covered loss.  - Personal liability: Covers injuries to third parties or damage to their property caused by the insured household.  """,  """  Adults who own or rent a house or apartment and want financial protection for their dwelling, belongings, and liability.  Ideal for first-time homeowners, families, and landlords seeking comprehensive home coverage.  """,  21, 75, "home"  ));   items.add(new Product(  "PROD\_07",  "Car Insurance",  "Insurance that protects your car and your liability arising from its use, covering damage, theft, third-party claims, and roadside emergencies.",  """  - Third-party liability (bodily injury/property damage): Covers injuries to others and damage to their property caused by your car.  - Collision: Pays for repairs to your car after a crash, regardless of fault (subject to deductible).  - Comprehensive: Covers non-collision losses (theft, fire, vandalism, falling objects, weather events).  - Medical payments / personal injury protection: Covers medical expenses for you and your passengers after an accident.  - Uninsured/underinsured motorist: Protects you if the at-fault driver has insufficient or no insurance.  - Roadside assistance & towing: Help for breakdowns, flat tires, dead batteries, and emergency towing.  - Glass coverage: Repairs or replaces damaged windshields and windows.  - Rental car / mobility allowance: Provides a temporary vehicle while yours is being repaired after a covered loss.  - Optional accessories & custom parts: Extends coverage to added equipment (sound systems, racks, custom wheels).  """,  """  Licensed drivers who own or lease a car and want financial protection for their vehicle and liability.  Ideal for commuters, families, and everyday drivers; optional endorsements available for ride-share or delivery use.  """,  18, 75, "auto"  ));   return items;  } } |

Son 7 productos, de distinta categoría. De manera similar a las entregas anteriores, cabe recalcar que cada producto tiene un **ID**, un **nombre**, una **descripción**, una **descripción de las coberturas de riesgo** y una propiedad que indica la **audiencia objetivo** (para nuestro propósito, éste es una de las propiedades más importantes del producto ya que nuestro Agente Recomendador intenta recomendar productos principalmente en base al perfil del cliente).

Además, y a diferencia de lo desarrollado en las partes anteriores de esta serie, he incorporado tres propiedades adicionales que nos permitirán realizar filtros específicos: **edad mínima**, **edad máxima** y **categoría**.

La lista anterior utiliza una *registro* (similar a un POJO) llamado **Product**:

|  |
| --- |
| package com.superchat.model;  public record Product(  String id,  String name,  String description,  String coveragesText,  String audienceText,  int ageMin,  int ageMax,  String category ) {} |

Ingesta de productos en la BD vectorial

La ingesta de productos en la BD PostgreSQL (con su extensión pgVector) queda a cargo de la siguiente clase utilitaria:

|  |
| --- |
| package com.superchat.ingestion;  import com.superchat.model.Product; import dev.langchain4j.data.document.Metadata; import dev.langchain4j.data.segment.TextSegment; import dev.langchain4j.data.embedding.Embedding; import dev.langchain4j.model.embedding.EmbeddingModel; import dev.langchain4j.store.embedding.EmbeddingStore; import lombok.extern.slf4j.Slf4j;  import java.util.List;  import static com.superchat.ingestion.ProductFields.\*;  @Slf4j public final class ProductIngestion {   private ProductIngestion(){}   public static void ingestAll(List<Product> products,  EmbeddingStore<TextSegment> store,  EmbeddingModel embeddingModel) {  for (Product p : products) {  *log*.info("Ingesting product: {} - {}", p.id(), p.name());  ProductIngestion.*ingestProduct*(p, store, embeddingModel);  }  *log*.info("Ingestion completed for {} products.", products.size());  }   public static void ingestProduct(Product p,  EmbeddingStore<TextSegment> store,  EmbeddingModel embeddingModel) {   // ===== Segmento AUDIENCE =====  Metadata audMd = new Metadata();  audMd.put(*META\_PRODUCT\_ID*, p.id());  audMd.put(*META\_NAME*, p.name());  audMd.put(*META\_AGE\_MIN*, p.ageMin());  audMd.put(*META\_AGE\_MAX*, p.ageMax());  audMd.put(*META\_CATEGORY*, p.category());  audMd.put(*META\_SEGMENT\_TYPE*, *SEG\_AUDIENCE*);   String audienceText = "Target Audience: " + p.audienceText().trim();  TextSegment audienceSeg = TextSegment.*from*(audienceText, audMd);   Embedding audienceEmb = embeddingModel.embed(audienceSeg).content();  store.add(audienceEmb, audienceSeg);   // ===== Segmento DETAILS =====  Metadata detMd = new Metadata();  detMd.put(*META\_PRODUCT\_ID*, p.id());  detMd.put(*META\_NAME*, p.name());  detMd.put(*META\_AGE\_MIN*, p.ageMin());  detMd.put(*META\_AGE\_MAX*, p.ageMax());  detMd.put(*META\_CATEGORY*, p.category());  detMd.put(*META\_SEGMENT\_TYPE*, *SEG\_DETAILS*);   String detailsText = """  Product ID: %s  Product Name: %s  Product Description: %s  Coverages:  %s  Target Audience: %s  """.formatted(  p.id(), p.name(), p.description().trim(), p.coveragesText().trim(), p.audienceText().trim());   TextSegment detailsSeg = TextSegment.*from*(detailsText, detMd);   Embedding detailsEmb = embeddingModel.embed(detailsSeg).content();  store.add(detailsEmb, detailsSeg);  } } |

El método **ingestAll** es el que realiza la carga de productos en la tabla product\_embeddings utilizando la clase de Langchain4j llamada EmbeddingStore.

Aquí me gustaría explicar con mayor profundidad este método ya que tiene varias cosas interesantes.

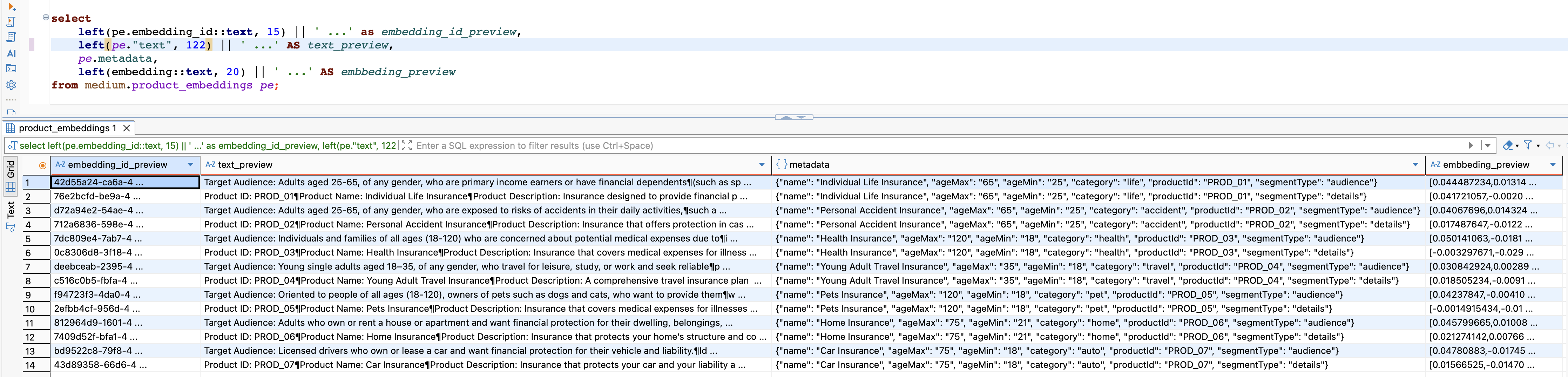
En primer lugar, quiero indicar que en la presente entrega usaremos una estrategia un poco diferente a la e la Parte 3 de esta serie:

Cargaremos en la tabla product\_embeddings dos embedings distintos:

* Un embedding para el texto que describe la audiencia objetivo. Esto nos permitirá hacer búsquedas semánticas más precisa en base a la descripción del perfil del cliente, lo cual veremos con más detalle más adelante.
* Un embedding para la descripción completa del prododucto (ID, nombre, descripción, coberturas y también su audiencia objetivo). Esto nos permitirá hacer búsquedas semánticas en base al texto completo del producto.
* Por cada uno los embedings cargaremos metadata, utilizando el objeto de Langchaing4j llamado Metadata.

A continuación un ejemplo (resumido) del contenido de la tabla después de la ingestión de los productos:

|  |
| --- |
| **select**  **left**(pe.embedding\_id::**text**, 15) || ' ...' **as** *embedding\_id\_preview*,  **left**(*pe*."text", 122) || ' ...' **AS** *text\_preview*,  *pe*.metadata,  **left**(embedding::**text**, 20) || ' ...' **AS** *embbeding\_preview*  **from** medium.product\_embeddings *pe*; |



Como veremos en unos párrafos más adelante, la metadata también nos será útil para aplicar filtros específicos a nuestra búsqueda, como por ejemplo en función a la edad del cliente. Esto lo haremos así ya que los modelos LLM no son muy buenos con los rangos de números, por lo cual haremos este filtro nosotros mismos manualmente utilizando las propiedades ageMax y ageMin que son parte de la Metadata cargada. Esto nos dará resultados mucho más robusto y precisos.

TODO: explicar que en la parte 3, siembre se envía la BD completa de productos al LLM en cada iteración, lo cual es poco eficiente en cuanto a tiempo de respuesta y costos, sobre todo si tenemos bases de datos grandes de productos. Por eso es mejor aplicar búsquedas semánticas personalizadas y filtros específicos usando la metadata, para luego enviar un conjunto reducido de productos al LLM para que éste lo utilice como contexto.

TODO: incluir un diagrama de arquitectura.