

# CuestGen

*Leo Aleksanteri Salomon Ketola*

*Desarrollo de Aplicaciones Multiplataforma*

*Curso 2024/2025*

*30 de junio 2025*



# Resumen

CuestGen es una aplicación web que combina el uso de Java con el framework Spring Boot que permite crear al usuario bancos de preguntas de las cuales se puede generar un cuestionario del estilo de un examen. Esto utiliza una inteligencia artificial para crear bancos de preguntas a partir de un texto o de un PDF.

Se pretendía utilizar inicialmente una inteligencia artificial como es Gemini IA con una conexión a nube, pero esta para el resto del diseño no encajaba por lo que se ha optado por el uso de Ollama para ejecutar los modelos de IA de forma local. El modelo elegido es Gemma3 que es una versión abierta de Gemini 2.0. Se probaron distintos modelos y con distintos tamaños, pero Gemma 3 de cuatro mil millones de parámetros era el modelo que mejores resultados daba en cuanto a tamaño, velocidad y calidad.

La solución en nube supondría también el uso de una base de datos en nube, pero este aplicativo utiliza un MongoDB local. Estas dos herramientas corren dentro de contenedores Docker que se crean utilizando Docker Compose.

Conseguir resultados uniformes de las IA de manera constante supone un reto importante, ésta se ha intentado resolver usando etiquetas de MongoDB y otras técnicas de programación para solamente guardar datos que correspondan a los campos que se quieran obtener.

Finalmente, el uso de una IA en local supone que toda la carga de ejecución cae sobre el usuario, esto significa que los resultados obtenidos variarán en velocidad dependiendo del hardware de la cual dispone el usuario. Si un usuario tiene un ordenador muy potente podría incluso utilizar un modelo de IA más grande para mejores resultados.

# Abstract

CuestGen is a web application which combines Java with the Spring Boot framework, this allows users to create question pools which can be used to generate exam style questionnaires. It uses an artificial intelligence to create question pools from text or from a PDF.

Initially it was thought of using Gemini AI with a cloud connection, but this did not fit the design of the rest of the application, so it ended up using Ollama to execute AI models locally. Gemma3 is the model that was selected in the end which is an open version of Gemini 2.0. Many models were tried, with different sizes but a 4b parameter Gemma 3 gave the best results for size, speed and quality.

The cloud solution would also mean the use of a cloud database, but this app uses a local MongoDB. These two tools run inside Docker containers that are created using Docker compose.

Getting consistent results from the AI was an important challenge, but the variance has been reduced thanks to the use of MongoDB tags and other programming techniques to only store data for fields we want to obtain.

Finally, the use of a local AI meant all the execution weight lied on the user, which meant the obtained results could vary in speed depending on the hardware the user was able to use. If a user had a very powerful computer, it could use a larger model for better results.

# ÍNDICE

1.	Introducción.....	5
2.	Planificación .....	6
3.	Análisis de requisitos.....	7
4.	Diseño .....	10
4.1	Dependencias .....	14
4.2	Generación Aumentada de recuperación (RAG) .....	15
4.3	Requerimientos.....	16
5.	Implementación .....	17
5.1	Configuración.....	17
5.2	Desarrollo.....	22
5.2.1	Modelo.....	22
5.2.2	Controlador.....	27
5.2.3	Vista.....	29
6.	Ánalisis de resultados .....	33
7.	Conclusiones y trabajo futuro.....	35
8.	Bibliografía .....	36

# 1. Introducción

El empleo de cuestionarios es una práctica habitual como mecanismo de evaluación de los conocimientos de una persona en diferentes situaciones tales como son la educación, formación laboral, evaluación, entre otros. Sin embargo, la creación manual de estos cuestionarios es especialmente repetitiva y tediosa cuando se requieren múltiples versiones de preguntas y respuestas en orden alterno.

Para resolver este problema, nace CuestGen, una aplicación web desarrollada con el objetivo de facilitar con el proceso de automatización en la creación de cuestionarios tipo examen. Esta herramienta está pensada tanto para profesores, que necesitan generar exámenes de forma rápida, como para estudiantes, que buscan practicar con diferentes modelos de examen para estar preparados para cualquier situación que se les presente en el examen.

CuestGen permite generar cuestionarios con preguntas y respuestas agrupados aleatoriamente, lo que ayuda a evitar la memorización mecánica y fomenta una evaluación más práctica y amena. Además, el usuario puede personalizar varios aspectos del cuestionario como el número de preguntas, el número de respuestas, los tipos de preguntas (opción múltiple, verdadero/falso, respuesta corta o respuesta larga).

Una de las funcionalidades principales de la aplicación es la gestión de bancos de preguntas. Los usuarios pueden crear sus propios bancos desde cero. También existe un transformador que permite convertir preguntas en texto plano o documentos PDF con temario ya existentes en bancos de preguntas de forma automática. Esto significa que no es necesario partir de un formato específico: incluso con solo el temario en PDF, CuestGen puede generar un conjunto de preguntas para crear el banco de preguntas.

Una vez creado el banco de preguntas, la aplicación generará un cuestionario aleatorio a partir de él. El usuario tiene la opción de incluir un anexo con las respuestas correctas al final del documento, lo cual es útil tanto para la corrección (estará en una página separada) como para la autoevaluación. Además, se ofrecerá una vista previa editable del cuestionario final, lo que permitirá realizar ajustes antes de exportarlo.

En cuanto a la exportación, CuestGen permite generar los cuestionarios en formato PDF, asegurando una presentación profesional y lista para imprimir o compartir.

En resumen, CuestGen combina flexibilidad, automatización y sencillez de uso en una única plataforma, eliminando muchas de las barreras habituales en la creación de exámenes personalizados. Tanto si eres docente como estudiante, esta herramienta está diseñada para ayudarte a optimizar tu tiempo y mejorar tus procesos de evaluación o estudio.

## 2. Planificación

Como en todo desarrollo de software, es fundamental contar con una planificación temporal del proyecto.

La duración total del proyecto es de aproximadamente 15 semanas, desde el 21 de marzo hasta la exposición final el 3 de junio, dividida en las siguientes fases de desarrollo:

- *El análisis de requisitos:* se representa la fase inicial. Durante esta etapa se identifican las necesidades del proyecto y se definen los objetivos. Se realiza un análisis de los requisitos funcionales, qué debe hacer la aplicación, y no funcionales, cómo debe comportarse, lo que permite establecer el alcance del sistema.
- *Fase de desarrollo:* en la cual se lleva a cabo la implementación del código, la lógica del sistema y la integración de los requisitos definidos.
- *La fase de documentación:* comprende la toma de notas, la redacción de la memoria del proyecto al igual que el registro de todas las decisiones y procesos desarrollados.
- *La fase de explotación:* la fase de exposición, donde finalmente se prepara y presenta el proyecto mediante una presentación en formato PowerPoint (PPTX). Este es el objetivo final de proyecto, crear una presentación del proyecto y demostrar su funcionamiento.

La imagen a continuación [Imagen 1] contiene el Diagrama de Gantt para la planificación del proyecto. En la parte superior del diagrama se indican las fechas estimadas para cada una de las fases. Este diagrama corresponde a un diagrama de Gantt, ampliamente utilizada en la gestión de proyectos es una representación gráfica del tiempo empleado en cada fase, para representar visualmente la planificación y el progreso de las tareas. Es habitual que estas fases se solapen ya que pueden complementarse unas a otras, por ejemplo, realizar pruebas para ver si es correcta el uso de una tecnología en el proyecto y cómo implementar su funcionalidad con otras, para que pueda funcionar correctamente.

Este diagrama ha sido elaborado utilizando la versión gratuita de TeamGantt, una aplicación web especializada en planificación de proyectos.

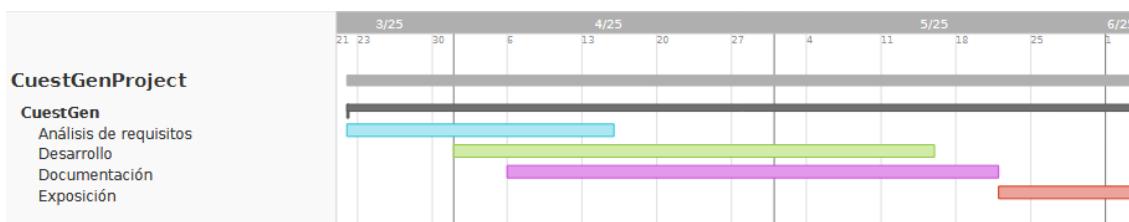


Imagen 1 Diagrama de Gantt

Creado usando la herramienta <https://www.teamgantt.com/>

### 3. Análisis de requisitos

CuestGen es una aplicación que requerirá de diversas herramientas para su implementación y su posterior explotación. En primer lugar, es esencial definir claramente el tipo de *software* que se desarrollará. En este caso se ha optado CuestGen como una aplicación web, facilitando así su acceso desde prácticamente cualquier dispositivo que disponga de un navegador web. Esta decisión responde a la necesidad de una aplicación multiplataforma, de fácil acceso e instalaciones mínimas.

El programa es una aplicación de generación de bancos de preguntas y sus correspondientes cuestionarios.

Un banco de preguntas se podrá crear de tres maneras distintas:

- *Manual*: Esta configuración tendrá una página donde se podrá establecer un título, una descripción y las preguntas de forma dinámica. Se pueden añadir preguntas, y dependiendo del tipo de pregunta añadir las respuestas y elegir la respuesta correcta. No se podrá añadir preguntas de tipos no registradores en la aplicación por parte del usuario. Las preguntas de opción múltiple tendrán más de dos respuestas, una pregunta de verdadero falso tendrá dos respuestas y las preguntas abiertas no tendrán respuesta definida. Existe la posibilidad de eliminar o añadir las respuestas y preguntas a necesidad del usuario.
- *Preguntas desde texto*: Esta función de la aplicación permite la creación de preguntas a partir de un banco de preguntas en texto plano. Es decir, si el usuario ya tiene un banco de preguntas que quiera utilizar en esta aplicación podrá pasársela en formato texto para que la pueda utilizar para generar cuestionarios. Esto usará una Inteligencia Artificial con **RAG** para poder transformar el texto al formato interno del programa.
- *Preguntas desde PDF*: A través de esta configuración será posible introducir un PDF y elegir el número de preguntas a generar. Esto usará una IA con **RAG** para poder transformar el PDF a preguntas en formato texto y del texto al formato interno del programa.

En la aplicación habrá páginas en las que se podrá visualizar un listado de los bancos de preguntas, que incluirá un enlace para poder visualizar los detalles de cada banco. Esto también se aplica a los cuestionarios. En este listado habrá un botón para poder borrar cada una de las opciones de la lista.

En la página de los detalles del banco de preguntas se creará un botón para generar el cuestionario, otro para editar y finalmente uno para borrar. Cuando se selecciona la opción de “Generar cuestionario” el sistema preguntará al usuario

de cuantas preguntas desea que sea el cuestionario en una ventana superpuesta al resto de la página. Una vez generado el cuestionario desde un banco de preguntas, se mostrará la página con los detalles del cuestionario.

A diferencia de los bancos de preguntas, la página de visualización de los detalles del cuestionario también será la que se utilice para editar sus propiedades. También incluirá un botón para exportarlo a formato PDF.

Los cuestionarios sólo se podrán borrar desde su listado.

En la parte superior de la página se implantará una barra de navegación para ir al inicio, ir al listado de bancos de preguntas o cuestionarios y un botón para crear los bancos de preguntas mediante uno de los métodos previamente descritos.

La información sobre los bancos de preguntas y cuestionarios se guardará de forma local en disco dentro de una base de datos.

No se necesitará de un inicio de sesión para poder acceder a la aplicación. Los datos que se guardan estarán totalmente en local

Ejemplo de un caso de uso para crear un cuestionario y exportarlo a PDF con soluciones.

Actor principal: Usuario. Actor secundario: Sistema.

Datos que usa: Archivo PDF, número de preguntas.

Precondiciones: la aplicación debe estar iniciada.

Postcondiciones:

Éxito: se recorre el caso de uso en su totalidad con éxito y se genera y descarga un PDF con soluciones sobre información que está incluida en dicho archivo.

Fallo: página de error o timeout.

Flujo principal:

1. El usuario abre la página inicial.
2. El sistema muestra la página inicial.
3. El usuario selecciona la opción de crear un banco de preguntas a partir de un pdf.
4. El sistema muestra el formulario para seleccionar un archivo e introducir un número de preguntas
5. El usuario selecciona examinar y selecciona un archivo pdf, introduce el número de preguntas a generar y selecciona la opción de Subir PDF.
6. El sistema hace llamada a la API de Ollama local y comienza a generar las preguntas y pasárlas a formato JSON.
7. El sistema muestra la página de detalles del banco de preguntas.

8. El usuario selecciona la opción de generar cuestionario.
9. El sistema abre un cuadro para introducir el número de preguntas
10. El usuario introduce un número de preguntas y selecciona generar
11. El sistema genera un cuestionario a partir del banco de preguntas y muestra el formulario de edición del cuestionario.
12. El usuario selecciona la opción de exportar cuestionario con soluciones.
13. El sistema crea la estructura del PDF y manda dicho al navegador web.
14. El navegador web inicia la descarga del archivo.

Flujo secundario:

6.a El sistema supera el tiempo especificado de generación de las preguntas.

Se vuelve al paso 5.

6.b El sistema no es capaz de generar el archivo vectorial.

Se vuelve al paso 5.

6.c El sistema no pudo crear las preguntas en la estructura correcta.

Se vuelve al paso 5.

6.d El sistema creó las preguntas con éxito, pero ocurrió una interrupción en la ejecución.

Recargar la página para comprobar si la ejecución ha sido un éxito en caso contrario volver al paso 5.

11.a El sistema no pudo encontrar el banco de preguntas correspondiente.

Se vuelve al paso 5.

## 4. Diseño

Para desarrollar un software funcional y completo se ha considerado esencial la combinación de múltiples tecnologías que permitan crear una aplicación robusta, portable y eficiente. Para la lógica de *backend* se ha optado por **Java** junto con el *framework* **Spring**, asegurando modularidad y facilidad de mantenimiento. La persistencia de datos se gestionará mediante la base de datos **MongoDB**. En cuanto a la portabilidad y facilidad de despliegue, se ha decidido utilizar **Docker** y **Docker Compose**, garantizando entornos uniformes y escalabilidad. La ejecución de los modelos de inteligencia artificial, necesaria para la generación automática de preguntas, será gestionada por **Ollama** utilizando el modelo **Gemma3**. Por su parte, el *frontend* se construirá combinando tecnologías estándar como **HTML**, **JavaScript** y **CSS**, junto con el motor de plantillas **Thymeleaf** para generar vistas dinámicas e interactivas.

Se ha optado por utilizar el lenguaje de programación **Java** dada mi experiencia formativa, así como por las diversas ventajas técnicas que ofrece, entre las que destacan su robustez, portabilidad, gran disponibilidad de documentación y soporte online. Es un lenguaje de propósito general que es adecuado para implementar aplicaciones web debido a su integración fluida con tecnologías de servidor e implantación multiplataforma gracias a su máquina virtual JVM. Es un lenguaje comúnmente usado para programas que utilicen el patrón de diseño *Modelo-Vista-Controlador* (MVC).

El lenguaje Java se emplea principalmente para el desarrollo de todo el código *backend* del proyecto CuestGen, incluyendo la gestión interna de bancos de preguntas, la generación y manejo de peticiones hacia los modelos de inteligencia artificial, así como las transacciones necesarias con la base de datos. Para realizar estas conexiones se ha utilizado Spring, que se explica a continuación.

Para facilitar la implantación de la lógica *backend* se ha escogido el framework de **Spring Boot**, el cual está construido sobre el framework Spring. Spring es un *framework* gratuito y de código abierto (bajo la licencia “Apache Licence 2.0”) con un amplio catálogo de aplicaciones desde IA Generativa, microservicios, *Cloud* y en este caso aplicaciones web. Facilita la inyección de dependencias al programa y ayuda en la integración de otras tecnologías como JPA. Spring Boot es un *framework* construido sobre Spring para facilitar el uso de Spring con configuraciones automáticas (minimiza configuración), servicios embebidos (ej. *Tomcat*) y facilita el empaquetado y despliegue de aplicaciones Spring.

Para decidir dónde se guardan los datos como son los bancos de preguntas que incluyen un título, una descripción, fecha de creación las preguntas y los cuestionarios que tienen los mismos campos que un banco de preguntas, pero siendo una selección de preguntas de un banco. Se guardarán en archivos **JSON** que seguirán la siguiente estructura JSON [Imagen 2]:

```
{
  "_id": ObjectId("AAA"),
  "title": String,
  "description": String,
  "questions": [
    {
      "_id": ObjectId("BBB"),
      "text": String,
      "type": Enum.QuestionType,
      "answers": [String, // Opcional
                 String,
                 String...]
    },
    "correctAnswer": String // Opcional
  ],
  {
    "text": String,
    "type": Enum.QuestionType,
  }
],
"creationDate": Date
}
```

*Cod Frag 1 Estructura base de datos Para un banco de preguntas o un cuestionario*

Como se puede observar en este esquema [Cod Frag 1], la estructura de los datos a guardar no es homogénea por tanto una base de datos relacional no cumpliría con su cometido. En consecuencia, se ha optado por utilizar una estructura de archivos JSON y el sistema gestor de bases de datos más empleado para este tipo de archivos es MongoDB un sistema gestor de bases de datos NoSQL.

Tiene integración con Spring de forma directa con la dependencia **Spring Data MongoDB** la cual se integra directamente con lo que ya está diseñado para el proyecto. Además, nos permite hacer nuestras configuraciones propias por si fuere conveniente.

¿Sería posible usar una base de datos relacional? Sí se podría optar por la utilización de una base de datos relacional, pero haría que la base de datos tenga muchos campos puestos en nulo, mientras que la flexibilidad de MongoDB es perfecta para este proyecto ya que no tiene una estructura rígida por la naturaleza de los tipos de preguntas.

Como herramienta para gestionar las dependencias he decidido usar por usar **Maven** (la otra opción siendo Gradle) por mi familiaridad dicha tecnología, además que Gradle se usa principalmente para aplicaciones móviles.

Para escribir el código y ejecutar el programa he usado un entorno de desarrollo integrado (IDE) llamado **IntelliJ IDEA Ultimate**, desarrollado por JetBrains. Está diseñado por defecto para el desarrollo de aplicaciones Java.

Por cómo se quiere procesar el texto y PDF es preciso utilizar una IA para hacer las transformaciones a JSON para poder guardarse en la base de datos y para poder generar las preguntas con una **IA generativa**. Como se quiere guardar toda la información en local he decidido utilizar **Ollama**, un programa para poder ejecutar Modelos de lenguaje (LLM) en local sin necesidad de un servidor. Es de código abierto y con licencia MIT.

Existen múltiples modelos de IA, en este caso me he inclinado por utilizar una IA de uso general como es **Gemma3**, un modelo abierto de Google basado en Gemini 2.0. Es uno de los modelos de IA más potentes que se pueden ejecutar en una única unidad de procesamiento gráfico (GPU). Por limitación de *hardware* y por una experiencia de usuario más rápida utilizo el modelo de cuatro mil millones de parámetros para generar los bancos de preguntas.

Para poder correr Ollama y MongoDB he decidido utilizar la tecnología de contenedores que ofrece Docker para una experiencia portable y multiplataforma.

Docker es un *software* basado en virtualización a nivel de sistema operativo, más concretamente contenedores. Permite empaquetar aplicaciones y todas sus dependencias en unidades aisladas ligeras y altamente portables. Además, a diferencia de máquinas virtuales tradicionales los contenedores utilizan el núcleo del sistema operativo de la máquina en la que se ejecutan y aíslan los procesos que se necesiten para ejecutar la aplicación.

Para facilitar la definición, gestión y configuración de la aplicación se utiliza **Docker Compose**, una herramienta que complementa Docker, que, a través de archivos YAML (docker-compose.yml, compose.yml) permite describir la configuración, propiedades y variables de entorno de las aplicaciones.

Para mostrar la aplicación web al usuario se usará el formato estándar de **HTML** con **CSS** y **JavaScript**. Estas tres tecnologías se complementan entre sí para crear una interfaz gráfica para la aplicación. Para su integración con Spring se usa **Thymeleaf**, que se explica en el apartado de 5.1 Dependencias.

Para el control de versiones se ha utilizado **GIT**, una tecnología ampliamente utilizada para tener un control de versiones online. El proyecto entero se ha ido subiendo a **GitHub**, plataforma usada para compartir repositorios GIT.

Un patrón de diseño de software **Modelo Vista Controlador** es comúnmente utilizado para implementar interfaces de usuario, datos y lógica de control. Marca una clara diferenciación entre *lógica de negocios* y muestra al usuario. Para proyectos de gran escala permite una mejor división del trabajo y un mejor mantenimiento. Se compone de tres partes, como bien su nombre indica,

Modelo, Vista y Controlador. El modelo es la parte que manejará los datos y la lógica de negocios. El controlador redirecciona las acciones del usuario a modelos y vistas. Las vistas se encargan del diseño y presentación que visualiza el usuario.

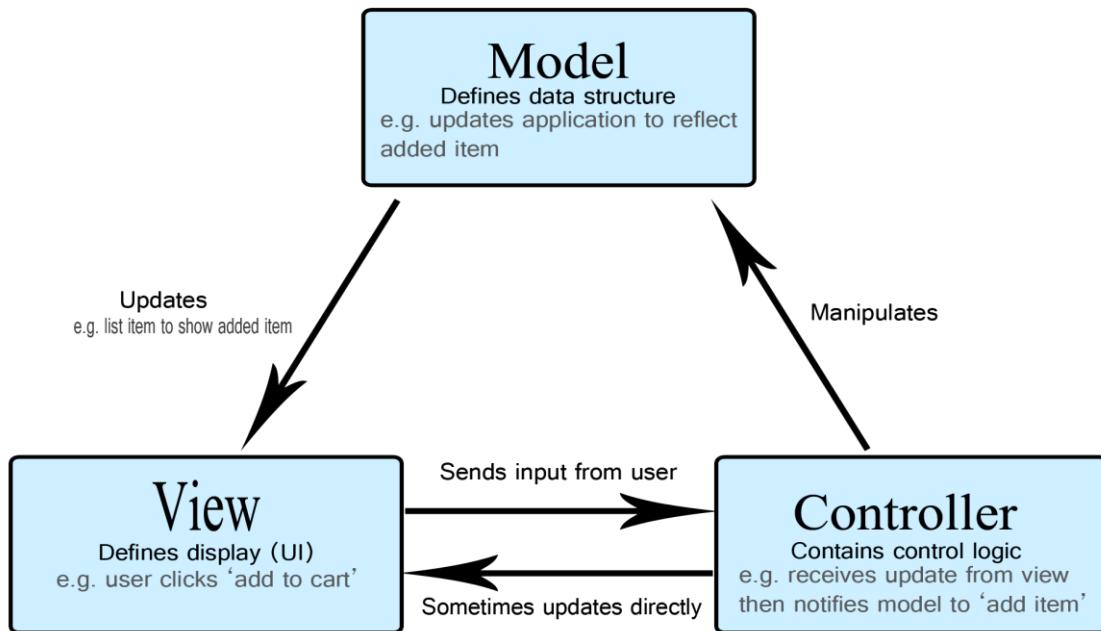


Imagen 2 Ex. Mozilla Corporation (2023, noviembre 13) MVC

<https://developer.mozilla.org/es/docs/Glossary/MVC>

El flujo deseado entre los diferentes componentes y páginas de la aplicación se puede observar en el siguiente diagrama de flujo [Imagen 3 para observar las diferentes maneras de navegar la aplicación.

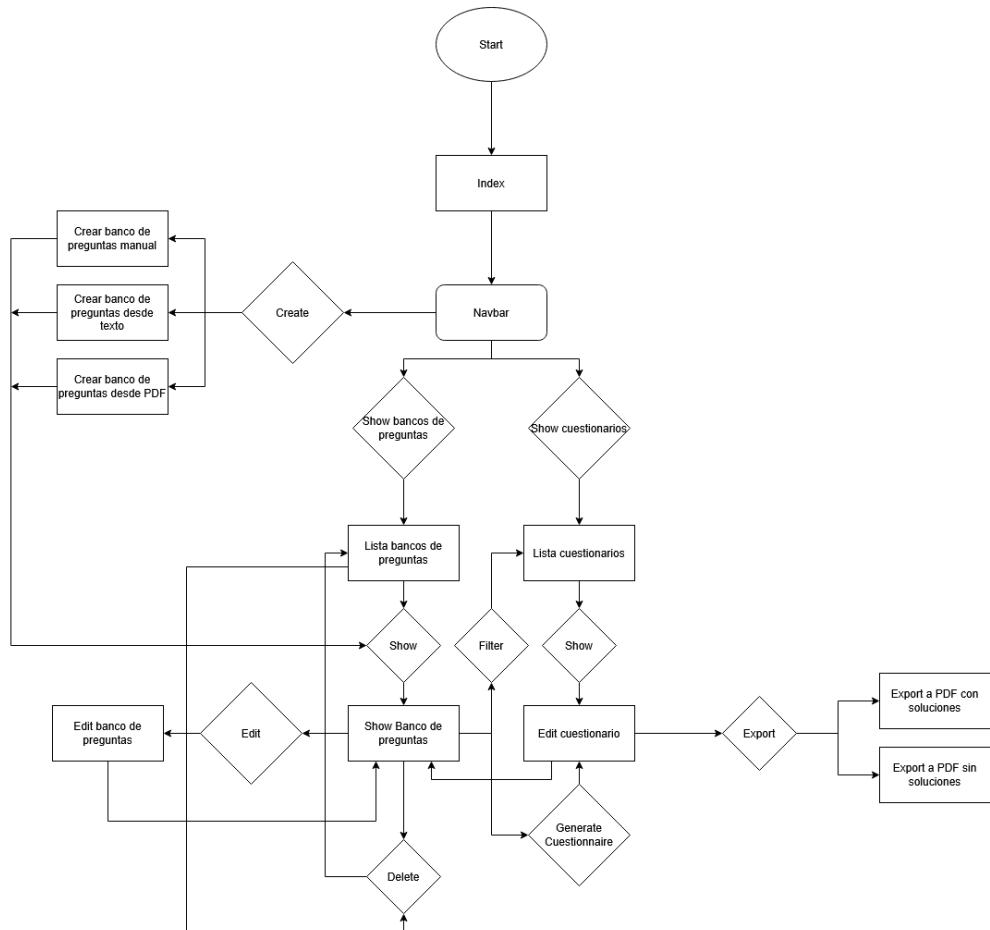


Imagen 3 Diagrama de flujo

Creado usando la herramienta <https://app.diagrams.net/>

## 4.1 Dependencias

Las dependencias utilizadas son las siguientes:

- *Spring Data MongoDB*: la conexión que realiza Spring a la base de datos de MongoDB.
- *Thymeleaf*: motor de plantillas para aplicaciones web que utiliza HTML, XML, Javascript, css o texto plano para que sean interpretadas por el navegador o el servidor.
- *Jackson*: biblioteca utilizada para serializar y deserializar objetos JSON.
- *Lombok*: biblioteca y plugin que reduce el uso de código repetitivo mediante anotaciones.
- *Spring AI Ollama*: permite la conexión automática de Ollama con Spring además de dar librerías para interactuar con la IA.
- *Spring AI Tika document reader*: módulo que se utiliza Apache Tika para leer y extraer texto de documentos de texto como PDF, DOCX, PPTX,

TXT, etc... y prepararlos para tareas de Retrieval Augmented Generation (RAG).

- *SpringBoot Docker Compose*: integra Docker Compose para que cuando se ejecute el programa también ejecute el archivo compose para que corra en el Docker Engine y no tener que ejecutar los contenedores de forma separada.
- *iTex7*: biblioteca para crear y manipular archivos PDF, se usa para la exportación de cuestionarios.
- *Spring Web*: librerías para crear aplicaciones web que utilicen Spring MVC. Usa Apache Tomcat para crear un contenedor embebido para levantar el servicio.

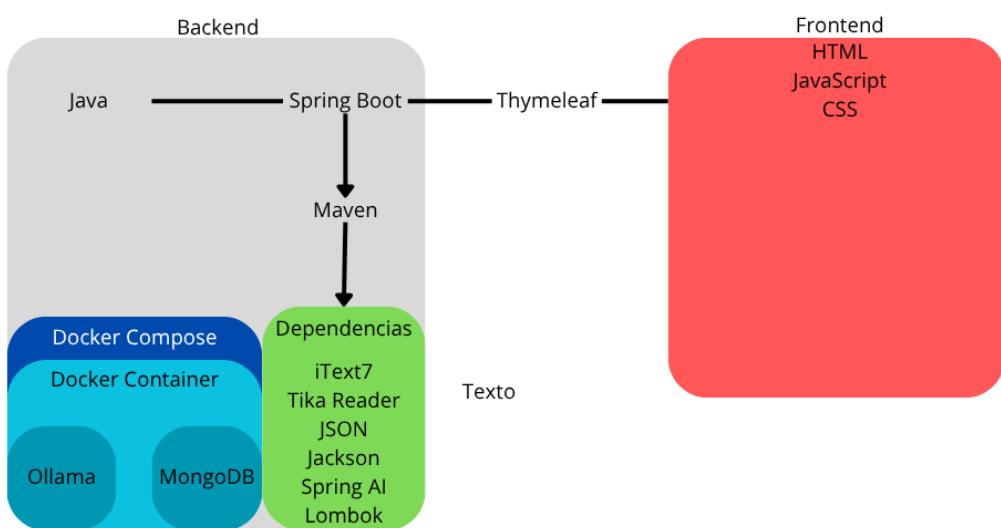


Imagen 4 Diagrama de componentes

## 4.2 Generación Aumentada de recuperación (RAG)

La **generación aumentada de recuperación** o *Retrieval-Augmented Generation* en inglés. Es una forma de optimizar los resultados de un LLM mediante la alimentación de información específica sin necesidad de modificar el modelo usado. Esta información puede ser información personalizada o información más actualizada que el modelo no tenía en su momento de entrenamiento (si no tiene búsqueda por internet). Permite al modelo de IA generativa otorgar resultados que están adaptados al **contexto** que se le ha embebido.

Para poder usar esta información la información debe ser transformada en representaciones numéricas (vectores) que se pueden guardar en archivos con información vectorial o una base de datos vectorial. Esta información se utilizará para hacer una búsqueda semántica, que en esencia hace que cuando al modelo se le realice una consulta también lo convierte en un vector y compara similitudes con los vectores almacenados. Los resultados se recuperan e inyectan como contexto para el prompt que posteriormente se envía al LLM.

## 4.3 Requerimientos

Debido a que el uso de la IA consume muchos recursos no todas las máquinas serán capaces de ejecutar la aplicación con las funcionalidades de IA.

Requerimientos de sistema para ejecutar el programa:

- *Docker engine y Docker compose* (Docker Desktop)
- *Java 17+*
- *Nvidia Container Toolkit o AMD Container Toolkit* (ROCM). Con WSL2
- Una GPU capaz de ejecutar un modelo de 4b de parámetros en Ollama

**Nvidia Container Toolkit** es una colección de bibliotecas y utilidades para habilitar al usuario construir y ejecutar contenedores que utilicen aceleración de GPU.

## 5. Implementación

### 5.1 Configuración

Para poder empezar con el proyecto es necesario tener instalado **Docker Engine** y **Docker Compose**. En Windows lo más habitual es instalar Docker Desktop, éste incluye ambos.

Después instalaremos las librerías de **Nvidia Container Toolkit** a través de un subsistema Linux de nuestra máquina (WSL2). Yo he utilizado Ubuntu.

Simplemente utilizando los siguientes comandos en el WSL2 se instala:

```
curl -fsSL https://nvidia.github.io/libnvidia-container/gpgkey | sudo gpg --dearmor -o /usr/share/keyrings/nvidia-container-toolkit-keyring.gpg \\ && curl -s -L https://nvidia.github.io/libnvidia-container/stable/deb/nvidia-container-toolkit.list | \\ sed 's#deb https://#deb [signed-by=/usr/share/keyrings/nvidia-container-toolkit-keyring.gpg] https://#g' | \\ sudo tee /etc/apt/sources.list.d/nvidia-container-toolkit.list1.
```

```
sudo apt-get update
```

```
sudo apt-get install -y nvidia-container-toolkit
```

Ahora configuraremos Docker para que pueda utilizar el Nvidia Container Toolkit.

```
sudo nvidia-ctk runtime configure --runtime=docker
```

Después es necesario reiniciar el Docker Daemon.

*Cod Frag 2 comandos configuración nvidia-ctk*

*Comandos de la guía <https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html>*

Una vez tengamos configurado nuestros contenedores de Docker podremos crear nuestro proyecto Spring. Para facilitar esta creación existe la herramienta online de **Spring Initializr** “<https://start.spring.io/>” (VMWare, 2023). En esta se puede elegir el tipo de herramienta de dependencias, lenguaje de programación versión y *metadata*. En este caso se ha elegido Maven, Java y la versión por defecto. En el metadata, pondremos la información que queramos sobre grupos, nombre, etc... En la parte de la derecha podremos elegir algunas de las dependencias.

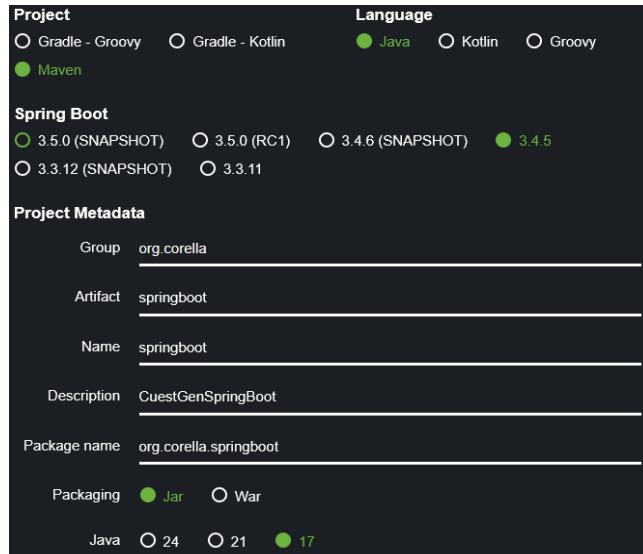


Imagen 5 Spring Initializr

Ex. <https://start.spring.io/>

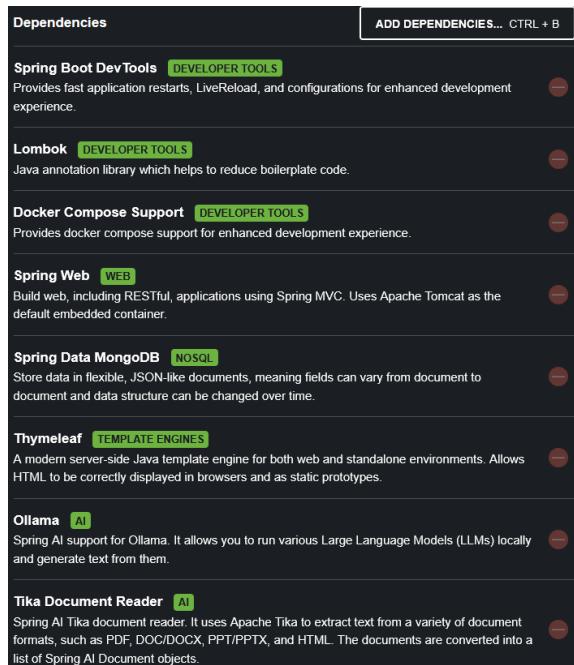


Imagen 6 Dependencias desde Spring Initializr

Ex. <https://start.spring.io/>

Además de estas dependencias debemos añadir **Spring AI Core**, **JSON**, **Jackson** e **iText7** [Cod Frag 3]:

```
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-core</artifactId>
    <version>1.0.0-M6</version>
</dependency>
<dependency>
    <groupId>org.json</groupId>
```

```

<artifactId>json</artifactId>
<version>20240303</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.18.3</version>
</dependency>
<dependency>
    <groupId>com.itextpdf</groupId>
    <artifactId>itext7-core</artifactId>
    <version>8.0.2</version>
    <type>pom</type>
</dependency>

```

Cod Frag 3 pom.xml fragmento

Se descarga y los abrimos con nuestro IDE, añadiremos algunas carpetas para tener una estructura más ideal para nuestro proyecto hasta tener algo similar a:

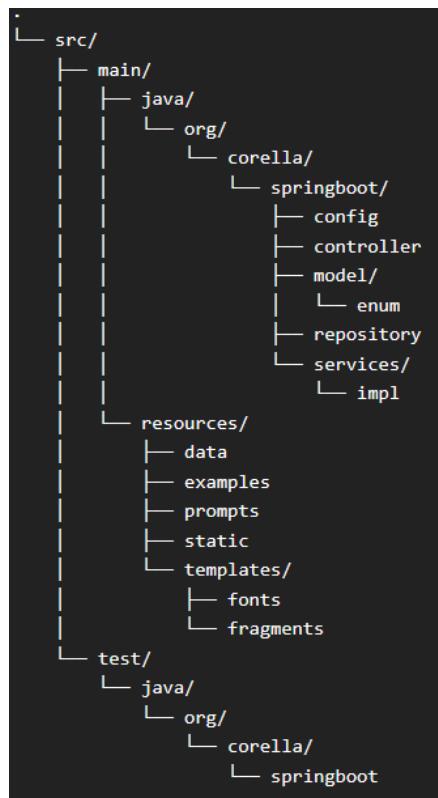


Imagen 7 Estructura de carpetas

Para nuestro compose a partir de la cual se ejecutarán los contenedores de **MongoDB**, **mongo-express** (por si fuera necesario administrar la base de datos) y Ollama tendremos este archivo para iniciar y configurar las imágenes y contenedores. Primero, se declara un servicio, se introduce su nombre de la imagen del repositorio de Docker o en otro caso un *Dockerfile* que incluya la información sobre la imagen. La propiedad “restart: always” permite que, si el

contenedor falla o se detiene, este vuelve a iniciarse cuando se reinicie el sistema operativo o cuando el contenedor se inicie tras la ejecución de Docker.

Las variables de entorno o de configuración se utilizan para las propiedades del propio programa. Los puertos a los que se expone para poder acceder a ellos son necesarios para que sea posible comunicarse con los contenedores desde fuera. Los volúmenes permiten guardar información persistente que no se destruya al eliminar el contenedor. Las *networks* sirven para comunicar contenedores. Ollama tiene una propiedad adicional que permite que utilice las propiedades de la aceleración de hardware que anteriormente se configuró.

```
services:
  mongodb:
    image: mongo:latest
    container_name: mongodb
    restart: always
    environment:
      MONGO_INITDB_ROOT_USERNAME: ${MONGO_INITDB_ROOT_USERNAME}
      MONGO_INITDB_ROOT_PASSWORD: ${MONGO_INITDB_ROOT_PASSWORD}
    ports:
      - ${MONGO_PORT}:27017
    volumes:
      - ./MongoDB/mongo_data:/data/db
    networks:
      - mongonetwork
  mongo-express:
    image: mongo-express
    container_name: mongo_express
    restart: always
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: ${MONGO_INITDB_ROOT_USERNAME}
      ME_CONFIG_MONGODB_ADMINPASSWORD: ${MONGO_INITDB_ROOT_PASSWORD}
      ME_CONFIG_BASICAUTH_USERNAME: user
      ME_CONFIG_BASICAUTH_PASSWORD: pass
      ME_CONFIG_MONGODB_ENABLE_ADMIN: true
      ME_CONFIG_MONGODB_SERVER: mongodb
      ME_CONFIG_MONGODB_URL: mongodb://${MONGO_INITDB_ROOT_USERNAME}:
        ${MONGO_INITDB_ROOT_PASSWORD}@mongodb:27017/?authSource=admin
    ports:
      - "${EXPRESS_PORT}:8081"
    depends_on:
      - mongodb
    networks:
      - mongonetwork
  ollama:
    image: ollama/ollama
    container_name: ollama
    restart: always
    ports:
```

```

    - "11434:11434"
volumes:
  - ./Ollama/ollama:/root/.ollama
deploy:
  resources:
    reservations:
      devices:
        - capabilities: [gpu]

networks:
  mongonetwork:
    driver: bridge

```

Cod Frag 4 compose.yml

Entre \$ y llaves (\${}) se podrá ocultar información sensible o variables de entorno en un archivo de *variables de entorno* .env . Es una manera para esconder información sensible para proyectos pequeños sin tener que utilizar técnicas de encriptación. Por ejemplo, un archivo. env puede contener los nombres de usuario, contraseñas, puertos internos, etc....

```
MONGO_INITDB_ROOT_USERNAME=MIUSUARIO
```

SpringBoot incluye un archivo de configuración application.properties que incluirá información sobre la base de datos MongoDB, el servlet para la subida de archivos, el cliente http y Ollama.

```

spring.application.name=CuestGenSpring
spring.data.mongodb.uri=mongodb://root:pass123@localhost:27017/
spring.data.mongodb.database=cuestgendata
spring.servlet.multipart.enabled=true
spring.servlet.multipart.max-file-size=10MB
spring.servlet.multipart.max-request-size=10MB
spring.http.client.read-timeout=30s
spring.http.client.connect-timeout=120s
spring.lifecycle.timeout-per-shutdown-phase=120s
server.port=8085
spring.ai.ollama.chat.options.model=gemma3
spring.ai.ollama.init.timeout=10m
spring.ai.ollama.init.max-retries=5
spring.ai.ollama.init.pull-model-strategy=when_missing

```

Cod Frag 5

También se incluyen tiempos de *timeout* más largos de lo habitual para contemplar las cargas que pueda tener la IA.

## 5.2 Desarrollo

El punto más habitual a la hora de empezar a desarrollar una aplicación es comenzar por el **modelo**, es decir los objetos que se guardarán en la base de datos y que interactuarán con los controladores y servicios.

A continuación, se incluye un **Diagrama de clases** [Imagen 8], es un diagrama UML que muestra los diferentes atributos, métodos, constructores, propiedades, implementaciones del proyecto en su totalidad.

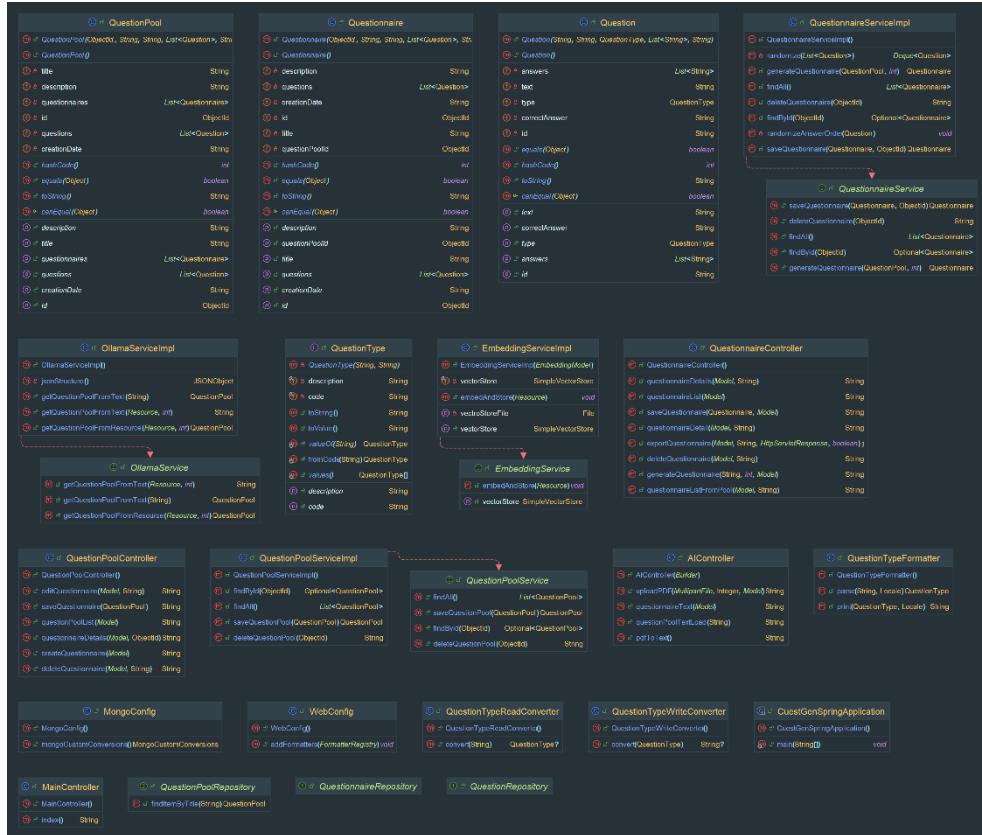


Imagen 8 Diagrama de clases del proyecto

### 5.2.1 Modelo

#### 5.2.1.1 Objetos del modelo

Los objetos son los componentes que interactúan con el sistema. Serán los **bancos de preguntas, cuestionarios y las preguntas**, cada una teniendo información que se guardará en la base de datos.

En el siguiente Diagrama de Clases [Imagen 9] muestra los diferentes atributos, funciones y propiedades que tienen las clases. El id es un identificador único que MongoDB nos proporciona automáticamente al insertar en la base de datos usa el tipo de datos **ObjectId** que usa un BSON de 12 bytes para asegurar que sea único.

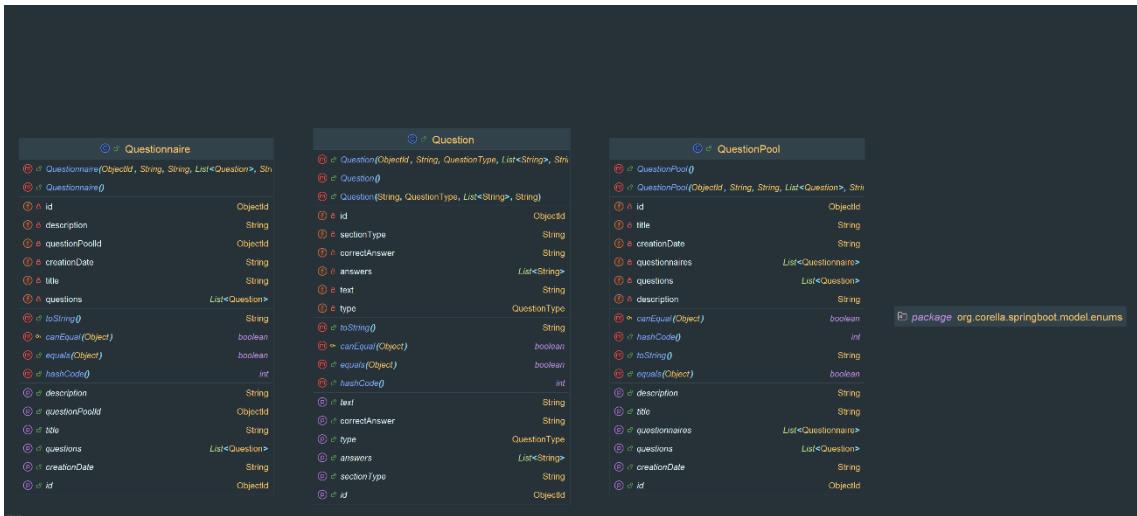


Imagen 9 Diagrama de clases de Model

Las clases llevan anotaciones **lombok** como `@Data`, `@AllArgsConstructor` y `@NoArgsConstructor` que permite reducir código repetitivo como todos los `getter`, `setter`, `constructor` de todos los parámetros y `constructor` sin parámetros. `@Document` es una anotación de Spring Data MongoDB para indicar en qué fichero se guardará ese objeto dentro de la base de datos.

Las clases **QuestionPool** (banco de preguntas) y **Questionnaire** (cuestionario) contienen los atributos “title” que es un título que se le dará al banco de preguntas o cuestionario que contendrá el tema del que trata, “description” es una descripción que contiene un breve texto que otorga mayor detalle que el título, “questions” es una lista que preguntas que contiene objetos de tipo **Question** (pregunta) y como último atributo en común de un banco de preguntas y cuestionarios, “creationDate” que incluye una fecha en formato “YYYY-MM-dd” (año-mes-día) para registrar el día de creación del objeto.

Los bancos de preguntas, además, tendrán un campo “questionnaires” para guardar un listado de los cuestionarios que se han creado a partir de ese banco de preguntas, y que utiliza una etiqueta de Spring Data MongoDB `@DocumentReference` para que pueda hacer referencia directa al objeto cuestionario creado a partir de su ObjectId.

Los cuestionarios contienen un campo “questionPoolId” que contiene el ObjectId del banco de preguntas de la que se basa.

Un **Question** (pregunta) es un objeto que contiene una etiqueta `@JsonIgnoreProperties` que sirve para enmendar errores que pueda cometer la IA al generar un banco de preguntas.

Además de su identificador tiene un “text” que es el cuerpo de la pregunta. “type” es un enumerado de tipo **QuestionType** para saber el tipo de preguntas, si será de opción múltiple, verdadero/falso, respuesta corta o respuesta larga. El campo “answers” es el listado de respuestas. El campo “correctAnswer” tiene la etiqueta

`@JsonInclude(JsonInclude.Include.NON_NULL)` para indicar a MongoDB que no incluya este campo si no está definido en el objeto Java con un valor no nulo.

### 5.2.1.2 Enumerados

El enumerado `QuestionType` contiene dos atributos que lo definen “code” un código para resumir el nombre del enumerado y “description” para aportar un texto del tipo en español. Contienen estos valores:

```
MULTIPLEOPTION("MO", "Opción Múltiple"),
BOOLEAN("BOOL", "Verdadero/Falso"),
LONGANSWER("LA", "Respuesta Larga"),
SHORTANSWER("SA", "Respuesta Corta");
```

Cod Frag 6 Enumerado

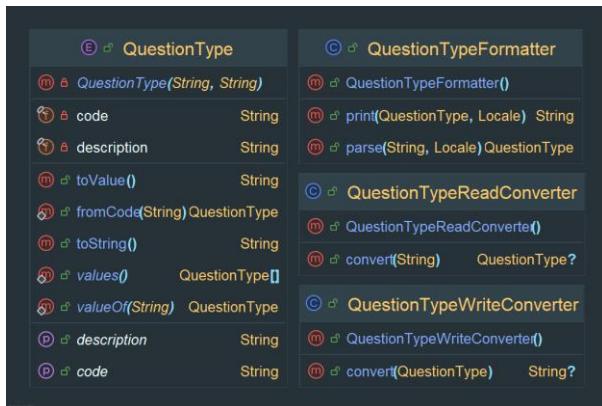


Imagen 10 Diagrama de clases de Enums

El enumerado contiene un método `fromCode` para transformar el código al tipo enumerado durante la serialización y deserialización del JSON.

El paquete también incluye dos converters que se incluye en la configuración de Mongo para indicar cómo debe **serializar** y **deserializar**. El formateador existe para solucionar un problema que existía con un selector en una plantilla HTML.

### 5.2.1.3 Servicios

En el modelo además de los objetos tendremos los servicios que implementarán la *lógica de negocio* de nuestro programa y la conexión a los *repositorios* que harán las transacciones que necesitemos con la base de datos.

Se ha dividido la implementación de los servicios en interfaces y por otro lado sus implementaciones en el paquete “impl”. Esto incluye lógica de cómo se realiza el guardado, llamadas a la API de Ollama, creación del archivo con información vectorial, búsquedas en la base de datos, etc....

La implementación de los servicios lleva una etiqueta `@Service` para indicar a Spring que esto es un servicio. Un servicio conecta varios repositorios y agrupa

su funcionalidad permitiendo que sean conectados automáticamente sin instanciación directa usando la etiqueta `@Autowired`.



Imagen 11 Diagrama de clases de Services

Los servicios de la [Imagen 11] se detallan a continuación:

### 1. EmbeddingService

**EmbeddingService** es el servicio que procesa el guardado de los texto o recursos a formato vectorial. **EmbeddingModel** es un modelo específicamente entrenado para generar vectores embebidos a partir de un recurso.

**SimpleVectorStore** es la manera en la que se guarda la información vectorial a un archivo vectorial utilizando el modelo embebido. En el [Cod Frag 7] se puede observar cómo se realiza el proceso de extracción de información del recurso, en este proyecto los PDF o textos planos, utilizando “*TikaDocumentReader*” anteriormente mencionado. Se divide el documento en tokens que determina de forma automática el “*TextSplitter*” de Spring AI y finalmente añade estos documentos a nuestro archivo vectorial. El archivo almacena únicamente un recurso a la vez, cada vez que un nuevo recurso es cargado el archivo vectorial es borrado y cargado con la nueva información.

```
public void embedAndStore(Resource resource) {
    File vectorStoreFile = getVectorStoreFile();
    if (vectorStoreFile.exists())
        if (!vectorStoreFile.delete())
            throw new RuntimeException("Error borrando vector store file");
    try {
        if (!vectorStoreFile.createNewFile())
            throw new RuntimeException("Error creando vector store file");
        TikaDocumentReader tikaDocumentReader = new TikaDocumentReader(resource);
        TextSplitter textSplitter = new TokenTextSplitter();
        List<Document> splitDocuments = textSplitter.split(tikaDocumentReader.get());
        vectorStore.add(splitDocuments);
        vectorStore.save(vectorStoreFile);
    } catch (Exception e) {
        e.printStackTrace();
        throw new RuntimeException("Error creating vector store file");
    }
}
```

Cod Frag 7 función embedAndStore

## 2. *OllamaService*

**OllamaService** es el servicio que hará las operaciones con llamadas a Ollama. Se utiliza un modelo con dos configuraciones distintas. Uno se utilizará para generar el output del programa en JSON, tendrá el output en JSON y una temperatura mínima (0,0) baja, la temperatura es la creatividad que se le permite a un modelo IA tener a la hora de generar una respuesta, para tener más consistencia a la hora de generar los JSON, y otro para generar las preguntas extraídas desde un PDF para luego ser procesadas para pasar a JSON con una temperatura superior para que sea más creativo a la hora de crear las preguntas. Utilizan el modelo **gemma3** de cuatro mil millones de parámetros.

Para no tener que escribir los “*prompts*” enteros en el código se utilizan los llamados “*PromptTemplates*” que son **plantillas** con el texto del *prompt* y unas variables puestas entre llaves {} para que se pueda pasar las variables que se quiera que sean dinámicas. Se pasan como parámetro dentro de un mapa de <String, Object>. Se lee el archivo donde tengamos el template y le añadimos nuestras variables para crear nuestro *prompt*. Este prompt se pasa a nuestro modelo de IA y hace una llamada con dicho prompt, devuelve una respuesta. De esta respuesta extraemos una cadena de caracteres. Con esta cadena podemos pasarlo a la IA de nuevo dependiendo del caso de uso. Si este es el resultado final que esperamos en formato JSON podemos deserializarlo utilizando *ObjectMapper* de **Jackson** para que lea los valores para deserializarlo a un objeto de tipo *QuestionPool*. Como la IA comúnmente comete errores es la razón por la cual la clase contiene `@JsonIgnoreProperties(ignoreUnknown = true)` para que pueda crear como mínimo una base para un banco de preguntas.

## 3. *QuestionPoolService*

**QuestionPoolService** es la clase que se encarga de realizar las llamas al repositorio para la búsqueda, guardado y borrado de **bancos de preguntas** en la base de datos y realizar las comprobaciones que sean necesarios como la comprobación de listas vacías y valores nulos.

## 4. *QuestionnaireService*

**QuestionnaireService** es la clase encargada principalmente de generar los **cuestionarios**. Para generar los cuestionarios utiliza el algoritmo para barajar de **Fisher Yates**. Un algoritmo que funciona en complejidad O(n), la cual, sigue esta lógica:

Para i desde n – 1 hasta 1 hacer

j = valor aleatorio con 0 <= j <= i

cambiar las posiciones de arr[j] y arr[i]

Esta implementación se puede realizar de la siguiente manera para nuestro programa:

```

Random r = new Random();
Question[] questionArr = questionList.toArray(new Question[0]);
List<Thread> threads = new ArrayList<>();
for (int i = questionList.size() - 1; i > 0; i--) {
    int j = r.nextInt( bound: i + 1);

    Question aux = questionArr[i];
    questionArr[i] = questionArr[j];
    questionArr[j] = aux;
}

```

*Cod Frag 8 Algoritmo de Fisher Yates*

Esto permite añadir una capa de aleatoriedad adicional al programa. Además de esto, baraja también el orden de las preguntas de tipo opción múltiple usando multithreading.

Aparte de esta funcionalidad tiene, igual que *QuestionPoolService*, una búsqueda, guardado y borrado de cuestionarios.

#### 5.2.1.4 *Repositorios*

Los **repositorios** son interfaces que implementan **MongoRepository** para realizar la conexión habilitada por Spring indicando en los genéricos de *MongoRepository<Clase, tipo de datos del Identificador>*. Estas clases no necesariamente contienen métodos a implementar ya que vienen gracias a *Spring Boot* para reducir código innecesario.

## 5.2.2 Controlador

Los **controladores** como ya se ha comentado, conectan la lógica del programa con lo que un usuario visualiza. Estas clases llevan la etiqueta *@Controller* para indicárselo a Spring que se usarán **plantillas HTML**. Estas clases gestionan las peticiones HTTP como GET y POST. Los métodos llevarán una etiqueta para distintas peticiones. Además, existe la operación *RequestMapping* que sirve como anotación genérica.



Imagen 12 Diagrama de clases de Controller

La clase **MainController** solamente contiene la página inicial que es mostrada al usuario al entrar en la aplicación. Internamente Spring también contiene una página “error” pero esta se procesa de forma automática lo cual hace que no sea necesaria su implementación explícita en el controlador.

El controlador de **bancos de preguntas** contiene operaciones para mostrar un único banco de preguntas, un listado de todos los bancos de preguntas, una función para añadir nuevos bancos de forma manual, guardado, edición y borrado.

Los **cuestionarios** son similares en funcionalidad, pero la forma en la que se muestran es distinta. En este caso solo se muestra el formulario de edición en vez de su página de detalles, desde una página de un banco de preguntas se llamará a la función de generación de un cuestionario que creará y guardará el cuestionario en la base de datos. También contiene una función de guardado, una función para mostrar los cuestionarios, una función para volver al banco de preguntas al que pertenece y finalmente la función de exportación.

La exportación recibirá un identificador de un cuestionario y un booleano para determinar si incluirá las soluciones o no. *HttpServletResponse* pertenece a la librería de *Java Jakarta* que nos permite, en esencia, enviar datos de vuelta al navegador web, en este caso una petición de descarga. Usando la librería de *iText7* y sus clases configuramos y construimos el PDF correspondiente para nuestro cuestionario. Incluyendo los campos de título, descripción, y las preguntas con sus respuestas, las soluciones irán incluidas o no dependiendo del booleano anteriormente mencionado. Para la fuente se ha utilizado una fuente no predeterminada por la librería, ésta viene incluida en la carpeta */resources/templates/fonts*, en la actualidad solamente es posible modificar el formato de la exportación dentro del código.

El **AIController** contiene todas las operaciones que utilicen nuestro **OllamaService** para hacer llamadas al modelo de IA. Se puede crear el cuestionario a través de texto, una función cargar dicho texto, el paso de recibir un pdf para pasarlo a un banco de preguntas y su subida.

Para recibir el PDF se usa el tipo de datos **MultipartFile** que es el tipo que devuelve el HTML, recibe también el número de preguntas que deberá generar. Este archivo se deberá transformar en un tipo que esté incluido en la interfaz **Resource** en este caso **ByteArrayResource** que podremos pasar a nuestra función de nuestro servicio para que se convierta en un archivo vectorial que el modelo IA pueda procesar. Una vez nos devuelva nuestro banco de preguntas lo guardaremos.

### 5.2.3 Vista

La **vista** o interfaz de usuario es la parte que el usuario puede visualizar para poder interactuar con la aplicación. Para implementar las vistas se ha usado **Thymeleaf** para utilizar las plantillas HTML con los controladores de Spring.

Estas plantillas utilizan las librerías de estilos de **Bootstrap** para darle un aspecto visual más agradable al usuario. Estas clases se utilizan simplemente indicando las distintas clases que contiene Bootstrap a cada una de las etiquetas HTML. Además de estos estilos también hay algunas modificaciones de estilos con **CSS**. Para obtener una mejor experiencia de usuario también se ha incluido fragmentos de **JavaScript** para incorporar fragmentos dinámicos como añadir preguntas, quitarlas y añadir y quitar respuestas.

Esta es la estructura de los HTML necesarios [Imagen 13]:

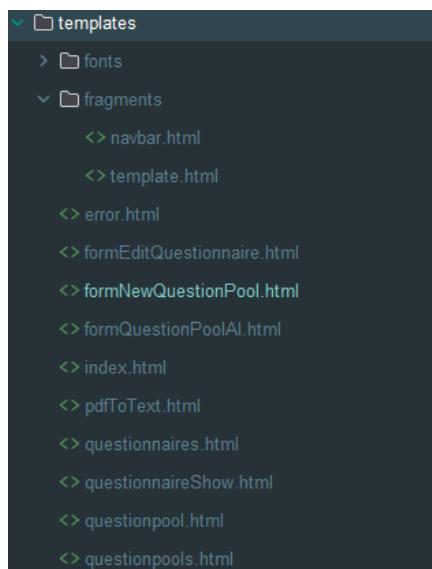


Imagen 13 Plantillas HTML

El fragmento *navbar.html* solamente contiene una barra de navegación que se encontrará en la parte superior de todas las páginas. *template.html* es una página base que se utilizará para crear el resto de las páginas.

El *index.html* o también llamado *landing page* contiene la página que se muestra al iniciar sin ninguna ruta adicional a la aplicación. Tiene una imagen, un título y los botones en grande para acceder a las funciones de creación.

La página *questionpools.html* incluye el listado de todos los bancos de preguntas. Tiene una tabla que incluye una sección con un enlace hacia los detalles de cada uno de los bancos de preguntas, un título, una descripción su fecha de creación y un botón de borrado.

Listado de bancos de preguntas				
Ver	Título	Descripción	Fecha de creación	Borrar
<a href="#">ID ⓘ</a>	Cuestionario-2025-05-18	Este cuestionario evalúa la comprensión del texto sobre las bases y aplicaciones de la Realidad Aumentada. Cubre temas como plataformas de desarrollo AR, objetivos de ARKit y ARCore, dispositivos de captura de imágenes y herramientas específicas para aplicaciones AR.	2025-05-18	<a href="#">X</a>
<a href="#">ID ⓘ</a>	Cuestionario-2025-05-18	Este cuestionario aborda conceptos fundamentales sobre la Realidad Aumentada (AR), incluyendo su definición, usos en distintas áreas, niveles de implementación, herramientas tecnológicas y perspectivas futuras del desarrollo de esta tecnología.	2025-05-18	<a href="#">X</a>

Imagen 14 Listado bancos de preguntas

La página *questionpool.html* contiene una tabla con la información más detallada de un banco de preguntas: su título, su descripción, sus preguntas que incluye texto de la pregunta su tipo, un listado de su respuesta si las tuviera y su respuesta correcta. Además de esta información hay cuatro botones en la parte superior, una para generar el cuestionario, otra para ver el listado de los cuestionarios que se han creado a partir de ese banco de preguntas, un botón para acceder al botón de edición y otra para borrar el banco de preguntas.



Imagen 15 Botones Banco de preguntas

Cuando se selecciona el botón de “Generar Cuestionario” aparecerá un campo creado utilizando JavaScript que nos permite introducir el número de preguntas que queremos que tenga el cuestionario. A esto se le denomina un elemento modal dialog. Este número no puede ser superior al número de preguntas que tiene el banco de preguntas, en dicho caso saltará un error.



Imagen 16 Modal Dialog

Cuando se selecciona el botón de Generar creará nuestro cuestionario y nos mostrará la página de detalles de cuestionario que a su vez es el formulario de edición de este.

El formulario *formEditQuestionnaire.html* nos permitirá editar cualquier detalle que veamos conveniente, ya sea si hay un error o queramos modificar una pregunta. No es posible cambiar el orden de las preguntas. En la parte inferior hay un botón para añadir una pregunta que utilizará un script de JavaScript para crear un nuevo bloque de pregunta. La opción de tipo limita que tipo de preguntas puede tener respuestas de forma dinámica. Para tipos que permiten respuestas como son la opción múltiple y verdadero/falso existe un botón que nos permitirá añadir y borrar nuevas respuestas de forma dinámica. Las respuestas contienen un botón para borrar el bloque completo.

En la parte superior están nuestros botones para guardar, exportar o ir al banco del que proviene el cuestionario.

The screenshot shows a web-based form for editing a questionnaire. At the top, there are four buttons: 'Guardar' (highlighted in green), 'Exportar Cuestionario con soluciones', 'Exportar Cuestionario sin soluciones', and 'Banco de preguntas original'. Below these are fields for 'Título' (Title) containing 'Cuestionario-2025-05-182025-05-22T20:02:23.634769400' and 'Descripción' (Description) containing 'Cuestionario nº2 Este cuestionario evalúa la comprensión del texto sobre las bases y aplicaciones de la Realidad Aumentada. Cubre temas como plataformas de desarrollo AR, obje...'. The main area is titled 'Preguntas' (Questions) and contains a question box with the text '¿Cuál de los siguientes NO es un componente clave para mostrar AR?'. Below it is a 'Opción Múltiple' (Multiple Choice) section. Under 'Respuestas' (Answers), there are four options: 'Gafas de realidad aumentada' (radio button not selected), 'Dispositivos móviles' (radio button not selected), 'Displays especiales' (radio button not selected), and 'Software de diseño gráfico' (radio button selected). To the right of each answer is a red 'X' button for deletion. At the bottom left is a blue 'Añadir Respuesta' (Add Answer) button, and at the bottom right is a red 'Quitar Pregunta' (Delete Question) button.

Imagen 17 Formulario Cuestionario

Se podrá elegir si se desea exportar el cuestionario PDF incluyendo sus soluciones o no.

La página de creación/edición de un banco de preguntas de forma manual se encuentra en *formNewQuestionPool.html*. Esta página es una página similar en funcionalidad a la vista en *formEditQuestionnaire.html* pero para los bancos de preguntas. A diferencia de esta solamente tiene dos botones para guardar, en dos lugares distintos: en la parte superior e inferior, pero con la misma funcionalidad.

La página *questionnaires.html* es el listado de los cuestionarios, si se selecciona desde la barra de navegación contendrá todos los cuestionarios, si se selecciona desde un banco de preguntas solamente contendrá los cuestionarios que han sido creados a partir de ese banco. Tienen un botón para ver los detalles de un cuestionario, es decir el formulario antes visto, su título, fecha de creación y un botón de borrado.

Para las funciones de creación que se pueden acceder desde la *navbar* o la página inicial. Tenemos el formulario antes mencionado, creación desde texto y creación desde PDF. La creación de un banco de preguntas desde un texto utilizará IA, lo que significa que la estructura no es necesario que sea exacta pero cuanta más información se le otorgue mejor será el resultado. *formQuestionPoolAI.html* es una página con un cuadro de texto y un botón para generarse. Para mejor resultado indicar claramente el título, descripción y las preguntas y sus respuestas correctas. Para mejor resultado es conveniente indicar el tipo de pregunta.

El formulario *pdfToText.html* es la página que solicita al usuario un archivo PDF y un número de preguntas a generar. Enviará el archivo en tipo *multiparty* verifica que sea de tipo PDF.

## 6. Análisis de resultados

La aplicación desarrollada es una solución web **funcional**, diseñada con una configuración mínima que permite su ejecución en **distintos entornos**. Se han realizado diversas **pruebas funcionales** que han involucrado la interacción de un usuario final con la plataforma, principalmente centradas en los flujos de **creación** y **exportación** de cuestionarios. Estos cuestionarios se pueden generar de manera manual, a partir de texto plano o desde documentos PDF, garantizando así la **variedad** en el origen de los contenidos.

Se han observado algunos aspectos relevantes durante la fase de testing que se ha realizado al final de la implementación. El tiempo de respuesta en la generación de preguntas varía considerablemente, y depende en gran medida de la capacidad gráfica y de procesamiento del equipo donde se ejecuta el sistema. Por esta razón, la utilización de modelos de inteligencia artificial alojados en la **nube** podría representar una alternativa interesante para ciertos usuarios, siempre que se cuente con una conexión a Internet estable.

En cuanto a los resultados generados por la inteligencia artificial, se ha verificado que estos pueden variar en función de la calidad y estructura del contenido proporcionado. Si bien la IA logra generar preguntas correctas, en ocasiones pueden aparecer errores como la asignación incorrecta de respuestas o la omisión del tipo de pregunta. Para mitigar estos posibles errores, se han implementado mecanismos de validación en el sistema y se ha puesto especial énfasis en ofrecer una experiencia de edición **cómoda** y **accesible**, de modo que el usuario pueda ajustar manualmente los cuestionarios según sus necesidades en caso de que se haya detectado un error por parte de la IA.

Se ha identificado también que el uso de archivos PDF especialmente largos o la solicitud de un número elevado de preguntas puede incrementar la probabilidad de fallos en la generación. Esto está directamente relacionado con las limitaciones actuales del motor de generación local y con el hardware disponible, pero este problema surge en la mayoría de los modelos IA. El uso de la RAG permite reducir sustancialmente los tiempos de ejecución y permite el uso de textos más largos, sin él, el límite de tokens de un modelo se superaría enormemente.

Además, se ha detectado que algunos modelos como *OpenAI GPT* para Spring ya incluyen en sus bibliotecas herramientas más avanzadas para la gestión de archivos **embebidos** y la generación de información **vectorial**. Estas soluciones podrían servir para optimizar y simplificar componentes como el servicio *EmbeddingService* y la integración de la IA, representando una línea interesante de mejora futura.

Por último, la aplicación no está empaquetada con un único lanzador, la aplicación se debería comprimir en un *jar* o configurar la aplicación para que sea ejecutado desde un *Dockerfile* que se incluya como imagen en el *compose*.

## 7. Conclusiones y trabajo futuro

Tras desarrollar y validar la aplicación **CuestGen**, se puede concluir que los objetivos principales del proyecto han sido conseguidos **satisfactoriamente**. La aplicación permite la **creación, edición y exportación** de cuestionarios generados a partir de bancos de preguntas que se han creado o generado utilizando Inteligencia Artificial.

El sistema demuestra un buen funcionamiento en diversos casos de uso, siendo una experiencia **intuitiva** y **fácil** para el usuario final desde el inicio de la aplicación hasta conseguir un cuestionario en un formato **exportable**. La interfaz es **simple** pero **intuitiva** para los usuarios, permitiendo una navegación **simple**, una edición de bancos y cuestionarios **seccional** que es capaz de enmendar errores de la IA y una exportación cómoda con simplemente la selección de un botón.

Un requisito que originalmente estuvo en el anteproyecto es la habilidad de **seccionalizar** los cuestionarios. Debido a la complejidad, el aprendizaje de esta y la gran variedad de resultados que devolvía la IA no ha sido posible incluir este requisito por el momento, pero con una ampliación futura y un modelo más grande y potente este requisito podría ser incluido.

Como ya se ha comentado, *Ollama* con el modelo de *Gemma3* tiende a tener un porcentaje de fallo alto por la limitación de un modelo pequeño porque estas no son capaces de retener información de un texto grande. Existen modelos en **nube** que requieren de una llave API para acceder a ellos como *Claude*, *Gemini* o *OpenAI*. Pero este formato perdería la propiedad de ser una aplicación completamente local.

En cuanto a la experiencia de usuario, la interfaz podría ser mejorada con la inclusión de ventanas de confirmación usando *JavaScript*, pantallas de carga y una mejor estructura a la hora de mostrar los detalles de un banco de preguntas.

Para facilitar la instalación se podría utilizar un instalador para que el usuario no tenga que configurar el *Nvidia Container Toolkit*. También debiera ser posible empaquetar el programa sin necesidad de utilizar *Docker Engine*.

Como conclusión, CuestGen presenta una solución **funcional, práctica y extensible** para la generación de cuestionarios, y establece una buena base para desarrollos de ampliación más avanzados que integren nuevas tecnologías cloud para un mayor procesamiento y cómputo de la información.

## 8. Bibliografía

Bibliografía:

(Estilo APA 7<sup>a</sup> ed.)

Universidad Autónoma de Madrid (2025, 18 de Marzo). *Citas y elaboración de bibliografía: el plagio y el uso ético de la información: Estilo APA 7<sup>a</sup> ed.*

[https://biblioguias.uam.es/citar/estilo\\_apa\\_7th\\_ed](https://biblioguias.uam.es/citar/estilo_apa_7th_ed)

Software y herramientas:

Apache Software Foundation. (s.f.). Apache Tika

[Biblioteca]. <https://tika.apache.org/>

Bootstrap Core Team (2011) Bootstrap

[Biblioteca] <https://getbootstrap.com/>

Docker Inc. (2013). Docker (versión 27.5.1)

[Software]. <https://www.docker.com/>

Docker Inc. (2014). Docker Compose (versión 2.36.1)

[Software]. <https://docs.docker.com/compose/>

FasterXML. (2012). Jackson (versión 2.14)

[Biblioteca]. <https://github.com/FasterXML/jackson>

iText Group. (2021). iText 7 (versión 7.2)

[Librería]. <https://itextpdf.com/>

MongoDB, Inc. (2024). MongoDB (versión 8.0)

[Base de datos]. <https://www.mongodb.com/>

Oracle Corporation. (2021). Java Platform, Standard Edition (versión 17)

[Plataforma]. <https://www.oracle.com/java/technologies/javase-downloads.html>

Ollama. (2025). Ollama (versión 0.7.0)

[Plataforma]. <https://ollama.com/>

Project Lombok. (2024). Project Lombok (versión 1.18)

[Biblioteca]. <https://projectlombok.org/>

Daniel Fernández (2025). Thymeleaf (3.1.3)

[Motor de plantillas]. <https://www.thymeleaf.org/>

VMware. (2023). Spring Initializr

[Herramienta]. <https://start.spring.io/>

VMware. (2025). Spring Boot (versión 3.5)

[Framework]. <https://spring.io/projects/spring-boot>

Audiovisuales:

Daily Code Buffer (2024, 25 de Agosto). *Java RAG Tutorial (with Local LLMs): AI for your PDFs*

[Vídeo]. Youtube. <https://youtu.be/LdXvJrLyl5o?si=ScJnRfpTJM1VXDhw>

Dan Vega (2024, 2 de Abril). *Spring AI Introduction: Building AI Applications in Java with Spring*

[Vídeo]. Youtube. <https://youtu.be/yvvjT0v3IpY?si=s8bT8AU4Zxy2XkH>

Dan Vega (2024, 19 de Abril). *Getting started with (Retrieval Augmented Generation) RAG in Java & Spring AI*

[Vídeo]. Youtube. <https://youtu.be/4-rG2qsTrAs?si=vGKqpkF5ck5lb25J>

Dan Vega (2024, 1 de Mayo). *Building a Spring Boot Reference Documentation Assistant with Spring AI & GPT-4*

[Vídeo]. Youtube.

[https://youtu.be/ZoPVGrB8iHU?si=2RHmWU\\_OUaODUz8P](https://youtu.be/ZoPVGrB8iHU?si=2RHmWU_OUaODUz8P)

Dan Vega (2024, 30 de Julio). *Getting Started with Ollama, Llama 3.1 and Spring AI*

[Vídeo]. Youtube.

<https://youtu.be/dffEF9ORVUg?si=n7Dpam8blWIqlzW1>

Dan Vega (2024, 22 de Octubre). *Java + RAG: Create an AI-Powered Financial Advisor using Spring AI* 

[Vídeo]. Youtube.

<https://youtu.be/6Pgmr7xMjiY?si=W76hYpVKqY1NpoKA>

Dev + Coffee (2021, 24 de Julio). *Dynamic HTML Content With Javascript*

[Vídeo]. Youtube. <https://youtu.be/EpCgFY-o9EI?si=amlfdxFmyuZ-JhpE>

freeCodeCamp.org (2018, 10 de Diciembre). *Learn JavaScript - Full Course for Beginners*

[Vídeo]. Youtube.

<https://youtu.be/PkZNo7MFNFg?si=ZCktCDR3PsYV7Qhx>

freeCodeCamp.org (2023, 19 de Enero). *Full Stack Development with Java Spring Boot, React, and MongoDB – Full Course*

[Vídeo]. Youtube.

<https://youtu.be/5PdEmeopJVQ?si=mo8Qev48bcIMzCY9>

Matt Williams (2024, 27 de Febrero). *How to run Ollama on Docker*

[Vídeo]. Youtube. <https://youtu.be/ZoxJcPkjirs?si=Ws9TRKqEbLu9PLm1>

Mervin Praison (2024, 23 de Febrero). *Ollama Embedding: How to Feed Data to AI for Better Response?*

[Vídeo]. Youtube.

<https://youtu.be/jENqvjkwmw?si=nRzNpvEwUS2mFOf7>

Programming Techie (2025, 19 de Marzo). *Spring Boot AI RAG (Retrieval Augmented Generation) Tutorial*

[Vídeo]. Youtube.

[https://youtu.be/SmDZV9\\_nAdY?si=pkMFCZsnuzlFu2OO](https://youtu.be/SmDZV9_nAdY?si=pkMFCZsnuzlFu2OO)

Prompt Engineer (2024, 7 de Diciembre). *Unlocking the Power of Ollama's Structured JSON Output*

[Vídeo]. Youtube.

<https://youtu.be/nz5c9d1fJZ0?si=EHvBS0gAvLGvNa8K>

Simplifying Tech (2023, 3 de Marzo). *How To Export Data To PDF Using Spring Boot App | Simplest way without using DB.*

[Vídeo]. Youtube. <https://youtu.be/QTysZf9Bgbk?si=sAG8WA40dtOtByzy>

Telusko (2025, 1 de Abril). *Spring AI Tutorial with OpenAI, Anthropic and Ollama*

[Vídeo]. Youtube. [https://youtu.be/mLsx\\_VInHI4?si=yTBOVSszls-y7Yhd](https://youtu.be/mLsx_VInHI4?si=yTBOVSszls-y7Yhd)

Telusko (2022, 11 de Octubre). *Java Spring Boot Mongodb Full Project*

[Vídeo]. Youtube. [https://youtu.be/kYiLzliHVY8?si=6dIIGuJCrl\\_EHH4m](https://youtu.be/kYiLzliHVY8?si=6dIIGuJCrl_EHH4m)

### Páginas Web:

Arquitectura Java (2022, 15 de Enero). *Spring @Service , usando el patrón Servicio*

<https://www.arquitecturajava.com/spring-service-usando-el-patron-servicio/>

Baeldung (2024, 11 de Mayo). Spring Data MongoDB - Configure Connection

<https://www.baeldung.com/spring-data-mongodb-connection>

Geeks for Geeks (2022, 19 de Diciembre). Shuffle a given array using Fisher-Yates shuffle Algorithm

<https://www.geeksforgeeks.org/shuffle-a-given-array-using-fisher-yates-shuffle-algorithm/>

Oracle (2007, Marzo). *Java SE Application Design with MVC*

<https://www.oracle.com/technical-resources/articles/java/java-se-app-design-with-mvc.html>

Nvidia Container Toolkit (s.f.). *Installing the NVIDIA Container Toolkit*

<https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/install-guide.html>

Ollama (2024, 8 de Abril). *Embedding models*

<https://ollama.com/blog/embedding-models>

Spring (s.f.). *Custom Conversions*

<https://docs.spring.io/spring-data/mongodb/reference/mongodb/mapping/custom-conversions.html>

Spring (s.f.). *Ollama Chat*

<https://docs.spring.io/spring-ai/reference/api/chat/ollama-chat.html>

Spring (s.f.). *Ollama embeddings*

<https://docs.spring.io/spring-ai/reference/api/embeddings/ollama-embeddings.html>

Spring (s.f.). *Retrieval Augmented Generation*

<https://docs.spring.io/spring-ai/reference/api/retrieval-augmented-generation.html>

Spring (s.f.). *Structured Output Converter*

<https://docs.spring.io/spring-ai/reference/api/structured-output-converter.html>

StackOverflow (s.f.). *When to implement WebMvcConfigurer to configure Spring MVC*

<https://stackoverflow.com/questions/56833181/when-to-implement-webmvcconfigurer-to-configure-spring-mvc>

W3Schools (s.f.). JavaScript Tutorial

<https://www.w3schools.com/js/>