Universidad Veracruzana

Facultad de Contaduría y Administración

Licenciatura: Ingeniería en Software.

Tema: Desarrollo de una API con Spring Boot

Experiencia Educativa: Desarrollo de aplicaciones

Docente: José Antonio Vergara Camacho

Alumno: Santes Primo José Francisco

Coatzacoalcos, Veracruz a 18 de Junio del 2024

Definición y Especificación de Requerimientos

El sistema debe cumplir con los siguientes requerimientos:

Requerimientos Funcionales:

1. Permitir la gestión de una colección de canciones, artistas y álbumes a través de una API REST.
2. Permitir la consulta de canciones, artistas y álbumes por diferentes criterios, como ID, título y nombre de artista.
3. Permitir la creación, actualización y eliminación de canciones, artistas y álbumes.
4. Proporcionar documentación clara y completa de la API REST para facilitar su uso por parte de desarrolladores externos.
5. Garantizar la integridad y consistencia de los datos almacenados en el cluster de MongoDB.

Requerimientos No Funcionales:

1. La API REST debe ser fácil de usar e intuitiva para los desarrolladores.
2. El sistema debe ser escalable para manejar un gran volumen de datos y usuarios concurrentes.
3. La API REST debe ser segura, protegiendo los datos de forma adecuada y evitando posibles vulnerabilidades.
4. El sistema debe ser eficiente en cuanto a uso de recursos, minimizando el consumo de CPU, memoria y ancho de banda.
5. El tiempo de respuesta de la API REST debe ser rápido, garantizando una experiencia fluida para los usuarios.

Restricciones:

1. El sistema debe utilizar Java 22 como lenguaje de programación.
2. Se debe utilizar Spring Boot como framework para el desarrollo de la API REST.
3. La base de datos debe ser MongoDB y se debe utilizar un cluster de MongoDB para garantizar la disponibilidad y la redundancia de los datos.
4. Se debe utilizar Docker para ejecutar el sistema en contenedores.

Arquitectura del Sistema

El sistema sigue una arquitectura basada en microservicios, lo que permite una mayor flexibilidad, escalabilidad y mantenimiento. Se sigue una estructura típica de aplicación basada en Spring Boot, donde se utilizan las siguientes capas y componentes:

Entidades

Representan las estructuras de datos principales de la aplicación, como Song, Artist y Album. Estas entidades están mapeadas a las colecciones correspondientes en la base de datos MongoDB.

DTOs (Data Transfer Objects):

Se utilizan para transferir datos entre la capa de controladores y la capa de servicios. Los DTOs se utilizan para evitar la exposición de la estructura interna de las entidades y proporcionar una representación más adecuada de los datos para la API REST.

Controladores (Controllers):

Son responsables de manejar las peticiones HTTP entrantes y de llamar a los métodos correspondientes en la capa de servicios. Los controladores están anotados con @RestController y definen los endpoints de la API REST.

Servicios (Services):

Contienen la lógica de negocio de la aplicación y se encargan de procesar las peticiones recibidas desde los controladores. Los servicios interactúan con los repositorios para realizar operaciones de lectura y escritura en la base de datos.

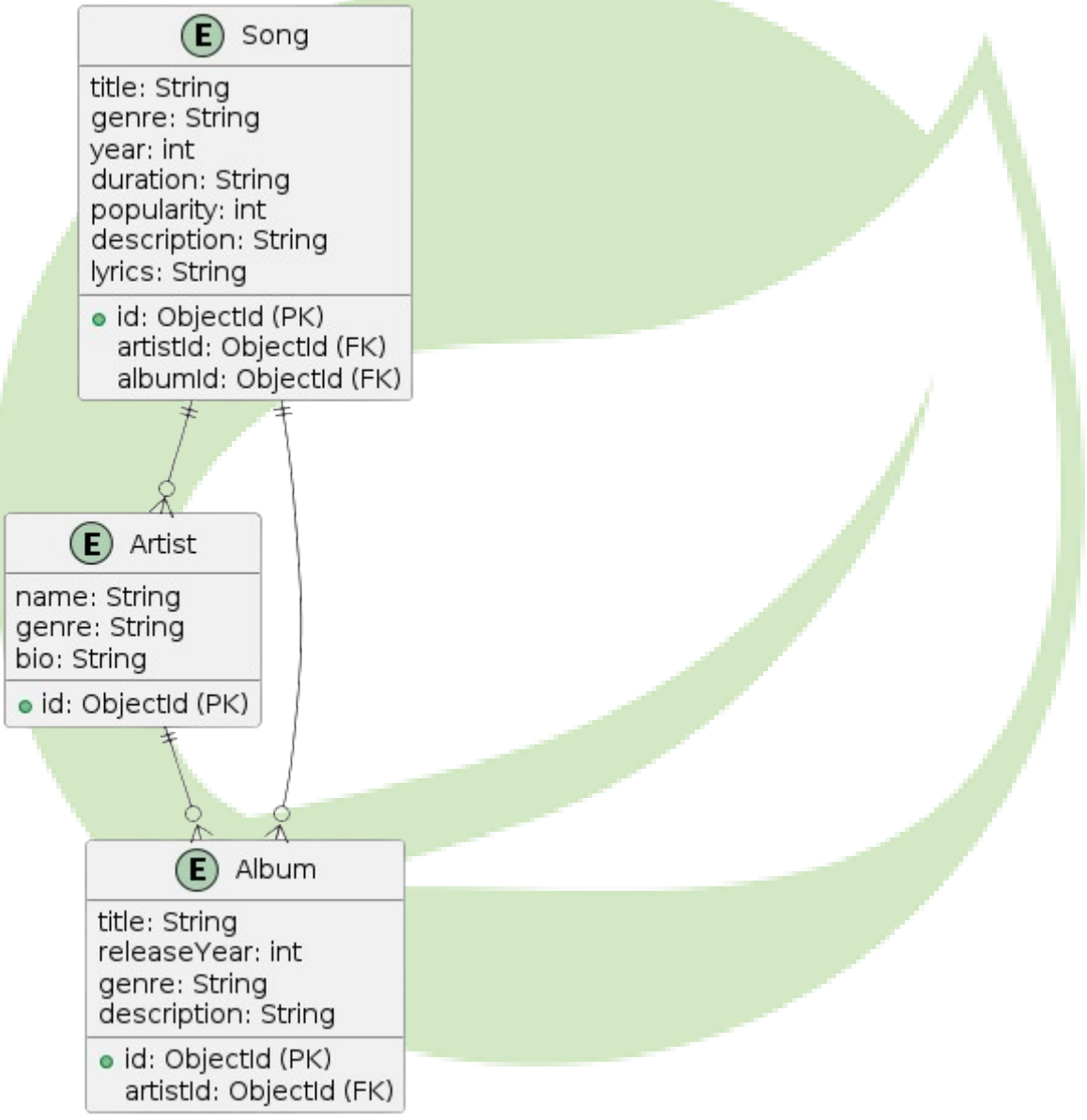
Repositorios (Repositories):

Se utilizan para interactuar con la base de datos MongoDB y realizar operaciones de lectura y escritura en las colecciones correspondientes. Los repositorios extienden la interfaz MongoRepository de Spring Data MongoDB.

Configuración de Spring:

Se utiliza configuración de Spring para definir los beans y las dependencias de la aplicación, así como para configurar aspectos como la conexión a la base de datos MongoDB y la generación de la documentación de la API con Swagger.

Modelo de datos



Descripción de procesos y servicios ofrecidos

Obtener todas las canciones:

- **Descripción**: Permite obtener una lista de todas las canciones almacenadas en la base de datos.
- **Endpoint**: GET /songs
- **Parámetros**: Ninguno
- **Respuesta**: Devuelve una lista de objetos JSON, cada uno representando una canción con sus atributos.

Obtener una canción por su ID:

- **Descripción**: Permite obtener una canción específica por su ID.
- **Endpoint**: GET /songs/{id}
- **Parámetros**: id (ID único de la canción)
- **Respuesta**: Devuelve un objeto JSON con los detalles de la canción correspondiente al ID proporcionado.

Crear una nueva canción:

- **Descripción**: Permite crear una nueva canción y almacenarla en la base de datos.
- **Endpoint**: POST /songs/save
- **Parámetros**: Objeto JSON con los atributos de la canción a crear.
- **Respuesta**: Devuelve un objeto JSON con los detalles de la canción creada, incluido su nuevo ID asignado por la base de datos.

Actualizar una canción por su ID:

- **Descripción**: Permite actualizar los atributos de una canción existente por su ID.
- **Endpoint**: PUT /songs/update/{id}
- **Parámetros**: id (ID único de la canción) y objeto JSON con los nuevos atributos de la canción.
- **Respuesta**: Devuelve un objeto JSON con los detalles de la canción actualizada.

Eliminar una canción por su ID:

- **Descripción**: Permite eliminar una canción existente por su ID.

- **Endpoint**: DELETE /songs/delete/{id}
- **Parámetros**: id (ID único de la canción)
- **Respuesta**: Devuelve un mensaje de confirmación indicando que la canción ha sido eliminada correctamente.

Obtener todos los artistas:

- **Descripción**: Permite obtener una lista de todos los artistas almacenados en la base de datos.
- **Endpoint**: GET /artists
- **Parámetros**: Ninguno
- **Respuesta**: Devuelve una lista de objetos JSON, cada uno representando un artista con sus atributos.

Obtener un artista por su ID:

- **Descripción**: Permite obtener un artista específico por su ID.
- **Endpoint**: GET /artists/{id}
- **Parámetros**: id (ID único del artista)
- **Respuesta**: Devuelve un objeto JSON con los detalles del artista correspondiente al ID proporcionado.

Crear un nuevo artista:

- **Descripción**: Permite crear un nuevo artista y almacenarlo en la base de datos.
- **Endpoint**: POST /artists/save
- **Parámetros**: Objeto JSON con los atributos del artista a crear.
- **Respuesta**: Devuelve un objeto JSON con los detalles del artista creado, incluido su nuevo ID asignado por la base de datos.

Actualizar un artista por su ID:

- **Descripción**: Permite actualizar los atributos de un artista existente por su ID.
- **Endpoint**: PUT /artists/update/{id}
- **Parámetros**: id (ID único del artista) y objeto JSON con los nuevos atributos del artista.
- **Respuesta**: Devuelve un objeto JSON con los detalles del artista actualizado.

Eliminar un artista por su ID:

- **Descripción**: Permite eliminar un artista existente por su ID.
- **Endpoint**: DELETE /artists/delete/{id}
- **Parámetros**: id (ID único del artista)
- **Respuesta**: Devuelve un mensaje de confirmación indicando que el artista ha sido eliminado correctamente.

Obtener todos los álbumes:

- **Descripción**: Permite obtener una lista de todos los álbumes almacenados en la base de datos.
- **Endpoint**: GET /albums
- **Parámetros**: Ninguno
- **Respuesta**: Devuelve una lista de objetos JSON, cada uno representando un álbum con sus atributos.

Obtener un álbum por su ID:

- **Descripción**: Permite obtener un álbum específico por su ID.
- **Endpoint**: GET /albums/{id}
- **Parámetros**: id (ID único del álbum)
- **Respuesta**: Devuelve un objeto JSON con los detalles del álbum correspondiente al ID proporcionado.

Crear un nuevo álbum:

- **Descripción**: Permite crear un nuevo álbum y almacenarlo en la base de datos.
- **Endpoint**: POST /albums/save
- **Parámetros**: Objeto JSON con los atributos del álbum a crear.
- **Respuesta**: Devuelve un objeto JSON con los detalles del álbum creado, incluido su nuevo ID asignado por la base de datos.

Actualizar un álbum por su ID:

- **Descripción**: Permite actualizar los atributos de un álbum existente por su ID.

- **Endpoint**: PUT /albums/update/{id}
- **Parámetros**: id (ID único del álbum) y objeto JSON con los nuevos atributos del álbum.
- **Respuesta**: Devuelve un objeto JSON con los detalles del álbum actualizado.

Eliminar un álbum por su ID:

- **Descripción**: Permite eliminar un álbum existente por su ID.
- **Endpoint**: DELETE /albums/delete/{id}
- **Parámetros**: id (ID único del álbum)
- **Respuesta**: Devuelve un mensaje de confirmación indicando que el álbum ha sido eliminado correctamente.

Documentación técnica del API REST

Propósito y Descripción

El API REST desarrollado con Spring Boot proporciona un conjunto de endpoints para la gestión de canciones, artistas y álbumes musicales. Utiliza MongoDB como base de datos y ofrece operaciones CRUD para cada entidad. La API sigue un diseño basado en microservicios, lo que permite una mayor flexibilidad y escalabilidad.

Endpoints Disponibles

- **Canciones (Songs)**
  - GET /songs: Obtiene todas las canciones.
    - Propósito: Obtener una lista de todas las canciones almacenadas en la base de datos.
  - GET /songs/{id}: Obtiene una canción por su ID.
    - Propósito: Obtener los detalles de una canción específica por su ID.
  - POST /songs/save: Crea una nueva canción.
    - Propósito: Crear una nueva canción y almacenarla en la base de datos.
  - PUT /songs/update/{id}: Actualiza una canción por su ID.
    - Propósito: Actualizar los atributos de una canción existente por su ID.
  - DELETE /songs/delete/{id}: Elimina una canción por su ID.
    - Propósito: Eliminar una canción existente por su ID.
- **Artistas (Artists)**
  - GET /artists: Obtiene todos los artistas.
    - Propósito: Obtener una lista de todos los artistas almacenados en la base de datos.
  - GET /artists/{id}: Obtiene un artista por su ID.
    - Propósito: Obtener los detalles de un artista específico por su ID.
  - POST /artists/save: Crea un nuevo artista.
    - Propósito: Crear un nuevo artista y almacenarlo en la base de datos.
  - PUT /artists/update/{id}: Actualiza un artista por su ID.
    - Propósito: Actualizar los atributos de un artista existente por su ID.

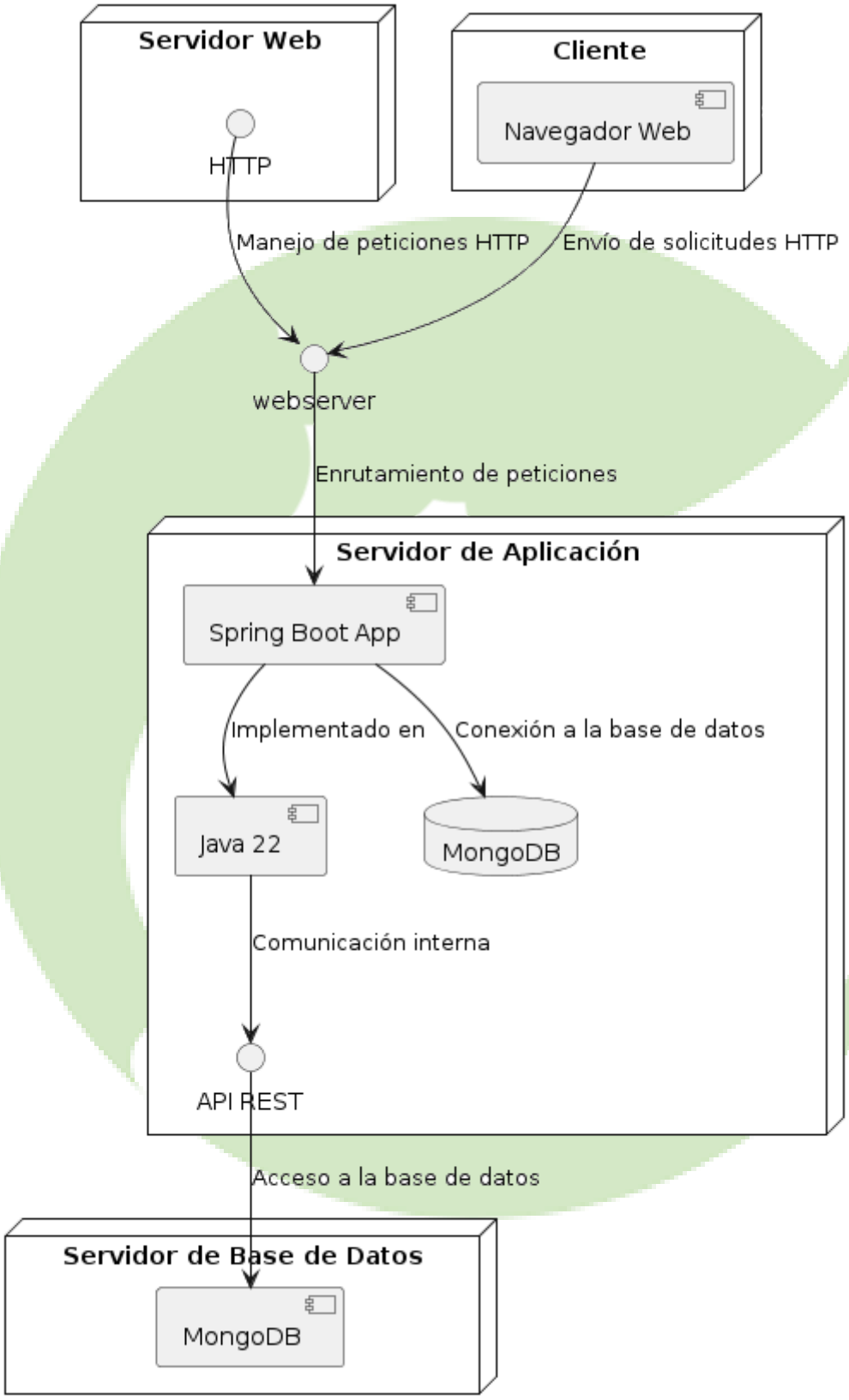
- DELETE /artists/delete/{id}: Elimina un artista por su ID.
  - Propósito: Eliminar un artista existente por su ID.

- **Álbumes (Albums)**
  - GET /albums: Obtiene todos los álbumes.
    - Propósito: Obtener una lista de todos los álbumes almacenados en la base de datos.
  - GET /albums/{id}: Obtiene un álbum por su ID.
    - Propósito: Obtener los detalles de un álbum específico por su ID.
  - POST /albums/save: Crea un nuevo álbum.
    - Propósito: Crear un nuevo álbum y almacenarlo en la base de datos.
  - PUT /albums/update/{id}: Actualiza un álbum por su ID.
    - Propósito: Actualizar los atributos de un álbum existente por su ID.
  - DELETE /albums/delete/{id}: Elimina un álbum por su ID.
    - Propósito: Eliminar un álbum existente por su ID.

Aspectos Técnicos Relevantes

- **Spring Boot**: Utilizado como framework para el desarrollo de la aplicación, facilitando la creación de servicios RESTful.
- **Lombok**: Utilizado para la generación automática de getters, setters y constructores, reduciendo la cantidad de código boilerplate.
- **MapStruct**: Utilizado para mapear entre entidades y DTOs, facilitando la conversión de objetos para la comunicación entre capas.
- **Mockito**: Utilizado para realizar pruebas unitarias y simular comportamientos de componentes.
- **MongoDB**: Utilizado como base de datos NoSQL para almacenar la información de canciones, artistas y álbumes.
- **Swagger/OpenAPI**: Utilizado para la generación automática de la documentación del API, facilitando su comprensión y uso por parte de los desarrolladores.

Diagrama de Despliegue



Herramientas utilizadas

- Java 22: Lenguaje de programación utilizado para desarrollar la aplicación.
- Spring Boot: Framework utilizado para desarrollar la aplicación API REST.
- MongoDB: Base de datos NoSQL utilizada para almacenar los datos de la aplicación.
- Docker: Plataforma utilizada para contenerizar la aplicación y facilitar su despliegue.
- Maven: Herramienta de gestión de proyectos utilizada para construir y gestionar las dependencias del proyecto.
- Postman: Herramienta utilizada para probar y documentar la API REST.
- Git y GitHub: Utilizados para control de versiones y colaboración en el desarrollo del proyecto.
- IntelliJ IDEA: Entorno de desarrollo integrado utilizado para desarrollar la aplicación.
- Jenkins: Herramienta de integración continua utilizada para automatizar el proceso de construcción, prueba e implementación de la aplicación.
- PlantUML: Utilizado para crear diagramas UML que representan la arquitectura y diseño del sistema.
- ChatGPT: Plataforma utilizada para interactuar con un modelo de lenguaje natural entrenado por OpenAI.
- MongoCompass: Herramienta de interfaz gráfica de usuario utilizada para interactuar con bases de datos MongoDB.