UNIVERSIDAD DE LAS FUERZAS ARMADAS (ESPE)

SEGUNDO SEMESTRE

CARRERAS TECNICAS

DEPARTAMENTO DE CIENCIAS EXACTAS

SEGUNDO PARCIAL

“Aplicación de Gestión de Tareas con Interfaz Gráfica y Base de Datos NoSQL”

AUTORES:

- Guerrero Steven.
- Samir Samande.
- Gabriel Sislema.

NRC:

1322

DOCENTE:

LUIS ENRIQUE JARAMILLO MONTAÑO

INTRODUCCION

En la actualidad, la gestión eficiente de tareas es fundamental tanto en entornos personales como profesionales. Con el avance de la tecnología, las aplicaciones informáticas han facilitado la organización y seguimiento de actividades, permitiendo a los usuarios optimizar su tiempo y aumentar su productividad.

Este proyecto tiene como objetivo desarrollar una aplicación de gestión de tareas con una interfaz gráfica intuitiva y una base de datos NoSQL (MongoDB) para el almacenamiento y recuperación eficiente de la información. La implementación se basará en los principios SOLID, garantizando un código modular, escalable y fácil de mantener.

A través de esta aplicación, los usuarios podrán crear, editar y eliminar tareas, así como asignarles estados y prioridades. Gracias a la flexibilidad de MongoDB, se podrá gestionar la información sin la rigidez de una base de datos relacional, lo que permitirá mayor adaptabilidad a las necesidades del usuario.

Este documento detalla el desarrollo del proyecto, abordando aspectos clave como el diseño de la interfaz, la arquitectura del software y la implementación de la base de datos, con el objetivo de proporcionar una solución práctica y funcional para la gestión de tareas.

1. OBJETIVOS

1.1 OBJETIVO GENERAL

Desarrollar una aplicación de gestión de tareas con una interfaz gráfica intuitiva y una base de datos NoSQL, aplicando los principios SOLID para garantizar un diseño modular, escalable y mantenible

1.2 OBJETIVOS ESPECÍFICO

Diseñar un modelo de base de datos NoSQL que permita almacenar, recuperar y gestionar tareas de manera eficiente.

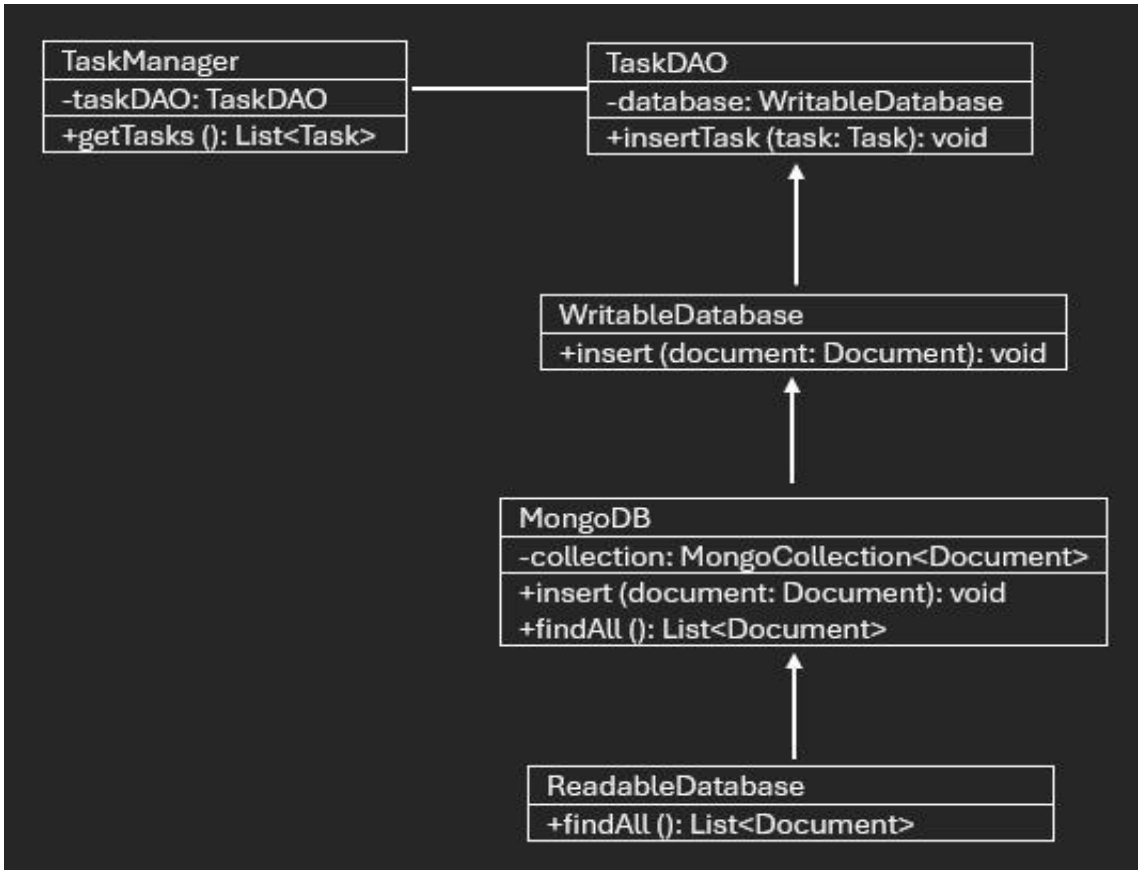
Implementar una interfaz gráfica amigable y accesible que facilite la interacción del usuario con la aplicación.

Aplicar los principios SOLID en la arquitectura del software para mejorar la mantenibilidad y escalabilidad del código.

Integrar la base de datos NoSQL con la interfaz gráfica para permitir la manipulación dinámica de las tareas.

2. MARCO TEÓRICO

Diagrama UML



1. TaskManager

- **Atributos:**
  - **taskDAO:** Una referencia al objeto TaskDAO.
- **Métodos:**
  - **getTasks():** Devuelve una lista de tareas (List<Task>).

2. TaskDAO

- **Atributos:**
  - **database:** Una referencia a un objeto WritableDatabase.
- **Métodos:**
  - **insertTask(task: Task):** Inserta una tarea en la base de datos**.**

3. WritableDatabase

- **Métodos:**
    - **insert(document: Document):** Inserta un documento en la base de datos**.**

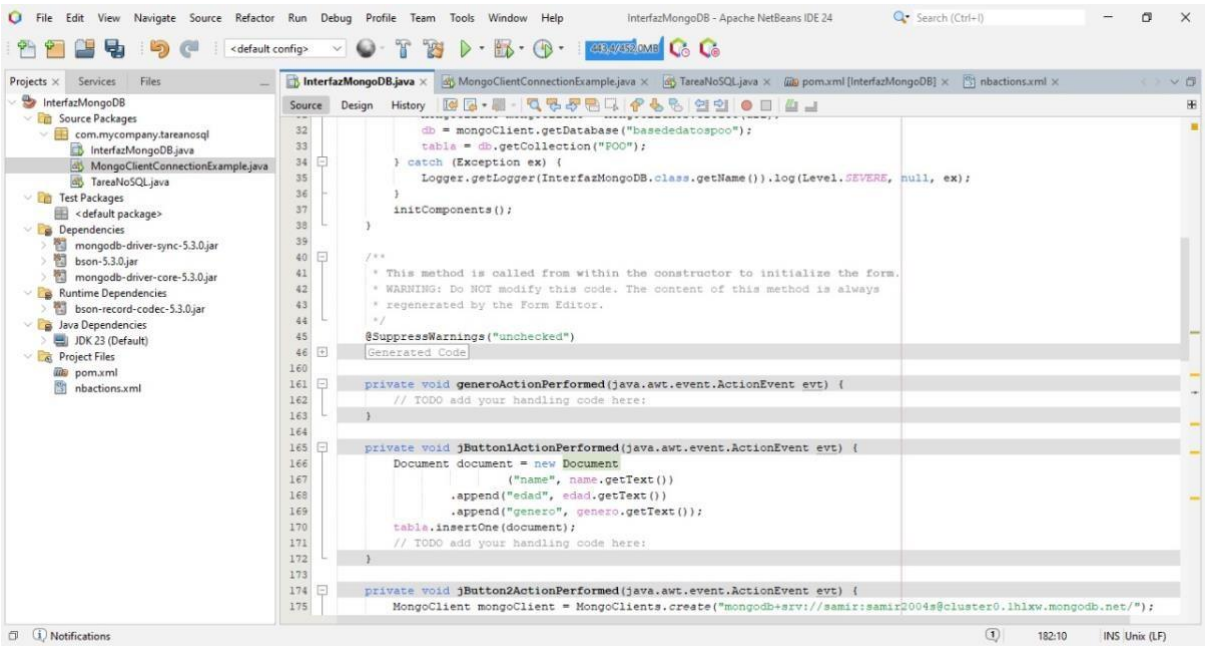
4. MongoDB (hereda tanto de WritableDatabase como de ReadableDatabase)

- **Atributos:**
    - **collection**: Un objeto MongoCollection<Document> que almacena los documentos.
- **Métodos:**
    - **insert(document: Document):** Inserta un documento en la colección de MongoDB.
    - **findAll():** Recupera todos los documentos de la colección como una lista (List<Document>).

5. ReadableDatabase

- **Métodos:**
    - **findAll():** Recupera todos los documentos como una lista (List<Document>).

CODIGO



Estructura del Proyecto

- **Proyecto:** "InterfazMongoDB".

Paquete de código fuente: com.mycompany.tareanosql, que contiene tres archivos

- **InterfazMongoDB.java:** Parece ser la clase principal de la interfaz gráfica.
- **MongoClientConnectionExample.java:** Maneja la conexión con MongoDB.
- **TareaNoSQL.java:** Contiene lógica relacionada con la gestión de datos en la base NoSQL.

Dependencias: Se incluyen varias librerías necesarias para trabajar con MongoDB

- mongodb-driver-sync-5.3.0.jar
- bson-5.3.0.jar

- mongodb-driver-core-5.3.0.jar
- bson-record-codec-5.3.0.jar
- Dependencias de Java: Se está utilizando JDK 23.

Explicación del Código en la Imagen

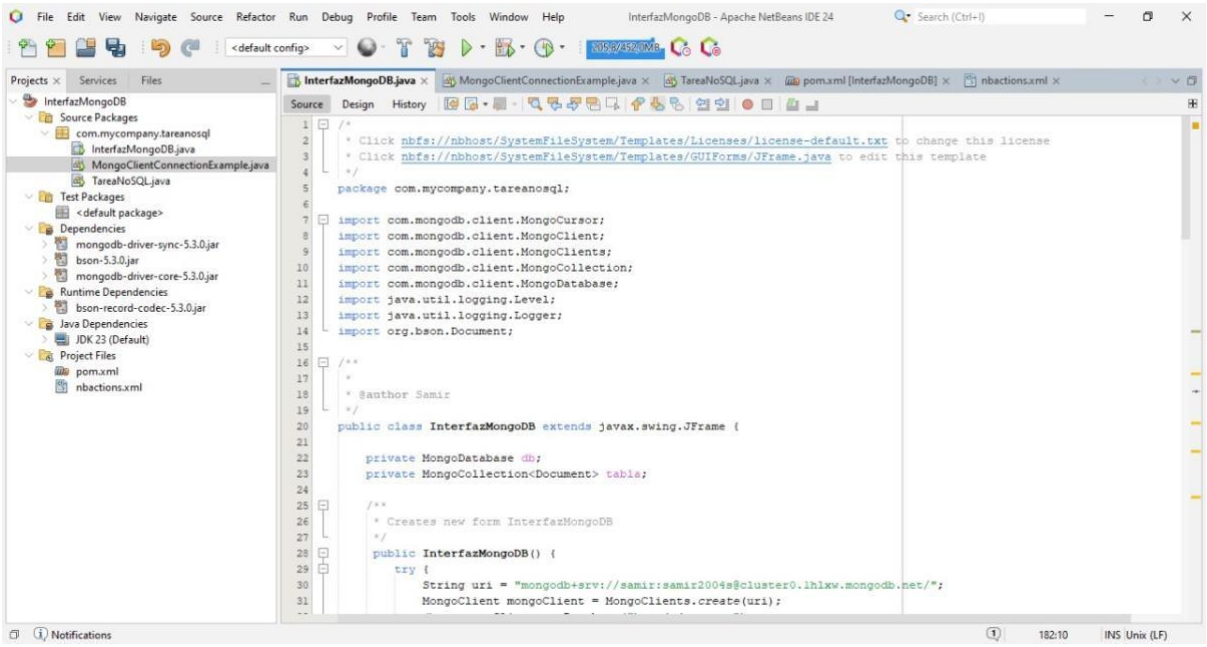
1. **Conexión con MongoDB Atlas**
   - **La línea MongoClient mongoClient =** MongoClients.create("mongodb+srv://samir:samir2004s@cluster0.lhlkw.mon godb.net/"); muestra que la aplicación se conecta a una base de datos MongoDB en la nube mediante MongoDB Atlas.
2. **Inserción de Datos en la Base NoSQL**
   - **En el método jButtonActionPerformed(evt), se crea un objeto Document con tres atributos:**
     - **"name":** Captura el texto ingresado en un campo de texto (name.getText()).
     - **"edad":** Captura la edad (edad.getText()).
     - **"genero":** Captura el género (genero.getText()).
   - Luego, este documento se inserta en la base de datos con tabla.insertOne(document);.

3. **Interfaz Gráfica y Eventos**

    • Se observa que la aplicación tiene botones (jButtonActionPerformed) y eventos (ActionEvent) que permiten que, al hacer clic en un botón, los datos ingresados se guarden en MongoDB.

4. **Uso de Dependencias de MongoDB**

    • Se están utilizando las librerías necesarias (mongodb-driver-sync-5.3.0.jar, bson-5.3.0.jar) para interactuar con MongoDB desde Java.



(MongoDatabase)

- Es una instancia de la base de datos de MongoDB.
- Se utiliza para acceder a una base de datos específica dentro del servidor de MongoDB.

Tabla (MongoCollection<Document>)

- Representa una colección dentro de la base de datos MongoDB.

- Se usa para realizar operaciones CRUD (Create, Read, Update, Delete) en una colección de documentos dentro de MongoDB.

Uri (String, dentro del constructor)

- Contiene la URL de conexión al servidor de MongoDB.

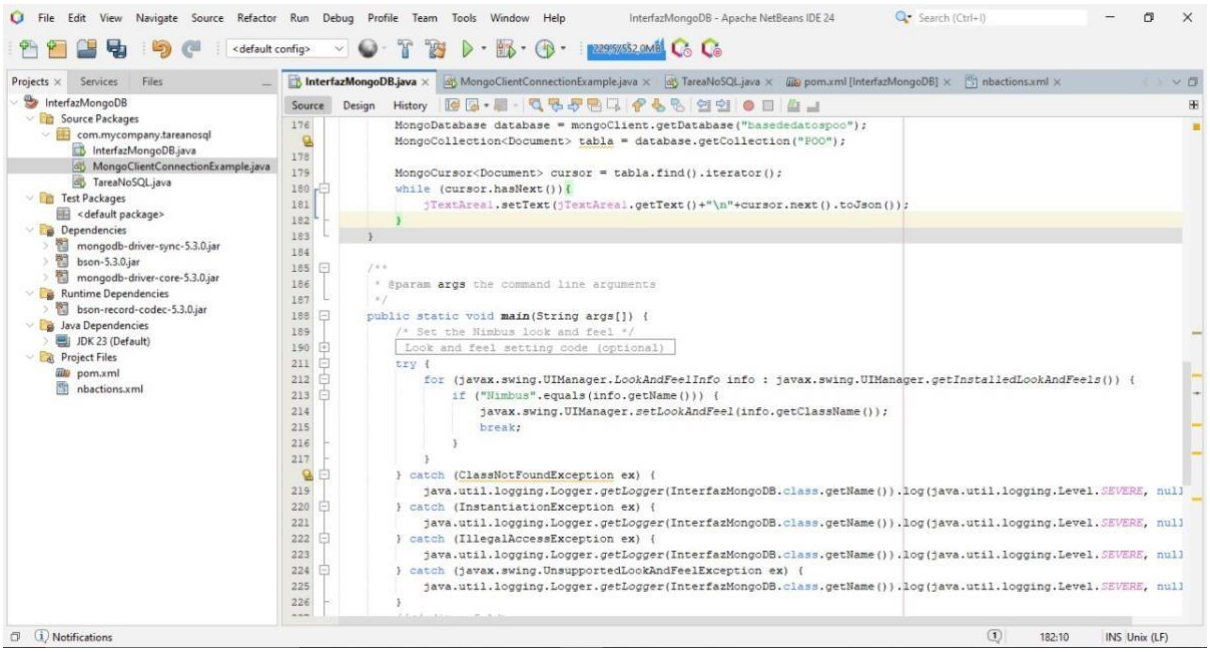
- Es necesario para establecer la conexión con la base de datos alojada en MongoDB

  Atlas u otro servidor.

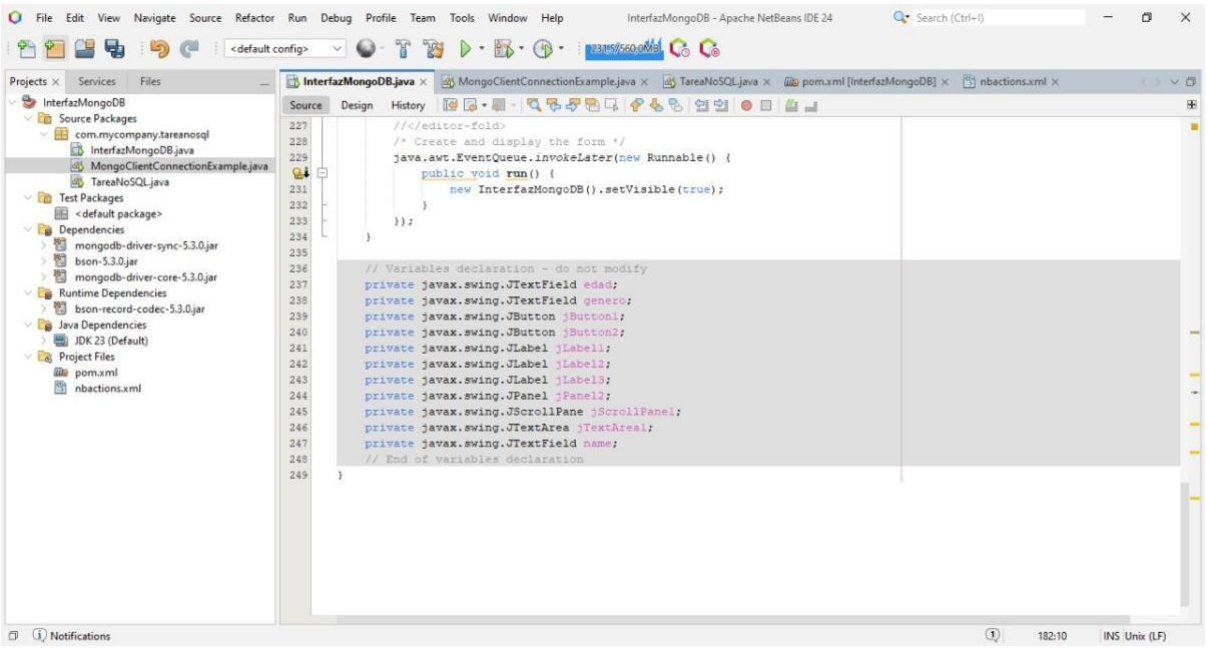
MongoClient (MongoClient, dentro del constructor)

- Es el cliente que se conecta a MongoDB utilizando la uri.

- Permite la comunicación entre la aplicación Java y el servidor de MongoDB



1. **database (MongoDatabase)**
    - Obtiene una referencia a la base de datos llamada "basedatospo".
    - Se usa para acceder a las colecciones dentro de esa base de datos.
2. **tabla (MongoCollection<Document>)**
    - Hace referencia a la colección llamada "FOO" dentro de la base de datos "basedatospo".
    - Permite realizar operaciones como inserciones, consultas, actualizaciones y eliminaciones en esta colección.
3. **cursor (MongoCursor<Document>)**
    - Obtiene un iterador sobre los documentos de la colección "FOO".
    - Se usa para recorrer todos los documentos de la colección uno por uno.
4. **while (cursor.hasNext())**
    - Verifica si hay más documentos en el cursor.
    - Mientras haya documentos, sigue iterando.
5. **jTextArea1.setText(jTextArea1.getText() + "\n" + cursor.next().toJson());**
    - Obtiene el siguiente documento del cursor y lo convierte a formato JSON.
    - Lo agrega al contenido actual del JTextArea, permitiendo que los datos de MongoDB se muestren en la interfaz gráfica.



Campos de Entrada (JTextField)

1. **edad (JTextField)**

- Campo de texto donde el usuario puede ingresar su edad.

2. **genero (JTextField)**

- Campo de texto donde el usuario puede ingresar su género.

3. **name (JTextField)**

- Campo de texto donde el usuario puede ingresar su nombre.

Botones (JButton)

4. **jButton1 (JButton)**

- Botón que probablemente se usa para ejecutar una acción (como guardar en la base de datos o consultar información).

5. **jButton2 (JButton)**

- Otro botón con una función distinta (puede ser para limpiar campos o cerrar la ventana).

Etiquetas de Texto (JLabel)

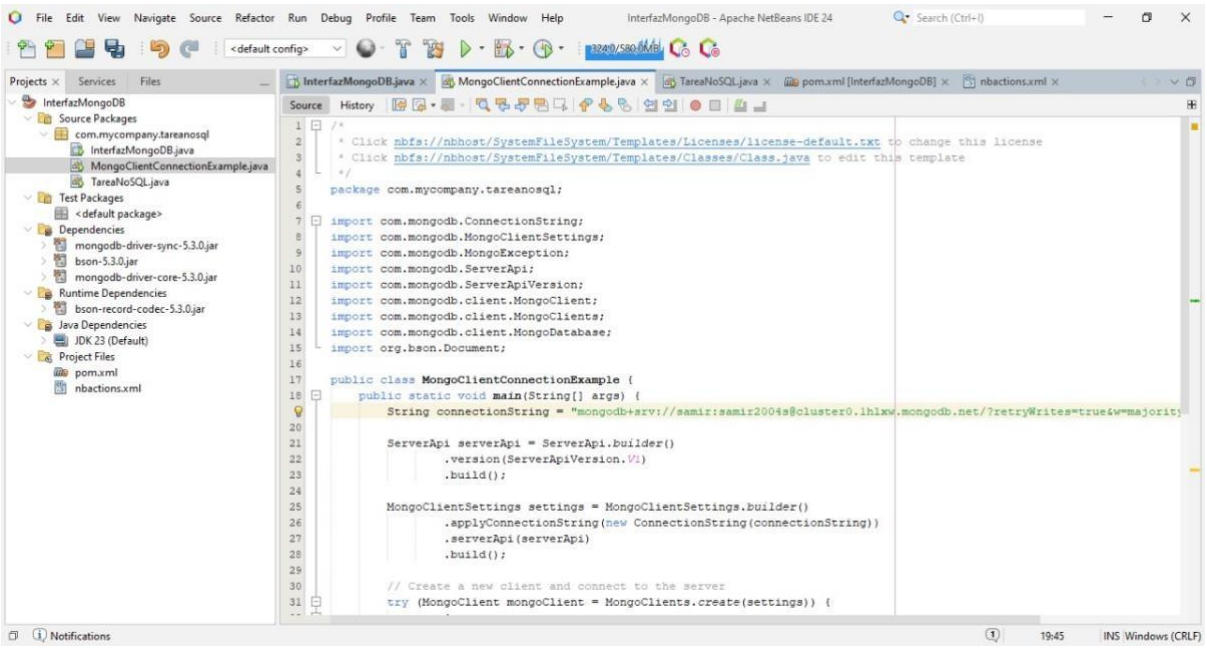
6. **jLabel1 (JLabel)**
    - Posiblemente muestra el texto "Edad" al lado del campo de edad.
7. **jLabel2 (JLabel)**
    - Posiblemente muestra el texto "Género" al lado del campo de género.
8. **jLabel3 (JLabel)**
    - Posiblemente muestra el texto "Nombre" al lado del campo de nombre.

Panel y Área de Desplazamiento

9. **jPanel2 (JPanel)**
    - Un panel contenedor que agrupa los elementos de la interfaz.
10. **jScrollPanel (JScrollPane)**
    - Panel de desplazamiento que permite desplazarse por jTextArea1 si su contenido es muy grande.

Área de Texto (JTextArea)

11. **jTextArea1 (JTextArea)**
    - Un área de texto donde se muestra información (posiblemente los datos extraídos de MongoDB).



String connectionString

- Contiene la cadena de conexión a la base de datos MongoDB.
- Usa el formato mongodb+srv://usuario:contraseña@cluster/... para conectarse a un clúster MongoDB alojado en la nube.
- En este caso, el usuario y la contraseña están en la URL.

ServerApi serverApi

- Se encarga de configurar la versión de la API de MongoDB.
- Se inicializa con ServerApi.builder() y se le asigna la versión ServerApiVersion.V1.
- Especifica cómo la aplicación se comunica con el servidor MongoDB.

MongoClientSettings settings

- Configura las opciones de conexión del cliente MongoDB.

- Usa el método MongoClientSettings.builder() para definir la configuración.
- Aplica la cadena de conexión (connectionString).
- Asocia la configuración de la API del servidor (serverApi).

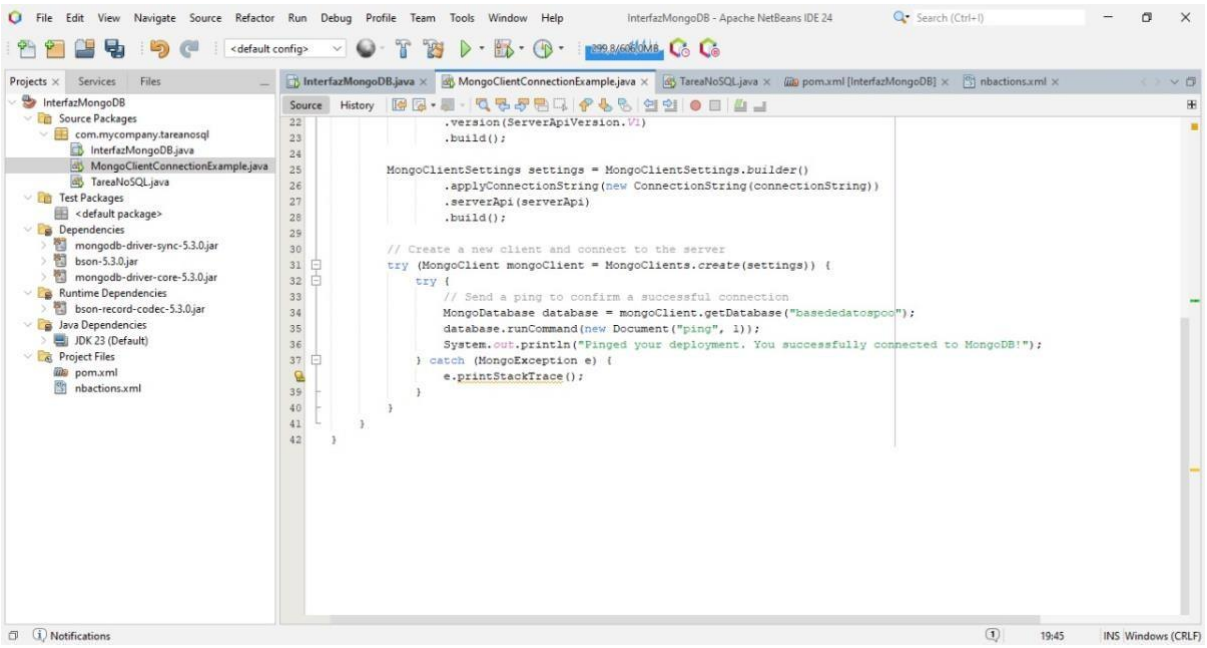
MongoClient mongoClient

- Es la instancia principal del cliente MongoDB.
- Se crea usando MongoClients.create(settings), lo que aplica la configuración establecida anteriormente.
- Se utiliza para conectarse al servidor MongoDB y realizar operaciones en la base de datos. **Variables connectionString** :Almacena la cadena de conexión a la base de datos MongoDB. **serverApi** : Configura la versión de la API del servidor MongoDB.

Settings:Define la configuración del cliente, incluyendo la conexión y la API del servidor. **mongoClient**:Crea un cliente MongoDB para establecer la conexión con el servidor.

Database: Representa la base de datos a la que se accede dentro de MongoDB.

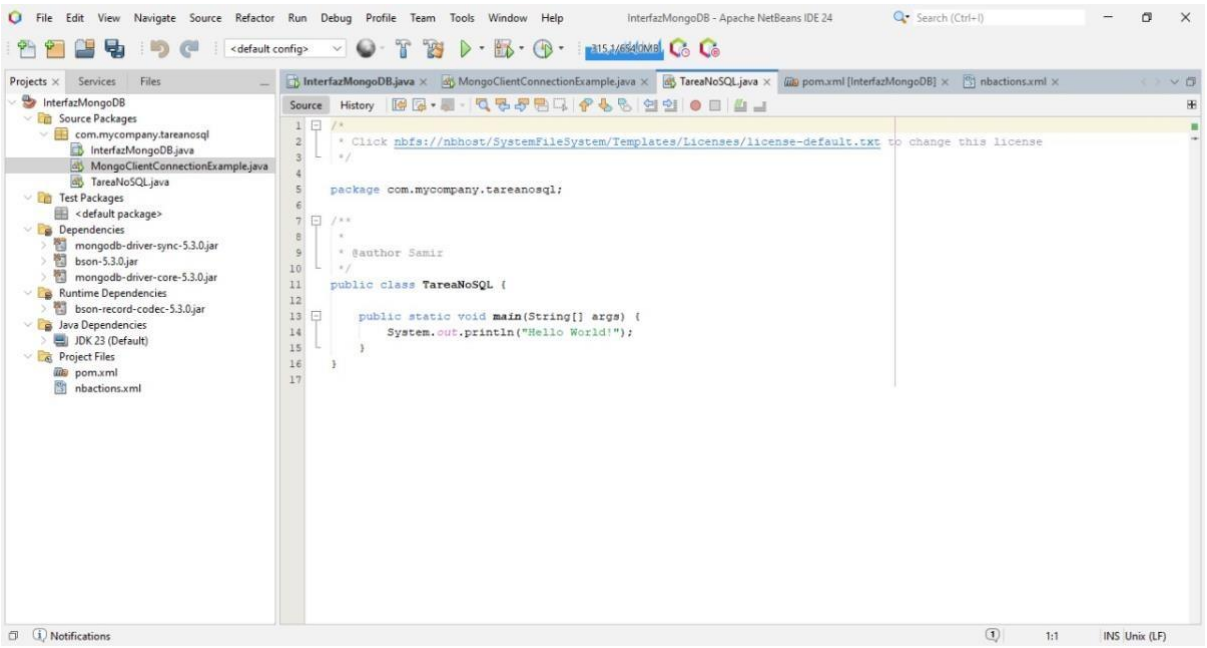
collection :Representa una colección dentro de la base de datos para manipular los datos. **doc (si existe)** :Almacena un documento con datos estructurados como un objet



connectionString: Contiene la URL de conexión a la base de datos MongoDB. **serverApi**: Configura la versión de la API del servidor MongoDB.

settings: Contiene los ajustes para crear una conexión a MongoDB, incluyendo la connectionString y serverApi. **mongoClient**: Es la instancia del cliente MongoDB que se utiliza para conectarse al servidor. **database**: Representa la base de datos "basededatospool" dentro de MongoDB.

e: Captura y maneja excepciones de tipo MongoException.



Package com.mycompany.tareanosql;

- Define el paquete en el que se encuentra la clase, en este caso, com.mycompany.tareanosql.

Public class TareaNoSQL

- Declara la clase pública TareaNoSQL, que es el punto de entrada del programa.

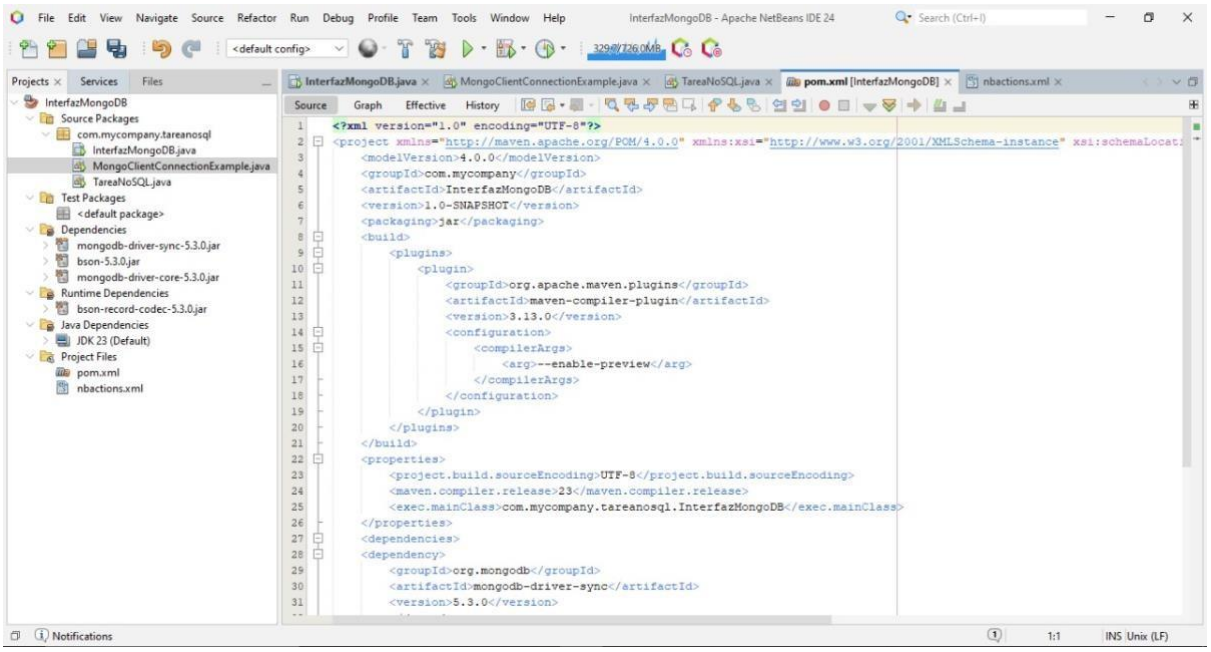
Método main(String[]args)

- Es el método principal de la aplicación, donde comienza la ejecución del programa.

- **args**: Representa los argumentos que pueden pasarse al programa desde la línea de comandos (en este caso, no se usan).

System.out.println("Hello World!");

- Imprime el mensaje "Hello World!" en la consola.



<groupId>: Define el identificador del grupo al que pertenece el proyecto. En este caso, es

"com.mycompany".

<artifactId>: Es el identificador único del proyecto dentro del grupo. Aquí se llama

"InterfazMongoDB".

<version>: Especifica la versión del proyecto, en este caso "1.0-SNAPSHOT", lo que indica que es una versión en desarrollo.

<packaging>: Define el formato de empaquetado del proyecto. "jar" indica que el resultado será un archivo .jar.

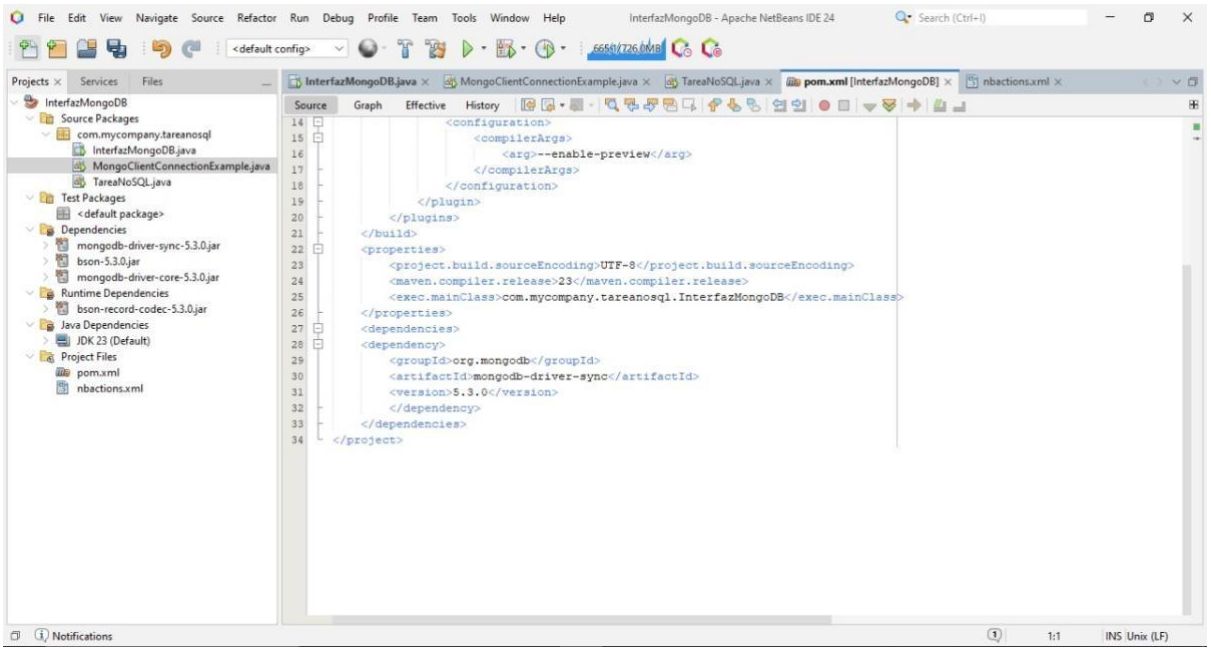
<build>: Contiene configuraciones relacionadas con la compilación del proyecto.

- **<plugins>**: Se define un plugin de compilación:

- **maven-compiler-plugin**: Se usa para compilar el código Java.
- **<version>3.13.0</version>**: Define la versión del plugin de Maven.
- **<compilerArgs>**: Contiene la opción --enable-preview, lo que sugiere que el código usa características en fase de prueba del JDK.

  <properties>: Configura propiedades generales del proyecto.
- **<project.build.sourceEncoding>**: Define la codificación del proyecto como "UTF-8".
- **<maven.compiler.release>**: Especifica la versión de Java a usar, "23" (JDK 23).
- **<exec.mainClass>**: Indica la clase principal del proyecto, "com.mycompany.tareanosql.InterfazMongoDB".

  <dependencies>: Contiene las dependencias necesarias para el proyecto.
- **mongodb-driver-sync (versión 5.3.0)**: Es el controlador de MongoDB para conexión síncrona.



Elementos principales del pom.xml

1. **<project>**: Define el proyecto Maven y su configuración.

2. **<groupId>com.mycompany</groupId>**: Es el identificador del grupo al que pertenece el proyecto.
3. **<artifactId>InterfazMongoDB</artifactId>**: Nombre del artefacto generado por Maven.
4. **<version>1.0-SNAPSHOT</version>**: Indica que es una versión en desarrollo.

5. **<packaging>jar</packaging>**: El proyecto se compilará como un archivo .jar.

6. **<build>**: Configura la compilación del proyecto.

    - **<plugin>**: Se usa el maven-compiler-plugin versión **3.13.0**, con el argumento --enable-preview para habilitar características experimentales del JDK.

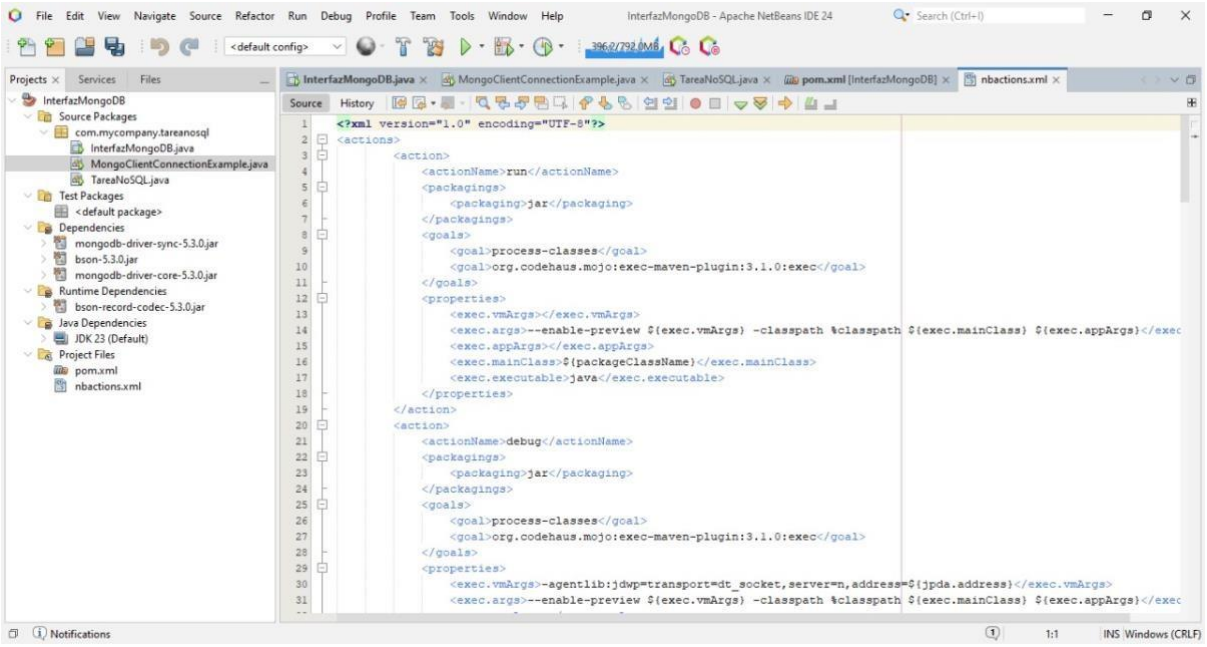
7. **<properties>**:

- **<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>**: Define la codificación del proyecto.

- **<maven.compiler.release>23</maven.compiler.release>**: Especifica que el proyecto usa **Java 23**.
- **<exec.mainClass>com.mycompany.tareanosql.InterfazMongoDB</exec.mainClass>**: Define la clase principal que se ejecutará.

8. **<dependencies>**:

    - **mongodb-driver-sync versión 5.3.0**: Es el driver que permite la conexión síncrona con **MongoDB**.



1. pom.xml (Configuración de Maven)

- **Define el proyecto**: Nombre, versión y empaquetado (jar).

- **Plugins**: Usa maven-compiler-plugin para compilar con JDK 23 y habilitar características experimentales.

- **Dependencias**: Incluye las librerías necesarias para trabajar con MongoDB

  (mongodb-driver-sync).

2. nbactions.xml (Acciones en NetBeans)

- **Define cómo ejecutar y depurar el proyecto** con Maven.
- **Acción run**: Ejecuta la aplicación con la clase principal definida.
- **Acción debug**: Configura el modo depuración con JDWP para conectar el depurador.

3. InterfazMongoDB.java (Clase Principal)

- Es el punto de entrada del programa (main).
- Posiblemente gestiona la conexión y operaciones con MongoDB.

4. MongoClientConnectionExample.java

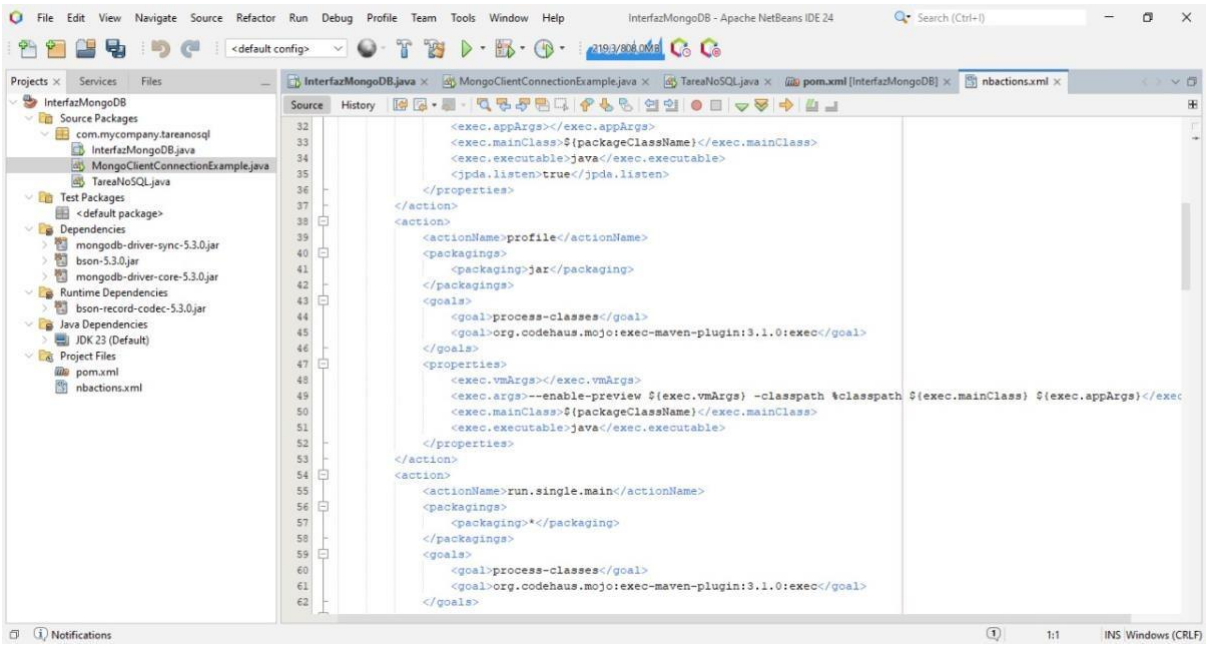
- Probablemente maneja la conexión con la base de datos MongoDB.
- Contiene código para establecer la conexión, insertar y consultar datos.

5. TareaNoSQL.java

- Posiblemente contiene operaciones específicas sobre MongoDB.
- Puede definir métodos para insertar, actualizar o eliminar documentos.

6. Dependencias (bson.jar, mongodb-driver-sync.jar, etc.) • mongodb-driver

- **-sync**: Permite conectarse y operar con MongoDB.
- **bson y bson-record-codec**: Manejan la conversión de datos en MongoDB.



Variables en el pom.xml

1. **<exec.vmArgs>**

- Se usa para definir los argumentos de la máquina virtual Java (JVM).
- En la imagen, permite habilitar la vista previa de características (--enablepreview) y establecer el classpath para la ejecución.

2. **<exec.args>**

- Contiene los argumentos que se pasarán al programa en tiempo de ejecución.
- Se observa que está configurado para incluir los argumentos de la ejecución (${exec.appArgs}).

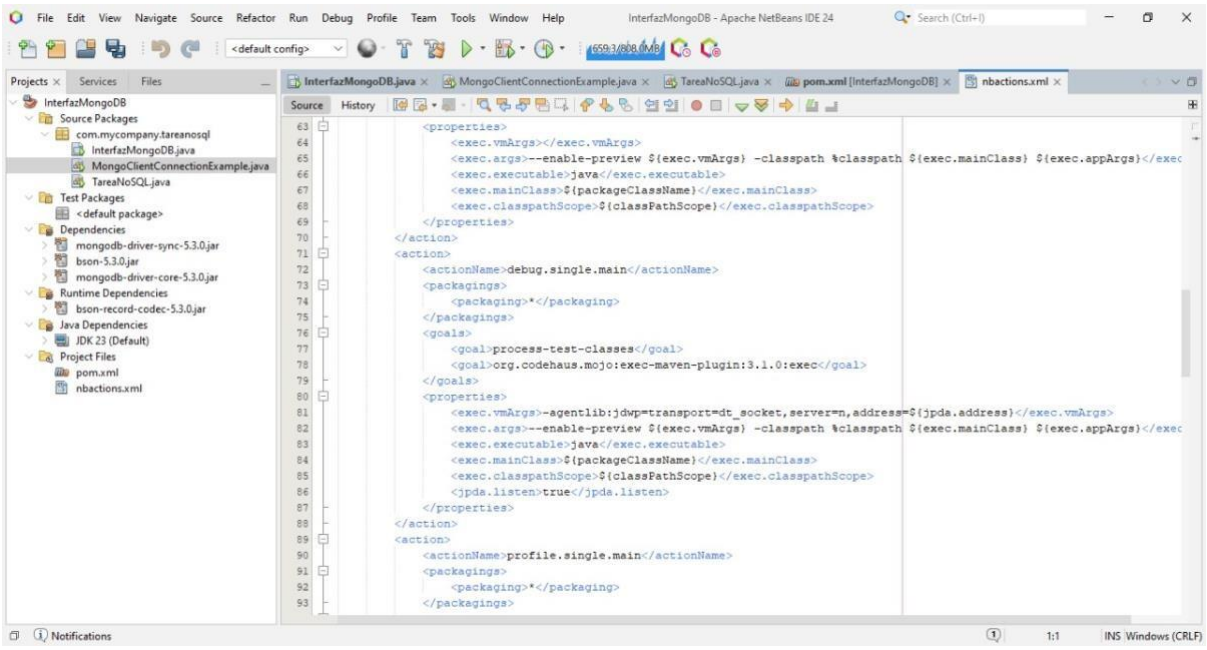
3. **<exec.mainClass>**

- Define la clase principal que se ejecutará en la aplicación.
- En este caso, usa la variable {packageClassName} para referirse a la clase principal.

4. **<exec.executable>**
    - Indica que el programa debe ejecutarse con el comando java.

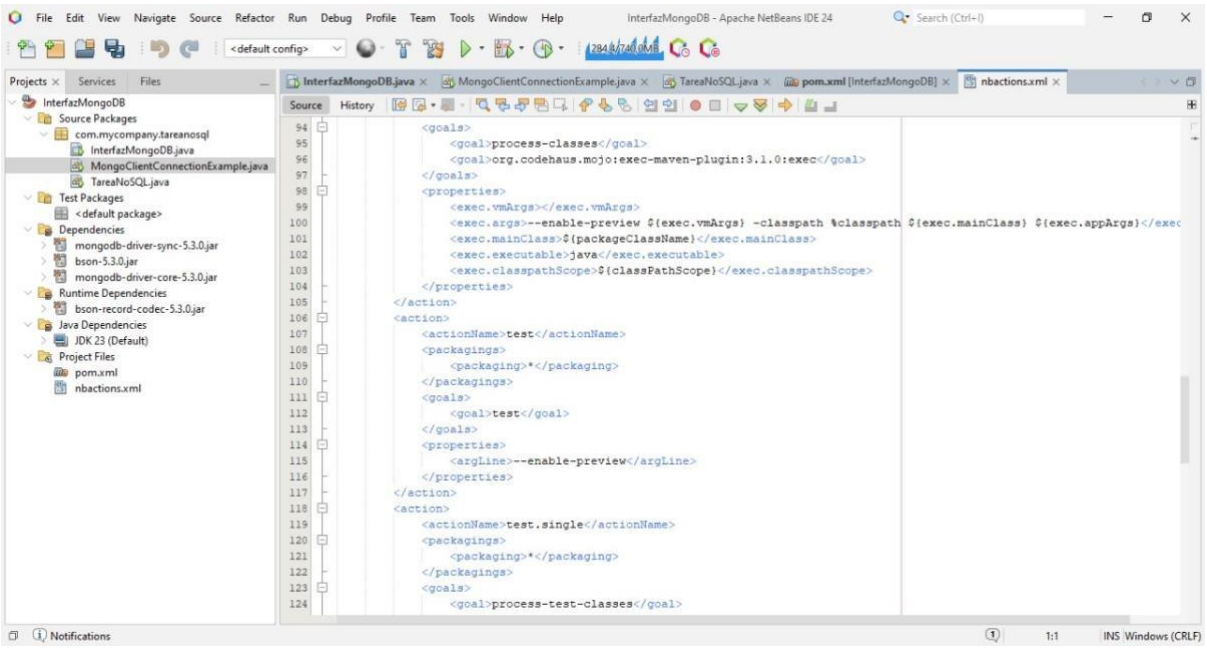
5. **<actionName>**
    - Define las acciones específicas dentro del pom.xml, como:
        - **profile**: Se asocia con la generación de un **JAR**.
        - **run.single.main**: Indica la ejecución de una clase principal específica.

6. **<goal>**
    - Define tareas de Maven que se ejecutarán.
    - Se usa **org.codehaus.mojo:exec-maven-plugin:3.1.0:exec**, un plugin para ejecutar clases Java directamente desde Maven.



1. **<actionName>debug.single.main</actionName>**
    - Esta acción configura la ejecución en **modo depuración**.
    - Se usa la opción agentlib:jdwp para habilitar la depuración remota en un socket.
2. **<exec.vmArgs>**
    - Contiene los argumentos de la JVM.
    - -agentlib:jdwp=transport=dt_socket,server=n,address=${jpda.address}
        - Habilita el depurador remoto usando el protocolo JDWP.
        - transport=dt_socket: Usa un socket para la comunicación.
        - server=n: Indica que el programa actuará como cliente y se conectará a un depurador remoto.
        - address={jpda.address}: Define la dirección para la conexión de depuración.

3. **<exec.args>**
    - --enable-preview: Activa las características en vista previa de Java.
    - -classpath %classpath ${exec.mainClass} ${exec.appArgs}
        - Establece el classpath y ejecuta la clase principal con sus argumentos.
4. **<exec.mainClass>**
    - Define la clase principal que se ejecutará (${packageClassName}).
5. **<exec.executable>**
    - Especifica que el ejecutable será java.
6. **<exec.classpathScope>**
    - Usa la variable {classPathScope}, lo que sugiere que el alcance del classpath se configura dinámicamente.
7. **<jpda.listen>true</jpda.listen>**
    - Habilita la escucha para depuración en modo JDWP.
8. **<actionName>profile.single.main</actionName>**
    - Se configura para **ejecutar el programa con perfilado**, posiblemente para analizar rendimiento o uso de memoria.



<goal>

- **process-classes**: Este objetivo se encarga de procesar las clases del proyecto, incluyendo la compilación y preparación antes de ejecutar o probar el código.
- **exec-maven-plugin:3.1.0:exec**: Ejecuta comandos o scripts definidos en la configuración del plugin Maven exec-maven-plugin.
- **test**: Ejecuta las pruebas definidas en el proyecto.

<properties>

- **<exec.vmArgs>**: Define los argumentos de la máquina virtual Java (JVM) que se pasan al momento de ejecutar el programa.
- **<exec.args>**: Especifica los argumentos de ejecución, incluyendo opciones como -enable-preview (para habilitar funciones de vista previa en Java), la ruta de clase (classpath) y la clase principal (mainClass) a ejecutar.
- **<exec.mainClass>**: Define el nombre de la clase principal que será ejecutada.
- **<exec.executable>**: Especifica el ejecutable que se usará, en este caso java.

- **<exec.classpathScope>**: Determina el alcance del classpath usado para la ejecución, asegurando que las dependencias necesarias estén disponibles.

<actionName>

- Representa el nombre de la acción o tarea personalizada en el flujo del proyecto.

  Ejemplos:

- test: Ejecuta las pruebas.

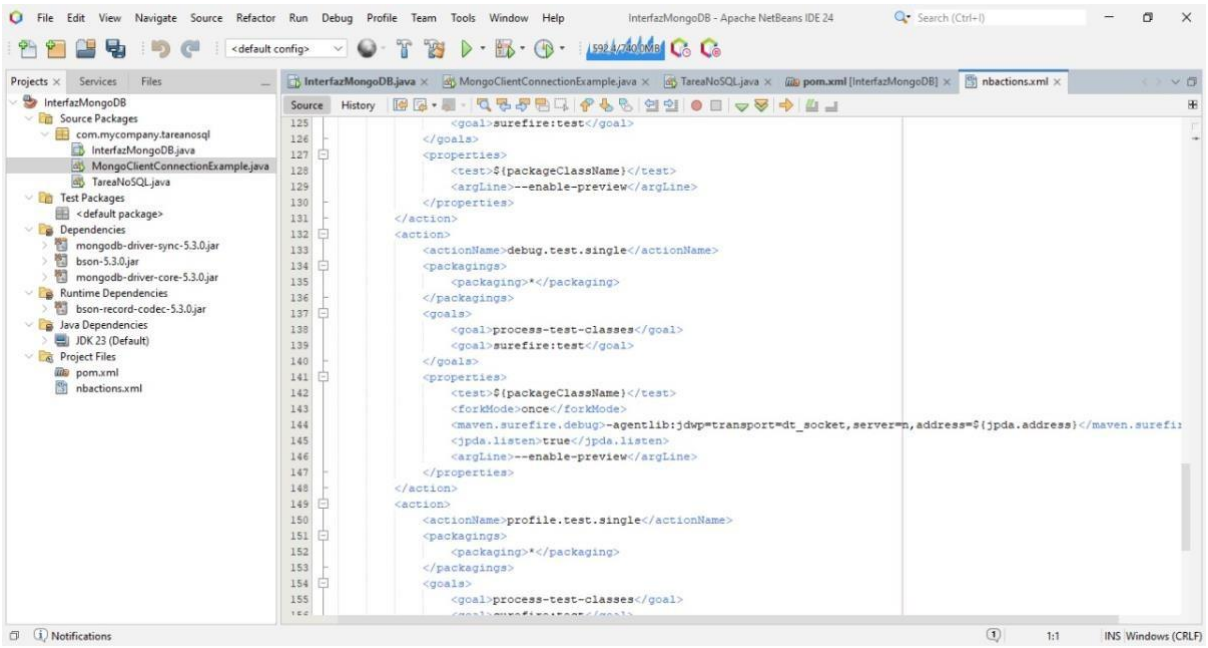
- test.single: Ejecuta clases o métodos de prueba individuales.

<packagings>

- Especifica el tipo de empaquetado del proyecto Maven, que puede ser jar, war, entre otros. El asterisco (*) indica que esta configuración es válida para cualquier tipo de empaquetado.

<argLine>

- Define argumentos específicos para la JVM. En este caso, el argumento --enablepreview habilita características de Java en estado de vista previa, útil para trabajar con versiones experimentales del lenguaje.



1. **Árbol del proyecto** (panel izquierdo):

    - **Nombre del proyecto**: InterfazMongoDB.
    - **Source Packages**:
        - Contiene paquetes con clases Java relacionadas, como:
        - InterfazMongoDB.java
        - MongoClientConnectionExample.java
        - TareaNoSQL.java

    • **Dependencias (Dependencies)**:

        - Incluye bibliotecas necesarias para trabajar con MongoDB, como:
        - mongodb-driver-sync-5.3.0.jar: Driver síncrono de MongoDB.
        - bson-5.3.0.jar: Biblioteca para manejar objetos BSON.

- mongodb-driver-core-5.3.0.jar: Core del controlador de MongoDB.

• **Runtime Dependencies**:

- bson-record-codec-5.3.0.jar: Codec para registros BSON.

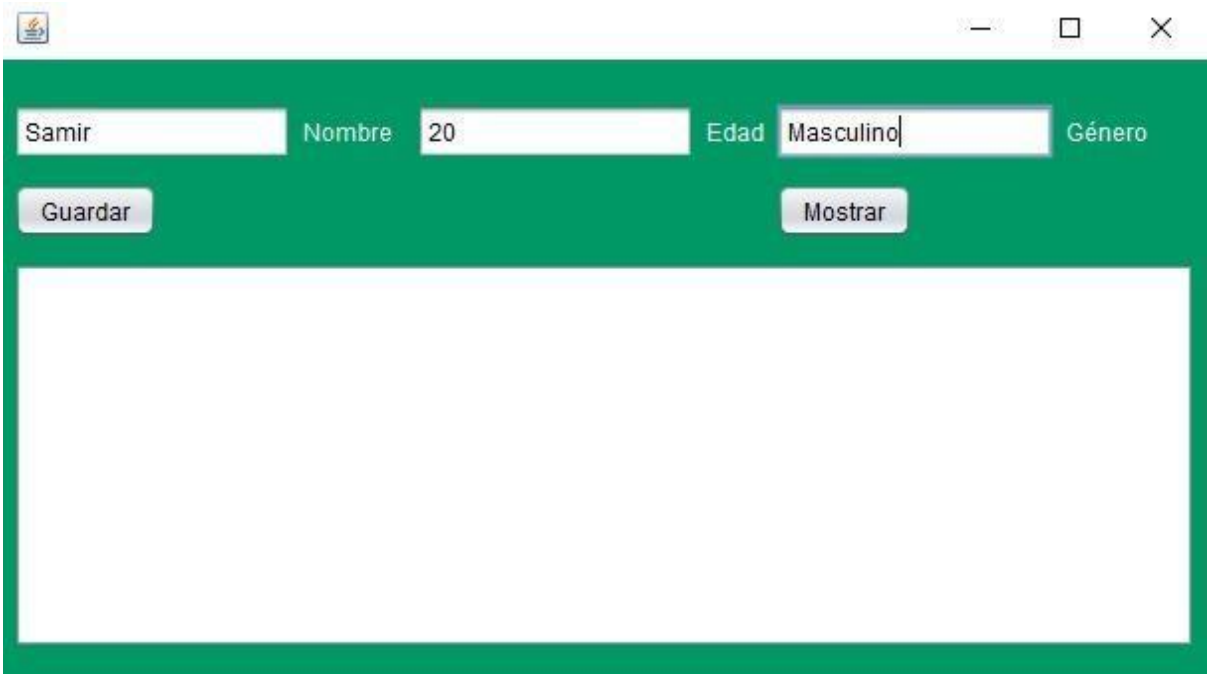
• **Java Dependencies**:

- Utiliza **JDK 23 (Default)** como versión de Java.

2. **Archivos del proyecto**:
   - pom.xml: Archivo de configuración de Maven para gestionar las dependencias y configuraciones del proyecto.
   - nbactions.xml: Archivo de configuración de acciones de NetBeans.
3. **Contenido del archivo pom.xml** (panel central):

   • Este archivo incluye configuraciones específicas de Maven, como:

   - **Acciones personalizadas (<action>)**:
   - Definición de objetivos (<goals>) y propiedades (<properties>) para ejecutar pruebas, depurar y realizar perfiles de ejecución.
   - Configuraciones para el uso de argLine con opciones como --enablepreview, que habilita características preliminares de Java.



Componentes principales de la interfaz:

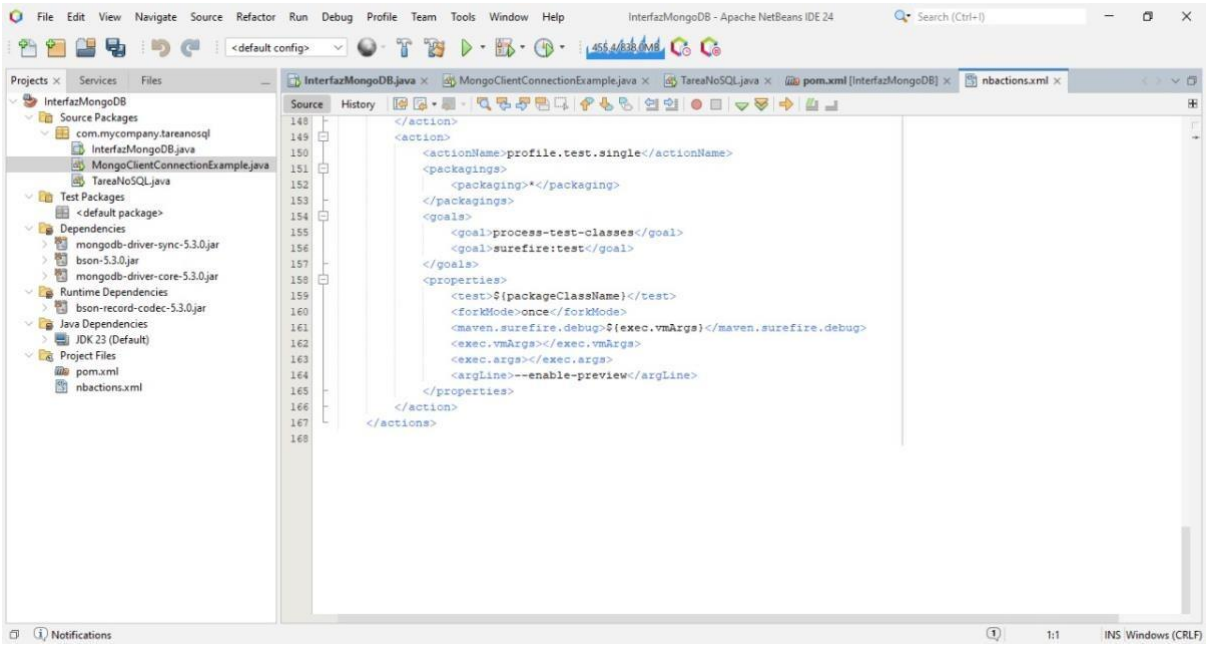
1. **Campos de texto**:
    - **Nombre**: Se muestra el texto "Samir". Es un campo para introducir el nombre de una persona.
    - **Edad**: Se muestra el valor "20". Es un campo para ingresar la edad de una persona.
    - **Género**: Se muestra el texto "Masculino". Es un campo para especificar el género.
2. **Botones**:
    - **Guardar**: Posiblemente se utiliza para almacenar los datos ingresados en algún tipo de estructura (como una base de datos, archivo, o lista en memoria).
    - **Mostrar**: Probablemente se utiliza para desplegar los datos almacenados en el área en blanco ubicada debajo de los botones.

3. **Área de texto** (en blanco):

- Espacio para mostrar los datos almacenados, como una lista o tabla de registros previamente guardados.

Funcionalidad esperada:

- **Entrada de datos**: El usuario puede llenar los campos de "Nombre", "Edad" y "Género".
- **Guardar**: Al presionar este botón, los datos ingresados deberían guardarse en una base de datos, archivo, o estructura interna del programa.
- **Mostrar**: Al presionar este botón, los datos guardados deberían aparecer en el área en blanco.



${packageClassName}

- Se usa en <test>${packageClassName}</test>, lo que indica que Surefire ejecutará una clase de prueba específica cuyo nombre de paquete se define en otra parte del pom.xml o en las propiedades del proyecto.

forKMode=once

- Indica que el proceso de prueba se ejecutará en un solo JVM en lugar de iniciar una nueva instancia para cada prueba.

${exec.vmArgs}

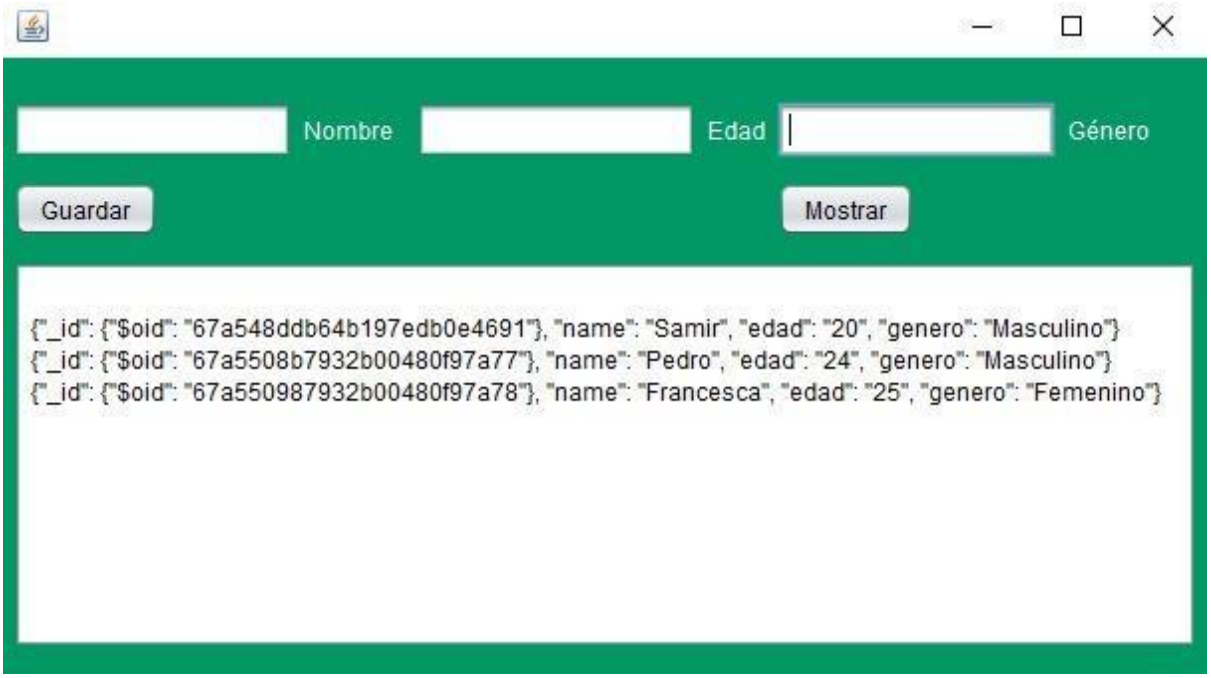
- Representa los argumentos de la máquina virtual (JVM) usados durante la ejecución de las pruebas.

${maven.surefire.debug}

- Se usa para habilitar la depuración de pruebas de Maven Surefire. Su valor probablemente se define en otro lugar del proyecto o en variables de entorno.

<argLine>--enable-preview</argLine>

- Indica que se habilitan características de Java en modo "preview", lo que es útil si el proyecto usa características experimentales de Java 23.



nombre

- **Descripción:** Variable que almacena el valor ingresado en el campo de texto "Nombre".
- **Tipo:** String (cadena de texto).
- **Uso:** Representa el nombre de la persona a guardar en la base de datos o lista. **edad**
- **Descripción:** Variable que almacena el valor ingresado en el campo de texto "Edad".
- **Tipo:** String o entero (idealmente debería ser un número entero, pero en la imagen parece manejarse como texto).
- **Uso:** Representa la edad de la persona y se usa para guardar o mostrar la información.

genero

- **Descripción:** Variable que almacena el valor ingresado en el campo de texto "Género".
- **Tipo:** String (cadena de texto).
- **Uso:** Representa el género de la persona (por ejemplo, "Masculino" o "Femenino").

boton_guardar

- **Descripción:** Variable asociada al botón "Guardar". Ejecuta una función o método para almacenar los datos ingresados en los campos de texto.
- **Tipo:** Evento o acción (trigger).
- **Uso:** Activa la función que guarda los datos en la base de datos o lista.

boton_mostrar

- **Descripción:** Variable asociada al botón "Mostrar". Ejecuta una función o método para recuperar y mostrar los datos almacenados.
- **Tipo:** Evento o acción (trigger).
- **Uso:** Activa la función que muestra los datos en el área de texto.

base_datos o lista_personas

- **Descripción:** Una estructura (como un arreglo o base de datos) que almacena los datos de las personas en formato JSON.
- **Tipo:** Lista o base de datos (como MongoDB).
- **Uso:** Contiene todos los objetos con información de las personas. Ejemplo:


```
{"_id": {"$oid": "67a548ddb64b197edb0e4691"}, "name": "Samir", "edad": "20", "genero": "Masculino"}
{"_id": {"$oid": "67a5508b7932b00480f97a77"}, "name": "Pedro", "edad": "24", "genero": "Masculino"}
{"_id": {"$oid": "67a550987932b00480f97a78"}, "name": "Francesca", "edad": "25", "genero": "Femenino"}
```

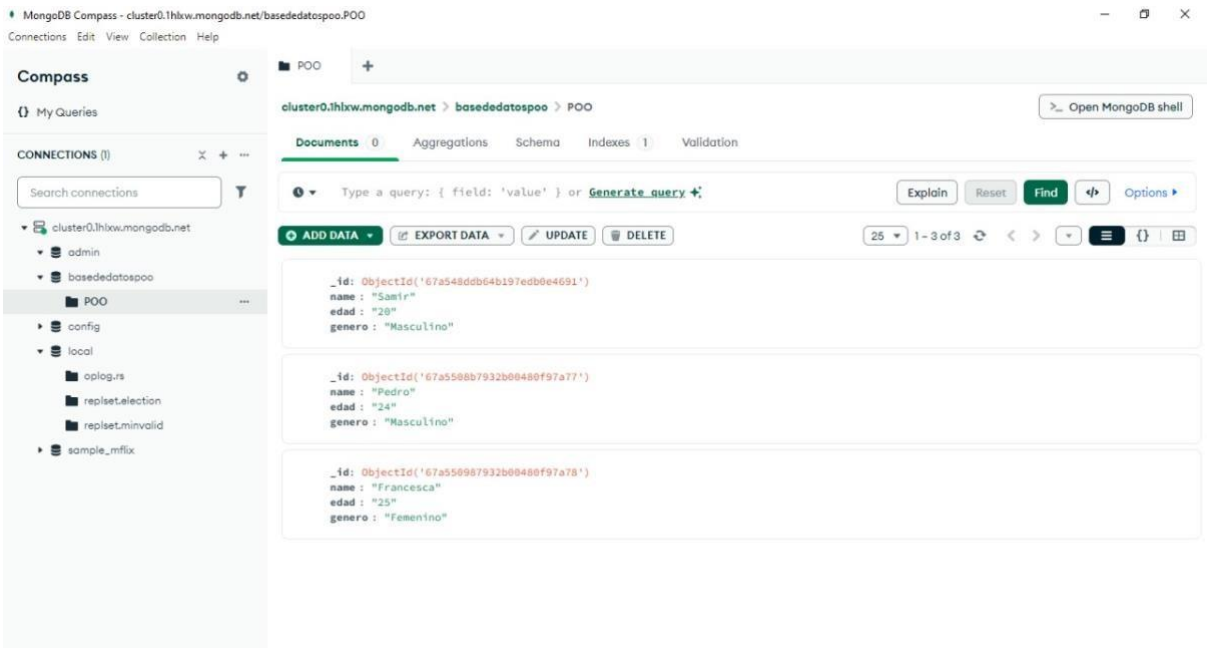

area_mostrar

- **Descripción:** Variable asociada al área de texto donde se despliegan los datos almacenados.
- **Tipo:** String o lista de strings.
- **Uso:** Muestra los datos recuperados de la base de datos o lista.

id_persona (opcional)

- **Descripción:** Identificador único para cada registro (como el _id que aparece en el formato JSON).
- **Tipo:** String.
- **Uso:** Se genera automáticamente al guardar un nuevo registro (probablemente con

MongoDB).



Variables del programa asociadas a esta base de datos:

1. **basededatospoo (Base de datos)**

- **Descripción:** Es el nombre de la base de datos que contiene las colecciones.

  En este caso, la colección relevante es POO.

- **Tipo:** Base de datos MongoDB.

- **Uso:** Almacenar y organizar colecciones que contienen documentos JSON.

2. **POO (Colección)**

- **Descripción:** Es una colección dentro de la base de datos basededatospoo.

- **Tipo:** Colección de MongoDB.

- **Uso:** Contiene documentos JSON relacionados con personas, organizados por campos como _id, name, edad y genero.

3. **_id (Identificador único)**

- **Descripción:** Identificador único generado automáticamente por MongoDB para cada documento.
- **Tipo:** ObjectId (un tipo especial de MongoDB).
- **Uso:** Permite identificar y diferenciar cada documento en la colección.
- **Ejemplo:** "ObjectId('67a548ddb64b197edb0e4691')".

4. **name (Nombre de la persona)**

- **Descripción:** Campo que contiene el nombre de una persona.
- **Tipo:** String (cadena de texto).
- **Ejemplo:** "Samir", "Pedro", "Francesca".

5. **edad (Edad de la persona)**

- **Descripción:** Campo que indica la edad de la persona.
- **Tipo:** String (debería ser un número entero para operaciones numéricas).
- **Ejemplo:** "20", "24", "25".

6. **genero (Género de la persona)**

- **Descripción:** Campo que indica el género de la persona.
- **Tipo:** String (cadena de texto).
- **Ejemplo:** "Masculino", "Femenino".

3. CONCLUSIONES

Aplicaciones basadas en bases de datos NoSQL, como MongoDB, son ideales para proyectos con estructuras de datos flexibles, ya que permiten almacenar información dinámica como tareas con atributos variables (nombre, prioridad, estado, etc.) sin necesidad de una estructura fija como en bases de datos relacionales.

La implementación de principios SOLID en el desarrollo de software mejora la mantenibilidad y escalabilidad del proyecto, asegurando que el código sea modular, fácil de entender y modificar, lo cual es crucial en sistemas con múltiples capas, como la interfaz gráfica, lógica de negocio y acceso a datos.

La combinación de interfaces gráficas con bases de datos NoSQL facilita la creación de herramientas visuales intuitivas para usuarios no técnicos, optimizando la experiencia de uso mientras se gestiona eficientemente la persistencia de los datos en aplicaciones modernas.

4. RECOMENDACIONES

Asegúrarse de implementar operaciones eficientes para evitar retrasos al guardar, actualizar o consultar datos desde la base de datos. Utiliza herramientas como índices en MongoDB para acelerar las búsquedas y limitar la cantidad de datos que se recuperan al mínimo necesario.

Diseña una interfaz gráfica que sea intuitiva y fácil de usar para los usuarios finales. Esto incluye usar etiquetas claras, botones accesibles y una estructura visual lógica que permita a los usuarios gestionar sus tareas sin dificultad.

5. BIBLIOGRAFÍA

Chodorow, K. (2019). *MongoDB: The definitive guide* (3.ª ed.). O'Reilly Media.

Banker, K. (2016). *MongoDB in action* (2.ª ed.). Manning Publications.

Russell, B. A. (2020). Practical MongoDB: Architecting, developing, and administering MongoDB. Apress.

Williams, E. (2021). Learn MongoDB 4.x: Design and build scalable and flexible applications. Packt Publishing.

Mehta, J., & Shah, M. (2022). *NoSQL with MongoDB in 24 hours*. Addison-Wesley.

Hows, D., Plugge, E., Membrey, P., & Hawkins, T. (2015). *MongoDB basics.* Apress.

Newton, J., & Smith, R. (2023). *Designing modern applications with MongoDB.*

O'Reilly Media.