

Informe Practica Programcion III : BAHRI MOHAMED FIRAS

NIE: Y9304323D

Introduccion:

En esta práctica, he intentado implementar todos los requisitos posibles. El código funciona bien en todos los apartados (Smart LLM, voz en consola TTS incluidos), pero no estoy seguro de si he utilizado las bibliotecas proporcionadas para implementar estas dos clases de manera adecuada o no. Dado que la ejecución depende de los argumentos pasados por el usuario, voy a intentar analizar todos los casos brevemente. La ejecución debe ser de la siguiente manera:

java -jar jllm.jar repository model view

Las opciones posibles:

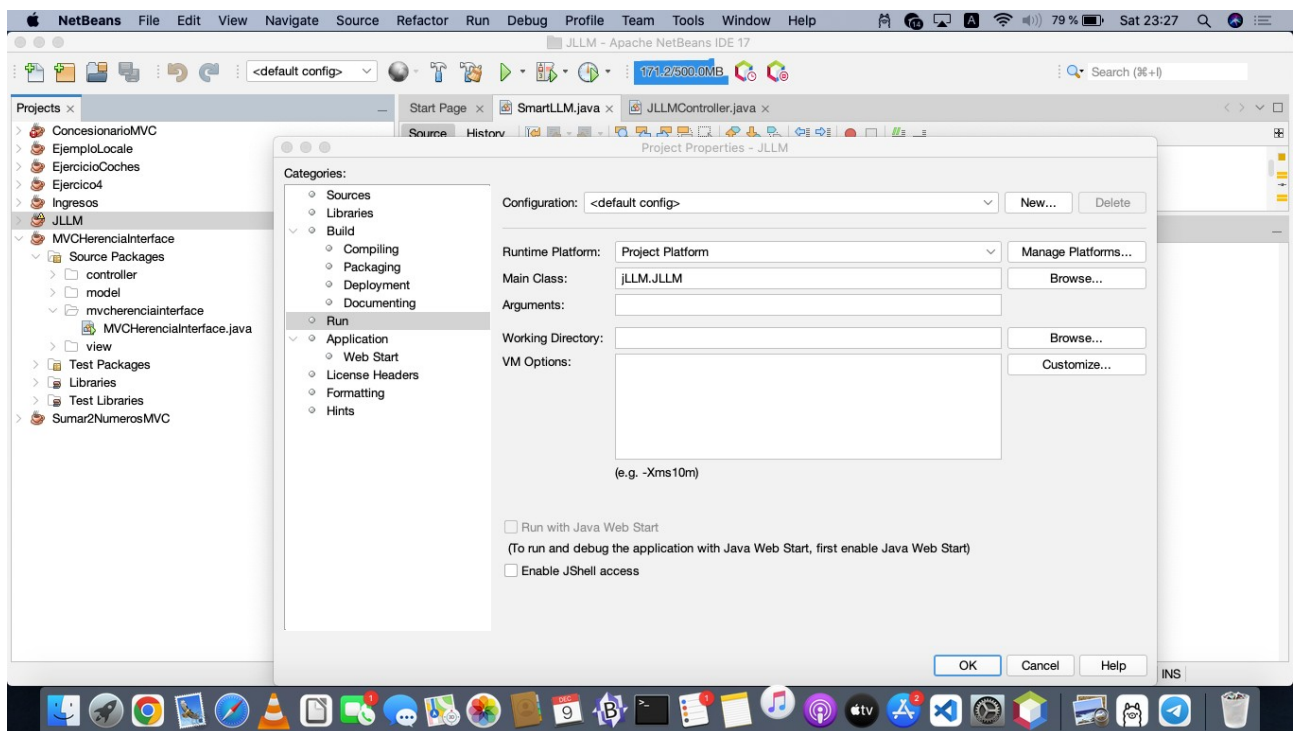
repository: xml,json

model: fake, csv, smart

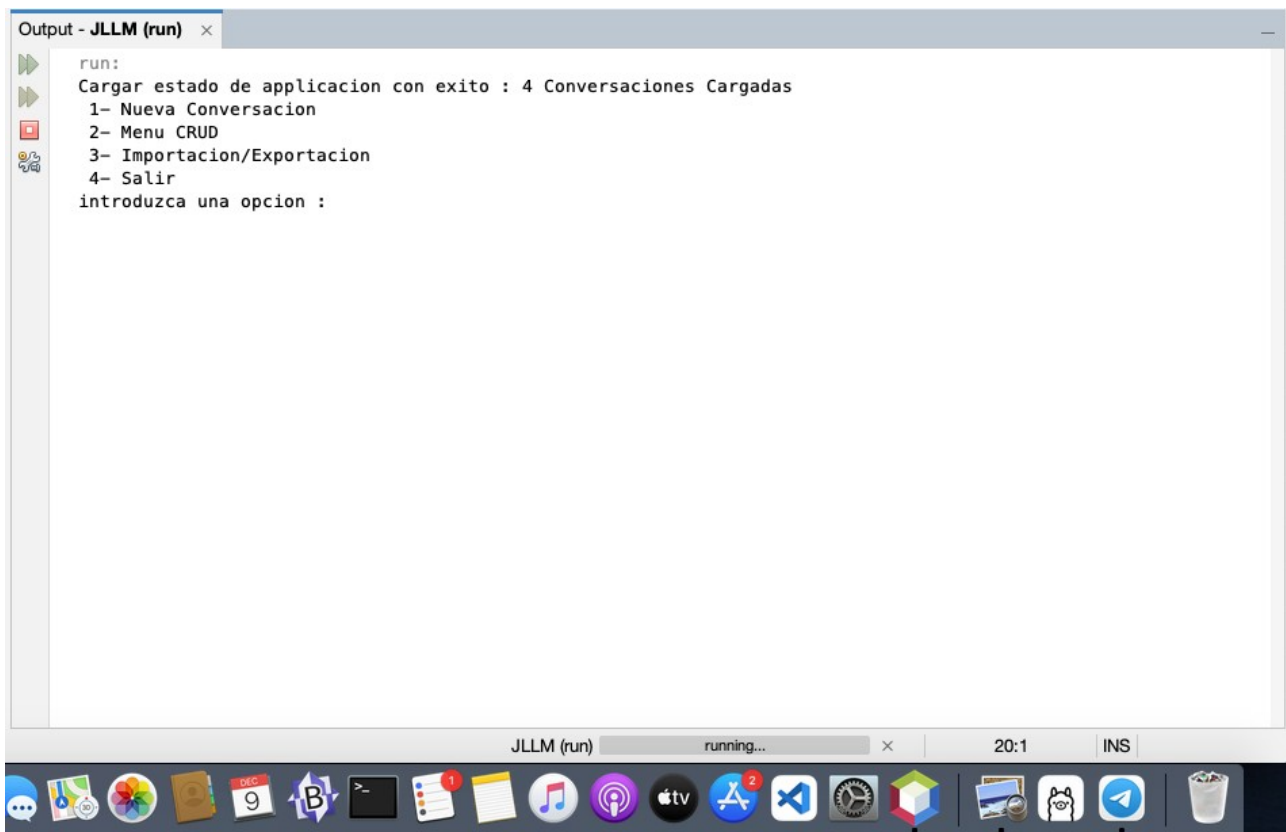
vista: console, voz

Caso 1: Por Default

En este caso, el usuario no ha pasado ningún argumento.



En la ejecución del programa, se muestra el menú principal. Ahora, todo depende de la opción introducida por el usuario. Dado que no se han pasado argumentos, nuestra interfaz se ejecutará con la clase hija SimpleConsole ApplicationView. El modelo trabajará con FakeLLm, y la importación/exportación de ficheros se realizará en formato JSON.



```
Output - JLLM (run) x
run:
Cargar estado de aplicacion con exito : 4 Conversaciones Cargadas
1- Nueva Conversacion
2- Menu CRUD
3- Importacion/Exportacion
4- Salir
introduzca una opcion :
```

Funcionalidad 1

Show ApplicationStart: Informar al usuario si se ha logrado cargar el estado de la aplicación con éxito o no, identificando el número de conversaciones cargadas. Esto se realiza mediante la deserialización del archivo jLLM.bin, el cual debe existir previamente y contener las conversaciones para cargar. En caso contrario, se imprimirá un mensaje de bienvenida.

Nota: Para cargar el estado de la aplicación, el archivo jLLM.bin debe existir en la carpeta jLLM que esté en el Escritorio.

Funcionalidad 2

Nueva Convercacion:

Después, se presentan las opciones posibles al usuario. Al elegir la primera opción, que es comenzar una conversación, se permite crear una conversación con un agente FakeLLM. Esta conversación termina cuando el usuario escribe (/salir). A continuación, se muestra un ejemplo de la ejecución.

Aquí, el usuario escribe su mensaje al agente, que debe ser diferente de (/salir), y se crea un array de mensajes. Dentro del bucle while, que se ejecuta siempre que el mensaje sea diferente de (/salir), se crea una instancia de la clase Mensaje con tres atributos:

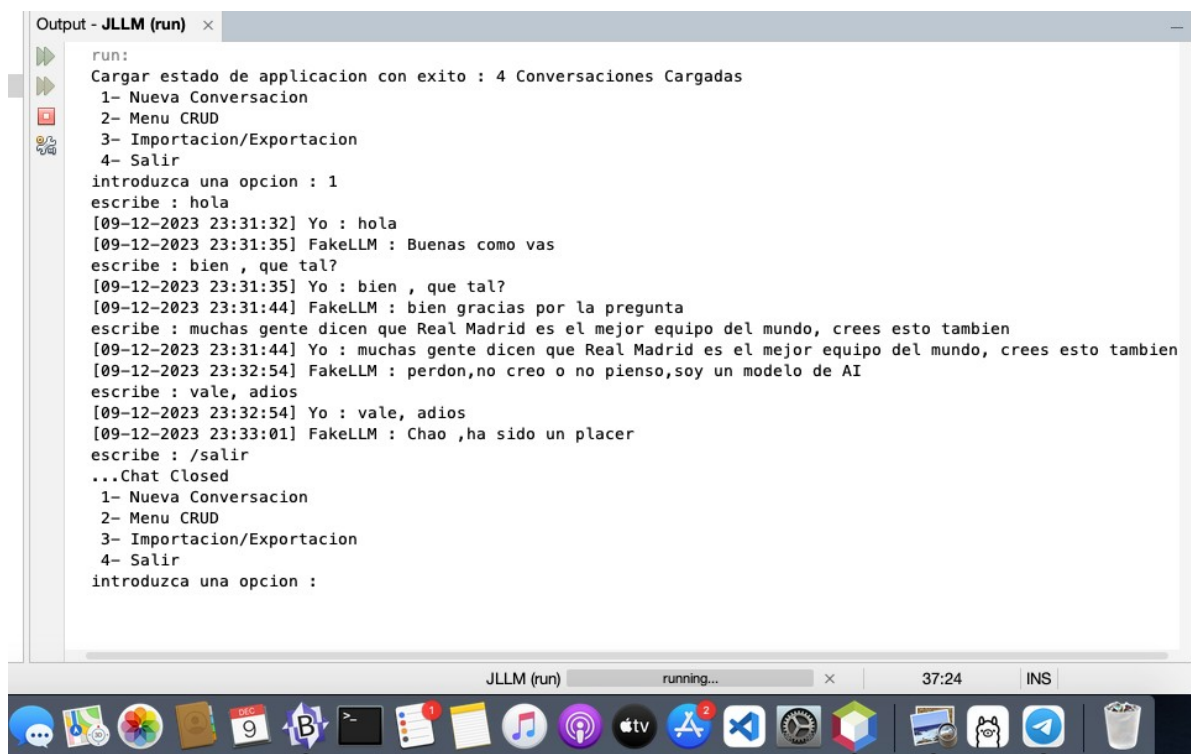
Time: Se crea un objeto Instant que representa el instante actual en el tiempo. Luego, se convierte en el número de segundos desde el inicio de la época (1 de enero de 1970, 00:00:00 UTC) con la llamada a getEpochSecond.

Contenido: Es el mensaje introducido por el usuario.

Emisor: Una variable de tipo String que tiene el valor "Yo" para identificar al usuario.

También se crea un array de mensajes que incluirá todos los mensajes. Cada vez que se crea una instancia de mensaje, se añade a este array en la clase View. Lo mismo ocurre en la clase Modelo cuando se recoge la respuesta. Este array se va pasando entre View y el Modelo a través del controlador.

Esto es un ejemplo de ejecución.

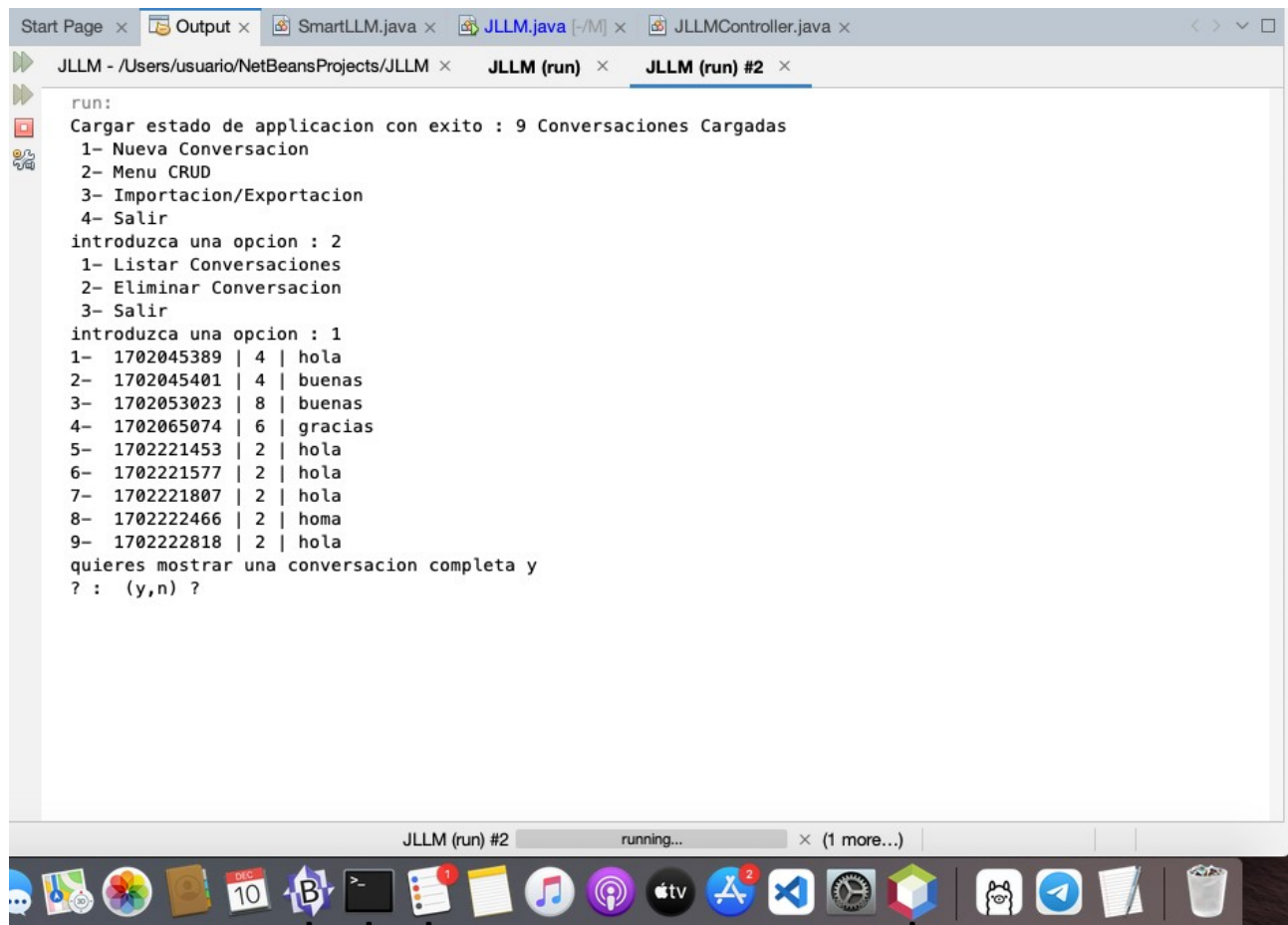


```
run:
Cargar estado de applicacion con exito : 4 Conversaciones Cargadas
1- Nueva Conversacion
2- Menu CRUD
3- Importacion/Exportacion
4- Salir
introduzca una opcion : 1
escribe : hola
[09-12-2023 23:31:32] Yo : hola
[09-12-2023 23:31:35] FakeLLM : Buenas como vas
escribe : bien , que tal?
[09-12-2023 23:31:35] Yo : bien , que tal?
[09-12-2023 23:31:44] FakeLLM : bien gracias por la pregunta
escribe : muchas gente dicen que Real Madrid es el mejor equipo del mundo, crees esto tambien
[09-12-2023 23:31:44] Yo : muchas gente dicen que Real Madrid es el mejor equipo del mundo, crees esto tambien
[09-12-2023 23:32:54] FakeLLM : perdon,no creo o no pienso,soy un modelo de AI
escribe : vale, adios
[09-12-2023 23:32:54] Yo : vale, adios
[09-12-2023 23:33:01] FakeLLM : Chao ,ha sido un placer
escribe : /salir
...Chat Closed
1- Nueva Conversacion
2- Menu CRUD
3- Importacion/Exportacion
4- Salir
introduzca una opcion :
```

Funcionalidad 3:

Menu Crud: Mostrar el menú CRUD con 3 opciones: Listar conversaciones, eliminar conversación y salir. Listar conversaciones: muestra todas las conversaciones actuales en nuestra aplicación. Este método está implementado en la clase JLLMModel y se llama ListarConversaciones. Consiste en imprimir el formato de la conversación (un método en

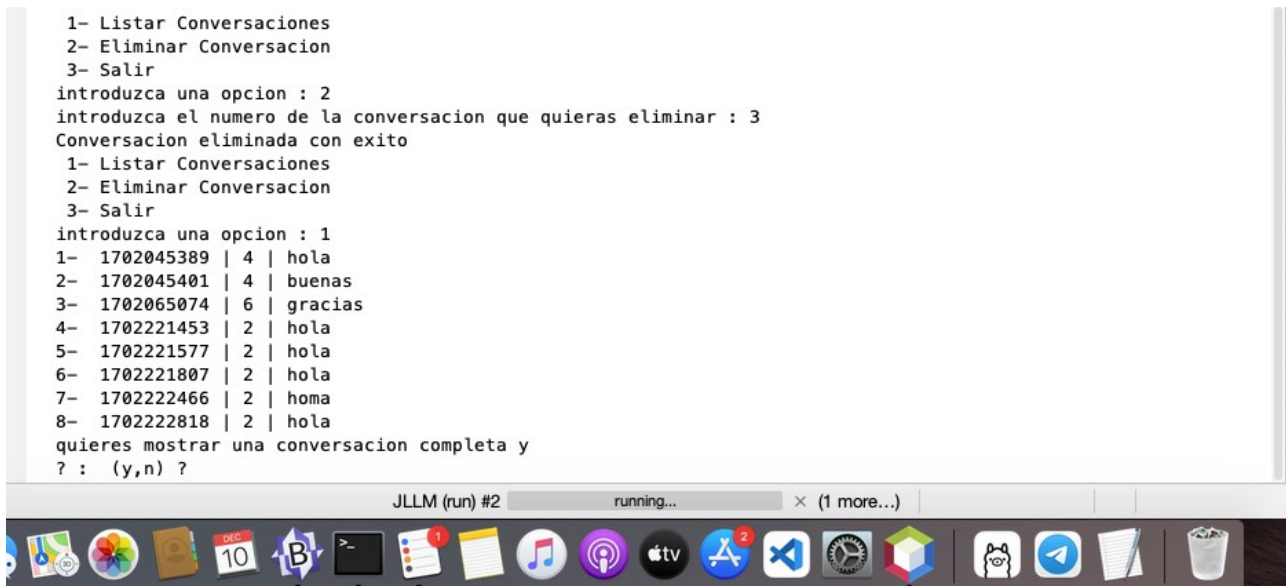
la clase Conversation llamado mostrarConversacion) con el uso del bucle for each. Además, se permite al usuario ver una conversación completa indicándola con su número. mediante el método unaConversacion. Todo esto se realiza si la conversación existe. ,es decir la lista de conversaciones no esta vacia . Esto es un ejemplo de la ejecucion .



```
run:
Cargar estado de aplicacion con exito : 9 Conversaciones Cargadas
1- Nueva Conversacion
2- Menu CRUD
3- Importacion/Exportacion
4- Salir
introduzca una opcion : 2
1- Listar Conversaciones
2- Eliminar Conversacion
3- Salir
introduzca una opcion : 1
1- 1702045389 | 4 | hola
2- 1702045401 | 4 | buenas
3- 1702053023 | 8 | buenas
4- 1702065074 | 6 | gracias
5- 1702221453 | 2 | hola
6- 1702221577 | 2 | hola
7- 1702221807 | 2 | hola
8- 1702222466 | 2 | homa
9- 1702222818 | 2 | hola
quieres mostrar una conversacion completa y
? : (y,n) ?
```

Al elegir eliminar conversación, se le pide al usuario que introduzca el número de la conversación que desea eliminar. El método eliminarConversacion en la clase JLLMModel se encarga de verificar si dicha conversación existe utilizando el método contains. Para que esto se implemente correctamente en la clase Conversation, se sobrescribe el método equals para identificar cada instancia de esta clase. Este procedimiento se realiza siempre que existan conversaciones. En este ejemplo, vamos a eliminar la conversación con el número 3 y mostrar la nueva lista de conversaciones.

```
1- Listar Conversaciones
2- Eliminar Conversacion
3- Salir
introduzca una opcion : 2
introduzca el numero de la conversacion que quieras eliminar : 3
Conversacion eliminada con exito
1- Listar Conversaciones
2- Eliminar Conversacion
3- Salir
introduzca una opcion : 1
1- 1702045389 | 4 | hola
2- 1702045401 | 4 | buenas
3- 1702065074 | 6 | gracias
4- 1702221453 | 2 | hola
5- 1702221577 | 2 | hola
6- 1702221807 | 2 | hola
7- 1702222466 | 2 | homa
8- 1702222818 | 2 | hola
quieres mostrar una conversacion completa y
? : (y,n) ?
```



Funcionalidad 4: Importación y Exportación

Esta funcionalidad implica la importación y exportación de datos, utilizando formato JSON en nuestro caso (por defecto). Estos dos métodos están presentes en la interfaz IRepository, la cual está implementada por las clases XMLRepository y JSONRepository.

Nota: Los archivos deben tener el nombre "input" (ya sea .json o .xml) y deben encontrarse en la carpeta "jLLM" que debe existir en el escritorio; de lo contrario, esta funcionalidad no funcionará.

Exportación:

El usuario tiene la opción de exportar todas las conversaciones mediante la llamada al método exportarTodos en la clase JLLMModel. Alternativamente, puede elegir exportar algunas conversaciones. En este caso, el usuario debe introducir el número total de conversaciones que desea exportar y los números de cada conversación (los números deben estar separados por espacio). En este método, he utilizado la colección Set para garantizar que no existan dos números repetidos y que el usuario no exporte dos veces la misma conversación. Este método está implementado con el nombre exportarAlgunas en JLLMModel. A continuación, se muestra un ejemplo de la ejecución de esta funcionalidad (exportación), junto con una captura del archivo input.json (que estaba vacío).

Todas las conversaciones:

Cargar estado de aplicacion con exito : 10 Conversaciones Cargadas

- 1- Nueva Conversacion
- 2- Menu CRUD
- 3- Importacion/Exportacion
- 4- Salir

introduzca una opcion : 3

- 1- Exportar
- 2- Importar
- 3- Salir

introduzca una opcion : 1

Quieres exportar todas las conversaciones (y,n) ? y

Exportacion con exito

- 1- Exportar
- 2- Importar
- 3- Salir

Algunas:

- 1- Exportar
- 2- Importar
- 3- Salir

introduzca una opcion : 1

Quieres exportar todas las conversaciones (y,n) ? n

introduzca el numero total de las conversaciones que quieras exportar : 3

introduzca los numeros de conversaciones que quieras exportar

Por favor los numeros deben separados por espacio

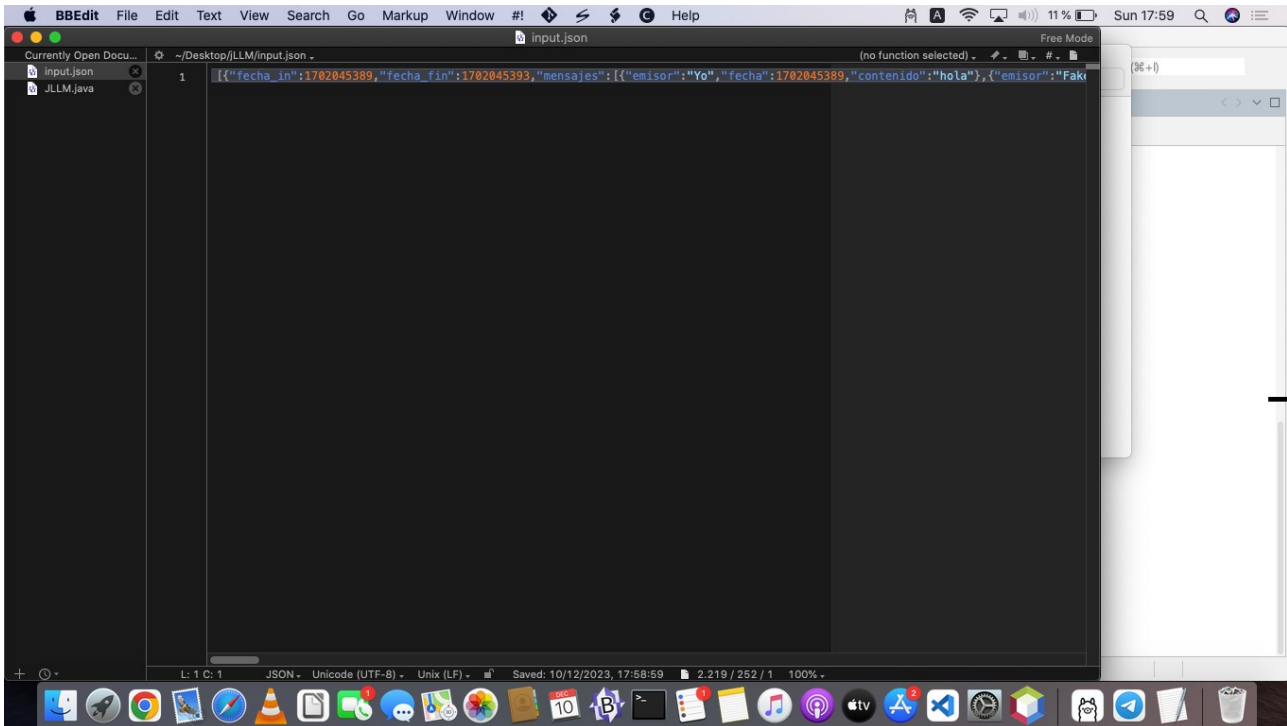
2 3 4

Exportacion con exito

- 1- Exportar
- 2- Importar
- 3- Salir

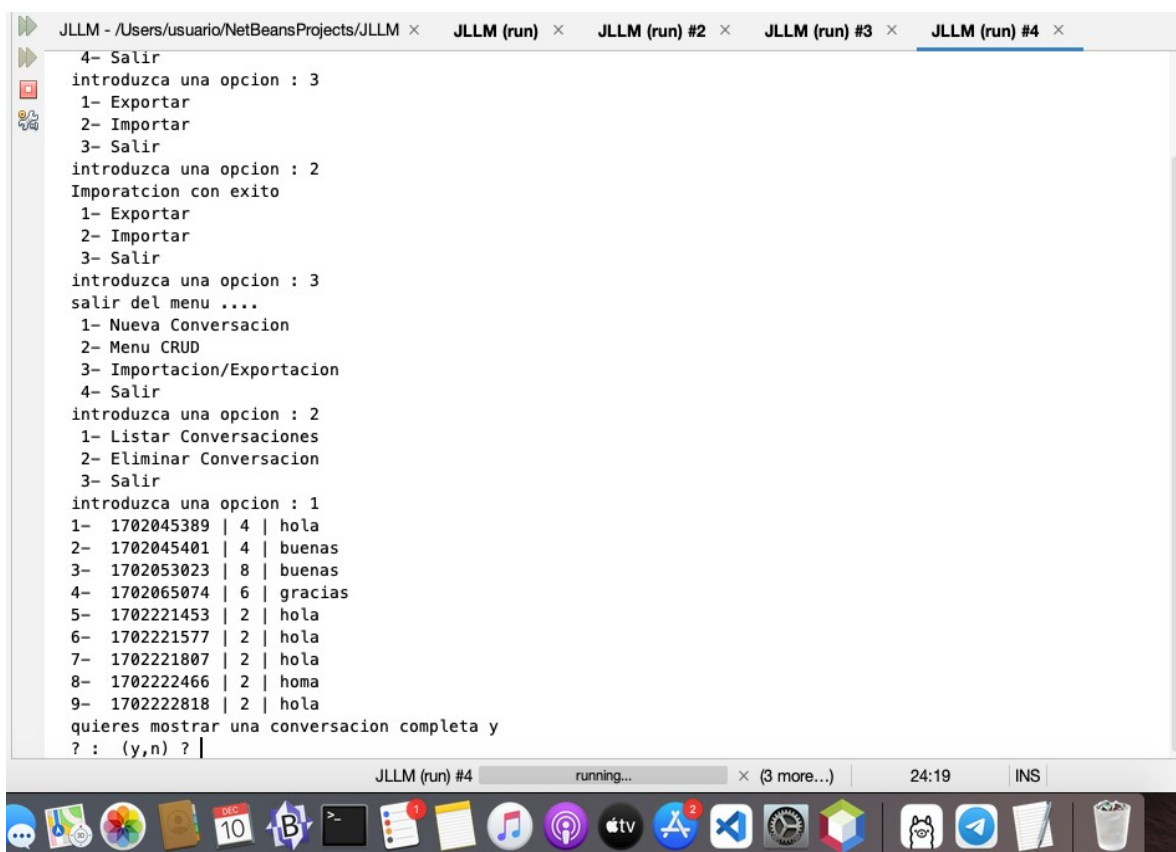
introduzca una opcion : |

Fichero input.json :



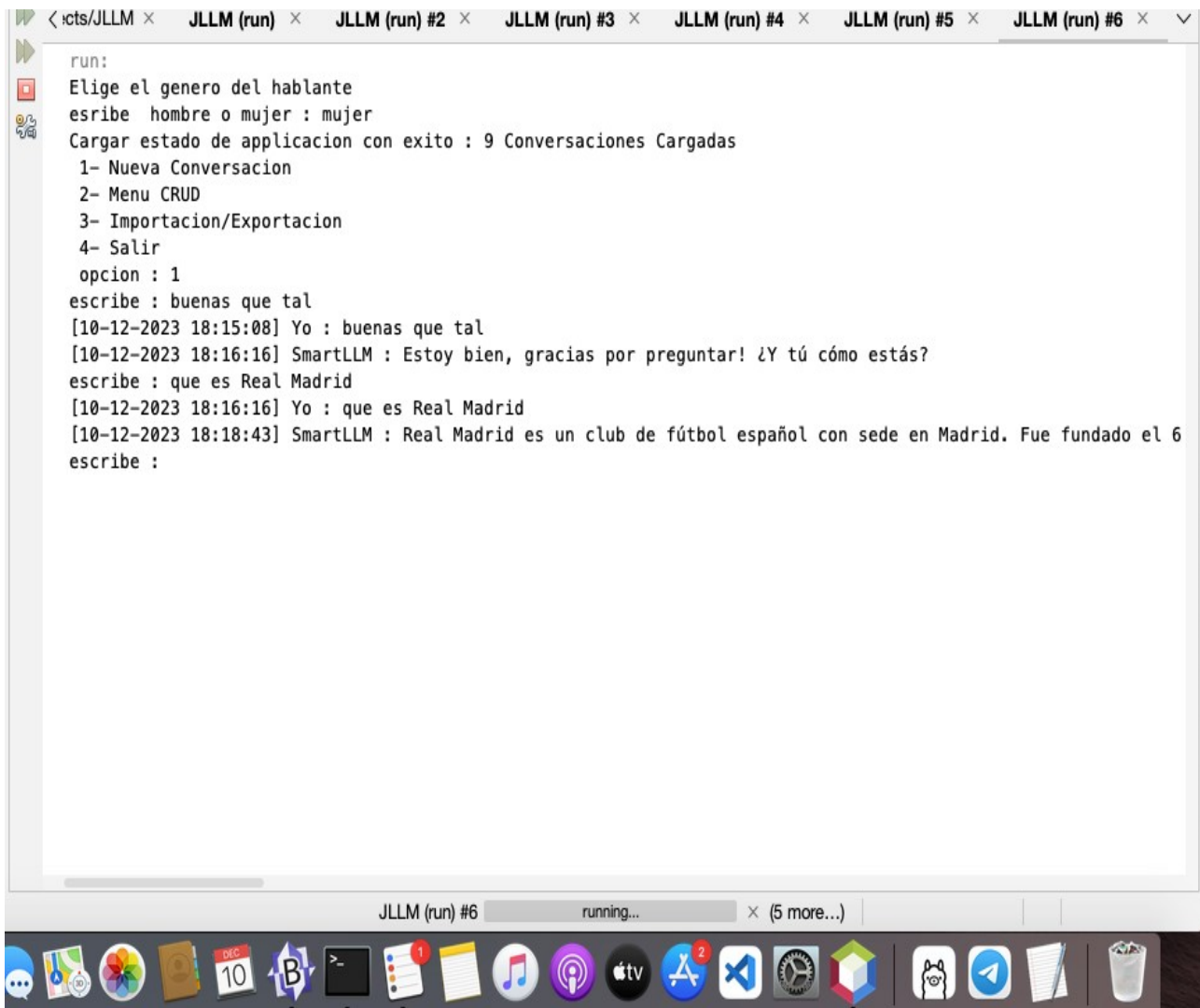
Importación:

La importación se puede realizar siempre y cuando las conversaciones que se vayan a importar no existan en nuestra aplicación, es decir, no deben ser conversaciones repetidas. A continuación, se muestra el proceso de importación de un archivo XML. Utilizando la biblioteca Jackson ObjectMapper, se ha utilizado StringBuilder para la concatenación de cadenas. El archivo 'input.xml' contiene las 9 conversaciones (en la primera ejecución). Después de eliminar una conversación, al realizar la importación, el número total de conversaciones vuelve a ser 9.



Caso 2: Ejecución con SmartLLM y VoiceConsoleView

Al principio, se le pide al usuario que elija el género del hablante (hombre o mujer). Cada opción del menú está reproducida con voz, se ha creado un método en la clase VoiceConsoleView que se denomina setVoice que tiene como argumento un String, al pasarlo se narra este String. Con el agente SmartLLM, se realizan llamadas y se consume la API que expone Ollama, pasando el texto a su método ollamaAPI.ask("modelo", entrada). La clase OllamaAPI interactúa con la API en el host proporcionado, y este método está implementado en el método speak de la clase SmartLLM. Este proceso tarda aproximadamente 3 minutos en obtener la respuesta. A continuación, se muestra un ejemplo de ejecución



```
run:
Elige el genero del hablante
escribe hombre o mujer : mujer
Cargar estado de aplicacion con exito : 9 Conversaciones Cargadas
1- Nueva Conversacion
2- Menu CRUD
3- Importacion/Exportacion
4- Salir
opcion : 1
escribe : buenas que tal
[10-12-2023 18:15:08] Yo : buenas que tal
[10-12-2023 18:16:16] SmartLLM : Estoy bien, gracias por preguntar! ¿Y tú cómo estás?
escribe : que es Real Madrid
[10-12-2023 18:16:16] Yo : que es Real Madrid
[10-12-2023 18:18:43] SmartLLM : Real Madrid es un club de fútbol español con sede en Madrid. Fue fundado el 6
escribe :
```


Caso 3: RandomCSVLLM

La ejecución con este modelo consiste en importar las frases de un archivo llamado RandomCSV con formato CSV. Este proceso está implementado en el método importarCSV de dicha clase. Para facilitar la importación, se ha creado una clase Frase con 3 atributos: tipo, longitud y contenido. A continuación, se muestra un ejemplo de su ejecución. En caso de la existencia de la coma en el parte contenido, lo he manejado con concatenar la tercera parte y la cuarta con “+” porque al final si existe una tercera comilla , la frase se divide en 4.

Nota: El archivo que trabaja con este modelo debe denominarse "RandomCSV.csv" y debe existir en la carpeta "jLLM" en el Escritorio, o este modelo no funcionará.

```
run:
Cargar estado de aplicacion con exito : 9 Conversaciones Cargadas
 1- Nueva Conversacion
 2- Menu CRUD
 3- Importacion/Exportacion
 4- Salir
introduzca una opcion : 1
escribe : hola
[10-12-2023 18:32:19] Yo : hola
[10-12-2023 18:32:21] RandomCSVLLM : ¡Qué alegría verte!
escribe : que tal?
[10-12-2023 18:32:21] Yo : que tal?
[10-12-2023 18:32:25] RandomCSVLLM : Eso es absolutamente cierto
escribe : adios
[10-12-2023 18:32:25] Yo : adios
[10-12-2023 18:32:27] RandomCSVLLM : Eso es absolutamente cierto
escribe : /salir
...Chat Closed
 1- Nueva Conversacion
 2- Menu CRUD
 3- Importacion/Exportacion
 4- Salir
introduzca una opcion :
```

Salir: Al elegir esta opción, se guarda el estado de la aplicación en el archivo jLLM.bin utilizando el método se hace la serialización de las conversaciones en dicho archivo y finaliza la ejecución de la aplicación.

```
Guardar estado de Aplicacion con exito
BUILD SUCCESSFUL (total time: 37 seconds)
```

Problemáticas:

En la clase VoiceConsoleView, al reproducirse la voz, todas las opciones se han reproducido simultáneamente. Por esta razón, he añadido un método sleep que se ejecuta después de cada llamada al método setVoice (que reproduce la voz) para detener el programa durante 3 segundos. No estoy seguro de si esto es correcto o si hay una manera más eficiente de hacerlo.

Como mencioné al principio, no estoy seguro del uso de las bibliotecas proporcionadas para implementar las clases SmartLLM y VoiceConsoleView, ya que creo que puede haber una manera más adecuada de hacerlo que lo que he utilizado.