SPEAKIT! V2.0

MANUAL DE USUARIO

Estructura general del sistema

Speakit es un sistema que lee documentos por salida de audio y los guarda indexados.

El sistema Speakit, permite ingresar un documento de texto, grabar todas las palabras que no estén registradas, y agregarlas a un diccionario que asocia palabras con su audio, y también permite reproducir las palabras contenidas en un documento.

En esta versión se agregó el módulo de indexación y almacenamiento de documentos, mientras que el módulo de audio se conservó y se mejoró. En una etapa posterior estos documentos se guardarán en forma comprimida.

El módulo Speakit, tiene la función de agregar el audio de una palabra. Además tiene la función de agregar un documento a la colección de documentos almacenados. Esta función realiza su tarea valiéndose del módulo documents. Puede almacenar documentos a pesar de que el módulo dictionary no contenga todas sus palabras. También tiene la función de obtener todas las palabras desconocidas de los documentos almacenados, lo cual sirve para grabar nuevas palabras.

En esta versión del sistema, audiofile sigue siendo de implementación secuencial pero el archivo que lo indexaba pasó a ser un Trie. El trie fue implementado sobre un archivo directo de registros por bloque que almacena los nodos del árbol y un archivo secuencial que se utiliza como índice para saber en que bloque buscar un nodo sabiendo su número. El archivo directo está implementado independientemente en la clase DirectRecordFile para poder ser reutilizado por otras estructuras que requieran el uso de un archivo directo.

El módulo *documentsstorage* tiene la funcionalidad de guardar, y buscar los documentos de texto guardados conociendo el offset dentro del archivo. Cuando se agrega un nuevo documento el módulo ftrs extrae los terminos relevantes, dejando de lado los q se repiten comunmente, tales como artículos, proposiciones, etc. Si se agregan al sistema un conjunto de documentos estos se indexan de una manera más óptima que ingresándolos de a uno, creando las listas invertidas sincronizadamente y utilizando un algoritmo de sort externo. Los documentos agregados, serán comprimidos en la próxima etapa con el módulo compression.

La búsqueda de documentos se realiza a través del módulo *ftrs* y se recuperan a través del módulo *documentsstorage*. Las listas invertidas del ftrs se almacenan en un archivo directo de registros por bloque que está indexado por un árbol B# para agilizar las búsquedas. El árbol B# está implementado en un módulo independiente y en el módulo ftrs se lo utiliza junto con un encoder que permite comprimir los términos en las hojas mediante front coding.

La garantía de funcionalidad y estabilidad del sistema está soportada por unas 200 pruebas automáticas, las cuales incluyen pruebas simples y pruebas de stress. Las pruebas de stress utilizadas garantizan por ejemplo que el árbol B# pueda indexar todas las palabras del idioma castellano (véase StressTreeTest con speakit/test/files/lemario.txt), u otras por ejemplo garantizan la robustez del sistema ftrs. Estas pruebas se pueden utilizar a modo de documentación técnica adicional para aprender sobre el funcionamiento y la utilización del sistema, por lo cual recomendamos su lectura y ejecución.

Abreviaturas utilizadas en la presente obra:

Tipos de datos:

B: Byte (complemento a dos en dos byte)
EC: Enteros Cortos (complemento a dos en dos byte)
E: Enteros (complemento a dos en cuatro bytes)
EL: Enteros Largos (complemento a dos en ocho bytes)
F: Fraccionarios (punto flotante)
C: Caracteres (con longitud exacta entre paréntesis)
V: Caracteres Variables (hasta 255) con prefijo de longitud

T: Texto (cantidad ilimitada de caracteres, incluyendo caracteres de control como salto de línea, retorno de carro, tabulación, fin de texto)
L: Lógicos (0: Falso o No, 1: Verdadero o Sí)

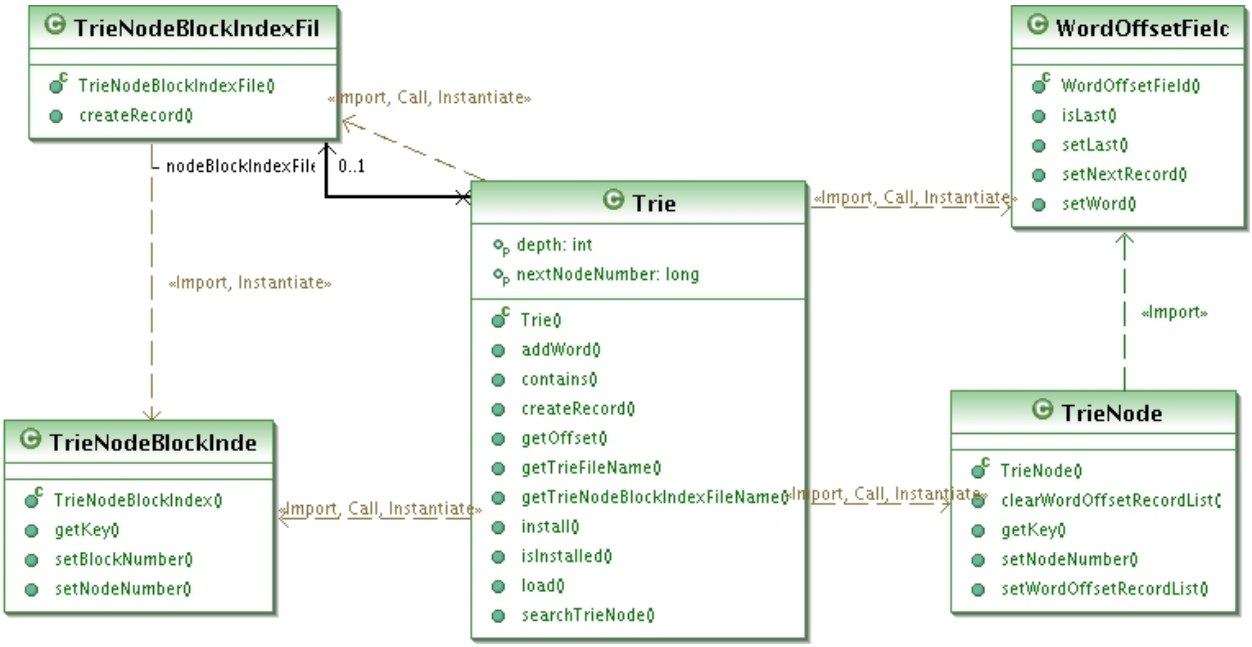
Estructuras genéricas de archivos de registros y de bloque

Archivo secuencial de registros (*SecuentialRecordFile*)

Representa a un archivo de registros de acceso secuencial. Permite insertar o buscar un registro. Las búsquedas se hacen en forma secuencial.

Trie

(Utilizado para el diccionario de audio)



Este modulo implementa un trie para el indice de "palabra-offset en el archivo de audio". Su profundidad es parametrizable y utiliza un archivo directo con registros variables por bloques (**DirectRecordFile**) para guardar los nodos del trie y, para acelerar la busqueda, otro archivo que contiene el numero de nodo y el bloque del archivo anterior donde se encuentra cada nodo. Asi, al buscar un nodo, se lo busca en el bloque adecuado y se evita recorrer secuencialmente el archivo de nodos del trie.

Definicion de los registros de los archivos:

Trie:
Conceptual: TrieNode**{(**numero_nodo**)i**,wordOffsetList**[(**palabra, proximo_nodo, es_ultimo**)+]}**

Logica: TrieNode**{**numero_nodo: EL, wordOffsetList**[**palabra: V, proximo_nodo:EL, es_ultimo:L**]}**

Indice de nodos por bloque
Conceptual: TrieNodeBlockIndex**{(**numero_nodo**)ie**, **(**numero_bloque**)i}**
Logica: TrieNodeBlockIndex**{**numero_nodo**:**EL, numero_bloque:E**}**

Archivo en bloques (*BasicBlockFileImpl*)

Esta clase permite el manejo de archivos por bloques de una forma básica. Tiene primitivas para crear un nuevo archivo de bloques, cargar uno previamente creado, agregar un bloque de bytes, obtener un bloque de bytes y guardar un bloque de bytes.

Archivo Directo de registros (*DirectRecordFile*)

Representa un archivo de registros de acceso directo. Tiene primitivas para crear un nuevo archivo de registros, cargar uno creado previamente, insertar un registro en un bloque, obtener un registro de un bloque, actualizar un registro, verificar si un registro existe y agregar un nuevo bloque.

Arbol B#

Representa un archivo de registros organizado en forma de árbol B#. Utilizado normalmente como índice de un archivo directo. Tiene las mismas primitivas que un archivo de registro, como por ejemplo insertar registro y obtener registro a partir de una clave. Si se utiliza el árbol como un índice, los registros que contiene son registros de índice, es decir, un registro consistente en una clave y el número de bloque del archivo directo.

Este árbol B# puede utilizar un encoder para guardar los datos en las hojas. El encoder puede utilizarse para comprimir, para encriptar o para hacer cualquier transformación de datos. En este proyecto se utiliza para hacer front coding de las palabras guardadas en el índice del ftrs.

Definición de nodos del arbol:
Nodo Indice:
Definicion conceptual:
NodoIndice**{(**nro_nodo**)i**,nivel,**(hijo_izquierd)ie**,elemento_indice**[**termino,**(**hijo_derecho)ie**]+}**
Definicion logica: NodoIndice**{**nivel:E, hijo_izquierdo:E, Array**[**cantidad_elementos:E, hash_cantidad_elementos:B, elemento_indice**(**termino:V,nro_nodo:E**)+]}**

Nodo Hoja:
Definicion conceptual:
NodoHoja**{(**nro_nodo**)i**,nivel**=0**,elemento_hoja**[**InvertedIndex_IndexRecord**]+**,**(**siguiente_nodo**)ie}**
Definicion conceptual: NodoHoja**{**Array**[**cantidad_elementos:E, hash_cantidad_elementos:B, **(**InvertedIndex_IndexRecord**)+]**, siguiente_nodo:E**}**

(nota: la definición de InvertedIndex_IndexRecord está en la sección de Indice Invertido)

Estructuras genéricas de campos utilizadas en esta versión

CompositeField

Campo abstracto que representa un campo compuesto por otros campos.

ArrayField

Campo compuesto por un arreglo de valores del mismo tipo y el campo de control que indica la cantidad de valores almacenados.

Algunos archivos específicos de este sistema

Índice de palabras del diccionario de audio

Implementado con un trie, cuyos nodos se almacenan en un archivo de registros directo.

Índice de nodos del trie del diccionario de audio

Implementado con un archivo secuencial. Utilizado para encontrar los nodos del trie de forma más eficaz.

Archivo de audios de palabras

Implementado con un archivo de registros secuencial. Almacena los audios de las palabras.

Índice de listas invertidas

Implementado con un árbol B#. Contiene registros de índice con clave y número de bloque. En este caso la clave es el término, y el número de bloque apunta a un bloque del archivo de listas invertidas.

Archivo de listas invertidas

Implementado con un archivo de registros directo por bloques. Almacena las listas invertidas. Las listas invertidas son un registro compuesto por un término y un arreglo de números de documentos y frecuencia. Los números de documentos coinciden con el offset donde se encuentra el documento en el archivo de documentos.

Archivo de documentos

Implementado con un archivo de registros secuencial. Almacena los documentos ingresados por el usuario.

Estructuras que relacionan los índices con los archivos directos/secuenciales

Diccionario de audio

Implementado con un trie. Cuando el audio de una palabra se busca en el diccionario primero se busca el número de offset del audio en el trie que indiza al archivo de audio. Una vez encontrado el offset con el trie, el diccionario de audio devuelve el audio de la palabra buscada.

Los nodos del trie están guardados en un archivo de registros en bloques, y además hay un índice de estos nodos, de manera que cuando se busca cada nodo primero se consulta al índice para saber en que bloque está y luego se busca el nodo en el bloque.

Indice invertido

Este módulo tiene una clase llamada InvertedIndex implementa el índice invertido que sirve para la recuperación de textos. En él se guardan los términos con sus respectivas listas invertidas.
Está compuesto por un archivo de datos y por un índice primario, exahustivo y selectivo.
El de datos está implementado con un archivo directo de registros variables.
El archivo indice es un bsharp del paquete **speakit.io.bsharptree**.

Definición de los registros:
Indice Invertido:
Conceptual: InvertedIndexRecord**{ (**termino**)i**, cantidad_documentos, maxima_frecuencia_local, InvertedList**[** Ocurrence**((**documento**)ie**, frecuencia_local**)+]}**
Logica: InvertedIndexRecord**{**termino: V, maxima_frecuencia_local: E, InvertedList**[** cantidad_ocurrencias:E,hash_cantidad_ocurrencias:B,Ocurrence**(** offset_documento:EL, frecuencia_local:E**)],** hash_registro:B**}**

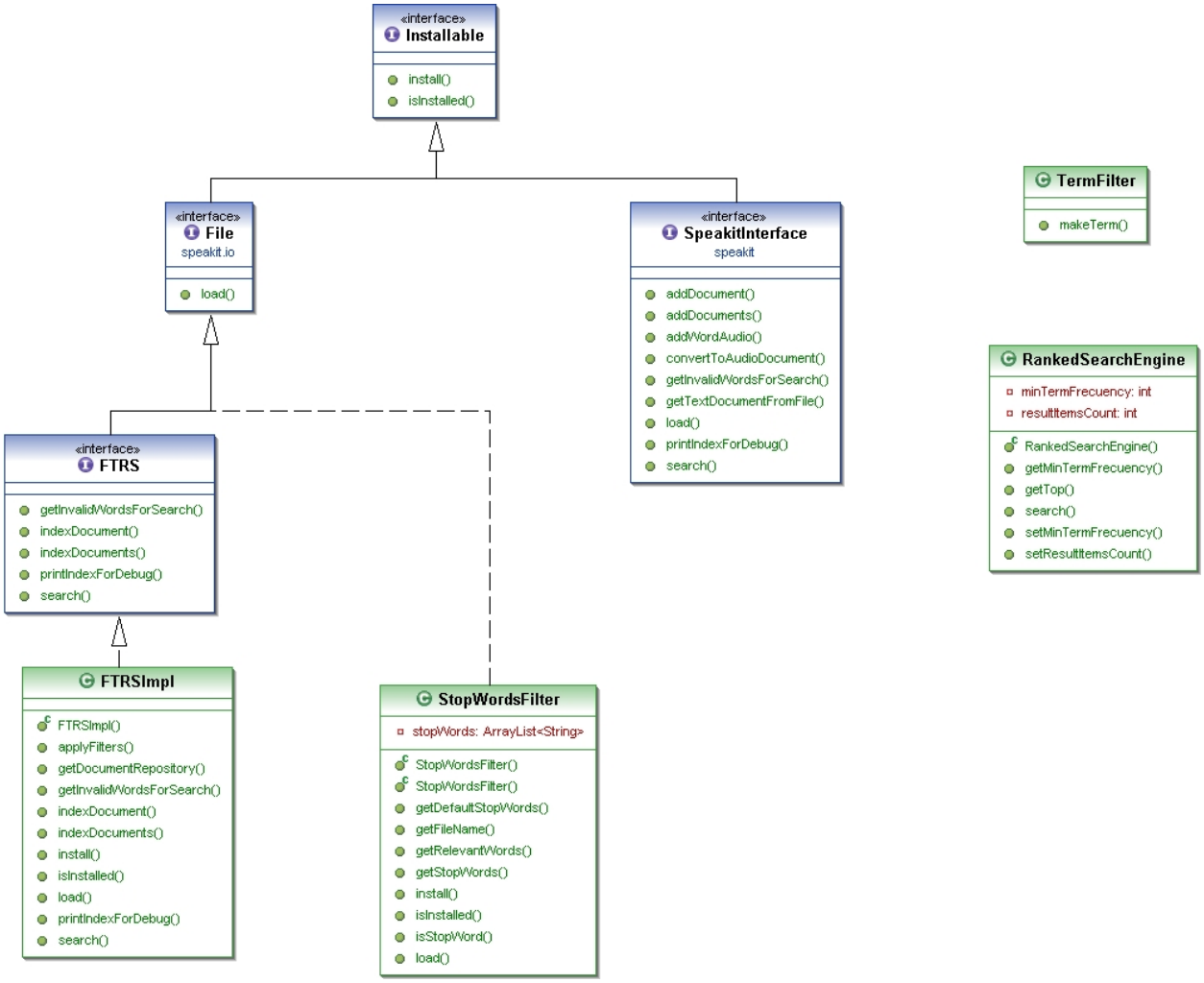
Indice del Indice Invertido:
Conceptual: InvertedIndex_IndexRecord**{(**termino**)i**,**(**bloque_datos**)ie}**
Logica: InvertedIndex_IndexRecord**{** FrontCodedStringField**[**cant_caracterers_coincidentes: EC,caracteres_finales: V**]**: V, numero_bloque: EL**,** hash_registro:B**}**

Resolución de consultas:

El sistema resuelve consultas utilizando el método vectorial, para lograrlo procede de la siguiente forma:

El módulo de FTRS recibe un documento de texto como consulta, lo hace pasar por unos filtros de palabras dejando al documento sólo con los términos relevantes. Estos filtros son los mismos que se usan para una indexación de documentos.
Luego para cada termino del documento se consulta al indice invertido y se obtiene un registro con lista de apariciones de ese en los documentos del sistema, ordenadas por frecuencia local.
Se ordena la lista de registros por relevancia del término. Un término es mas importante que otro si aparece en menos documentos.
Por cada registro se recorre la lista de apariciones, se obtiene el identificador de cada documento y se lo agerga a la lista de resultados. Estos documentos se agregan hasta que la lista de resultados llegue al limite de 10 documentos.
Luego por cada id de documento se consulta al modulo DocumentStorage para obtener el texto completo.

Diagrama de clases del indice invertido



Indexación de multiples documentos:

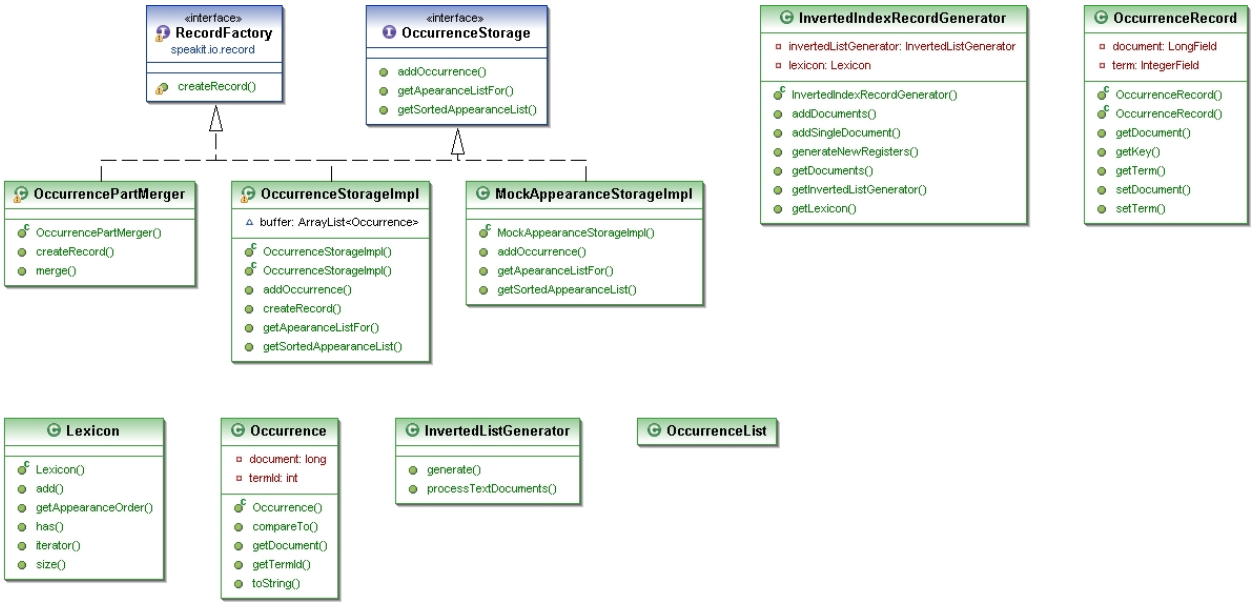
Obtencion de las listas invertidas de cada termino

Para la indexación de documentos se opto por el mecanismo de ordenamiento denominado sort Interno, que va cargando en memoria una porción del archivo desordenado y genera n -1 particiones ordenadas de igual tamaño y una que puede ser menor. El mecanismo escribe las particiones en un archivo secuencial, usando registros de longitud fija donde se almacena [(nro_de_termino) (nro_de_documento)] . Soportando repetidos para poder procesar la frecuencia de cada termino dentro de los documentos.
Estas particiones se unen mediante un mecanismo de merge que consulta las particiones ordenadas previamente generadas y devuelve un único archivo ordenado, del mismo tipo que el generado por el algoritmo de ordenamiento.

Stop Words:

Se conoce como stop words a las palabras que pueden encontrarse comunmente en casi cualquier documento de texto, tales como artículos, preposiciones, conectores, etc.
A la hora de realizar una busqueda, si se incluyen las stop words, estas palabras estarian presentes en multiples documentos, ensuciando asi los resultados. Para evitar esto, no se tienen en cuenta a la hora de indexar documentos como asi tampoco a la hora de realizar busquedas.

Diagrama de clases del paquete indexador



Problemas encontrados y soluciones propuestas

Integridad de datos:

Trabajando con archivos muchas veces sucedía que por un mal funcionamiento en los algoritmos de persistencia se graben y carguen datos en disco de forma erronea o incompleta, provocando fallas en los lugares donde se procesaban. Era muy dificil detectar si el problema provenía de datos o de la logica de procesamiento. Para facilitar la localización de la fuente del problema se implementó un método que permitía detectar datos corruptos al momento de leerlos de disco, así por lo menos sabemos si el origen es o no de datos.
El método consiste en guardar al final de cada registro un campo byte con un hash de todos sus datos. Luego al cargar un registro de disco se verifica si el campo hash coincide con el calculado sobre todos los datos del registro. Si no coincide el dato es inválido, de lo contrario es altamente probable de que sea válido.

A veces los datos corruptos hacían imposible cargar todo el registro y calcular el hash. Por ejemplo cuando el registro tenia campos de tipo array y se corrompía el campo de control que indicaba la cantidad de elementos, a veces era tan elevado que el sistema se quedaba sin memoria. Par solucionarlo aplicamos el mismo método de hash usado en registros para cada campo de control de ese tipo.

MANUAL DE USUARIO

Instrucciones de uso

Instalación del sistema

Para instalar y correr el sistema se requiere de la herramienta **Apache Ant**. Esta herramienta puede conseguirse mediante el gestor de paquetes de su distribución de Linux o descargandolo de la página **http://ant.apache.org** . Si se elige esta última opción se deben seguir las instrucciones de instalación de la herramienta proporcionadas en la página.

Para compilar o correr la aplicación, utilizando la consola del sistema operativo, ingresar al directorio **speakit** y ejecutar alguno de los siguientes comandos:

ant clean - Usando este comando se borran las carpetas utilizadas para compilar y distribuir la aplicación.

ant compile - Con este comando se compila la aplicación, generando los archivos .class dentro de la carpeta **build** del proyecto.

ant run - Al ejecutar este comando se compila y se ejecuta la aplicación.

ant - Si se ejecuta el comando ant, sin utilizar ningún parametro, por default se ejecuta el comando **ant run**.

Utilización del sistema

Al iniciar la aplicación podemos ver la siguiente pantalla:

```
Menu Principal
1.- Procesar un archivo de Texto
2.- Procesar varios archivos de Texto
3.- Reproducir Archivo
4.- Realizar una consulta


0.- Salir
```

Aqui tenemos 5 opciones:

- **Procesar archivo de Texto**: En este módulo el sistema pregunta por un archivo de texto para ser leido. Si el archivo está dentro de la carpeta de la aplicación no es necesario ingresar la ruta al mismo, solo se debe ingresar el nombre. Una vez ingresado el nombre del archivo, el sistema reconoce las palabras que ya están agregadas al diccionario y las que no se encuentren en este, se requerirá que sean grabadas por el usuario. Luego de grabar cada palabra el sistema reproducirá la palabra, preguntará al usuario si la palabra se grabó correctamente y dará la posibilidad de regrabarla si asi lo desea el usuario.

```
Speak It!
Menu Principal
1.- Procesar un archivo de Texto
2.- Procesar varios archivos de Texto
3.- Reproducir Archivo
4.- Realizar una consulta


0.- Salir
1
Leer archivo de Texto

Ingrese la ruta a continuación:
(Si su archivo es '1.txt' sólo presione ENTER)
hamlet.txt
El documento contiene palabras desconocidas, que deberá grabar a
continuación.

Palabra 'ser'. (ENTER para grabar).

Grabando... (ENTER para detener).
```

```
Reproduciendo...(ENTER para confirmar. N para volver a grabar)

Palabra 'o'. (ENTER para grabar).

Grabando... (ENTER para detener).

Reproduciendo...(ENTER para confirmar. N para volver a grabar)

Palabra 'no'. (ENTER para grabar).

Grabando... (ENTER para detener).

Reproduciendo...(ENTER para confirmar. N para volver a grabar)

Palabra 'esta'. (ENTER para grabar).

Grabando... (ENTER para detener).

Reproduciendo...(ENTER para confirmar. N para volver a grabar)
n
Palabra 'esta'. (ENTER para grabar).

Grabando... (ENTER para detener).

Reproduciendo...(ENTER para confirmar. N para volver a grabar)

Palabra 'es'. (ENTER para grabar).

Grabando... (ENTER para detener).

Reproduciendo...(ENTER para confirmar. N para volver a grabar)

Palabra 'la'. (ENTER para grabar).

Grabando... (ENTER para detener).

Reproduciendo...(ENTER para confirmar. N para volver a grabar)

Palabra 'cuestion'. (ENTER para grabar).

Grabando... (ENTER para detener).

Reproduciendo...(ENTER para confirmar. N para volver a grabar)

El documento fué agregado con éxito.
```

Si se ingresa un nombre de archivo que el sistema no puede encontrar, se emitirá un mensaje de error y la aplicación volvera al menu inicial.

```
Speak It!
Menu Principal
1.- Procesar un archivo de Texto
2.- Procesar varios archivos de Texto
3.- Reproducir Archivo
4.- Realizar una consulta


0.- Salir
1
Leer archivo de Texto

Ingrese la ruta a continuación:
(Si su archivo es '1.txt' sólo presione ENTER)
lalala.txt
No pudo encontrarse el archivo 'lalala.txt'.
Speak It!
Menu Principal
1.- Procesar un archivo de Texto
2.- Procesar varios archivos de Texto
3.- Reproducir Archivo
4.- Realizar una consulta


0.- Salir
```

- Procesar varios archivos de Texto: Funciona igual a la opción anterior, sólo que permite ingresar mas de un documento a la vez. Los nombres de archivo se deben escribir uno a continuación del otro separándolos por una coma.

```
Menu Principal
1.- Procesar un archivo de Texto
2.- Procesar varios archivos de Texto
3.- Reproducir Archivo
4.- Realizar una consulta


0.- Salir
2
Ingrese cada una de las rutas de los documentos que desea ingresar
separadas por coma
hamlet.txt,speakit.txt

Los documentos ingresados contienen palabras desconocidas que deberá
grabar a continuación
Palabra 'una'. (ENTER para grabar).
```

- **Reproducir Archivo**: Con esta opción podemos reproducir las palabras grabadas con anterioridad en la aplicación. Se solicitará nuevamente el nombre del archivo ingresado y el sistema indicará por pantalla las palabras a reproducir consecutivamente y se escuchará su audio a continuación.

```
Speak It!
Menu Principal
1.- Procesar un archivo de Texto
2.- Procesar varios archivos de Texto
3.- Reproducir Archivo
4.- Realizar una consulta


0.- Salir
3
Ingrese la ruta a continuación:
(Si su archivo es '1.txt' sólo presione ENTER)
hamlet.txt
Se va a reproducir el siguiente documento
Ser o no ser: ésta es la cuestión!
```

- Realizar una consulta: Escriba los términos por los que quiere buscar y el sistema desplegará un listado de los documentos más relevantes.

```
Speak It!

Menu Principal
1.- Procesar un archivo de Texto
2.- Procesar varios archivos de Texto
3.- Reproducir Archivo
4.- Realizar una consulta


0.- Salir
4
Ingrese la consulta
cuestion
Los documentos encontrados para la consulta realizada se muestran a
continuacion:
1 : Ser o no ser: ésta es la cuestión!....
```

Si quiere reproducir uno de los documentos listados, presione 1 y a continuación escriba el número de documento que quiere escuchar y el sistema se lo leerá.

```
Si desea reproducir algun documento presione 1
Para realizar una nueva consulta presione 2
Para ir al menu principal presione 0
1
Elija el numero de documento que desea reproducir
1
Se va a reproducir el siguiente documento
Ser o no ser: ésta es la cuestión!
```

- **Salir**: Esta opción permite salir de **SpeakIt**.

```
Menu Principal
1.- Procesar un archivo de Texto
2.- Procesar varios archivos de Texto
3.- Reproducir Archivo
4.- Realizar una consulta


0.- Salir
0
Terminado.
```

Ejemplos

Junto a la distribución del sistema se incluyen 2 test cases con documentos de ejemplos y los archivos binarios correspondientes. Para poder usar efectivamente estos ejemplos se deben colocar (junto a los binarios) dentro de la carpeta raiz del proyecto.

Test Case 1

De estos test cases, todos los archivos del primero han sido grabados por nosotros (archivos de ejemplo desde el 1 al 10).
Para probar, hemos realizado las siguientes busquedas con su resultado:

Búsqueda	Resultado
ser	0 resultados. Es una ***stop word***
la cuestion	texto de ejemplo1.txt
ausente	texto de ejemplo5.txt
presente en la república	texto de los archivos ejemplo5.txt, ejemplo7.txt, ejemplo8.txt, ejemplo9.txt
jefe de la república	texto de los archivos ejemplo7.txt,ejemplo8.txt,ejemplo9.txt

no sabía lo que hacia	0 resultados. Las palabras no estan indexadas
corona de espinas	texto de los archivos ejemplo3.txt,ejemplo10.txt

Test Case 2

Hemos incluido tambien 3 archivos más de ejemplo, para probar el ingreso de los archivos y su grabación en el sistema. Al intentar ingresar estos 3 archivos, el sistema pedira que ingresemos los siguientes terminos (los otros no son pedidos porque ya estan ingresados)

evitar, robo, por, sus, politicos, los, discipulos, pudieron, vez, atrapado, soldados, romanos, su, calaban, fondo, cada, persona, lugar