Encapsular el acceso a una aplicación BASIC/MS-DOS

Sistemas Legados

Pedro Allué Tamargo (758267) Juan José Tambo Tambo (755742)
Jesús Villacampa Sagaste (755739)

11 de noviembre de 2020

Índice

1. Introducción

En esta práctica se pide la realización de un *wrapper* sobre una aplicación legada de un sistema *MS-DOS*. Para ello se va a utilizar un emulador y mediante capturas de pantallas de la interacción con la misma y un software de reconocimiento de caracteres (*OCR*) se van a extraer los datos.

2. Esfuerzos invertidos

- Pedro Allué Tamargo: Creación de la API REST, organización de la estructura de proyectos, implementación de la clase *Wrapper*, redacción de la memoria.
  - Tiempo invertido: 10 horas
- Juan José Tambo Tambo:
- Jesús Villacampa Sagaste:

3. Descripción de la aplicación legada

Para interaccionar con la aplicación legada se va a utilizar el emulador *DosBox* (el incluido junto al enunciado de la práctica). Para ejecutar la aplicación se va a ejecutar el *script* `database.bat` que ejecuta el emulador y la aplicación legada.
La aplicación legada consiste en un programa de *MS-DOS* que gestiona una biblioteca de cintas con distintos programas y/o juegos (Figura 1).

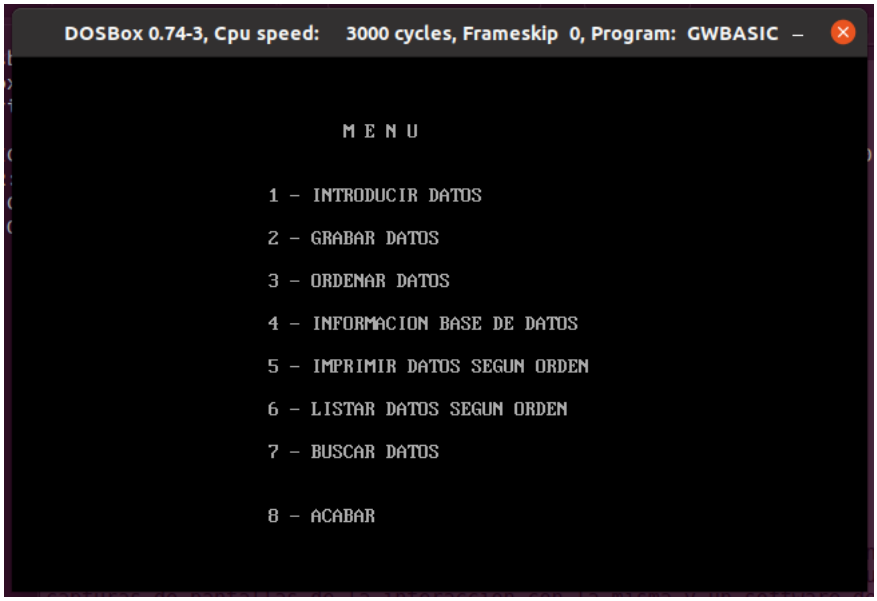

Figura 1: Captura de pantalla del menú principal de la aplicación legada

Con esta aplicación el usuario podía introducir nuevos registros que hacían alusión a cintas con los campos: *Nombre*, *Tipo*, *Cinta*. También se añade un campo que gestiona la propia aplicación correspondiente al número de registro que le corresponde en la base de datos.

4. Tecnología elegida

Se ha elegido *Java* como lenguaje de programación para esta tarea. Para implementar el servidor web se ha utilizado el *framework SpringBoot* por su facilidad de uso. Se va a utilizar un servidor de aplicaciones *Apache Tomcat* incluido en el *framework* de *Spring*. Se va a exponer una API REST para obtener los datos del servidor. Para la creación de la página web y la interacción con la *API REST* del servidor se va a utilizar *Vue.js*, un *framework* de *JavaScript*.

5. Implementación del Wrapper

5.1. Preparación del entorno

Para la realización de esta práctica se va a utilizar el sistema operativo *Ubuntu*, se debe instalar el emulador *DosBox*. Para ello se ejecutarán las siguientes órdenes:

```
sudo apt update
sudo apt -y install dosbox
```

En primer lugar se deberá configurar el emulador *DosBox* para que monte el directorio de la aplicación legada en un disco virtual (*C*) y para que ejecute el programa. Para ello se añadirán las siguientes líneas al fichero de configuración ubicado en el directorio `$HOME/.dosbox`. Se añadirán las siguiente líneas al fichero de configuración (`dosbox-0.74-3.conf`) en la sección `[autoexec]`:

```
[autoexec]
# Lines in this section will be run at startup.
# You can put your MOUNT lines here.

MOUNT C $HOME/.dosbox/c/
C:
cd Database
GWBASIC.BAT
```

Tras eso se va a proceder a crear el directorio `$HOME/.dosbox/c` con el contenido de la aplicación legada.

Para poder utilizar el reconocimiento de caracteres (*OCR*) se debe instalar la librería *Tesseract* para el Sistema Operativo. Para instalarla se ejecutarán los siguientes comandos:

```
sudo apt update
sudo apt -y install tesseract-ocr
```

De cara al reconocimiento de caracteres en las capturas de pantallas se va a utilizar otro binario para conseguir la posición de la ventana del emulador. Tras una búsqueda en la web se encontró un *POST* en *StackExchange*[1] que comentaba el uso de la herramienta *wmctrl*. Se va a proceder a instalar el binario *wmctrl* mediante las siguientes órdenes:

```
sudo apt update
sudo apt -y install wmctrl
```

5.2. Frontend del Wrapper

Para la realización de la parte del *frontend* se ha utilizado *Vue.js*, un *framework* de *JavaScript* de código abierto que permite la construcción de interfaces de usuario de una manera sencilla y agradable.
Este *framework* nos permite mostrar el contenido de manera dinámica en la página web a través de llamadas a la *API REST* del servidor.
Para la elaboración del código en *HTML*, se ha utilizado la herramienta *Bootstrap*, de código abierto y de uso gratuito que dispone de plantillas *HTML* y *CSS* para interfaces de usuario que permite que éstas sean usables y accesibles. Esta herramienta cuenta con plantillas de todo tipo desde distintas estructuras como pueden ser listas, tablas, filas o columnas hasta botones ya diseñados, lo que ha permitido que la implementación haya sido tanto poco costosa como intuitiva para el usuario.
Para el proceso de desarrollo se ha utilizado la herramienta `npm`, que es un gestor de dependencias de *JavaScript*. Sirve para instalar y gestionar versiones de paquetes y librerías, y permite ver en tiempo real el resultado de la página web en un navegador web tan solo guardando los cambios efectuados en el código.

Metemos captura de como queda la página??? **SI**

Para integrar este *frontend* con el código del servidor de *backend* se ha creado un *script BASH* para compilar y copiar el resultado de este proyecto al servidor *backend*. El código del *script* es el mostrado en el código 5.2.

[1] `https://unix.stackexchange.com/questions/14159/how-do-i-find-the-window-dimensions-and-position-accurately-including-decora`

```
#!/bin/bash

# Build and copy project to Wrapper_MSDOS static folder

# Variables
dist_dir="./dist"
wrapper_dir="../Wrapper_MSDOS"
static_dir="$wrapper_dir/src/main/resources/static"

# Script
npm run build

# If compilation returns exit code = 0 then copy dist to wrapper's static
    directory
if [ $? -eq 0 ]; then
    rm -rf $static_dir/*
    cp -r $dist_dir/* $static_dir
fi
```

5.3. Backend del Wrapper

El lenguaje utilizado para la realización de la práctica ha sido *Java*, ya que, además de ser un lenguaje muy flexible y cómodo para los integrantes del grupo, se tiene experiencia con las herramientas que se utilizarán durante el desarrollo de la práctica.

Esto es redundante porque se ha comentado en el apartado de *Tecnología elegida.*

Para la realización de la parte del *Backend* se ha utilizado el *framework SpringBoot*. Se ha escogido esta tecnología principalmente debido a que *Spring* incluye un servidor *apache Tomcat*, lo que facilita mucho su configuración y despliegue. Para la integración de *Spring* se ha utilizado *Gradle*, un gestor de dependencias simple y fácil de utilizar que se ha utilizado anteriormente en otras asignaturas.
COMENTAR ALGO MÁS DE LAS TECNOLOGÍAS?

Como se ha mencionado anteriormente, se expone una *API REST* mediante la cual el cliente puede acceder a distintos recursos proporcionados por el servidor a través de las opciones que se muestran en la página principal. Para ello, se han creado tres *endpoints*:

- */filterByName*: Obtiene todos los registros cuyo nombre coincide con el nombre que se le pasa como parámetro.
- */filterByTape*: Obtiene todos los registros que pertenecen a la tapa cuyo nombre se pasa como parámetro.
- */getRecords*: Obtiene el número de registros almacenados en la aplicación.

Para poder abrir el emulador desde la aplicación se ha utilizado el comando *dosbox*, el cual se es inicializado en cada llamada a la *API REST* mediante el comando *ProcessBuilder*. Al final de cada llamada, el proceso es destruido mediante el comando *destroy*. Tras ser llamado, se realiza una espera de 5 segundos para que todo el proceso se inicie de forma correcta.

La comunicación entre el servidor y el programa *DosBox* se realiza mediante la clase *Robot* de *Java* (java.awt.Robot). De esta manera se permite generar eventos de ratón y teclado en el servidor, comunicándose de esta manera con el emulador *DosBox*. El problema de la clase *Robot* es que utiliza una objetos de tipo *Key* para realizar las distintas acciones. Para poder indicar a la clase *Robot* la acción a realizar en función de la tecla o acción de ratón utilizada, se ha creado otra clase llamada *RobotAdapter* inspirada en un código encontrado en la web.[2]

[2] `https://stackoverflow.com/questions/1248510/convert-string-to-keyevents`

Para obtener y procesar la respuesta del emulador, se utiliza *Tesseract*, una herramienta de reconocimiento de caracteres (*OCR*). Inicialmente se ha utilizado el *dataset* adjuntado con los archivos de la práctica, el cual no reconocía correctamente algunos caracteres, por lo que se decidió probar otro conjunto de datos de entrenamiento para observar si existía una mejora en los resultados. Se utilizó un *dataset* obtenido de *Github*[3], con el cual se obtuvieron peores resultados que con el inicial, por lo que finalmente se utilizó el conjunto proporcionado con los archivos de la práctica. Aún así, algunas palabras y caracteres no son reconocidas de forma correcta por el *OCR*, por lo que los resultados pueden ser erróneos en ocasiones.
Para indicar al objeto de la clase *OCR* el lugar en la pantalla que debe procesar, se ha utilizado el comando *wmctrl*, el cual obtiene la posición exacta de cada una de las ventanas que aparecen en el escritorio en el momento de su invocación. Para obtener la de la aplicación *DosBox*, se utiliza el comando *wmctrl* mediante `ProcessBuilder(WMCTRL_PATH, lG")` y se analiza la salida producida con *BufferReader*.De esta manera se obtiene las coordenadas x e y las cuales se utilizan para crear un objeto de tipo *BufferedImage* mediante una función de la clase *Robot* `robot.createScreenCapture`. Este objeto es analizado por el *OCR* mediante la función `ocr.doOCR`, la cual devuelve el contenido de la imagen en formato *String*

[3] `https://github.com/tesseract-ocr/tessdata/blob/master/spa.traineddata`