FE-Monitor: Explorador de archivos con consola de comandos

Miguel Pérez Alonso

Mayo 2021

Resumen– FE-Monitor es un explorador de archivos que funciona sobre sistemas operativos Windows. Esta aplicación permite al usuario tener varias carpetas abiertas de manera simultánea sin tener que preocuparse por como colocarlas en el escritorio, ya que la aplicación las organiza en un total de tres pestañas donde cada pestaña puede tener abiertas dos carpetas. Además, ofrece al usuario la posibilidad de crear accesos directos a sus carpetas favoritas para facilitar el acceso a las mismas. Pero la parte más importante de esta aplicación es que ofrece un modo desarrollador que habilita una consola de comandos en la parte inferior de la ventana, permitiendo al usuario utilizar comandos de Windows y una selección de comandos Linux, además de ofrecer unos comandos con una sintáxis totalmente nueva que permite operar directamente sobre las carpetas que se encuentren abiertas en las ventanas de la aplicación sin la necesidad de especificar la ruta de las mismas.

Palabras clave– Explorador de archivos, Windows, Comandos

Abstract– FE-Monitor is a file explorer which works over Windows operative systems. This application allows the user to have multiple folders opened simultaneously, without having to care about how to dispose them over the screen, the application organizes them in three different tabs, and every tab can have two folders opened. In addition, it offers to the user the possibility to create shortcuts to his favorites folders to ease the access to them. But the most important part of this application is the developer mode which enables a command console at the bottom of the application window, allowing the user to use Windows commands, a selection of Linux commands and a complete new commands with a new syntax which allows the user to operate directly with the opened folders without needing to specify the path.

Keywords– File explorer, Windows, Commands

1 Introducción

Es muy común encontrar a gente que en su día a día utiliza el explorador de archivos de su sistema operativo, ya sea en su puesto de trabajo o en su vida personal. Independientemente del nivel de conocimientos tecnológicos del usuario, todos deben lidiar con esta herramienta para gestionar los archivos y directorios de su ordenador.

En muchas ocasiones, pueden darse situaciones en las que tenemos que operar sobre dos carpetas diferentes, teniendo que abrir dos ventanas independientes del explorador de archivos y colocarlas de manera que nos facilite el movimiento de archivos o carpetas entre estas.

Así surgió la idea de FE-Monitor, una aplicación escritorio pensada para funcionar en sistemas operativos Windows, que nos facilitará la operabilidad entre carpetas de nuestro ordenador. Es importante destacar que esta aplicación pretende llegar al mayor número de usuarios posibles, tanto expertos en tecnología como usuarios sin formación específica en este campo, así como para gente mayor y estudiantes. Lo que pretende FE-Monitor es facilitar la gestión de ficheros respecto al sistema de explorador de archivos actual, pudiendo tener diferentes secciones de la pantalla mostrando el contenido de diferentes directorios, ya que muchas veces nos enfrentamos a la situación de tener que traspasar archivos de una carpeta a otra y no es un proceso ágil a no ser que estés acostumbrado.

El hecho de que la aplicación opere sobre el sistema operativo Windows permite centrar esfuerzos en desarrollarla sin preocupaciones de compatibilidad con otros sistemas, consiguiendo emplear la mayor parte del tiempo en mejorar su operabilidad con el sistema operativo elegido. Además, el hecho de elegir Windows como plataforma facilita su ex-

pansión y difusión en el mercado, ya que, en la actualidad, existen en el mundo más de un billón de dispositivos funcionando con Windows[1], si añadimos que Windows es el sistema operativo más utilizado desde hace más de nueve años[2], se asegura una estabilidad en el mercado para poder posicionarnos. Aunque Windows haya sido elegido como sistema operativo principal en este desarrollo, se plantea a futuro adaptar esta aplicación a sistemas operativos Linux y MacOS, permitiendo que la aplicación llegue a más usuarios.

El abanico de posibles usuarios interesados es muy amplio, ya que todo usuario que quiera gestionar sus archivos es un usuario potencial de la aplicación, independientemente de su edad, nivel de conocimiento tecnológico y profesión. Contará con una interfaz gráfica para que su uso resulte lo más familiar posible, además, se podrá activar un terminal embebido que nos permitirá operar en las diferentes carpetas con total facilidad y agilidad.

El terminal embebido está ideado para aceptar comandos de sistemas operativos Windows y unos comandos personalizados creados para esta aplicación, así como una selección de comandos Linux y Git. Gracias a estos últimos, se añade una funcionalidad extra a la aplicación ya que podremos manejar nuestro repositorio desde la misma aplicación y transferir archivos entre nuestras carpetas locales y las carpetas remotas localizadas en servidores de control de versiones.

Los objetivos de este proyecto tienen dos orígenes y fines diferenciados. Por una parte el desarrollo de una interfaz gráfica compuesta por un árbol de directorios, donde se mapean todos los discos duros conectados al ordenador en el momento del inicio de la aplicación, también cuenta con tres pestañas diferentes y dos ventanas en cada pestaña, en las que se puede visualizar el contenido de dos directorios diferentes y operar sobre ellos, ya sea para crear un archivo nuevo en alguno de ellos, borrar alguno ya existente o renombrarlos. Además, se permite al usuario arrastrar archivos y carpetas de una ventana a otra, facilitando así los movimientos de ficheros.

Por otra parte, se encuentra el desarrollo una consola de comandos que permite al usuario operar con sus archivos y carpetas utilizando comandos de Windows y una selección de comandos Linux, los relacionados con la gestión de ficheros. Esta consola se puede activar y desactivar, de manera que los usuarios que no necesiten esta funcionalidad no verán el espacio de su aplicación comprometido por una funcionalidad sin uso.

Estos objetivos constituyen el Minimum Viable Product(MVP)[3] del proyecto, y por lo tanto lo que se esperaba asumir en el tiempo asignado para el desarrollo. El MVP aporta muchas ventajas al desarrollo del proyecto, ya que permite que la aplicación crezca sin perder funcionalidad en cada release. Además, permite adaptarse a las necesidades de los usuarios, nutriéndose del feedback recibido en los diferentes sondeos. Gracias al uso del MVP, se protege al proyecto y su desarrollo de sufrir cambios drásticos que penalicen en tiempo y recursos el avance del mismo.

- E-mail de contacto: miguel.perezal@e-campus.uab.cat
- Mención realizada: Ingeniería del Software
- Trabajo tutorizado por: Cristina Romero Tris
- Curso 2020/21

Como funcionalidades adicionales, en futuras implementaciones, la aplicación admitirá comandos de Git y permitirá al usuario visualizar en una de las ventanas de la aplicación el contenido de un repositorio basado en Git, como podría ser Bitbucket. Por último, la aplicación debería contar con un sistema donde capture las operaciones realizadas y las guarde en un fichero de log para su posterior análisis.

Estas últimas funcionalidades descritas no forman parte del MVP, y, por lo tanto, no han sido implementadas en el tiempo de duración del proyecto. Aún así, es importante tenerlas en cuenta ya que aportarán un valor añadido considerable a la aplicación final.

2 METODOLOGÍA Y RESULTADOS

Para el correcto desarrollo del proyecto, se establecieron cuatro etapas diferenciadas. Estas etapas eran la etapa de análisis y estudio previo (2.1), la etapa de desarrollo de la interfaz gráfica (2.2), la etapa de desarrollo de la consola de comandos (2.3) y la etapa de testing final (2.4). Se decidió que el desarrollo se llevaría a cabo siguiendo el patrón MVVM (Model-View-ViewModel)[4], muy utilizado en las aplicaciones de .Net Framework.

Este patrón organiza los diferentes archivos de nuestro proyecto para asegurar una buena escalabilidad y mantenimiento en el tiempo. La parte de la vista (View) se compone de los ficheros XAML que conforman la interfaz visible e interaccionable para el usuario. En el modelo (Model), se encuentran las lógicas de interacción con la interfaz, como por ejemplo el control de qué teclas accionan qué eventos. Por último, en el modelo de vista (View model), se opera con los elementos de la vista cuyo valor no es estático, como por ejemplo, el contenido de un TextBox cuyo valor es extraído de una respuesta a un comando enviado por el usuario.

En las siguientes subsecciones se explican las etapas del proyecto con las tareas asignadas en cada una, así como cambios en la planificación o sucesos destacables surgidos durante el desarrollo.

2.1. Análisis y estudio previo

Esta primera etapa constituye el inicio del proyecto. Durante esta etapa se realizaron los estudios previos necesarios para el correcto desarrollo e implementación de las diferentes funcionalidades. También se dedicó parte del esfuerzo de esta etapa a construir el ecosistema de la aplicación conformado por un repositorio en Bitbucket donde se mantiene una versión actualizada del código, una página de Jira vinculada al repositorio para poder planificar correctamente el trabajo y disponer de un registro de la actividad, y un servidor de orquestación de Jenkins, vinculado también al repositorio de Bitbucket para poder automatizar los procesos de build y testing cada vez que se detecten cambios en la rama master.

Las etapas de desarrollo siempre van precedidas de un estudio previo, en caso de que con el estudio principal no se hayan cubierto todas las necesidades, y se finalizan con una etapa de testing sobre la implementación realizada. Así pues, dentro de esta etapa encontramos las siguientes tareas:

- Elaboración de personas, para definir correctamente el

target principal de la aplicación, así se facilita la elaboración de tests, formularios y sondeos teniendo claro a que público van dirigidos.

- Sondeos a usuarios reales, a través de la herramienta de Google forms se realizaron diferentes encuestas a usuarios potenciales. En estos formularios es importante extraer información de aspectos que utilizan más frecuentemente en su explorador de archivos, pero más importante saber que cosas no les gustan o les parecen poco intuitivas.

  Gracias a estos sondeos se descubrió que había necesidades importantes que tenían los usuarios y que FE-Monitor podía cubrir, así que, una vez se estudiaron las respuestas de los usuarios, se hizo un cambio en la planificación del proyecto para centrar más esfuerzos en mejorar la interfaz de la aplicación. En el anexo se puede encontrar una comparativa del primer prototipo de FE-Monitor y el prototipo actualizado según las necesidades de los usuarios.

- Captación de requerimientos, utilizando los resultados de las encuestas anteriores, así como con las experiencias propias utilizando un explorador de archivos, se creó documentación sobre requerimientos de manera que puediese ser utilizado en las siguientes etapas del proyecto.

  Estos requerimientos fueron reunidos en un documento de especificación de requerimientos donde se encuentran tablas como la mostrada a continuación:

ID	**REQ-F-12**
TÍTULO	ABRIR DESDE EL ÁRBOL DE DIRECTORIO
DESCRIPCIÓN	La aplicación debe permitir al usuario abrir una carpeta en una ventana configurable seleccionándola desde el árbol de directorios.
ACCEPTANCE	Arrastramos una carpeta desde el árbol de directorios hasta una de las ventanas configurables. En esta ventana se muestra el contenido de la carpeta, ya sean archivos u otras carpetas.
PRIORIDAD	H
PADRES	REQ-F-11; REQ-F-01

Fig. 1: Ejemplo de requerimiento

  Para estas tablas se definieron los campos, ID, Título, Descripción, Acceptance, Prioridad y Padres. El ID se compone del tag REQ, seguido de F o NF, indicando si el requerimiento es de tipo funcional o no funcional, y un entero que incrementa en uno por cada requerimiento creado. El campo Acceptance indica el criterio a seguir para determinar que ese requerimiento se ha cubierto con una implementación funcional. En cuanto a la prioridad se utiliza un sistema de clasificación dividido en prioridad alta, media y baja (H, M, L). Esta prioridad determina el orden en el que se debían abordar las implementaciones, teniendo en cuenta su impacto en el funcionamiento de la aplicación y el valor que añade a la misma para el usuario final.

- Elaboración de diagrama de casos de uso, mediante la herramienta PlantUML, utilizando los casos de uso extraídos a raíz de la captación de requerimientos. El diagrama se ha adjuntado en la sección de apéndice al final del documento para facilitar su lectura e interpretación.

- Puesta en marcha del ecosistema Jenkins-Bitbucket-Jira. Para esta tarea se creó una cuenta de Atlassian así como un repositorio de Bitbucket y un proyecto de Jira. El servidor de orquestación Jenkins se instaló en la máquina local donde se llevó a cabo el desarrollo para agilizar esta parte del proceso, ya que Jenkins se instala como servicio y, siempre que el PC esté encendido y el servicio en marcha, puede funcionar sin problemas. Para vincular Jenkins y Bitbucket se instaló el plugin de Bitbucket para Jenkins y se creó un Webhook para Jenkins en Bitbucket.

  Además, se instaló el plugin de MSBuild en Jenkins para poder hacer el proceso de build del proyecto de manera automática cada vez que Jenkins detecta un cambio en la rama master del repositorio.

2.2. Desarrollo de la interfaz gráfica

La siguiente etapa se centró en el desarrollo de la interfaz gráfica común tanto para los usuarios con más conocimientos informáticos como para los usuarios sin tanta formación en este campo. Tanto este desarrollo como el de la consola de comandos se hizo utilizando C# como lenguaje de programación y Visual Studio 2017 como IDE. Esta parte del trabajo se compuso de las siguientes tareas:

- Implementar el esqueleto del patrón MVVM.
- Implementar árbol de directorios.
- Implementar las pestañas.
- Implementar las ventanas internas para abrir carpetas.
- Implementar barra de carpetas favoritas.

2.2.1. Implementación MVVM

El primer paso en el desarrollo, no estuvo directamente relacionado con implementaciones o desarrollos de funcionalidades, sino que tuvo que ver con la creación del esqueleto de la aplicación para poder implementar correctamente el modelo MVVM. Para ello, se crearon dos carpeta, una llamada View y la otra llamada ViewModel. En la primera se encuentran las vistas y modelos, ya que estos últimos se crean automáticamente y se vinculan a los archivos de vista. Por otra parte se encuentras los arcivos de modelo de vista en la carpeta ViewModel.

Para vincular una vista con su modelo de vista, se debe crear un objeto de la clase ViewModel, de la ventana que corresponda, en el modelo de la vista que queremos vincular. Con el objeto ya creado, se debe indicar que el DataContext de esa vista será el modelo de vista. Simplemente con la línea $DataContext = ObjetoViewModel$ se puede vincular.

Por último, es importante corregir el archivo App.xaml para que el campo $StartupUri$ apunte a la carpeta View y al fichero XAML que corresponda a la ventana de inicio. Una vez hechos estos cambios, ya se pudo proceder a los desarrollos correspondientes.

2.2.2. Implementación árbol de directorios

Para implementar el árbol de directorios se utilizó la clase Directory del namespace System.io, propio del .NET Framework. A través de esta clase, se extraen las unidades de

disco conectadas al sistema en el momento del arranque de la aplicación. Los elementos extraídos se almacenan en un TreeView para poder ser consumidos desde la vista. Se utilizaron unas imágenes gratuitas para representar los discos duros y las carpetas. A continuación, se muestra como quedó el resultado final de esta implementación.

Fig. 2: Árbol de discos

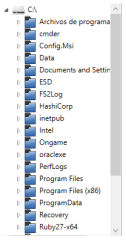


Fig. 3: Árbol de discos desplegado

2.2.3. Implementación pestañas

Una vez se desarrolló el árbol de directorios, se procedió al desarrollo de las pestañas. Estas pestañas permiten al usuario poder tener hasta seis ventanas donde poder abrir diferentes directorios. Se definió un número máximo de tres pestañas y la posibilidad de poder cambiar el título de las mismas para poder organizar las ventanas a elección del usuario. Para poder cambiar el título de la pestaña, basta con hacer doble click en la misma, de esta manera aparecerá una ventana para que el usuario introduzca un nuevo título. Estos datos son serializables por lo que una vez se cambia el título, este aparecerá con el nuevo nombre aunque cerremos la aplicación.

Estas pestañas se implementaron a través de un TabControl el cual abarca todo el ancho de la ventana, respetando el espacio ocupado por el árbol de directorios.

Fig. 4: Pestañas por defecto

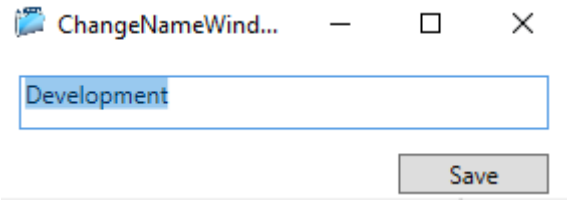


Fig. 5: Ventana para cambio de título

2.2.4. Implementación ventanas

Cada pestaña debía habilitar dos secciones de la pantalla para poder abrir diferentes directorios. Para referirse a estas secciones se utilizará el nombre ventanas configurables. Dichas ventanas ofrecían, en un primer desarrollo, la opción de abrir una carpeta que se encuentre en el árbol de directorios de la izquierda, haciendo doble click sobre la carpeta deseada. El contenido de la carpeta se muestra en la primera ventana disponible, y en caso de que las dos ventanas de una pestaña estén ocupadas, se realiza un cambio automático de pestaña para ocupar la siguiente ventana disponible.

Al abrir una carpeta, se activa un encabezado que contiene unas flechas de navegación, la ruta del directorio abierto y un botón para cerrar la carpeta y dejar otra vez la ventana libre.

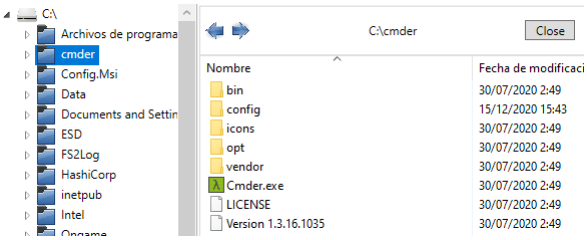


Fig. 6: Carpeta abierta en ventana configurable

2.2.5. Implementación barra de favoritos

Una funcionalidad con la que no se contaba en un inicio es la barra de favoritos. Actualmente, el explorador de archivos de Windows permite guardar rutas a directorios en lo que denomina 'Acceso rápido', pero a pesar de existir esta funcionalidad, muchos usuarios a través de las encuestas indicaron que les faltaba la opción de tener unas carpetas favoritas de fácil acceso. Por ello, se tomó la decisión de incluir una barra de favoritos en el margen superior de la pantalla.

Para añadir una carpeta a favoritos, se implementó un menú contextual donde, al hacer click derecho en cualquier carpeta del árbol de directorios, nos aparece la opción de añadir a favoritos. Si elegimos esta opción, la carpeta aparecerá en el margen superior con un icono de carpeta y su nombre en la parte inferior. En caso de que queramos borrar la carpeta de favoritos, haciendo click derecho en dicha carpeta nos aparecerá otro menú contextual con la opción de borrar de favoritos.

Por último, para abrir una carpeta situada en la barra de favoritos, bastará con hacer click sobre ella para que ocupe la primera ventana libre, de la misma manera que funciona con el árbol de directorios.

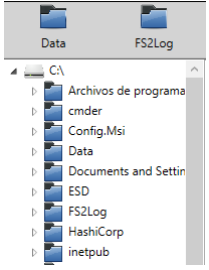


Fig. 7: Barra de favoritos con dos carpetas

2.3. Desarrollo de la consola de comandos

Una vez desarrollada la interfaz común, se comenzó con la implementación de la consola de comandos. Esta parte del proyecto en un inicio formaba parte del proyecto de Visual Studio de FE-Monitor, pero posteriormente se tomó la decisión de aislarlo en un proyecto diferente que generase una DLL, y que ésta se cargase en tiempo de ejecución en la aplicación resultante. Esta decisión fue tomada, ya que la funcionalidad de la consola de comandos puede ser utilizada en diferentes proyectos no relacionados con FE-Monitor. Las tareas que compusieron esta etapa fueron:

- Desarrollo de la interfaz de consola.
- Implementación de comandos Windows.
- Implementación de la interpretación de comandos Linux.
- Implementación de los comandos propios de FE-Monitor.

2.3.1. Desarrollo de la interfaz de consola

El primer paso en el desarrollo de la consola de comandos fue la creación de una interfaz sobre la que trabajaría el usuario. Esta interfaz se compone de un encabezado en el que figura el directorio en el que se encuentra el usuario, información de autoría de la aplicación y el cuadro de texto donde el usuario puede escribir los comandos. En un prinicipio, se pretendía hacer una ventana de personalización de estilo para la consola donde se pudiese elegir el color de fondo, de fuente, tamaño de la fuente y tipografía. Valorando las respuestas de las encuestas se veía que la mayoría de los usuarios no daban demasiada importancia a este aspecto, por lo que se minimizó a dos estilos predefinidos que pueden ser cambiados con sólo un click a un botón. Esto no quiere decir que la idea de poder personalizar la consola se haya descartado, simplemente fue aplazada en el tiempo para poder priorizar otras funcionalidades más interesantes de la consola de comandos.

Mediante el modelo de vista, se controla que cada vez que se envía un comando se cree una nueva línea con el formato adecuado, es decir, que comience con el carácter '#' y que este no pueda ser borrado. Al comienzo, los comandos no estaban implementados por lo que las pruebas se realizaron respondiendo con el texto 'comando X enviado', de esta manera podía comprobarse el correcto funcionamiento sobre el que se trabajaría posteriormente.

Por último, se creó un evento que se activa cuando el usuario pulsa la teclas Ctrl + D, para que la ventana cambie entre el modo usuario, sin consola de comandos, y el modo desarrollador, con consola de comandos. Salvando la consola de comandos, la interfaz de desarrollador y la de usuario son completamente iguales y tienen la misma operabilidad.

Fig. 8: Consola de comandos

Fig. 9: Segundo estilo de consola

2.3.2. Implementación de comandos Windows

Una vez se implementó la interfaz de la consola, así como el control sobre el comportamiento de la misma a nivel visual, se comenzó con la implementación de los comandos de Windows. Esta parte tenía una facilidad mayor que las dos siguientes ya que C# tiene una gran compatibilidad con las funciones relacionadas directamente con el sistema operativo Windows y la creación de procesos.

Al comienzo del desarrollo se implementó una clase para recibir los comandos, enviarlos al terminal de Windows y devolver la respuesta al controlador del terminal de FE-Monitor. Esta implementación podía funcionar perfectamente, pero, se decidió hacer un pequeño cambio dado que según la planificación del proyecto se podían cumplir los tiempos sin demasiado problema, por lo que se aisló esta nueva clase en un proyecto separado llamado CommandInterpreter. Este proyecto genera una DLL al ser compilado y puede ser cargada desde FE-Monitor o desde cualquier otro proyecto. La ventaja de esta nueva forma de implementar la consola de comandos es, principalmente, poder aprovechar la consola de comandos para otros desarrollos y poder compartirla en un repositorio público para que otros desarrolladores pueden utilizarla y colaborar en su mejora.

Para utilizar el intérprete de comandos, basta con añadir la referencia a la DLL compilada y hacer llamadas a la función SendCommand de la clase Manager. No es necesario hacer la distinción entre comandos Windows, Linux u otros desde la aplicación que llama a la DLL ya que esta se encarga de interpretar el comando y mandarlo al terminal de Windows.

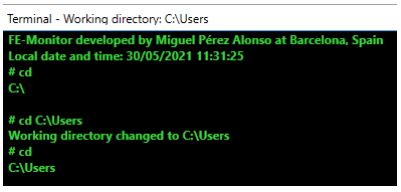


Fig. 10: Ejemplo de comandos Windows

2.3.3. Implementación de comandos Linux

Para la siguiente tarea se hizo un estudio previo de los comandos Linux que más podrían interesar para gestionar archivos y directorios de nuestro PC.

El primer comando que se implementó fue el comando pwd, dada su facilidad de operación y uso. Para este primer comando se comprobaba el valor del comando y si este coincidía con pwd se sustituía por el comando cd. Esta primera versión era muy simple y servía únicamente para comprobar el éxito de la operación de traducción de comandos.

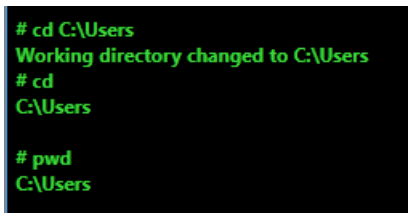


Fig. 11: Ejemplo de comando PWD

Obviamente, esta forma de proceder no era la más óptima ni la que aseguraba una mayor escalabilidad del intérprete de comandos, por lo que, una vez se llegó a este punto de la implementación, se creó una clase Linux Interpreter para poder interpretar todos los comandos elegidos de Linux. Esta clase contiene un diccionario donde la clave es el comando de Linux en formato string y el valor su equivalente en Windows también en formato string.

Para hacer la traducción el intérprete busca el comando introducido por el usuario en el diccionario de los comandos Linux y lo sustituye por su equivalente en Windows. Era importante en este punto que el intérprete respetase los parámetros que no eran el comando y que no sustituyese absolutamente todo el texto, sino solo el comando de Linux.

De esta manera se consiguió implementar los comandos para borrar y mover directorios, así como los mismos para ficheros.

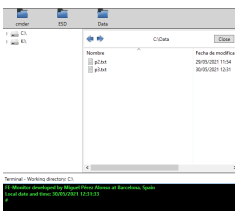


Fig. 12: Carpeta con dos archivos

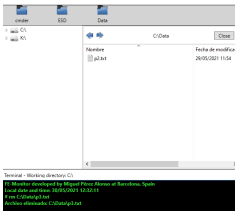


Fig. 13: Comando para borrar archivo

2.3.4. Implementación de comandos propios

La última implementación relacionada con la consola de comandos son los comandos propios de FE-Monitor. Estos comandos pretenden utilizar las ventajas de tener múltiples ventanas abiertas de manera simultánea y la capacidad de utilizar la consola de comandos sin tener que abrir otra aplicación.

Después de sopesar diferentes formas en las que podrías funcionar estos comandos se creó una sintáxis propia que permite exprimir al máximo todas las cualidades de FE-Monitor. Estos comandos se componen del nombre de la ventana sobre la que se quiere actuar, por defecto el nombre de una ventana es windowX donde la X es el número de la ventana empezando por 1, la primera ventana de la primera pestaña, y 6 la última ventana de la última pestaña, seguido de un símbolo $>$ y a continuación seguiría de manera obligatoria el comando a utilizar y de manera opcional el archivo afectado y el directorio de destino, en ese orden.

Por ejemplo, si el comando que queremos utilizar es lista el contenido de la ventana número tres utilizaríamos el comando 'window3 $>$ cd'. En este caso no es necesario ni un nombre de fichero ni una ruta de destino. Ahora bien, si lo que queremos utilizar es un comando para transferir archivos de la primera ventana a la segunda utilizaríamos el comando 'window1 $>$ movefile file_name.ext window2'. En este último caso se utilizan todos los componentes de comando permitidos.

Además, el comando movefile es un comando propio, ya que no es un comando de Windows ni uno de Linux. Este comando se creó después de crear otro comando para mover fichero y directorios que se explicará a continuación, de esta manera se podía hacer una distinción entre ambos comandos.

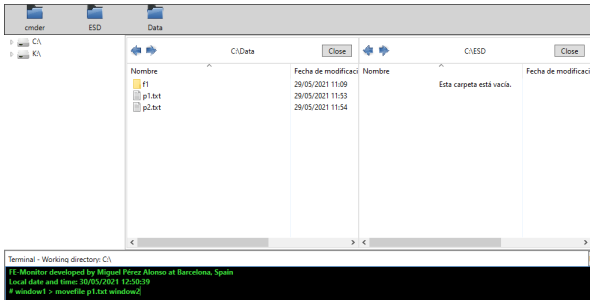


Fig. 14: Comando para mover archivo

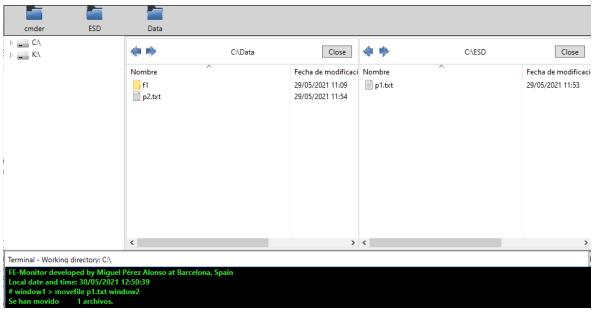


Fig. 15: Archivo movido

El siguiente comando relacionado con el movimiento de archivos fue el comando $movebulk$. Este comando permite mover lotes de archivos para evitar tener que enviar el mismo comando cambiando un parámetro varias veces, así se consigue que la experiencia de usuario mejore y se aporta un valor añadido muy interesante a la consola de comandos.

La sintáxis de este comando puede variar en función de las necesidades del usuario, en caso de que quiera elegir ciertos archivos y directorios utilizará una sintáxis o si quiere mover todas las carpetas de un directorio a otro utilizará otra sintáxis.

El primer caso se consigue utilizando el comando de la siguiente manera, 'windowX $>$ movebulk [file1.ext,file2.ext] windowY', de esta manera los archivos file1 y file2 que se encuentran en la ventana X serán transferidos a la ventana Y.

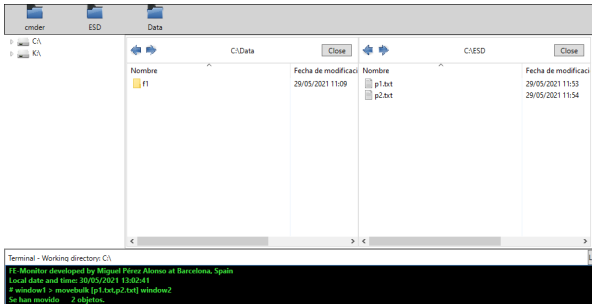


Fig. 16: Comando para mover varios archivos

Esta misma sintáxis no excluye la transferencia de directorios, el usuario podría perfectamente utilizar el comando de la siguiente manera 'windowX > movebulk [file1.ext,folder1] windowY' consiguiendo así transferir el archivo file1 y la carpeta folder1 de la ventana X a la ventana Y.

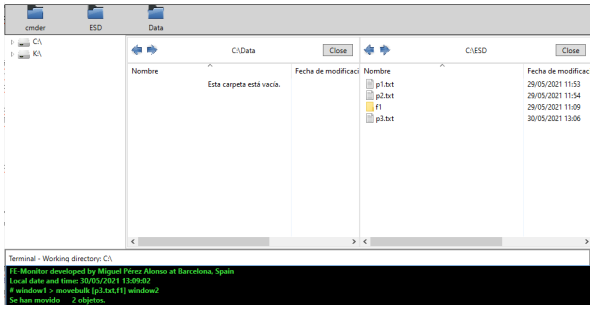


Fig. 17: Comando para mover un archivo y una carpeta

Por último, este comando también admite mover archivos y carpetas indicando el tipo. Por ejemplo, si el usuario quiere mover todas las carpetas de una ventana a otra podría utilizar el comando como se ha indicado anteriormente o, para facilitar las cosas, podría utilizar el comando de la siguiente manera 'windowX > movebulk folders windowY'. Esta nueva sintáxis transfiere todos los objetos de tipo carpeta que se encuentren en la ventana X a la carpeta que abierta en la ventana Y.

Es importante destacar la importancia de facilitar la transferencia de archivos y carpetas mediante la consola de comandos para que la gente que está acostumbrada a trabajar de esta manera encuentre más ágil su experiencia de trabajo, así como para que los usuarios menos acostumbrados al terminal de comandos puedan tener una primera experiencia más sencilla y cómoda.

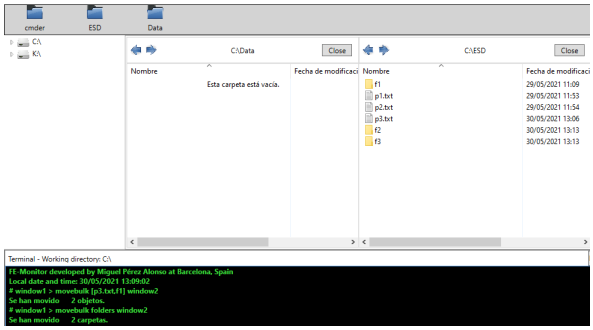


Fig. 18: Comando para mover todas las carpetas

2.4. Testing final con usuarios

En la última etapa del proyecto, se realizarán un seguido de tests en el conjunto de la aplicación. Con el desarrollo ya finalizado se probará el conjunto de la aplicación, para ello, se reunirá a un conjunto de usuarios potenciales y se les guiará para realizar ciertas acciones, una vez completen esta prueba deberán completar un formulario de Google Forms para registrar sus experiencias. Por otra parte, se reunirá a otro grupo de usuarios potenciales y se les pedirá que utilicen la aplicación con total libertad, y una vez más deberán completar un formulario al finalizar la sesión. Cuando esta parte haya concluido, se recogerán todos los resultados y se elaborarán unas conclusiones sobre el estado de la aplicación y las posibles mejoras y arreglos a realizar.

Este periodo de pruebas con usuarios reales abarcará esa última etapa del proyecto con el objetivo de poder reflejar en el documento final los resultados de las pruebas y mejorar así las secciones de resultados y conclusiones, basándose en las experiencias de los usuarios al utilizar FE-Monitor.

2.5. Valoración de los resultados

Al final de la etapa de desarrollo, se ha conseguido tener una aplicación funcional que cumple las expectativas esperadas según la planificación. Se ha podido implementar una interfaz de usuario completamente operativa y adaptada según el estudio realizado para poder captar el mayor número de usuarios posible. Una aplicación intuitiva y que cubra las necesidades de los usuarios ha sido siempre el objetivo principal de este proyecto, por lo que se ha tenido muy en cuenta los resultados de las encuestas para poder conseguir el mejor resultado.

En cuanto al terminal de comandos, los resultados entran dentro de lo esperado, teniendo en cuenta el punto que se quería alcanzar durante las horas destinadas al proyecto. Se ha desarrollado un terminal que admite comandos de Windows, una selección de comandos Linux, y se ha creado una sintáxis totalmente nueva para poder exprimir al máximo la utilidad de las múltiples ventanas sobre las que nos permite operar FE-Monitor.

Este desarrollo ha tenido en cuenta en todo momento el MVP de la aplicación y se ha conseguido que todos los avances formasen un módulo funcional que pudiese ser utilizado por el usuario sin que su experiencia se viese perjudicada por algún motivo.

En esta útlima etapa de proyecto, se centrarán esfuerzos en que los usuarios prueben la aplicación, ya que han sido los propios usuarios los que han permitido mejorar FE-Monitor y adaptarlo a sus necesidades.

2.6. Conclusiones

FE-Monitor está resultando en una aplicación con múltiples funcionalidades, todas ellas de carácter bastante intuitivo, que puede mejorar la experiencia de los usuarios a la hora de utilizar un explorador de archivos. Tiene las cualidades para que un abanico bastante amplio de usuarios lo utilice con total comodidad, y esa era precisamente la intención cuando se pensó en una aplicación que uniese una herramienta que utiliza todo el mundo a diario, con la posibilidad de mejorarla y añadiendo una herramienta como es

la consola de comandos, utilizada por un grupo importante de usuarios.

Es importante que este proyecto siga creciendo y escalando para poder posicionarse como una aplicación de referencia. Los propios usuarios serán partícipes de este crecimiento aportando sus impresiones sobre el uso de FE-Monitor, y, además, se podrá contar con la ayuda de los desarrolladores que lo deseen al poder subir el código del intérprete de comandos a un repositorio público.

FE-Monitor cuenta con una buena base en su ecosistema de desarrollo, gracias al uso de Bitbucket y Jenkins en conjunto, así como la elección de C# y .NET, dos tecnologías bastante nuevas y que cada vez están ganando más seguidores en el mercado de la programación, permitiendo que el proyecto pueda seguir creciendo y mantenerse vivo durante años.

Como conclusiones más personales, debo decir que realizar este proyecto ha supuesto un crecimiento profesional importante, debido a tener que hacerme cargo de un proyecto creado desde cero, utilizando sólo las librerías que incluye por defecto el framework de .NET, y pudiendo montar un esquema basado en la filosofía DevOps. Obviamente aún queda mucho por mejorar en los aspectos concernientes a la gestión de proyectos y es importante seguir creciendo como profesional, pero este proyecto ha podido encender una chispa para seguir superando los retos que vaya encontrando.

REFERENCIAS

[1] Microsoft (2018) ”Windows Devices”[en línea]. Disponible en: https://news.microsoft.com/bythenumbers/en/windowsdevices

[2] Shanhong Liu (Octubre de 2020) ”Global market share held by computer operating systems 2012-2020, by month”[en línea]. Disponible en: https://www.statista.com/statistics/268237/global-market-share-held-by-operating-systems

[3] Colaboradores de Wikipedia (Septiembre de 2020) ”Producto viable mínimo”[en línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Producto_viable_m%C3%ADnimo&oldid=128951661

[4] David Britch (Agosto de 2017) ”Patrón Model-View-ViewModel”[en línea]. Disponible en: https://docs.microsoft.com/es-es/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

[5] Equipo Jenkins (Febrero de 2021) ”Bitbucket Server Integration”[en línea]. Disponible en: https://plugins.jenkins.io/atlassian-bitbucket-server-integration/

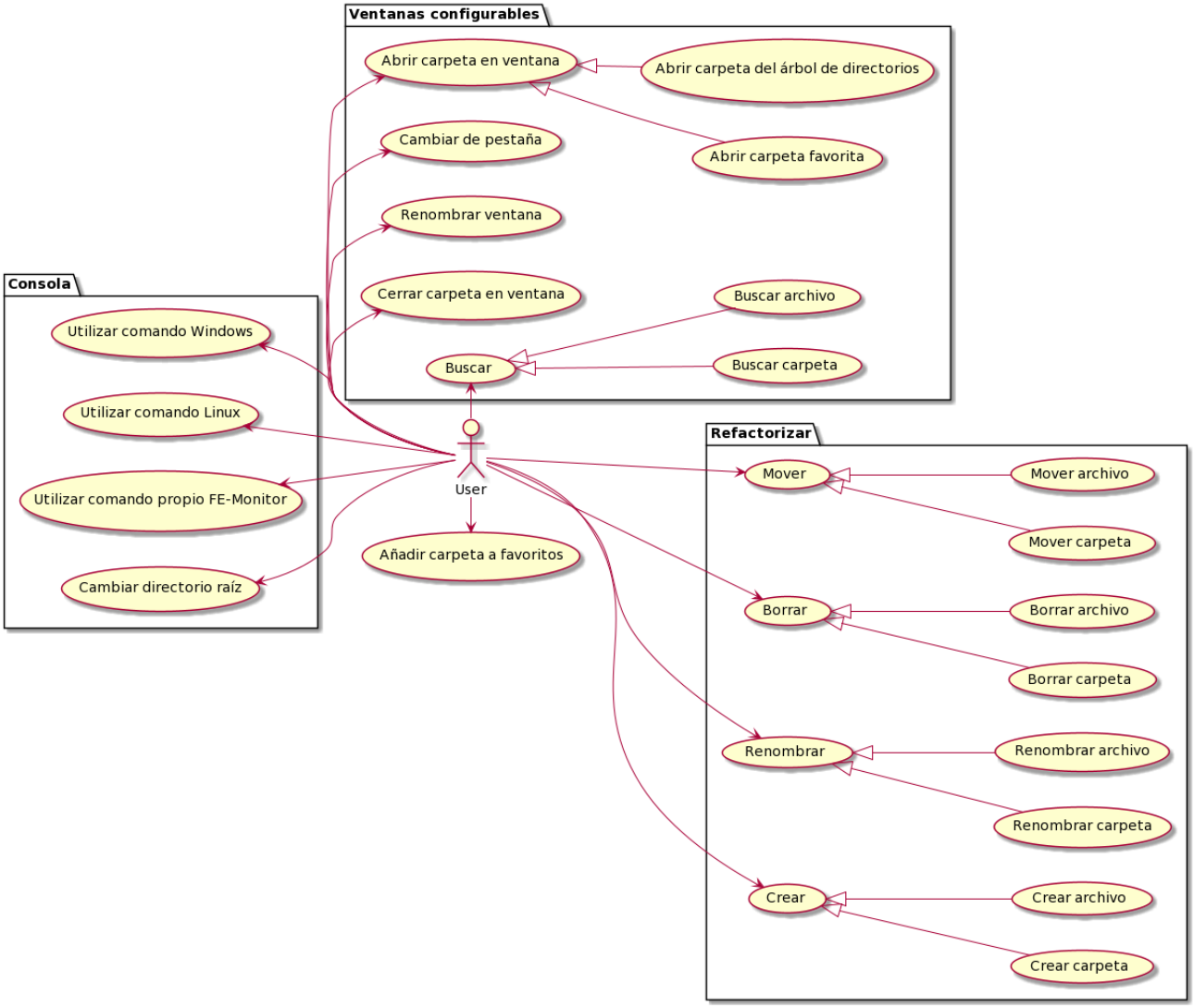
[6] Comunidad Bitbucket (Marzo de 2021) ”Link Bitbucket with Jira”[en línea]. Disponible en: https://confluence.atlassian.com/bitbucketserver/link-bitbucket-with-jira-776640408.html

[7] Microsoft (2021) ”Process Clase”[en línea]. Disponible en: https://docs.microsoft.com/es-es/dotnet/api/system.diagnostics.process?view=net-5.0

[8] LeModa (Mayo de 2021) ”Windows and Unix command line equivalents”[en línea]. Disponible en: https://www.lemoda.net/windows/windows2unix/windows2unix.html

APÉNDICE

A.1. Diagrama casos de uso



A.2. Evolución interfaz