Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

sl-pr-2

Encapsular el acceso a un *mainframe* IBM ESA/390

Autor 1:	Gracia Picó, Martina - 795809
Autor 2:	Toral Pallás, Héctor - 798095
Autor 3:	Carrizo Pérez, Daniel - 821674
Grado:	Ingeniería Informática
Curso:	2022-2023

08 de Enero de 2023

Índice

01

CREACIÓN DEL PROYECTO

Para la realización del proyecto se ha hecho una aplicación Python usando Flask para el despliegue en la web, y con el objetivo de transformarlo en una ventana y separar la aplicación del navegador se ha usado WebView.

Es importante comentar que para poder realizar el proyecto tuvimos que descargarnos el emulador wc3270 y guardar el ejecutable en la carpeta de nuestro proyecto, para que el mismo proyecto pudiese lanzarlo.

02

CONEXIÓN CON EL EMULADOR

Para conectar nuestra aplicación con el emulador de la aplicación legada hemos usado Py3270. Esta biblioteca de Python cuenta con funciones que interactúan con el emulador. Como había algunas funciones que la biblioteca no detectaba en vez de usar "*pip install py3270*"tuvimos que añadir el código de la biblioteca en directamente en nuestro proyecto (fichero *py3270.py).*

El proyecto cuenta con un código principal que se encuentra en *app.py* el cual usa funciones de el fichero *emulator.py* y de *py3270.py.* Además para la GUI del usuario se han usado ficheros *html* para cada una de las pantallas de nuestra aplicación.

Con todo esto la configuración de nuestro proyecto, por ahora, es la siguiente:

```
\GestorDeTareas
|- \lib
|- emulator.py
|- py3270.py
|- \templates
|- index_inicio.html
|- index_inicio_error.html
|- index_inicio_ocupado.html
|- tareas.html
|- app.py
|- wc3270.exe (ejecutable del emulador)
```

Para que el código fuese legible las funciones que tenían que ver con interacción con el emulador se agruparon en un fichero a parte del de la aplicación principal (fichero *emulator.py*).

2.1. EXPLICACIÓN DE *EMULATOR.PY*

El fichero *emulador.py* cuenta con 5 funciones que sirven para gestionar la interacción entre nuestra aplicación y la aplicación legada. Este fichero también cuenta con 4 funciones auxiliares.

La función *emulador(mylogin, mypass)* es la encargada de lanzar el emulador y de introducir en la aplicación legada las claves de inicio de sesión introducidas por el usuario. Esta aplicación devuelve un 0 en caso de que el inicio de sesión haya sido exitoso, devuelve 1 en caso de que el usuario o la contraseña sean incorrectos, y devuelve 2 en caso el usuario ya este en uso, es decir, ya haya un inicio de sesión con ese usuario.

```
1 def emulador(mylogin, mypass):
2   global e, active_window
3   # Main
4   host = "155.210.152.51"
5   port = "3270"
6
7   e = Emulator(visible=True)
8   e.connect(host + ':' + port)
```

```
  time.sleep(delayScreen)

  # Pataalla inicio
  time.sleep(delayScreen)
  e.send_enter()

  # Pantalla Login
  time.sleep(delayScreen)
  ## Usuario
  e.wait_for_field()
  e.send_string(mylogin)
  e.send_enter()
  ## Contrasena
  e.wait_for_field()
  e.send_string(mypass)
  e.send_enter()

  # Chequear correcto inicio de sesion
  time.sleep(delayScreen)
  inicio = inicio_correcto()
  if inicio==0:
    # Pantalla previa a comandos
    time.sleep(delayScreen)
    e.wait_for_field()
    e.send_enter()
    time.sleep(delayScreen)
    e.wait_for_field()
    e.send_string('PA1')
    e.send_enter()

    # Pantalla comandos
    time.sleep(delayScreen)
    e.wait_for_field()
    e.send_string('tareas.c')
    e.send_enter()
    return 0
  elif inicio==1:
    e.terminate()
    return 1
  elif inicio==2:
    e.terminate()
    return 2
```

Todas estas comprobaciones de inicio de sesión las realiza una de las funciones auxiliares, *inicio_correcto()* chequeando lo que se muestra en la pantalla de la aplicación legada.

```
def inicio_correcto():
  line=e.string_get(7,2,24)
  if line=="Userid is not authorized":
    return 1
  line=e.string_get(7,2,18)
  if line=="Password incorrect":
    return 1
  line=e.string_get(1,1,16)
  if line.rstrip()=="Userid is in use":
    return 2
  return 0
```

También encontramos la función *pantalla()* la cual se encarga de plasmar en un fichero la pantalla que muestra el emulador en el momento en el que la llaman, esto lo hace leyendo las líneas de la pantalla y escribiéndolas en un fichero.. Esta función es usada por otras funciones, no por la aplicación principal, puesto que otras funciones necesitan el texto que muestra el emulador para poder realizar diferentes acciones.

```
def pantalla(filename="pantalla.txt"):
  time.sleep(0.5)
  screen_content = ''
  for row in range(1, 43 + 1):
    line = e.string_get(row, 1, 79)
    screen_content += line + '\n'
```

```
  archivo = open(filename, "w")
  archivo.write(screen_content)
  archivo.close()
```

Contamos también con otras funciones que se encargan ya de la gestión de tareas en sí. La primera de estas funciones es *assign_tasks()*. A esta función le pasan el tipo de tarea ("General.º .Específica"), la fecha, la descripción y el nombre. Este último será vació en el caso de que el tipo de tarea sea "General". En esta función accedemos a la opción de 1 que nos permite asignar una tarea, dependiendo del tipo de tarea accedemos a la opción 1 o 2 de asignar tareas. Luego insertamos la fecha, la descripción, pero en el caso de una tarea específica antes de la descripción insertaremos el nombre.

```
def assign_tasks(tipo:str, fecha:str, desc:str, nombre:str):
  desc = '"' + desc.replace(" ", " ") + '"'
  nombre = '"' + nombre.replace(" ", " ") + '"'

  e.send_string("1")
  e.send_enter()
  e.delete_field()

  if tipo=="General":
    e.send_string("1")
    e.send_enter()
    e.delete_field()
    e.send_string(fecha)
    e.send_enter()
    e.delete_field()
    e.send_string(desc)
    e.send_enter()
    e.delete_field()

  elif tipo=="Especifica":
    e.send_string("2")
    e.send_enter()
    e.delete_field()
    e.send_string(fecha)
    e.send_enter()
    e.delete_field()
    e.send_string(nombre)
    e.send_enter()
    e.delete_field()
    e.send_string(desc)
    e.send_enter()
    e.delete_field()
  e.send_string("3")
  e.send_enter()
  e.delete_field()
```

La segunda es *get_tasks_general()*, la cual se encarga de obtener un listado de tareas de tipo general y la información de cada una. También se encarga de procesar la información y devolverla en un formato correcto. Esta función es una función auxiliar puesto que no la usa nuestra app directamente.

```
def get_tasks_general(file="pantalla.txt"):
  resultado = []
  for num_line in range(0, 43 + 1):
    line=read_line(num_line,file)
    if line!=0:
      if line.find("TOTAL TASK")!=-1:
        return resultado
      else:
        partes = line.split(" ")
        if partes[0]=="TASK":
          temp ={"fecha":partes[3],"descripcion":partes[5].strip('"')}
          resultado.append(temp)
  return resultado
```

La tercera es *get_tasks_specific()*, la cual se encarga de obtener un listado de tareas de tipo especifico y la información de cada una. También se encarga de procesar la información y devolverla en un formato correcto.

Esta función es una función auxiliar puesto que no la usa nuestra app directamente.

```
def get_tasks_specific(file="pantalla.txt"):
  resultado = []
  for num_line in range(0, 43 + 1):
    line=read_line(num_line,file)
    if line!=0:
      if line.find("TOTAL TASK")!=-1:
        return resultado
      else:
        partes = line.split(" ")
        if partes[0]=="TASK":
          temp = {"fecha":partes[3],"ombre":partes[4].strip('"'),"descripcion":partes[5].
    strip('"')}
          resultado.append(temp)
  return resultado
```

La cuarta función es *view_tasks()* cuya principal función es avanzar por las pantallas del emulador para luego llamar a las funciones auxiliares *get_tasks_general()* y *get_tasks_specific()*, y juntar sus dos resultados y devolvérselos a la aplicación principal.

```
def view_tasks():
  resultado=[]
  e.send_string("2")
  e.send_enter()
  e.delete_field()
  e.send_clear()
  e.send_string("1")
  e.send_enter()
  e.delete_field()
  pantalla()
  general = get_tasks_general()
  e.send_clear()
  e.send_string("2")
  e.send_enter()
  e.delete_field()
  pantalla()
  e.send_string("3")
  specific = get_tasks_specific()
  e.send_enter()
  e.delete_field()
  resultado = general + specific
  return resultado
```

La quinta y última función es *exit_tasks()* que se encarga de cerrar el emulador.

```
def exit_tasks():
  global e
  e.send_string("3")
  e.send_enter()
  e.delete_field()
  e.send_string("off")
  e.send_enter()
  e.delete_field()
  time.sleep(0.5)
  e.terminate()
```

2.2. EXPLICACIÓN DE *APP.PY*

La aplicación cuenta con un código principal en el que se despliega la aplicación y se crea la ventana del Gestor de tareas (wc3270). A parte, tiene también 5 funciones que son las encargadas de gestionar las diferentes peticiones que realicen a nuestra aplicación. Además hay 1 función que solo se ejecutará al cerrar la aplicación.

La primera función se encarga de la gestión en la ruta /". Esta simplemente se encarga de renderizar ïndex_inicio.html"donde el usuario podrá iniciar sesión con sus credenciales. (*mylogin=GRUPO_03*, *mypass=SECRETO6*).

```
@app.route('/')
def index():
  return render_template('index_inicio.html')
```

La segunda función se encarga de la gestión en la ruta /ini". En esta se inicia sesión en el emulador llamando a la función *emulador(mylogin, mypass)* de *emulator.py*. Dependiendo de lo que devuelva la función se renderizará una pagina *html* u otra. En el caso de que devuelva 0 renderiza "tareas.html", si devuelve 1 renderiza ïndex_inicio_error.html", y si devuelve 2 renderiza ïndex_inicio_ocupado.html".

```
@app.route('/ini', methods=['POST'])
def ini():
  last_user = request.form['usuario']
  last_passwd = request.form['contrasena']
  e = emulador(last_user,last_passwd)
  if e==0:
    return render_template('tareas.html')
  elif e==1:
    return render_template('index_inicio_error.html')
  elif e==2:
    return render_template('index_inicio_ocupado.html')
```

La tercera función se encarga de la gestión en la ruta /assignGeneral". A esta función se le pasan como parámetros tipo, fecha y descripción, los cuales, los ha insertado el usuario. Usa la función *assign_tasks()* de *emulator.py* para guardar la tarea que el usuario ha insertado, y como es necesario que se muestren las tareas que hay guardadas se llama también a la función *view_tasks()* para que al renderizar "tareas.html"se carguen los datos de todas las tareas guardadas.

```
@app.route('/assignGeneral', methods=['POST'])
def assignGeneral():
  tipo = "General"
  fecha = request.form['fechaGeneral']
  desc = request.form['descripcionGeneral']
  nombre = ""

  assign_tasks(tipo, fecha, desc, nombre)
  data = view_tasks()
  return render_template('tareas.html', data=data)
```

La cuarta función se encarga de la gestión en la ruta /assignEspecifica". A esta función se le pasan como parámetros tipo, fecha, descripción y nombre, los cuales, los ha insertado el usuario. Tiene un funcionamiento similar al de la función anterior.

```
@app.route('/assignEspecifica', methods=['POST'])
def assignEspecifica():
  tipo = "Especifica"
  fecha = request.form['fechaEspecifica']
  desc = request.form['descripcionEspecifica']
  nombre = request.form['nombreEspecifica']

  assign_tasks(tipo, fecha, desc, nombre)
  data = view_tasks()
  return render_template('tareas.html', data=data)
```

La quinta función se encarga de la gestión en la ruta /exit". Esta función llama ala función *exit_tasks()* de *emulator.py* y se encarga de eliminar el fichero que se usa para redireccionar la salida del emular y poder trabajar con los datos del mismo. Termina la función renderizando ïndex_inicio.html"

```
@app.route('/exit', methods=['POST'])
def exit():
  exit_tasks()
  if os.path.exists("pantalla.txt"):
    os.remove("pantalla.txt")
  return render_template('index_inicio.html')
```

En este fichero contamos como hemos comentado antes con una función encargada de gestionar el caso de que cierren la aplicación de forma abrupta. Tiene la misma funcionalidad que la función anterior, solo que no renderiza ninguna pantalla, puesto que se cierra la aplicación.

```
def on_application_exit():
  exit_tasks() # Ejecuta tu funcion antes de cerrar la aplicacion
  if os.path.exists("pantalla.txt"):
    os.remove("pantalla.txt")

# Registra la funcion on_application_exit para que se ejecute al cerrar la aplicacion
atexit.register(on_application_exit)
```

03

DIFICULTADES ENCONTRADAS

La mayor dificultad encontrada fue la interacción entre nuestro programa y el emulador. Conseguimos encontrar una biblioteca de Python que realizaba la mayoría de interacciones con el emulador.

Otra dificultad fue el hecho de gestionar cuando la pantalla se llenaba el programa fallaba porque hacía falta una interacción más para que volviese al principio. Lo solucionamos limpiando la pantalla cada vez que realizamos una acción.

Por último, encontramos problemas a la hora de crear el ejecutable, pero explicaremos esta cuestión en el CREACIÓN DEL EJECUTABLE.

04

FUNCIONAMIENTO DEL PROGRAMA

4.1. EJECUCIÓN DEL PROGRAMA

Para ejecutar el programa se requiere un sistema operativo Windows. Para ejecutarlo basta con clicar sobre el archivo .bat "Gestor de Tareas.bat.[el] se encargará de lanzar la aplicación.

4.2. USO DEL PROGRAMA

Una vez lanzada la aplicación lo primero que muestra es la ventana para iniciar sesión como se observa en la *Figura 1*. El programa no permite al usuario iniciar sesión si el nombre de usuario o la contraseña están vacíos. Una vez introducidos los datos, si son incorrectos mostrará un mensaje de error como se muestra en la *Figura 2*, en caso de que sean correctos pero ya haya otro usuario conectado con esas credenciales mostrará un mensaje como se muestra en la *Figura 3*.

Una vez iniciada la sesión el programa realiza un iteración con el sistema legado para avanzar a la pantalla deseada.

A continuación se muestra la ventana que permite al usuario seleccionar el tipo de tarea a crear (*Figura 3*). Al seleccionar el botón "Nueva tarea"se abre una ventana emergente donde el usuario introduce los datos necesarios para crear dicha tarea (los datos no pueden ser vacíos).

Una vez rellenos los campos se presiona el botón "Guardarz el programa empieza a interactuar con el sistema legado para guardar dicha tarea. Una vez que la tarea se haya guardado se volverá a la pantalla principal con la diferencia de que aparece la nueva tarea, tal y como se muestra en la *Figura 4*.

05

CREACIÓN DEL EJECUTABLE

La creación del ejecutable no hemos podido realizarla de forma satisfactoria. Hemos probado diferentes módulos como por ejemplo pyinstaller, pyoxidizer, py2exe. Seguimos los pasos para la realización del ejecutable según la documentación de cada módulo, pero no pudimos conseguir que funcionará.

Tenemos la teoría de que al usar Flask que es para servicios web no es posible realizar un ejecutable como tal. Como alternativa hemos realizado un .bat que se encarga de crear un entorno donde instala las dependencias y tras esto lanza el gestor de tareas. Al finalizar el programa el bat se encarga de la eliminación de dicho entorno. Hemos decidido realizar esto así porque es para un entorno de prácticas. En un caso real tendríamos un bat para instalar el entorno, otro para lanzar el programa y otro para eliminar el entorno simulando la instalación, la ejecución y la desinstalación del programa respectivamente.

5.1. EXPLICACIÓN DE *GESTORDETAREAS.BAT*

GestorDeTareas.bat es un archivo que lanza otros dos archivos .bat. Este fichero se ha creado para cumplir con uno de los requisitos de la práctica que es que se tiene que poder lanzar haciendo click a un solo fichero. Este fichero lanza un primer fichero (*launcher.bat*) que es el encargado de lanzar el programa y luego lanza un segundo fichero (*uninstaller.bat*) que es el encargado de eliminar el entorno virtual. Esta eliminación se hace en un fichero a parte, porque en el caso de querer lanzar varias veces la aplicación el no eliminar el entorno virtual acelera mucho el relanzar la aplicación.

El archivo *launcher.bat* comienza chequeando si Python está instalado:

```
:: Verificar si Python esta instalado
python --version > nul 2>&1
if %errorlevel% neq 0 (
  echo Python no esta instalado. Por favor, instale Python antes de continuar.
  pause
  exit /b 1
)
```

Luego crea un entorno virtual y lo activa:

```
:: Crear el entorno virtual
python -m venv "%VENV_PATH%"

:: Activar el entorno virtual
echo Creando entorno virtual en %VENV_PATH% si es necesario ...
call "%VENV_PATH%\Scripts\activate.bat"
```

Luego instala las dependencias:

```
:: Instalar las dependencias de tu aplicacion
echo Instalando dependencias si es necesario ...
%PIP% install -r "%REQ_PATH%\requirements.txt" > nul 2>&1
```

Luego lanza la aplicación pero el .bat sigue ejecutándose:

```
:: Ejecutar la aplicacion Flask y esperar a que termine
cd GestorDeTareas
echo Lanzando Gestor de tareas (x3270) ...
start /wait python ".\app.py"
```

El archivo *uninstaller.bat* elimina el entorno virtual creado en *launcher.bat*, si no existiera dicho entorno , no haría nada.

```
:: Eliminar el entorno virtual
echo Eliminando entorno virtual...
rmdir /s /q "%VENV_PATH%"
echo Eliminado
```

06

TAREAS Y DEDICACIÓN

Tarea	Martina	Héctor	Dani
Sesión de prácticas	3h	0h	3h
Pruebas con el emulador	30min	30min	30min
Implementación de funciones de *emulator.py*	10h	10h	20h
Implementación de funciones de *app.py*	10h	10h	20h
Creación del fichero "html"	15min	15min	15min
Creación del ejecutable	2h	2h	5h
Total	**25h y 45 min**	**22 h y 45 min**	**48h y 45 min**