Pied Piper OS

Javier Emiliano Escobedo Padilla
Rodrigo Orozco Hidalgo

Video

https://youtu.be/AMX-v-1mgk8

Code

Disponible en

GIT: https://github.com/solvedrex223/Final_project_OS.

para acceder directamente a los módulos del **Filesystem**(cada modulo esta en su respectivo paquete):

https://github.com/solvedrex223/Final_project_OS/tree/Unix_Release/venv/Web/web_server/file_system/FileSystem/Packages

Disponible en

para acceder directamente a los **módulos del Sistema de compression**(cada modulo esta en su respectivo paquete):

https://github.com/solvedrex223/Final_project_OS/tree/Unix_Release/venv/Web/web_server/file_system/FileCompSys

Disponible en

Para acceder al servidor web se debe entrar aquí:

https://github.com/solvedrex223/Final_project_OS/tree/Unix_Release/venv/Web/web_server

El servidor web se divide en diferentes apps para los diferentes componentes del OS:

Web server - Master file_system gui

Executive summary

The purpose of this project is to develop a file compression and storage service, run through a linux based OS in a web server.

There are multiple parts in this project. In this presentation we are showing the design of the file system that we will be using for the os.

For more information around the WebDev, FileCompression, and current state of the project please visit https://github.com/solvedrex223/Final_project_OS.

Problem

1. Access your files everywhere at anytime.
2. High data storage consumption.
3. High costs for data storage and maintenance.

Solution

The web application will provide the user a file storage and administration service, through friendly GUI.

Furthermore, the web app will provide advanced users access to a console interface.

The system BTS will compress and decompress files as needed to optimize data storage.

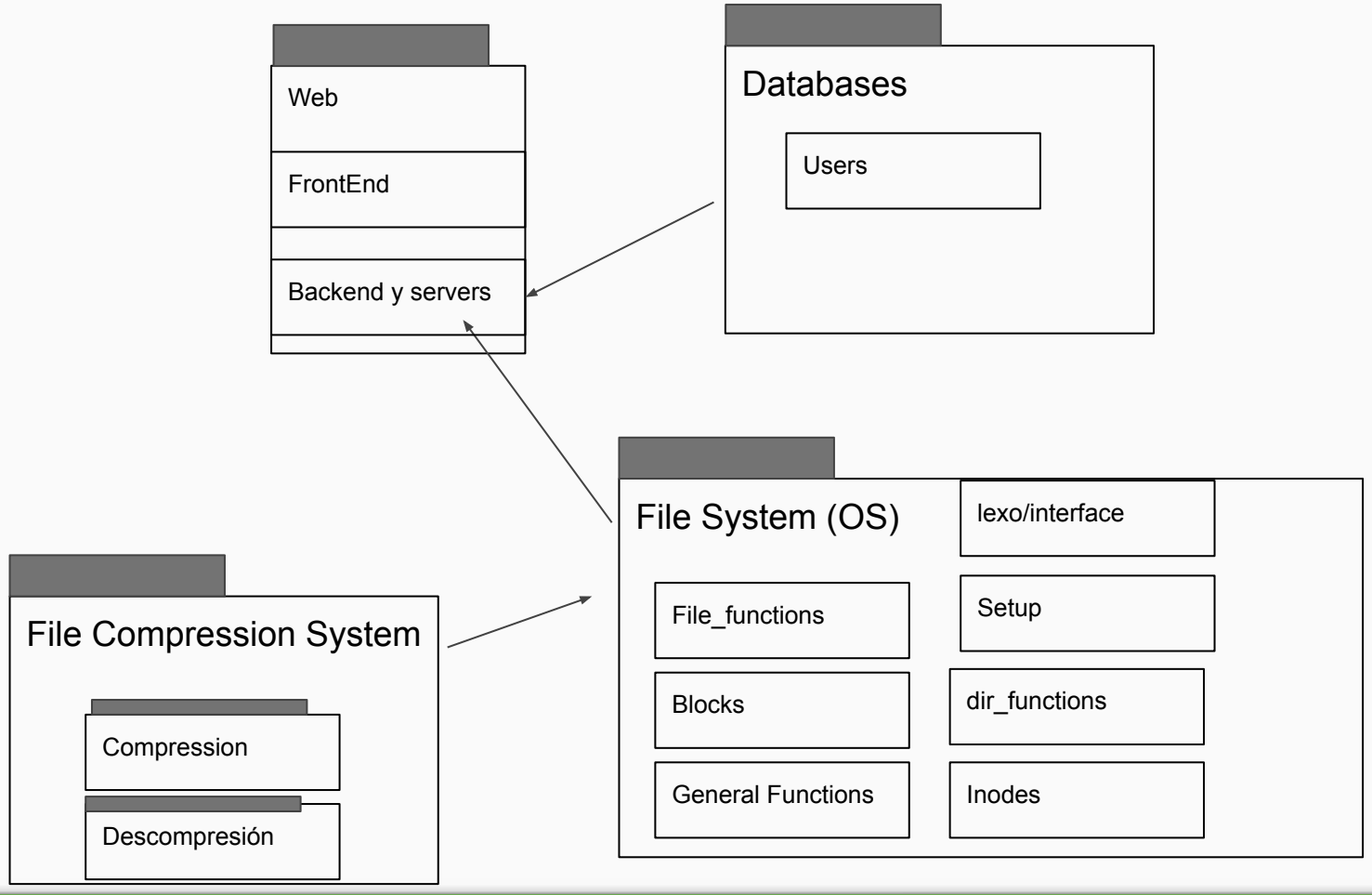
Benefits

Low storage costs.

Availability of your information everywhere.

Arquitectura

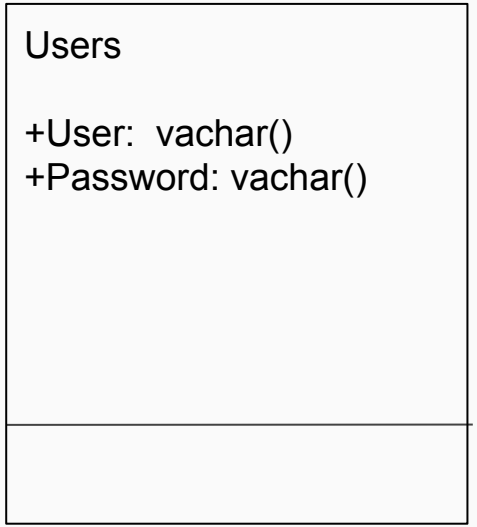
Packet Diagrams



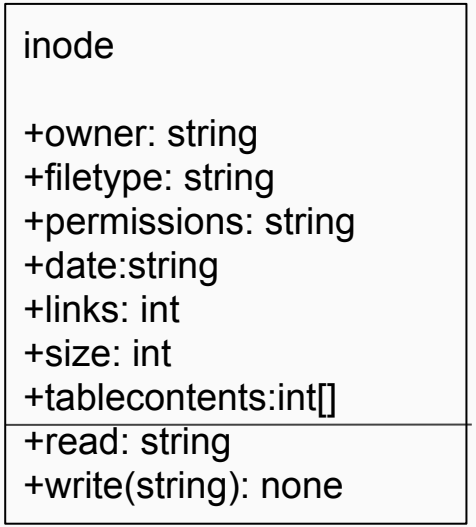
UML Diagram a de Paquetes

Class Diagrams

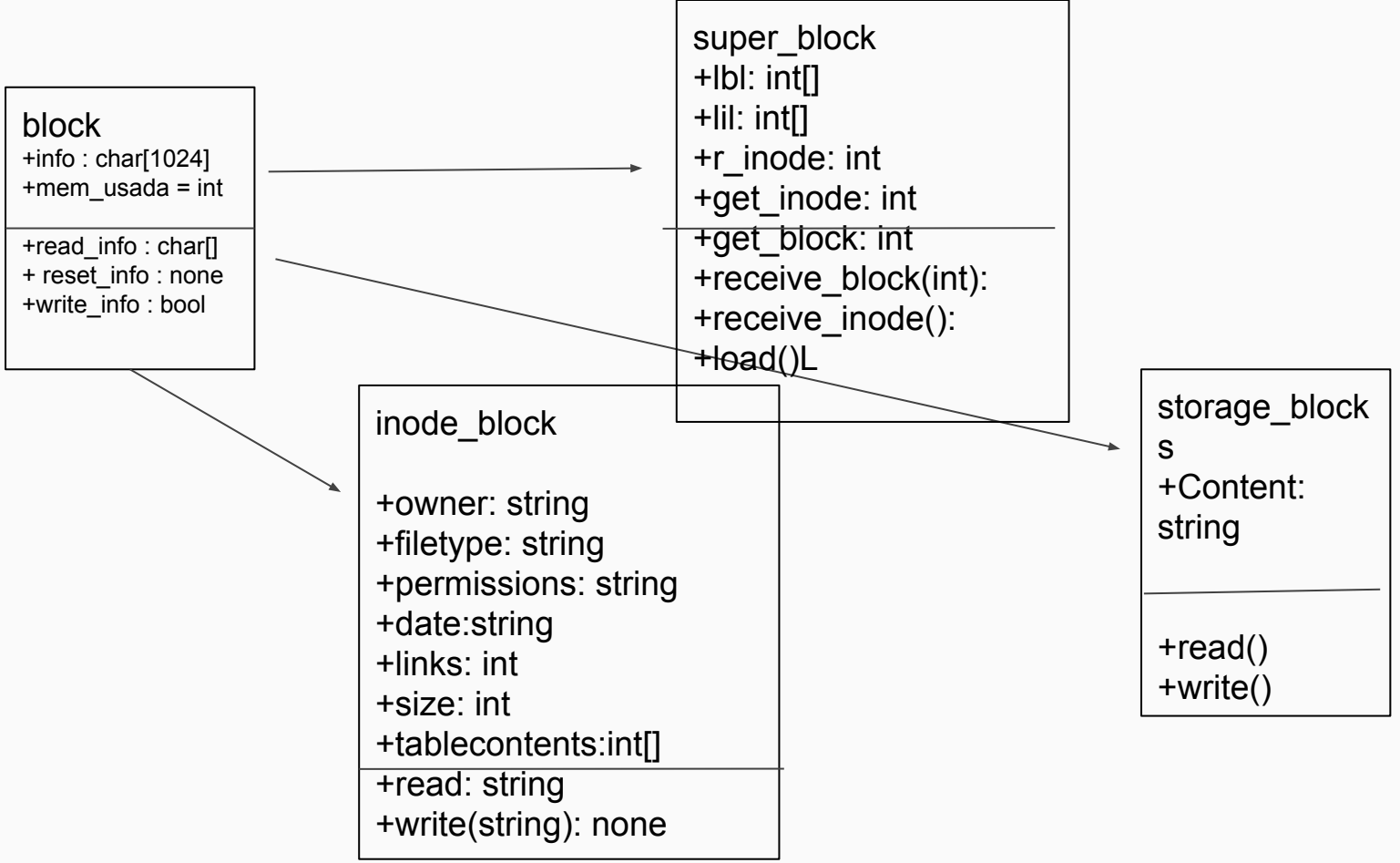
DataBase



Inodes


inode
+owner: string
+filetype: string
+permissions: string
+date:string
+links: int
+size: int
+tablecontents:int[]
+read: string
+write(string): none

Blocks


block
+info : char[1024]
+mem_usada = int
+read_info : char[]
+ reset_info : none
+write_info : bool
super_block
+lbl: int[]
+lil: int[]
+r_inode: int
+get_inode: int
+get_block: int
+receive_block(int):
+receive_inode():
+load()L
inode_block
+owner: string
+filetype: string
+permissions: string
+date:string
+links: int
+size: int
+tablecontents:int[]
+read: string
+write(string): none
storage_block
s
+Content:
string
+read()
+write()

Storage

Sistema tendrá:

1,000,000 bloques de 1k cada uno, representados por un archivo txt seriado.

1,000 bloques se destinarán a los inodos. en total habrá 16k inodos.

SuperBlock:

LBL contains 8* ints (4 bytes each)

LIL contains 8* ints (4 bytes each).

Inode:

owner: owner, group (stored as bin numbers) DC= 2 Bytes

file_type: “d” directory , “-” file, “0” available, “l” link DC= 1 Bytes

access_permissions: owner, group, others. DC= 2 Bytes

File access times: last modified 8(bytes) format MMDDYYYY DC= 8 Bytes

number_of_links: representing the number of names the file has in the directory hierarchy. (3byte) no file can be referenced more than 16,777,215 times. DC= 3 Bytes

size: Data in a file is max 4,294,967,295 4GB, sored as a 4 bit bin number DC= 4 Bytes

Table of contents: max_file_size(17,247,248,000 | 17GB) DA [0:7], SIA [8], DIA [9], TIA [10] DC= 44 Bytes

Inode size 64 bytes.

storage_block

fits 1024 bytes | 1024 chars | 256 4byteints.

Inode_block:

16 Inodes per Block.

Algorithms

Algoritmos de parseo

Algoritmos Utilizados:

```
B = number_bytes

def string_to_int(string): Parsero de cadena a entero O(|string|)
def int_to_string(num, bytes): Parseo de entero a cadena. O(B)
def write_str_bin(file, string): Escritura a disco de String a
Bytes. O(|string|)
def bytes_to_string(byte_array): Parseo lectura Bytes a String.
O(B)
```

Data Block

Algoritmos Utilizados:

```
B = number_bytes

def read_info(self) #Reads binary from file and updates self.info
with string formatted info RT = O(B)
def reset_info(self): #Resets the block RT = O(B)
def write_info(self): #Writes the whole Block to disk RT = O(B)
```

Superblock (Use guide)

```
'''

superblock:
super_block(1) -> LIL
super_block(2) -> LBL

use
init()
load()
code ...

Useful SuperBlocks methods
load()
free_block()
free_inode()
receive_block(int)
receive_inode(int)
all other methods should not be called on regular basis since
they are used to built the "Useful ones".
'''
```

SuperBlock

Algoritmos Utilizados:

```
B = number_bytes

def __init__(self, id): if id == 1 -> LIL, if id == 2 -> LBL O(1)
def write_info(self): #Actualiza el contenido del Bloque y lo
escribe a disco. O(|contenido|)
def append(self, num): Añade un bloque al final de a la LIL o LBL.
y actualiza la información O(|LIL| o |LBL|). tarda lineal por el
update.
def update_info(self): #Updates self.info to be up to date with
LBL or LIL O(|LIL| o |LBL|)
def load(self): #Reads the file content and sets up the LBL or
LIL O(B)
def pop(self): Elimina un bloque al final de a la LIL o LBL. y
actualiza la información. O(|LIL| o |LBL|). tarda lineal por el
update.
def free_block(self): #Frees a Memory Block from the LBL, returns
the freed Block. O(|LBL|), toma este tiempo porque se realiza el
update.
def refill_LBL(self): #Refills the the LBL when would empty,
returns the pending BLock. O(|LBL|), toma este tiempo porque se
realiza el update.
def recieve_block(self, block_no): Recibe un bloque liberado por
un archivo borrado, lo añade  a la LBL y actualiza disco, utiliza
el mismo algo que unix. O(|LBL|),por el update.
```

SuperBlock

Algoritmos Utilizados:

```
B = number_bytes

def clean_LBL(self,block_no): se acciona cuando la LBL se vacia,
funciona igual que el de Unix. O(|LBL|)

def receive_inode(self, inode_no): recibe un inodo cuando se
elimina un archivo, directorio, etc y actualiza el disco, funciona
igual que el de Unix. O(|LIL|)

def change_rem_inode(self,inode_no): Actualiza el remember inode y
escribe los cambios en disco. O(|LIL|) por el update.

def clear_inode(self, inode_no): Elimina el inodo del disco.
O(Inode_size)

def free_inode(self): Devuelve un no_inodo para asignarselo a un
nuevo archivo, dir, etc y actualiza disco. O(|LIL|) por el update.
```

Inode Use Guide

```
'''

Inodes:
Inode(int) -> instantiates a inode object that should be used
with existing inodes, needs to be read() before used.
Inode(*) -> Should be used when creating a new inode.

Useful SuperBlocks methods
write()
read()
delete()
parse_all()
binarize_all()
all other methods should not be called on regular basis since
they are used to built the "Useful ones".
'''
```

Inode

```
def parse_all(self): #Parses the binary read from file content to
workable in memory info. calls the following self descriptive
methods.
    self.parse_size()
    self.parse_toc()
    self.parse_group()
    self.parse_owner()
    self.parse_access_permissions()
    self.parse_links()
```

O(inode_size)

Inode

```
def binarize_all(self): binariza toda la informacion en memoria
para prepararla para escritura a disco.
    self.binarize_owner()
    self.binarize_group()
    self.binarize_size()
    self.binarize_links()
    self.binarize_toc()
    self.binarize_access_permissions()
    return True
```

O(inode_size)

Inode

```
def get_block(self): #returns the block the inode is located.
   return 3 + self.id//16

def get_offset(self): #returns the offset of the inode
   return ((self.id-1)%16) * 64
```

O(1)

Inode

```
def write(self): writes the inode content to disk, before wirting
it, the info should be binarized.
O(inode_size)

def read(self): reads the inode content from disk, after reading
it the info gets parsed.
O(inode_size)

def delete(self): changes the filetype in the disk to “0” .
O(1)
```

Setup

```
if __name__ == "__main__":
    set_hard_drive()
    set_inodes_list()
    set_inodes()
    set_block_list()
```

Setup

```
   set_hard_drive() # sets ablank hard drive at a given location
that contains 1,000,000 .block files   O(no_blockfiles)


set_inodes_list() # sets the Inode list for initial use, not the
first 8 inodes are already assigned.O(blocksize)


set_inodes() #sets all the inodes in disk covers blocks 3-1002.
O(no_inodeBlocks)


set_block_list() #sets the block list through the disk starting in
BLock 2 and also writes iteratively the extensions of the LBL unix
style throug the disk.   O(no_blockfiles)
```

Create A File / Dir / ect

```
create a File(File name, user, file_content)

no_inodo = LIL.free_inode()
inodo = Inode("File name", "user", no_inodo, |file_content)
no_blocks  = ceil(|file_content| / 1024)
assign_blocks(no_block, file_content){
      for x in no_block:
            get_Block
            #primeros 8
                  write_Block(file_content[x: 1024 *x])
            #Los demas
                  DFS(x)
                  write_Block(file_content[x: 1024 *x])
}

inodo.binarizeall()
inodo.write()
return True
```

Remove A File / Dir / ect

```
remove_A_File(Inode_no)
inodo = Inode(Inode_no)
no_blocks  = ceil(|file_content| / 1024)
free_blocks(no_block){
      for x in no_block:
            get_Block
            #primeros 8
                  LBL.Receive_Block()
            #Los demas
                  DFS(x)
                  LBL.Free_Block()
}
LIL.receive(Inod_no)
return True
```

Read A File / Dir / ect

```
Read a File(no_inode):

inodo = Inode(no_inode)
inodo.parseall()
cnt = 0
content =””
for file in inodo.TOC.
      if cnt <=8:
            content += read(file)
      else:
            DFS(file)
            content += read(file)
      cnt ++

return content
```

CD

```
CD(no_inode):
inodo = Inode(no_inode)
inodo.load()
return inodo
```

Lexo

```
def lexo():
    ruta = '/'
    user = 'root'
    cwd = Inode(id= 2)
    cwd.read()
    files = find_files(cwd)
    while (1):
        msg = input()
        if msg  == mkdir [dest]: mkdir(dest)
        if msg  == mkfile [dest]: mkfile(dest)
        if msg  == cd [dest]: cd(dest)
        if msg  == rm [dest]: rm(dest)
        if msg  == rmdir [dest]: rmdir(dest)
        if msg  == ls: ls()
        if msg  == cat [dest]: cat(dest)
```

File comrpession

Estado

Construido en base al algoritmo de Huffman.

Ya codeado y funcional, solo necesita adaptarse la lectura y escritura al Sistema de archivos en desarrollo.

Server

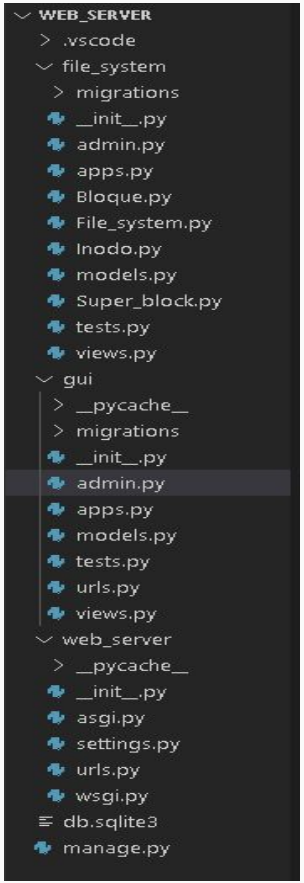
Servidor Web

El servidor web está creado en Django utilizando python.

Django tiene ciertas ventajas que nos da contra otras opciones para facilitar el desarrollo del OS:

1. Apps: las apps son distintas partes del servidor que funcionan en conjunto, dándonos la habilidad de poder desarrollar las diferentes partes del OS por separado.
2. Documentación: Django es de los frameworks más utilizados en el mundo, por lo que la documentación de ésta es muy vasta.
3. Escalabilidad: Django es de los frameworks más robustos, por lo que tenemos muchas opciones de optimización y recursos que podemos usar.

Archivos


WEB_SERVER
.vscode
file_system
migrations
__init__.py
admin.py
apps.py
Bloque.py
File_system.py
Inodo.py
models.py
Super_block.py
tests.py
views.py
gui
__pycache__
migrations
__init__.py
admin.py
apps.py
models.py
tests.py
urls.py
views.py
web_server
__pycache__
__init__.py
asgi.py
settings.py
urls.py
wsgi.py
db.sqlite3
manage.py

Web Server

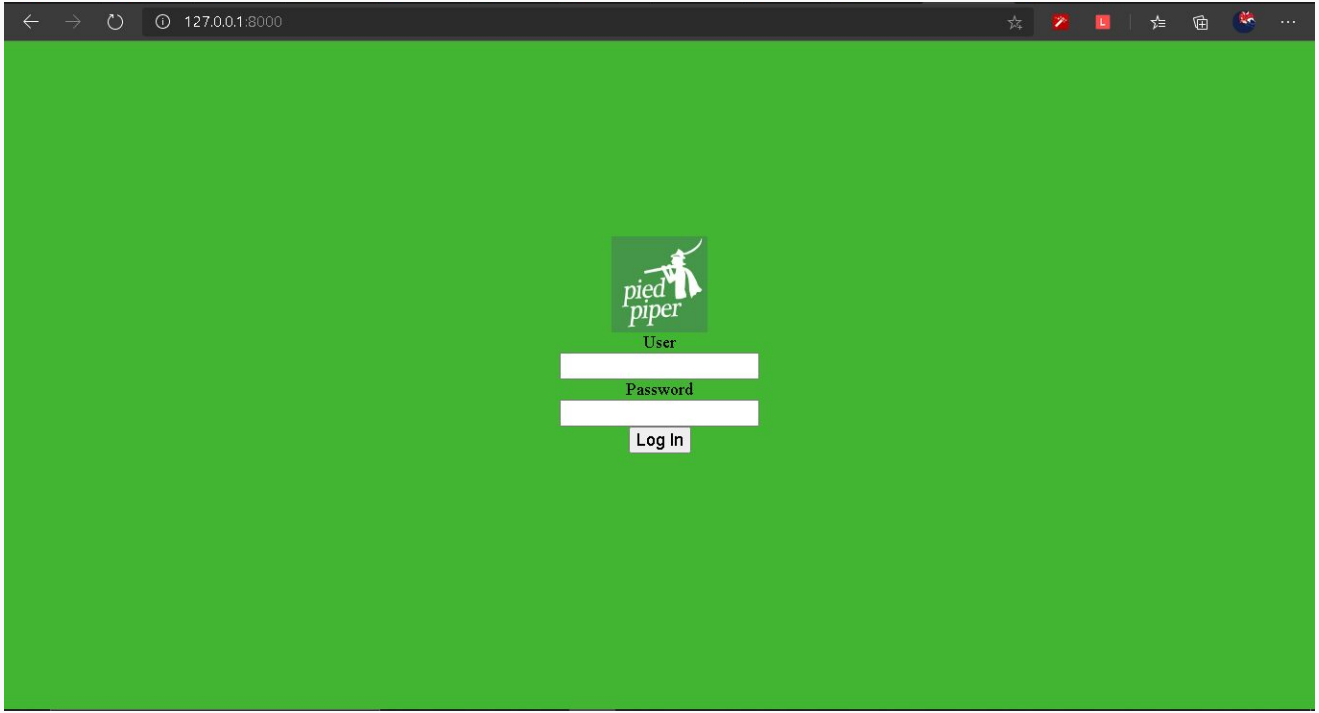
El apartado de Web Server es el corazón del servidor. Aquí se controlan los URLs, sus direcciones y cómo se comunican las diferentes apps. EL archivo a destacar para poder mantener todo el servidor es settings.py

File_system

Aquí se encuentran las clases y los métodos que utilizamos para poder llamar al File System. Todas las apps que necesiten modificar documentos deberán llamar a alguna clase del File System para poder realizar dichas modificaciones.

GUI

Esta app es la que se dedica a mostrar al usuario la interfaz gráfica con la que interactúa con el OS.



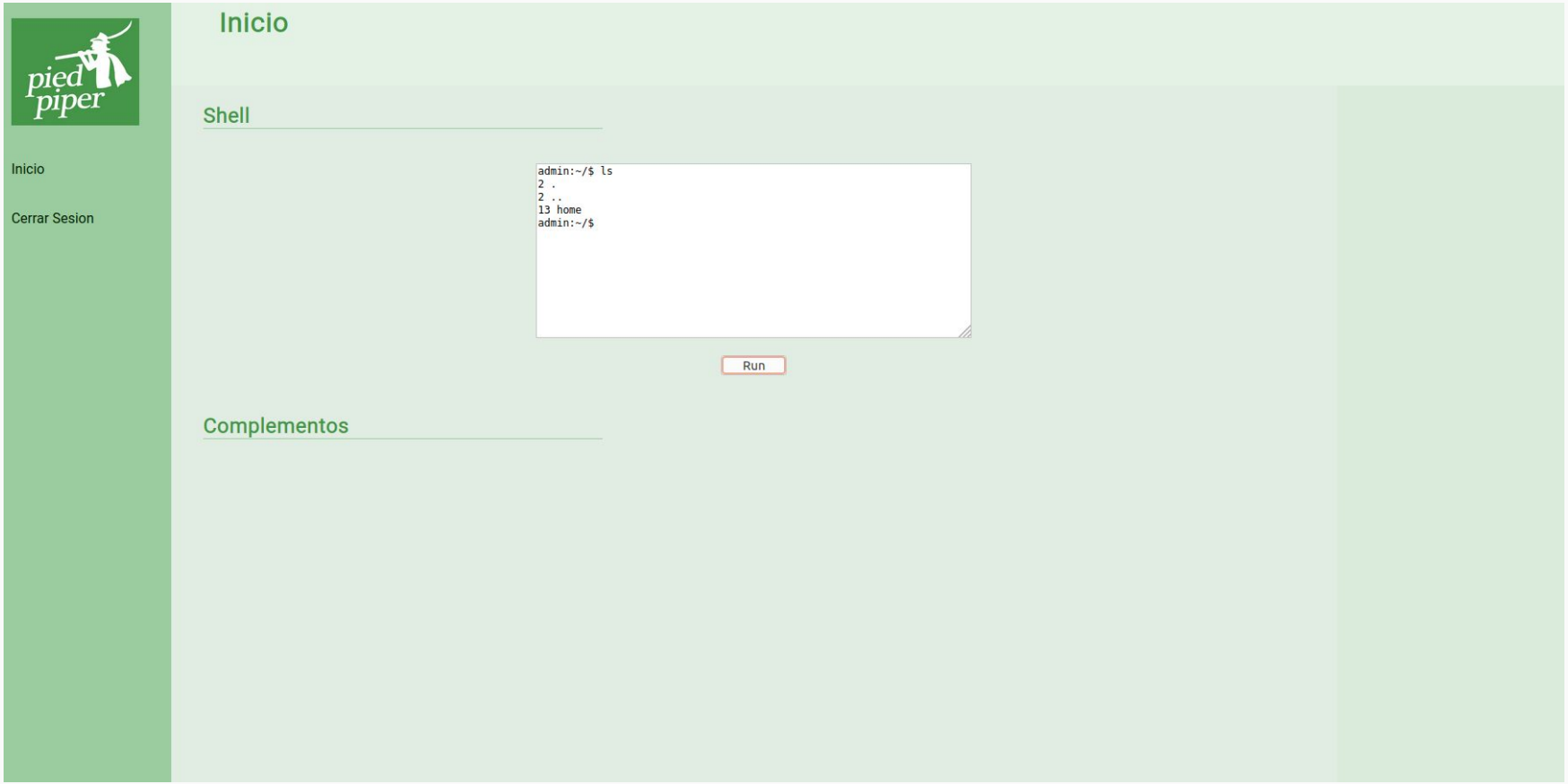
Usuarios

Los usuarios son manejados con una base de datos en SQLite que es una base de datos dentro de python. Ésta se usa para checar los logins de los usuarios y se toman de ahí los privilegios que tiene cada usuario para poder modificar archivos e utilizar los diferentes comandos dentro de la terminal.

Registro de Usuarios y Login

Los usuarios se registran a través de un formulario que guarda en la base de datos el usuario y la contraseña que se ingresan. Así mismo, se utiliza otro formulario para validar el login. Éste último formulario regresa el valor que equivale a los privilegios que tiene el usuario.

Gui


Inicio
Inicio
Cerrar Sesion
Shell
admin:~/$ ls
2 .
2 ..
13 home
admin:~/$
Run
Complementos