

Proyecto ClassAdmin

ÍNDICE

1. [Introducción](#)
 - 1.1. [Síntesis](#)
 - 1.2. [Justificación del proyecto](#)
 - 1.3. [Objetivos](#)
2. [Desarrollo](#)
 - 2.1. [Definición del problema](#)
 - 2.2. [Análisis del sistema](#)
 - 2.3. [Diseño](#)
 - 2.4. [Implementación](#)
 - 2.5. [Conclusiones](#)
 - 2.6. [Líneas de investigación futuras](#)
 - 2.7. [Bibliografía](#)
 - 2.8. [Anexos](#)
3. [GNU Free Documentation License](#)

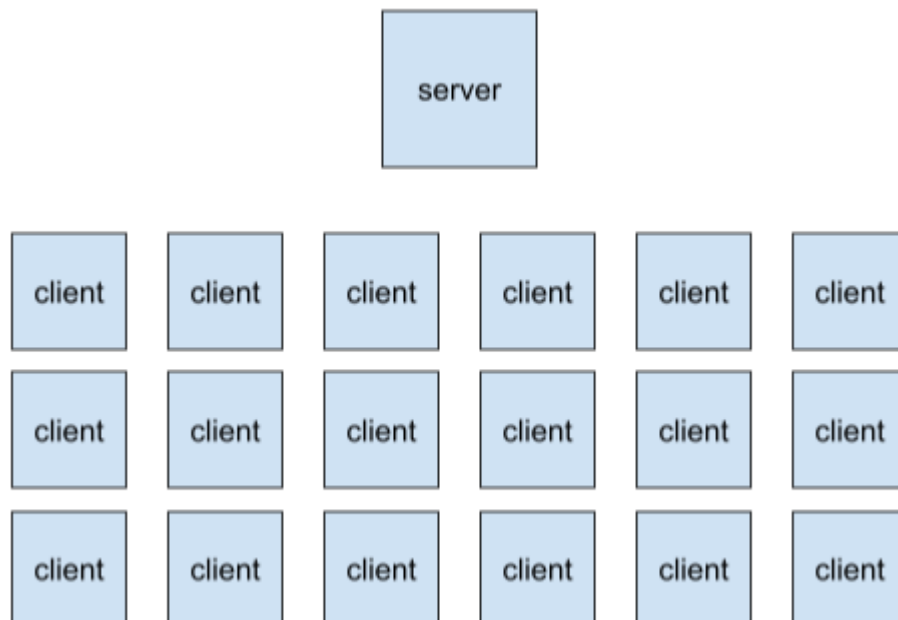
Proyecto ClassAdmin

1. Introducción

1.3. Síntesis

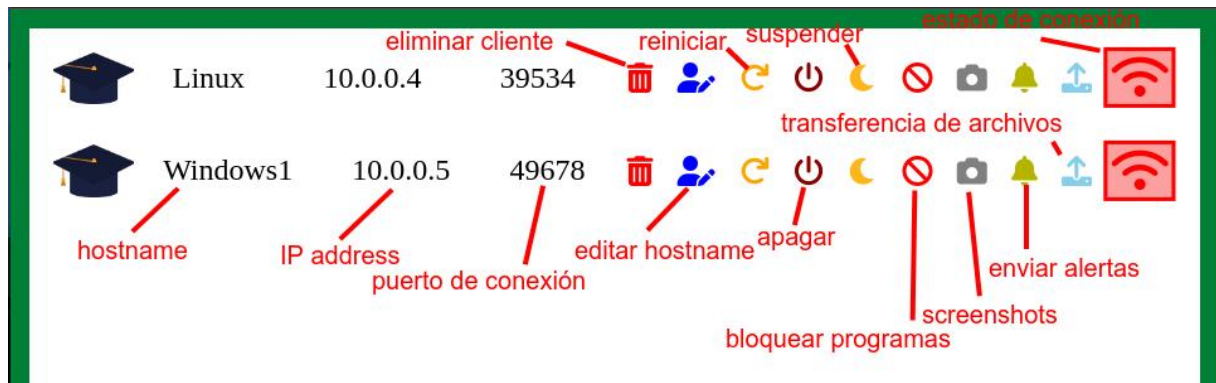
El proyecto estará desarrollado completamente en Python 3, con scripting Powershell y Bash para según qué funcionalidades. El funcionamiento del software es el siguiente.

Se instalará un servidor que estará localizado en el ordenador del profesor y este servicio que hace de servidor estará en escucha de conexiones clientes. Es decir, supongamos que los usuarios son los alumnos y estos encienden los ordenadores. Pues al encenderse cada ordenador se ejecutará un servicio cliente (agente) que se conectará al servidor.



El profesor puede manipular el servidor a través de la aplicación web Django, que es un dashboard en que se tiene que loguear con contraseña y OTP, y al loguearte este te muestra los clientes conectados y desconectados. Desde Django se pueden hacer diferentes operaciones con cada cliente.

Proyecto ClassAdmin



- Hostname: Se muestra el nombre del ordenador del cliente.
- Dirección IP: Se muestra la dirección IP del cliente.
- Puerto: Se muestra el puerto por el cual el cliente está conectado al servidor.
- Eliminar cliente: Elimina el registro del cliente en la base de datos de ClassAdmin.
- Reiniciar: Reinicia el ordenador del cliente
- Apagar: Apaga el ordenador del cliente
- Suspender: Bloquea y suspende el ordenador del cliente.
- bloquear programas: Bloquea los programas instalados en el PC.
- Screenshots: Ejecuta una captura de pantalla en el escritorio del cliente y lo guarda desde el cliente en una carpeta compartida creada en el servidor.
- Alertas: Permite enviar alertas al cliente
- Subir archivos: Permite subir archivos (.pdf,.txt,.png,etc.) al cliente.
- Cerrar sesión: Cierra la sesión del dashboard.
- Cambiar configuración: Permite editar la contraseña de la sesión, así como cambiar el OTP, el puerto, el usuario por el cual ejecutará las notificaciones que se mostrarán y descargarse los códigos de recuperación.

current password

new password

repeat new password

new port

Current port 7788

Notificaciones

run notification as user...

Save notifications user

The notification run as user usuario

* The notifications settings will be applied at restart the service

+

↓

save

Proyecto ClassAdmin

-

1.4. Justificación del proyecto

He escogido un proyecto propio por meta personal. Mi objetivo era entender y comprender el funcionamiento interno de cualquier aplicación web que controle de manera remota hosts. Casos como Microsoft Azure, Google Cloud, AWS, etc. Obviamente no está tan bien elaborado, pero la lógica de funcionamiento y la comunicación es la misma o parecida (conexiones sockets). Aparte también abarca lo que sería todos los módulos dados en estos 2 años de curso.

ISO / ASO: Subprocesos, hilos, scripts en powershell y bash así como la instalación y configuración de este proyecto (ClassAdmin)

PAX: Concepto de un socket en redes y de direcciones IPs así como puertos.

GBD/ABD: Creación de la DB con las tablas y el usuario para conectarse a este.

LLM / IAW: Creación de la aplicación web con Django

LLM: Hacer el frontend de la aplicación

IAW: Hacer el backend con MVC con las tecnologías Django y la creación de la API.

SXI: El despliegue de la aplicación web Django bajo un servicio apache2, así como también la utilización del archivo "hosts" para hacer resolución DNS localmente. También el despliegue en Azure.

SAD: La más importante, ya que el proyecto está desarrollado con todos mis posibles conocimientos sobre seguridad. Desde añadir SSL a la web Django (también visto en SXI) hasta encriptación punto a punto de las conexiones. Y la encriptación de la contraseña al almacenarla en la DB o enviar el login así como la utilización de un OTP

Está todo programado con Python 3 desde el servidor y el cliente hasta la aplicación web y API Rest. También utilizando scripts de powershell/bash

Proyecto ClassAdmin

para algunas funcionalidades.

A parte de que los conocimientos que puedo adquirir durante el desarrollo de este, me pueden servir bastante en un futuro, sobre todo en el mundo de la ciberseguridad.

1.5. **Objetivos**

Los objetivos principales del proyecto serán:

Control remoto de un equipo desde una aplicación web.

Bloquear aplicaciones remotamente desde un dashboard

Transferencia de archivos del servidor al cliente

Practicar mucho más la programación en Python 3

Profundizar más en el módulo sockets de Python 3

Licenciar el proyecto con GPLv3 (GNU Public License version 3) así como GFDL (GNU Free Document License)

Aprender y saber utilizar Microsoft Azure

Entender mejor el funcionamiento interno de Windows

Mejorar los conceptos de PowerShell

Automatización de la instalación y desinstalación del programa

Creación de servicios en Windows (mediante un programa tercero NSSM)

2. Desarrollo

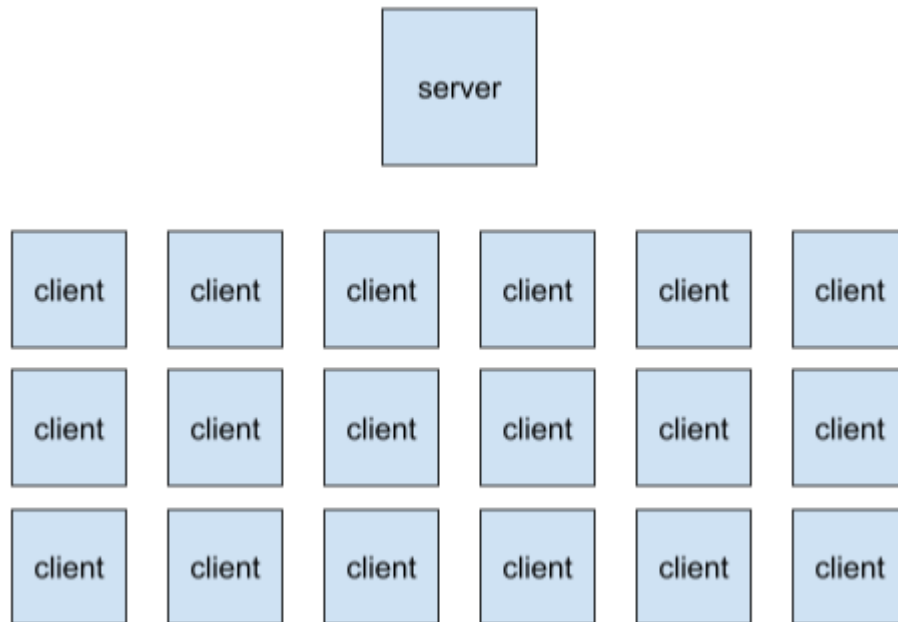
2.3. **Definición del problema**

Un instituto necesita tener el control remoto de los equipos de las aulas de sus alumnos, así como la restricción de programas como juegos, ocio, programas que no se quieran usar temporalmente, etc. Además poder enviar alertas,

Proyecto ClassAdmin

modificar el hostname de un equipo, apagar, reiniciar, suspender el equipo y hacer una captura de pantalla, así como la posibilidad de enviar archivos remotamente a sus alumnos individualmente. El sistema se necesita para el uso de exámenes cuando lo hagan en el ordenador.

2.4. Análisis del sistema



La solución que se ha ofrecido es la creación de un dashboard web que al profesor le permita desde el panel administrativo ver todos los clientes (ordenadores de los alumnos) que están conectados al servidor (ordenador del profesor) y poder interactuar con cada uno de ellos haciendo, haciendo las funciones anteriormente explicadas.

El sistema tendrá las siguientes tecnologías:

- Python 3

Está desarrollado con python ya que es un lenguaje de programación muy versátil y legible. Además se puede programar el software para que sea multiplataforma (Windows/Linux) sin mucha dificultad y con posibilidad de compilar el código si se quiere, gracias a la librería PyInstaller.

- Django

Proyecto ClassAdmin

Utilizo Django ya que es un framework de Python que te permite crear aplicaciones webs MVC

- Sockets

Utilizo el módulo sockets en python para permitir conexiones entre equipos y poder hacer así un panel de administración de equipos.

- Multiprocesos

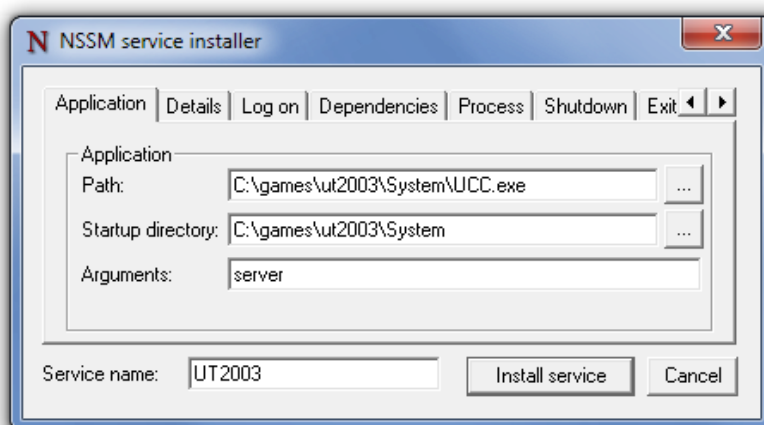
Los servicios, tanto cliente como servidor, ejecutan subprocesos, por ejemplo en el servidor por cada cliente que se conecta, su conexión se crea en un subproceso. Permitiendo así múltiples conexiones de clientes en el servidor.

- Hilos

Los servicios también tienen hilos que permiten ejecutar partes del código de manera paralela.

- NSSM

El NSSM es un programa que facilita la creación de servicios para Windows. Es el que utilizo para crear el servicio (cliente o servidor) en windows.



- Django Rest Framework

Lo utilizo para crear la API (Application Programming Interface) que permite desde la web o los servicios, interactuar con la base de datos de

Proyecto ClassAdmin

ClassAdmin

- Apache2/Xampp

Es el servicio web que se utiliza para levantar la aplicación web. Apache2 en caso de Linux y Xampp en caso de Windows

- MariaDB

Es la base de datos que se utiliza para almacenar los datos de los clientes que se conectan, así como el servidor. Se utiliza mariaDB, pues es una base de datos de código abierto y de software libre.

- Powershell

Los servicios cliente (ClassAdmin), utilizan scripts powershell en caso de Windows para cambiar el nombre, listar los programas existentes en el sistema operativo así como bloquearlos y desbloquearlos.

- Bash

Los servicios cliente (ClassAdmin), utilizan scripts bash en caso de Linux para cambiar el nombre, listar los programas existentes en el sistema operativo así como bloquearlos y desbloquearlos.

- Microsoft C++ Build Tools

Este software es de Microsoft y se debe de instalar en el caso de que se quiera instalar el servidor ClassAdminS en un Windows, ya que en el proyecto se utiliza el módulo psutils y este lo requiere.

El proyecto se desarrolló en este periodo de tiempo, no es 100% exacto:

2021

JANUARY S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	FEBRUARY S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28	MARCH S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	APRIL S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
MAY S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	JUNE S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30	JULY S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	AUGUST S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
SEPTEMBER S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30	OCTOBER S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	NOVEMBER S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30	DECEMBER S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

2022

JANUARY S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	FEBRUARY S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28	MARCH S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	APRIL S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
MAY S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	JUNE S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30	JULY S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	AUGUST S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
SEPTEMBER S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30	OCTOBER S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	NOVEMBER S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30	DECEMBER S M T W T F S 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

Desarrollo de la aplicación web y su base de datos: Se desarrolla la parte frontend y el backend con modelo-vista-controlador (MVC) gracias al framework Django de python. También la creación de la base de datos MariaDB.

Creación de la API: Django permite crear una API gracias a Django REST Framework, este tendrá los siguientes endpoints:

/status: GET, POST → Muestra los estados o añade nuevos

/status/<name>: PUT,DELETE,PATCH → Actualiza o elimina un estado

/clients: GET, POST → Muestra los clientes o añade nuevos

/clients/<id>: PUT,DELETE,PATCH → Actualiza o elimina un cliente


/clients/<address>/<hostname>: PUT,DELETE,PATCH →


Actualiza o elimina un cliente a partir de que una dirección IP y hostname coincidan

/servers: GET, POST → Muestra los servidores o añade nuevos

/servers/<id>: PUT,DELETE,PATCH → Actualiza o elimina un servidor


Proyecto ClassAdmin


 **Creación del servidor:** Se crea el servidor con Python 3.10 y se ejecuta como servicio.

 **Creación del cliente:** Se crea el cliente (agente) con Python 3.10 y se ejecuta como servicio.

 **Interactuar con los clientes desde el servidor mediante la web:**

Aquí es donde se crea el pipeServer y pipeClient, que gracias a este microservicio, permite que el servidor pueda enviar mensajes al cliente, desde la aplicación desde las funcionalidades de los botones (editar hostname, apagar, reiniciar...)

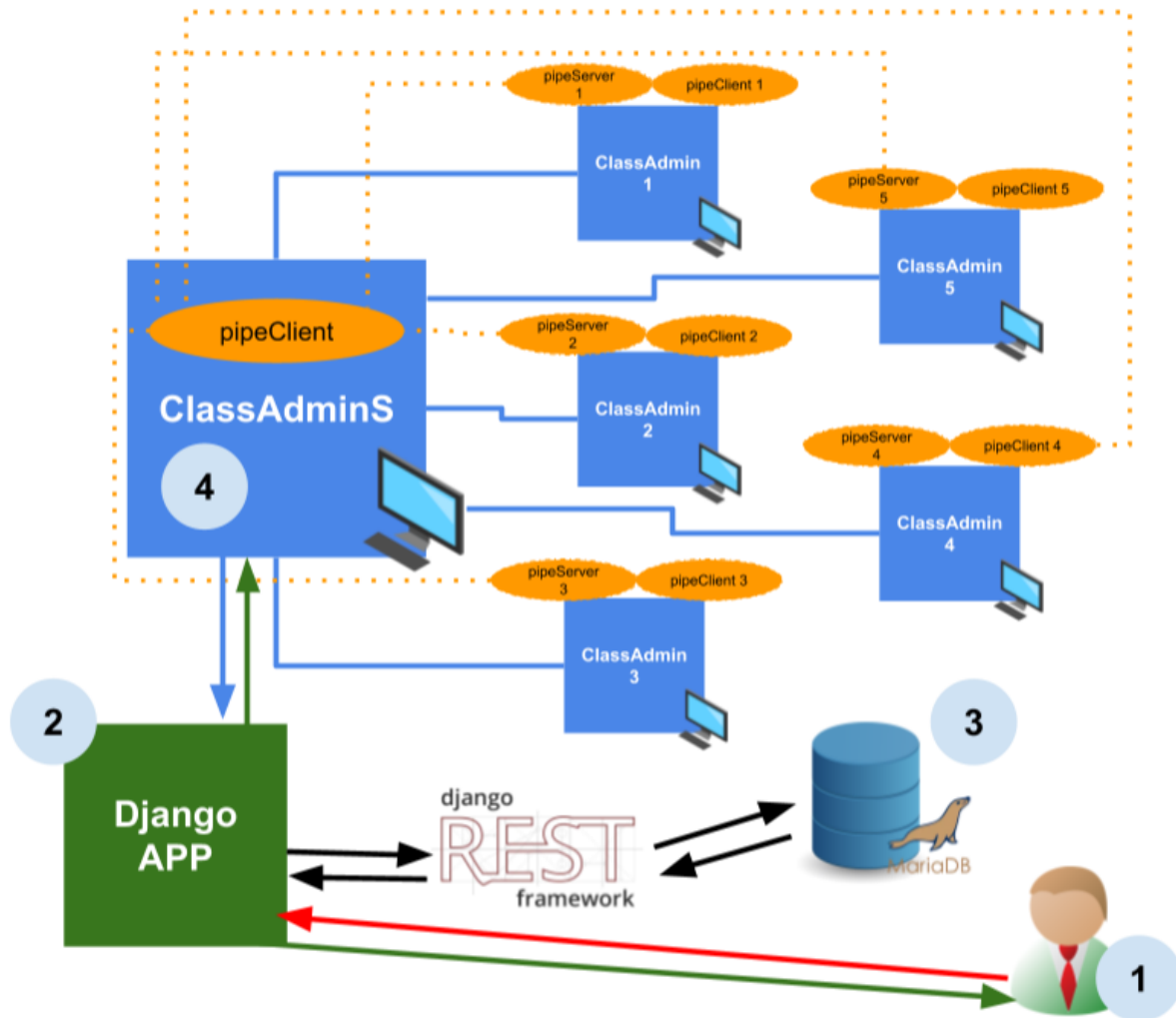
 **Despliegue del proyecto en Microsoft Azure:** El proyecto se desarrollo en local con máquinas virtuales (Virtualbox). Pero se desplegó en Microsoft Azure para mejor facilidad en el servidor.

 **Documentación y presentación del proyecto:** Se hace la documentación del proyecto en Google Docs y la presentación en Google Presentations.

2.5. Diseño

El diseño del sistema es el siguiente:

Proyecto ClassAdmin



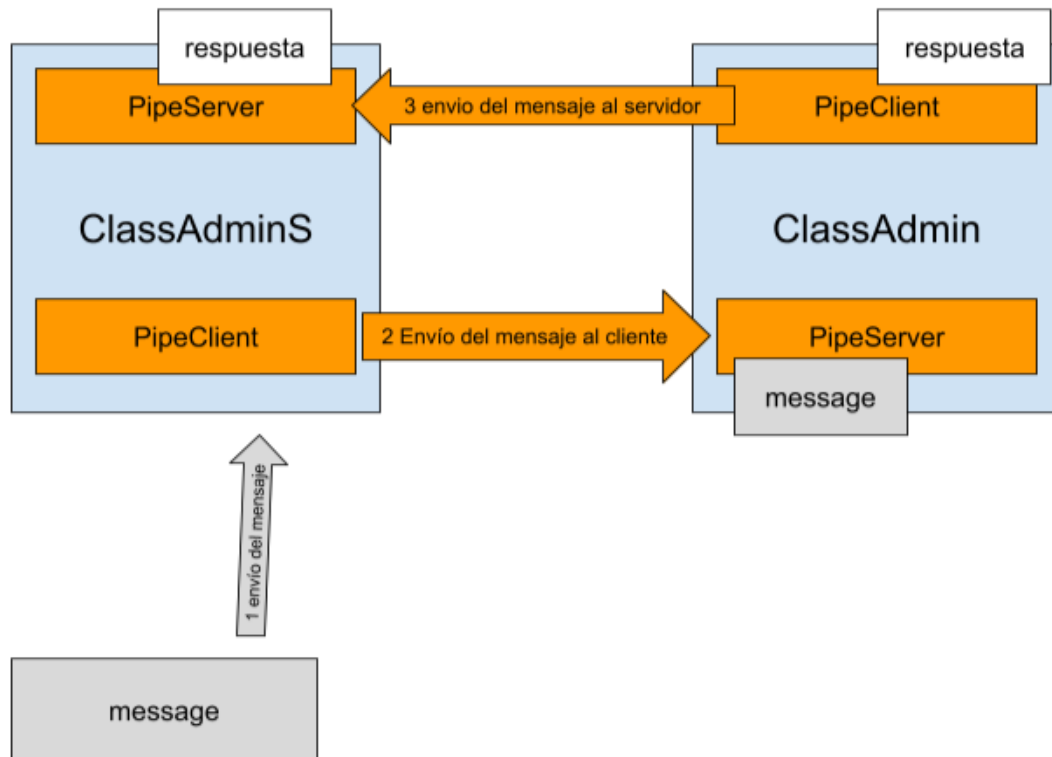
Existen dos servicios en este sistema. ClassAdminS (la S es de servidor), que estará en escucha de conexiones. Y ClassAdmin, que es el servicio cliente (agente) que al iniciar el ordenador, se iniciará el servicio, y establecerá una conexión con el ClassAdminS.

Una vez establecida la conexión, se mostrará en la aplicación web, la conexión y el usuario administrador, podrá interactuar con el cliente establecido, haciendo las funciones permitidas en el dashboard.

El administrador puede interactuar con los clientes conectados, gracias al pipeServer que tiene el cliente que permite al servidor ClassAdminS conectarse a este mini-servidor y enviar las instrucciones para que ejecute las funcionalidades indicadas.

Lo explico más claramente:

Proyecto ClassAdmin



El servidor ClassAdminS aparte de ser un servidor también hace de cliente para enviar mensajes (pipeClient). Y el servicio cliente ClassAdmin aparte de ser el cliente de ClassAdminS también es servidor para recibir los mensajes que el mini cliente le envía. (pipeServer).

En resumen, las conexiones sockets entre ClassAdminS y ClassAdmin se compone de dos partes:

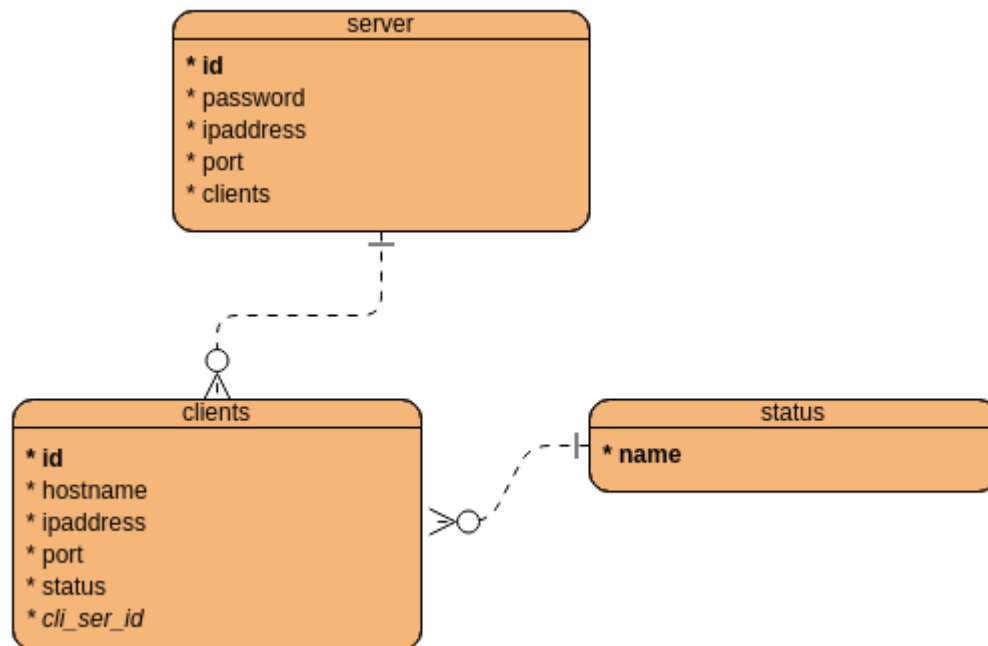
1. servicio ClassAdminS y los clientes ClassAdmin, la arquitectura es servidor-cliente
2. PipeServer y pipeClient que permite el envío de mensajes entre nodos.
 - a. PipeServer: Es el servidor que establece la conexión y recibe el mensaje para interactuar.
 - b. pipeClient: Es el cliente y es el que se conecta al pipeServer y envía un mensaje para finalmente "mandar ordenes" al cliente de lo que tiene que hacer. La conexión finaliza al enviar el mensaje.

Es gracias a pipeServer y pipeClient que el servidor puede ejecutar operaciones en los clientes y estos pueden enviar mensajes a otros

Proyecto ClassAdmin

clientes.

El sistema tiene una pequeña base de datos donde almacena los registros de los clientes, así como los datos del servidor, aquí muestro el diseño de entidad-relación.



La tabla "server" almacena los datos del servidor: dirección IP, puerto, número de clientes que se pueden conectar simultáneamente y la contraseña para acceder al dashboard.

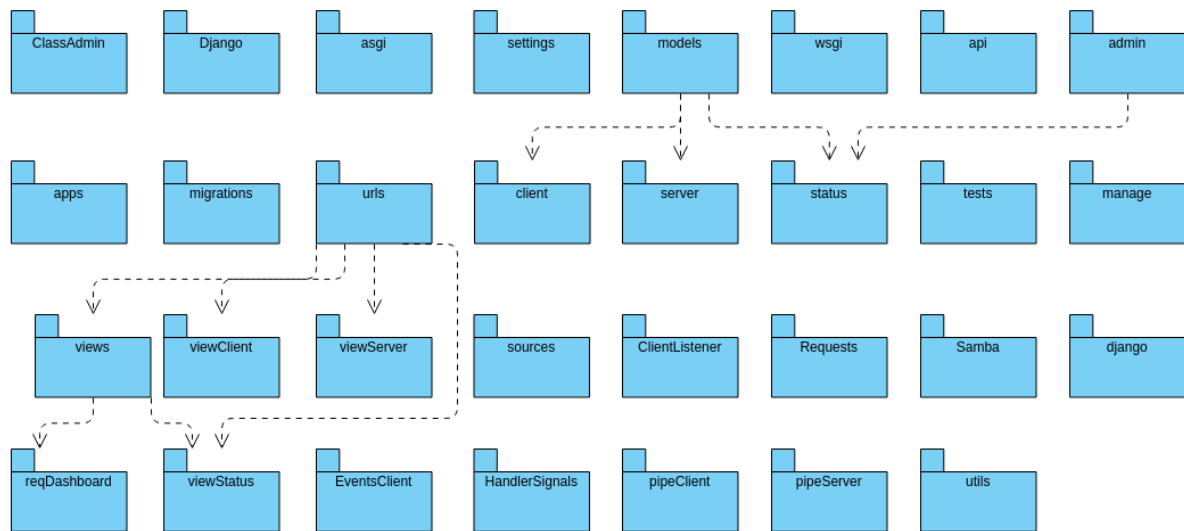
En la tabla "clients" se almacenará todos los clientes registrados, estén conectados o no. Para cada registro será un cliente registrado. Este contendrá el nombre del ordenador, la dirección ip, el puerto por el cual está conectado, si está conectado o no el cliente y en qué servidor está conectado.

Después, está la tabla "status" que contiene únicamente los estados de las conexiones de los clientes. Gracias al estado se mostrará en la web de un modo u otro.

La programación que se ha hecho orientada a objetos (POO). Así que mostraré el diseño UML de las clases, y los paquetes utilizados.

Proyecto ClassAdmin

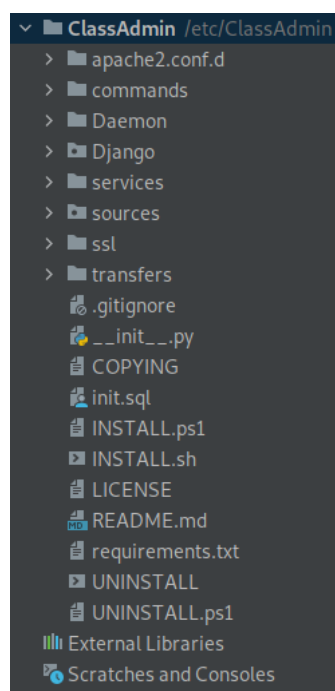
Los paquetes que se utilizan son los siguientes:



A parte de los paquetes que contiene el proyecto, es necesario instalar las siguientes librerías, (también se puede ver en el archivo requeriments.txt de la carpeta raíz):

django==3.2.7, pyotp, json, hashlib, base64, qrcode, requests, io, random, binascii, math, os, sys, platform, psutil, mysql.connector, pymysql, pyscreenshot, pysmb, socket, multiprocessing, threading, ssl, time, urllib3, re, datetime, signal, logging, certifi

Cómo está orientada a objetos (POO) vamos a mostrar un esquema UML



de las clases. Pero primero de todo vamos a mostrar la estructura que sigue el proyecto y después mostraremos las clases.

apache2.conf.d: En esta carpeta contiene los archivos de configuración de apache2/Xampp para su instalación.

commands: Aquí es donde se almacenarán los comandos necesarios para el proyecto.

toast32.exe/toast64.exe: Es el ejecutable que permite mostrar una notificación en Microsoft Windows.

Proyecto ClassAdmin

Daemon: Se almacenan los archivos .service de Linux.

Django: La carpeta donde está la aplicación web Django.

services: Es donde están los scripts Python ClassAdmin.socket y ClassAdminS.socket que son los archivos principales del proyecto. Sin estos no iniciarán los servicios.

sources: Es la carpeta principal donde se almacena la mayoría de dependencias de los archivos ClassAdmin.socket y ClassAdminS.socket (la extensión .socket es inventada para identificar que son los scripts Python que despliega el servidor o cliente.)

ssl: Se almacenan los certificados SSL, este contiene ClassAdmin.crt, ClassAdmin.csr y ClassAdmin.key.

transfers: Esta carpeta contiene dos carpetas más:

.screenshots: Esta carpeta se utiliza para almacenar todas las capturas hechas remotamente a los clientes. Para ello esta carpeta es compartida como ClassAdminS_Screenshots. Desde la aplicación se puede hacer una captura al cliente. Pues desde el cliente se ejecuta la función para capturar la pantalla y la imagen generada se almacena en la carpeta compartida ClassAdminS_Screenshots, teniendo permisos de escritura/lectura. Las credenciales son: ClassAdmin/12345678.

Server: Esta carpeta se utiliza para almacenar todos los archivos que se quieran transferir a un cliente. El cliente debería de ir a la carpeta ClassAdmin/transfers/server y vería el archivo que desde la aplicación el administrador le ha enviado.

INSTALL: Los archivos INSTALL.ps1 y INSTALL.sh permiten la instalación de los servicios ClassAdmin o ClassAdminS más cómodamente. Aunque hay un documento donde muestra los pasos para instalarlo.

Nota: La instalación del servicio ClassAdminS (servidor) en INSTALL.ps1 (Windows) no está programado. Por falta de tiempo.

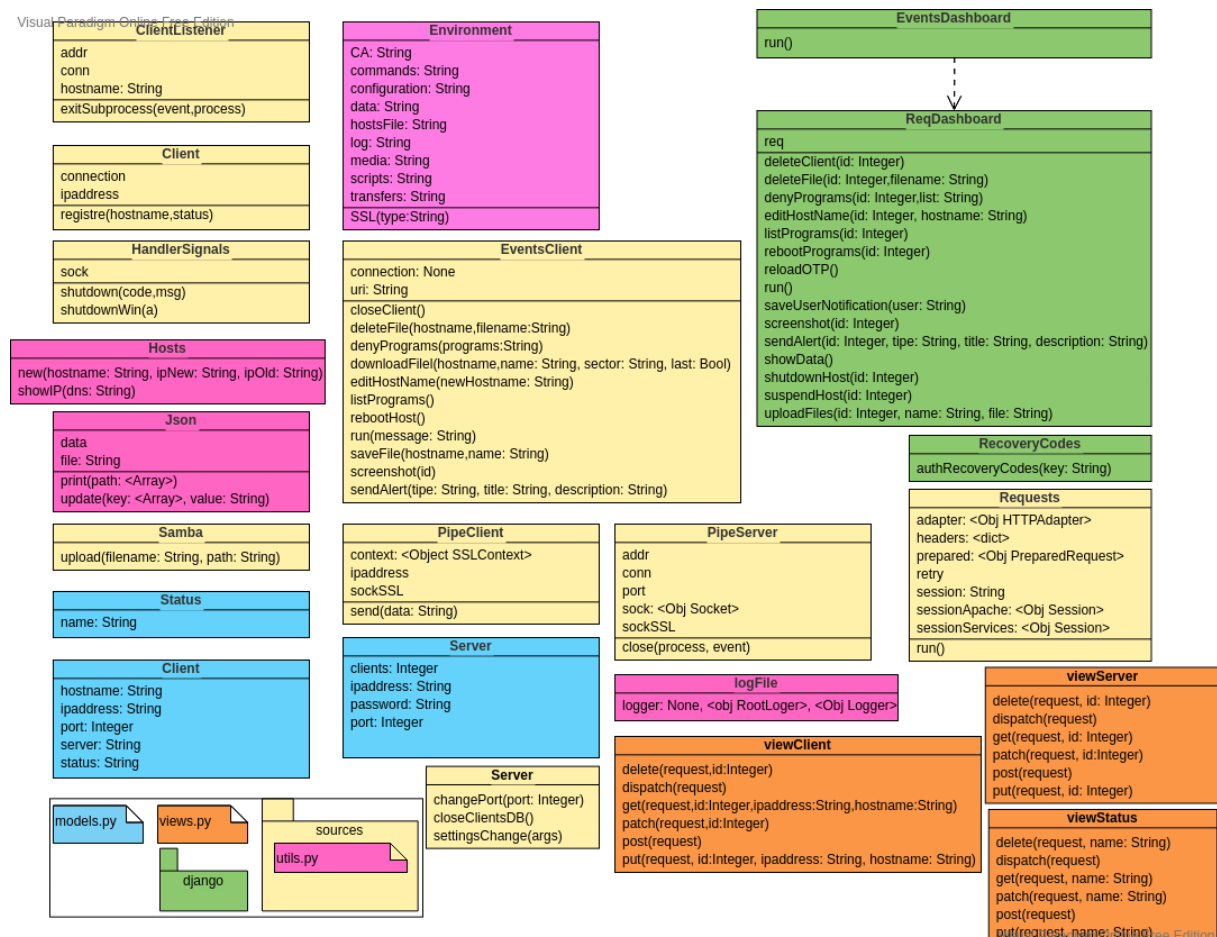
Proyecto ClassAdmin

UNINSTALL: Los archivos UNINSTALL.ps1 y UNINSTALL.sh permiten la desinstalación de los servicios ClassAdmin y ClassAdminS. Se desinstalarán todos los archivos fuentes, así como configuraciones. **NO se desintalan los programas.**

requirements.txt: En este archivo TXT, se especifica todos los paquetes necesarios para el buen funcionamiento del software.

IMPORTANTE: Si da errores durante la ejecución del programa. Mira los archivos log /var/log/ClassAdmin.log o sino en /var/log/syslog.
En caso de estar en Windows mira en la carpeta raíz del proyecto, que habrá un archivo ClassAdmin.log o en el registro de eventos > Aplicacion

El esquema UML de las clases es la siguiente:



models.py: Diseños de las tablas de DB.

Status: Diseño de la tabla "status" en la base de datos ClassAdmin

Client: Diseño de la tabla "client" en la base de datos ClassAdmin

Proyecto ClassAdmin

Server: Diseño de la tabla "server" en la base de datos ClassAdmin

views.py: Todos los métodos de la API de ClassAdmin.

viewClient: Esta clase contiene los métodos POST,PUT,DELETE,PATCH asociados a la tabla "client".

ViewStatus: Esta clase contiene los métodos POST,PUT,DELETE,PATCH asociados a la tabla "status".

ViewServer: Esta clase contiene los métodos POST,PUT,DELETE,PATCH asociados a la tabla "server".

django: Trabaja con las funciones para control remoto en los clientes.

ReqDashboard: Esta clase es padre de la clase EventsDashboard y se utiliza para llamar a las funciones de los clientes (shutdown,edit hostname>alert,etc.). Desde JavaScript se hace un fetch() a la ruta (/dashboard), cuya ruta lo redirige a la clase EventsDashboard. La clase reqDashboard únicamente contiene los métodos principales para las funcionalidades.

EventsDashboard: Esta clase es hija de la clase reqDashboard. Es la que se encarga de ver, si el POST es una acción o una notificación y dependiendo de ello, ejecutará las funciones correspondientes. Cuyas funciones están en reqDashboard.

RecoveryCodes: Esta clase se encarga de generar los códigos de recuperación del OTP, así como eliminandolos mientras se vayan usando.

utils.py: Este archivo contiene todas las clases y funciones que se vayan a usar en todo el proyecto.

Environment: Esta clase contiene todas las "variables de entorno" que indican las rutas de directorios de recursos que se quieran usar.

Hosts: Esta clase trabaja con el archivo hosts del equipo. Añadiendo el classadmin.server, o modificando su dirección IP si este se ha cambiado.

Proyecto ClassAdmin

Json: Esta clase permite trabajar más cómodamente con archivos JSON, mostrando y eliminando la llave indicada.

LogFile: Esta clase se encarga de registrar en el archivo log de classadmin (classadmin.log) un mensaje.

sources: Esta carpeta contiene la mayoría de funciones y clases que se usan en el proyecto.

ClientListener: Esta clase es usada al crear el subproceso del cliente.

Esta clase es responsable de interactuar (intermediario) entre el cliente y el servidor, permitiendo el envío de mensajes al cliente y recibirlos.

Client: Esta clase se encarga de registrar los clientes conectados, añadiéndolos a la base de datos de ClassAdmin.

HandlerSignals: Es usada para cerrar los servicios (ClassAdmin.socket) o (ClassAdminS.socket)

EventsClient: Esta clase es usada por el servicio ClassAdmin, aquí están las funciones llamadas desde la aplicación web.

Samba: Esta clase se encarga de guardar un archivo en la carpeta compartida ClassAdminS_Screenshots.

PipeClient: Esta clase es usada para enviar datos a una dirección ip específico (al cliente normalmente)

PipeServer: Esta clase es usada para recibir los mensajes enviados desde pipeClient.

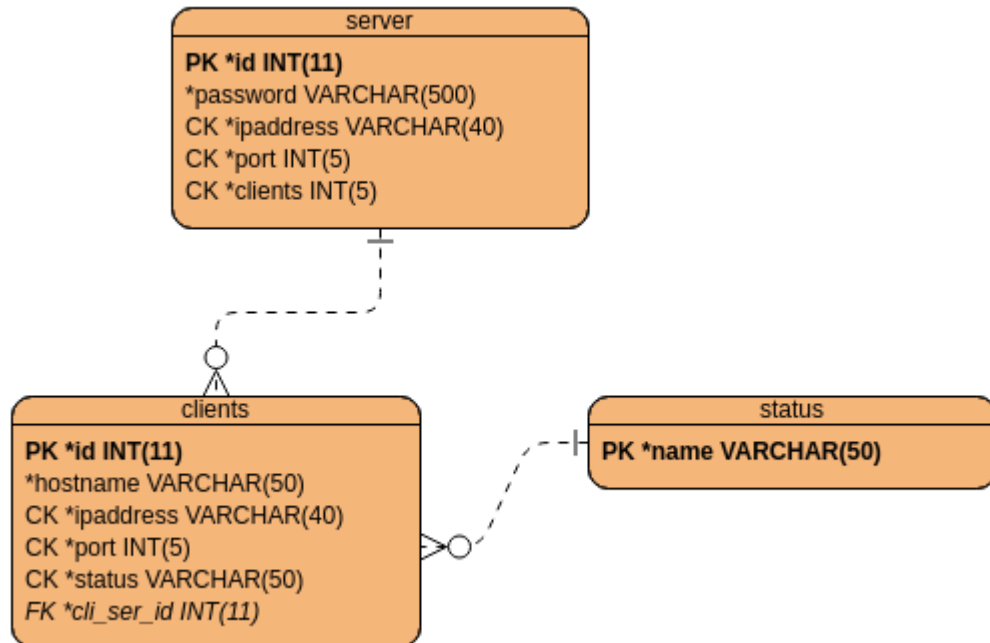
Requests: Esta clase se encarga de facilitar las llamadas a la API de ClassAdmin.

Server: Esta clase es usada para cambiar la dirección IP del servidor (ClassAdminS), así como su puerto y cerrar todos los clientes.

2.6. Implementación

Proyecto ClassAdmin

En el proyecto se ha desplegado una pequeña base de datos para almacenar los datos de conexión de cada cliente, así como su nombre. También los datos del servidor.



server:

password: Se almacena la contraseña en SHA512

ipaddress: Se almacena la dirección IP del servidor

port: Se almacena el puerto en el que está el servidor

clients: Se indica la cantidad de clientes que se pueden conectar a la vez.

clients:

hostname: Se almacena el nombre del PC del cliente conectado.

ipaddress: Se almacena la dirección IP del servidor

port: Se almacena el puerto en el que está el servidor

status: Se indica si el cliente está conectado o no (CONNECTED/DISCONNECTED).

Proyecto ClassAdmin

cli_ser_id: Se especifica en que servidor está conectado.

status:

name: Se indica el tipo de estado que tiene el cliente. Si está conectado o no.

La base de datos tiene el usuario ClassAdmin con contraseña 12345678, este usuario solo tiene acceso a la base de datos ClassAdmin. Ninguno más.

En el servicio Apache2, se utiliza los módulos:

- mod_wsgi: Este módulo de Apache2 es utilizado para programar Python en apache2
- rewrite: Este módulo de Apache2 se utiliza para redirigir las urls.

En el servidor mariaDB de Linux se ha configurado en el archivo /etc/mysql/mariadb.conf.d/50-server.cnf se a configurado las siguientes directivas:

- bind-address = 0.0.0.0: Le permitimos que cualquier IP puede hacer peticiones y conectarse al servidor de base de datos.
- max_allowed_packet = 1G: Le indicamos la longitud de los paquetes a enviar y recibir del servidor
- max_connections = 1000: Le indicamos el número de clientes conectados simultáneamente.

Proyecto ClassAdmin

```
88 text files.
86 unique files.
24 files ignored.

github.com/AlDanial/cloc v 1.86  T=0.04 s (1587.2 files/s, 124960.9 lines/s)
```

Language	files	blank	comment	code
Python	34	205	520	1618
JavaScript	10	74	134	991
CSS	4	5	51	784
PowerShell	6	32	65	358
Bourne Shell	4	14	61	221
HTML	3	0	30	162
XML	6	0	0	91
Bourne Again Shell	1	2	8	53
SQL	1	5	10	41
JSON	1	0	0	38
Markdown	1	2	0	15
SUM:	71	339	879	4372

El proyecto en resumen tiene unos 4372 líneas de código y 71 archivos de código, teniendo en cuenta 88 archivos de texto, 24 archivos ignorados durante el escaneo y 86 archivos únicos. Calcularía que el proyecto tiene unos 269 archivos en total.

Procedemos a explicar la instalación de los servicios. Esta parte la tenéis en otro documento aparte.

Recomiendo ejecutar el instalador de Powershell o Bash para más facilidad. Si se quiere instalar el ClassAdminS en Windows, se deberá de instalar a mano. Perdona por las molestias.

Proyecto ClassAdmin

2.7. Conclusiones

Con este proyecto, he aprendido bastante más y mejor sobre el funcionamiento de los sockets en Python 3.

También he aprendido a como instalar y crear servicios en Windows, aunque no de forma nativa, sino ayudándome de un programa tercero.

Además he profundizado más sobre el framework Django así como la creación de una API con Django Rest Framework.

El proyecto me ha sido muy difícil llevarlo a cabo pero al final, lo tengo hecho. Solo espero que os funcione a la hora de probarlo. En fin de facilitaros la instalación he creado un instalador (INSTALL.sh e INSTALL.ps1).

Incluso he hecho que los servicios ClassAdmin corran en sistemas Windows.

Este software lo he licencia bajo GPLv3, he estudiado un poco los pasos para añadir la licencia. Todo el procedimiento que se debe de hacer.

También ha sido una gran mejora del proyecto anterior personal. Se puede ver en github.com/cleanet/sockets. Pues se ha mejorado el rendimiento de los servicios con el fin de no consumir tanta CPU. Evitando procesos zombies y otros errores.

2.8. Líneas de investigación futuras

Ni en broma este software lo pondría en producción en un entorno real. Falta mucho camino para ello. Solo por empezar, tendría que compilar todo el código con la librería (Pyinstaller). Sí lo compilase sería más fácil para el usuario, ya que no debería de preocuparse de las dependencias ni que python3 esté instalado, pues estaría en el mismo ejecutable.

Proyecto ClassAdmin

Serían varios ejecutables. Además, si estuviera compilado, compartiría el código fuente, con el fin de cumplir con la licencia GPLv3.

También se podría asociar un cliente con un número móvil y este poder recibir notificaciones de eventos hechos en el cliente.

También ha sido una gran mejora del proyecto anterior personal. Se puede ver en github.com/cleanet/sockets. Pues se ha mejorado el rendimiento de los servicios con el fin de no consumir tanta CPU. Evitando procesos zombies y otros errores.

2.9. Bibliografía

Sinceramente, no podría especificar todos los sitios webs de los cuales se ha investigado y aprendido. Solo puedo indicar las grandes plataformas, desde donde he podido obtener la información requerida.

Documentación

<https://docs.djangoproject.com/en/4.0/>

<https://www.django-rest-framework.org/>

<https://docs.python.org/3/>

<https://docs.python.org/3/library/socket.html?highlight=socket#module-socket>

<https://requests.readthedocs.io/en/master/>

<https://realpython.com/python-requests/>

<http://nssm.cc/commands>

<http://nssm.cc/usage>

<https://docs.microsoft.com/en-us/powershell/module/>

Proyecto ClassAdmin

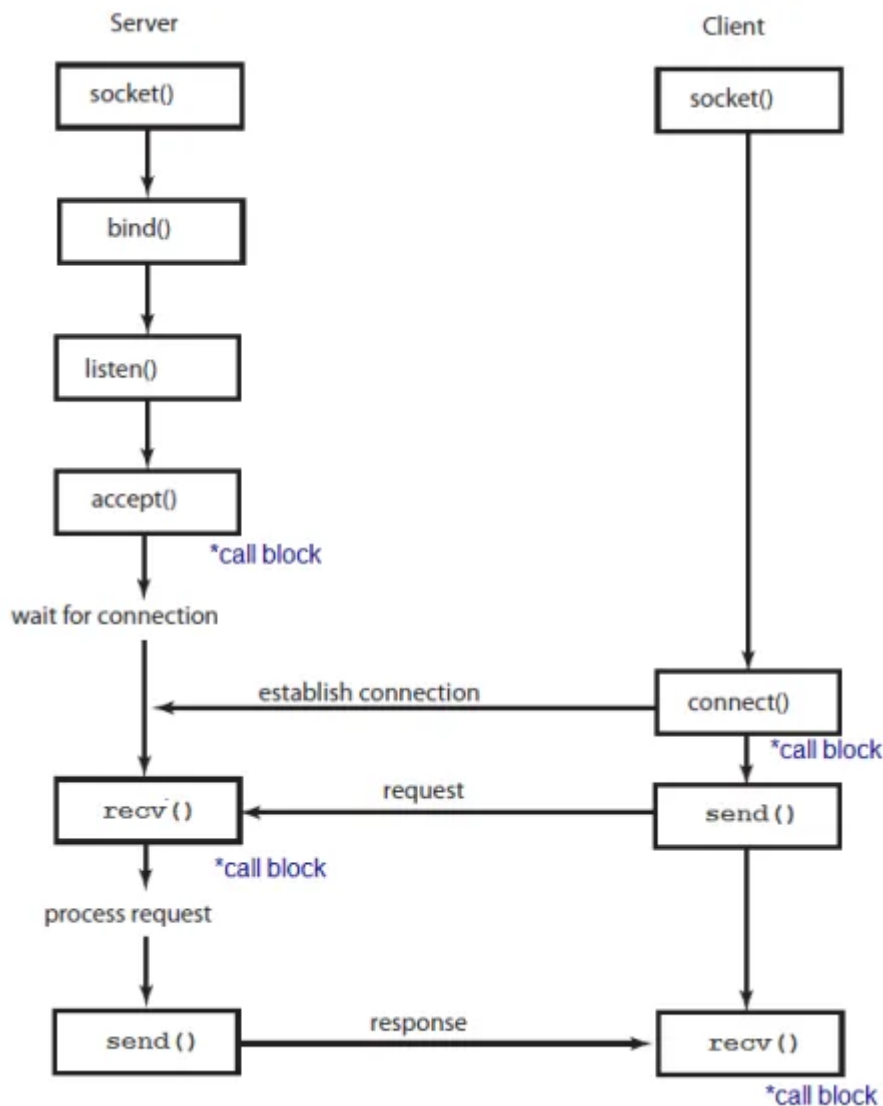
<https://www.gnu.org/licenses/gpl-howto.es.html>

<https://www.gnu.org/licenses/fdl-howto.es.html>

Tutoriales

https://www.youtube.com/watch?v=hL52_nB5QSw

Imágenes de guía



preguntas en foros

<https://foro.elhacker.net/profiles/drakaris-u562026.html>

<https://stackoverflow.com/users/9951542/cleanet>

Proyecto ClassAdmin

<https://stackoverflow.com/questions/1364173/stopping-python-using-ctrlc/21460045>

<https://code-maven.com/catch-control-c-in-python>

sitios oficiales de los programas externos usados

___<https://nssm.cc/>

___<https://www.apachefriends.org/index.html>

___<https://visualstudio.microsoft.com/visual-cpp-build-tools/>

2.10. Anexos

En el servidor, el código secreto del OTP (cambiar el QR code) está en el archivo de configuración sources/data.json

```
{,
  "OTP": {
    "key": "YLBDRH5VBTEWK3QRYBHLIZQL",
    "recoveryCodes": [
      "546-879",
      "057-352",
      "034-439",
      "529-604",
      "293-183",
      "205-751",
      "120-745",
      "609-578",
      "650-047"
    ]
  },
}
```

En el archivo data.json no solo se configura el OTP, sino también puedes configurar los datos de conexión para la base de datos.

También podemos configurar los datos de la carpeta compartida. Es decir, que servidor va a

```
{,
  "DB": {
    "host": "127.0.0.1",
    "user": "ClassAdmin",
    "password": "12345678",
    "database": "ClassAdmin"
  },
}
```

conectarse, con que usuario y contraseña, así como el nombre de la carpeta compartida.

Se puede dar el caso que en GNU/Linux no se muestra las notificaciones. Comprueba que en services/ClassAdmin.conf la

Proyecto ClassAdmin

directiva "notifications" este en "true". Sí esto es así y sigue sin funcionar. Esto significa que en tu distribución GNU/Linux la variable de entorno DISPLAY es diferente a :0, por la cual haremos lo siguiente:

1. Ejecuta en tu terminal

```
echo $DISPLAY
```

2. Ahora con la salida del comando anterior, ves a sources/utils.py y en la línea 177 de la función Notify(), y reemplaza el :0 por la salida del comando anterior (siendo diferente a :0)

```
def Notify(title:str,message:str,output=True):
    status = Json(Environment.configuration).print(["notifications"])
    if(json.loads(status)):
        user = Json(Environment.configuration).print(["user"])
        architecture = platform.architecture()[0]
        zenity = f"zenity --notification --title \"{title}\" --text \"{message}\" --window-icon=\"{Environment.media}/images/ClassAdminLogo.png\" --display :0"
        toast = f"toast{architecture.replace('bit', '')} --title \"{title}\" --message \"{message}\" --icon \"{Environment.media}/images/ClassAdminLogo.png\" "
        command = f"su {user} -c '{zenity}'" if platform.system().upper() == 'LINUX' else toast
        os.system(command)
    if output:
        print(message)
    return message
```

3. También debes de editar el archivo sources/eventsClient.py en la línea 90 de la función showAlert(), y reemplaza el :0 por la salida del comando anterior (siendo diferente a :0)

```
@staticmethod
def showAlert(type:str,title:str,description:str) -> bool:
    if type=="notification":
        Notify(title,description,False)
    else:
        zenity = f"zenity --{type} --title '{title}' --text '{description}' --window-icon='{Environment.media}/images/ClassAdminLogo.png' --display :0"
```

3. GNU Free Documentation License

Se otorga permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre GNU, Versión 1.3 o cualquier versión posterior publicada por la Free Software Foundation; sin secciones invariantes, sin textos de portada y sin textos de contraportada. Se incluye una copia de la licencia en el archivo GFDL.txt. Se debe de mantener la licencia original, así como nombrar al autor original quien creó el software.