



TRABAJO FINAL DE GRADO

# JOB SCHEDULER

DESARROLLO DE UN SISTEMA DE  
PROGRAMACIÓN DE TAREAS Y MONITOREO DE  
RECURSOS EN LINUX

Autor: Marcos Zaragoza Ríos

ASIR

Curso 2024/25

IES LA VEREDA

Tutor: Youssef Majdoub Amajoud



## **Introducción**

*En la administración de sistemas operativos, la automatización de tareas y la monitorización de recursos son esenciales para garantizar la eficiencia y la continuidad del servicio. Este proyecto presenta el desarrollo de un Sistema de Programación y Monitoreo de Tareas para sistemas Linux, diseñado para simplificar y optimizar la gestión de tareas administrativas y de mantenimiento.*

*Inicialmente, la idea principal del proyecto era utilizar Apache Airflow para la gestión de tareas. Sin embargo, debido a su complejidad y a las limitaciones encontradas en nuestro entorno específico, se decidió desarrollar desde cero una solución en Python.*

*El sistema desarrollado facilita la definición y ejecución de tareas automatizadas, así como la supervisión continua de los recursos del sistema. Mediante una interfaz intuitiva, los administradores pueden programar actividades rutinarias, recibir alertas ante situaciones críticas y generar informes que contribuyen a una gestión más efectiva del entorno operativo.*

*Las funcionalidades del sistema incluyen la creación de copias de seguridad, la limpieza periódica de la papelera de reciclaje, el envío de alertas por correo electrónico sobre el uso crítico de recursos y la generación de informes de usuarios. Los resultados obtenidos demuestran la efectividad del sistema en la automatización de tareas rutinarias y la supervisión continua de los recursos, contribuyendo así a la optimización de la administración del sistema.*

### **Abstract**

*In system administration, task automation and resource monitoring are essential to ensure efficiency and service continuity. This project presents the development of a Task Scheduling and Monitoring System for Linux systems, designed to simplify and optimize the management of administrative and maintenance tasks.*

*Initially, the main idea of the project was to use Apache Airflow for task management. However, due to its complexity and the limitations encountered in our specific environment, it was decided to develop a solution from scratch in Python.*

*The developed system facilitates the definition and execution of automated tasks, as well as the continuous monitoring of system resources. Through an intuitive interface, administrators can schedule routine activities, receive alerts for critical situations, and generate reports that contribute to more effective management of the operational environment.*

*The system's functionalities include creating backups, periodically cleaning the recycle bin, sending email alerts about critical resource usage, and generating user reports. The obtained results demonstrate the system's effectiveness in automating routine tasks and continuously monitoring resources, thus contributing to system administration optimization.*

## **Resum**

*En l'administració de sistemes operatius, l'automatització de tasques i la monitorització de recursos són essencials per garantir l'eficiència i la continuïtat del servei. Aquest projecte presenta el desenvolupament d'un Sistema de Programació i Monitorització de Tasques per a sistemes Linux, dissenyat per simplificar i optimitzar la gestió de tasques administratives i de manteniment.*

*Inicialment, la idea principal del projecte era utilitzar Apache Airflow per a la gestió de tasques. No obstant això, a causa de la seva complexitat i de les limitacions trobades en el nostre entorn específic, es va decidir desenvolupar des de zero una solució en Python.*

*El sistema desenvolupat facilita la definició i execució de tasques automatitzades, així com la supervisió contínua dels recursos del sistema. Mitjançant una interfície intuïtiva, els administradors poden programar activitats rutinàries, rebre alertes davant situacions crítiques i generar informes que contribueixen a una gestió més efectiva de l'entorn operatiu.*

*Les funcionalitats del sistema inclouen la creació de còpies de seguretat, la neteja periòdica de la paperera de reciclatge, l'enviament d'alertes per correu electrònic sobre l'ús crític de recursos i la generació d'informes d'usuaris. Els resultats obtinguts demostren l'efectivitat del sistema en l'automatització de tasques rutinàries i la supervisió contínua dels recursos, contribuint així a l'optimització de l'administració del sistema.*

# Índice de contenidos

<b>1. INTRODUCCIÓN.....</b>	<b>5</b>
1.1 MÓDULOS A LOS QUE IMPLICA.....	5
1.2 BREVE DESCRIPCIÓN DEL PROYECTO.....	6
<b>2. ESTUDIO PREVIO.....</b>	<b>7</b>
Situación de la Empresa.....	7
Necesidades de la Elaboración del Proyecto.....	7
<b>3. DISEÑO.....</b>	<b>8</b>
<b>3.1 DISEÑO GENERAL.....</b>	<b>8</b>
<b>3.2 DISEÑO DETALLADO.....</b>	<b>9</b>
<b>4. IMPLANTACIÓN.....</b>	<b>11</b>
4.1 Entorno de Desarrollo utilizado.....	11
4.2 Procesos Desarrollados.....	11
<b>5. RECURSOS.....</b>	<b>13</b>
5.1 Herramientas hardware.....	13
5.2 Herramientas software.....	13
5.3 Sistemas operativos y paquetes necesarios.....	15
5.4 Personal.....	15
<b>6. RESULTADOS Y PRUEBAS.....</b>	<b>16</b>
6.1 Resultados Obtenidos.....	16
6.2 Análisis de Desempeño.....	16
6.3 Pruebas Realizadas.....	16
<b>7. MANUAL DE USUARIO Y GUÍA DE INSTALACIÓN.....</b>	<b>18</b>
7.1 Manual de Usuario.....	18
7.1.1 Estructura de Ficheros.....	18
7.1.5 Visualizar y Gestionar Tareas.....	24
7.2 Guía de instalación.....	33
4.2 Desarrollo de la Aplicación.....	37
<b>8. CONCLUSIONES.....</b>	<b>38</b>
8.1 Alcance y Logros.....	38
8.2 Desafíos, Problemas y Lecciones Aprendidas.....	38
8.3 Futuras Mejoras.....	39
8.4 Conclusión Final.....	39
<b>9. REFERENCIAS.....</b>	<b>41</b>
<b>10. ANEXOS.....</b>	<b>42</b>

# 1. INTRODUCCIÓN

## 1.1 MÓDULOS A LOS QUE IMPLICA

Este proyecto se basa en las habilidades y conocimientos adquiridos a lo largo de los cursos de primero y segundo de ASIR (Administración de Sistemas Informáticos en Red). A continuación, se detallan los módulos más relevantes y cómo sus contenidos han sido aplicados en el desarrollo de este trabajo:

- **Administración de Sistemas Operativos (ASO):** Este módulo ha sido fundamental para entender el entorno en el cual se ejecutan los trabajos programados, así como para manejar los comandos necesarios para la limpieza de archivos y monitoreo del sistema.
- **Servicios en Red (SER):** La configuración de servicios de red y la comprensión de protocolos de comunicación han sido fundamentales para implementar funcionalidades como el servidor SFTP y SMTP. Esto ha facilitado la transferencia segura de archivos de respaldo y la integración de servicios de notificaciones por correo electrónico.
- **Implantación de Aplicaciones Web (IAW):** Este módulo ha sido fundamental para el desarrollo e implementación de la interfaz web. Ha proporcionado los conocimientos necesarios para desplegar aplicaciones web seguras y eficientes, facilitando así la interacción de los administradores con el sistema a través de una plataforma accesible y fácil de usar.
- **Administración de Sistemas Gestores de Bases de Datos (SGBD):** Este conocimiento se utilizó para almacenar configuraciones y logs de las tareas programadas, asegurando que toda la información relevante esté disponible.
- **Seguridad y Alta Disponibilidad (SAD):** La comprensión de las buenas prácticas de seguridad ha sido crucial para asegurar el entorno en el que se desarrolla esta aplicación. Esto incluye la configuración segura del servidor SFTP y la restricción de permisos para usuarios específicos, garantizando así la protección de los datos y la integridad del sistema.
- **Lenguaje de Marcas (LM):** Los conocimientos adquiridos en este módulo han sido aplicados en el diseño y estructuración de las páginas web que componen la interfaz del sistema. Utilizando lenguajes de marcas como HTML y CSS, se ha logrado crear una presentación clara y coherente de las funcionalidades disponibles, mejorando la experiencia del usuario y la usabilidad de la aplicación.

## 1.2 BREVE DESCRIPCIÓN DEL PROYECTO

Este proyecto es un Job Scheduler o más conocido como Programador de tareas, una herramienta diseñada para automatizar diversas tareas administrativas y de mantenimiento en un sistema. El objetivo principal es facilitar la ejecución de tareas repetitivas como la limpieza de la papelera, el envío de correos electrónicos con reportes del sistema, y la copia de seguridad de datos. Además, se busca proporcionar herramientas para monitorear el uso de memoria y disco, y alertar sobre posibles problemas antes de que afecten el rendimiento del sistema. La configuración flexible permite a los usuarios definir y configurar sus propias tareas programadas mediante un archivo de configuración sencillo.

El uso del proyecto es intuitivo. Los administradores pueden definir qué tareas desean automatizar, especificando la frecuencia y los detalles de cada una. El sistema se encarga de ejecutar las tareas programadas en los intervalos establecidos, asegurando que las actividades críticas se realicen sin intervención manual. Asimismo, ofrece funcionalidades para supervisar los recursos del sistema y notificar a los administradores sobre situaciones que requieran atención, promoviendo una gestión proactiva del entorno informático.

El proyecto está desarrollado en Python, aprovechando su versatilidad y la disponibilidad de bibliotecas que facilitan la programación de tareas y la monitorización del sistema. Se utilizaron librerías como `schedule` para la programación de tareas y `psutil` para la supervisión de recursos. Además, se implementó una interfaz web utilizando Flask, que permite a los administradores interactuar con el sistema de manera sencilla y accesible, programar nuevas tareas, monitorear las existentes y gestionar las alertas y reportes a través de una plataforma intuitiva. Asimismo, se configuró un servidor SFTP seguro para la transferencia de archivos de respaldo, garantizando la protección de los datos y facilitando una gestión eficiente de los mismos.

## 2. ESTUDIO PREVIO

### Situación de la Empresa

La empresa en la que se desarrolla este proyecto es de tamaño mediano y cuenta con una infraestructura de IT bastante compleja. Esta infraestructura incluye varios servidores, estaciones de trabajo y dispositivos de red que son esenciales para las operaciones diarias de la empresa. Con el crecimiento de la empresa, las demandas sobre su infraestructura de IT también han aumentado, creando una serie de necesidades específicas que este proyecto busca resolver.

### Necesidades de la Elaboración del Proyecto

La empresa realiza diversas tareas de mantenimiento y administración que son repetitivas y consumen mucho tiempo. Estas tareas, como la limpieza de la papelera de reciclaje de los servidores, la realización de copias de seguridad y la generación de informes sobre el uso de recursos del sistema, actualmente se realizan de manera manual. Esto no solo consume tiempo, sino que también aumenta el riesgo de errores humanos. La automatización de estas tareas mediante un Job Scheduler reducirá significativamente la carga de trabajo del equipo de IT y mejorará la eficiencia de estas operaciones.

Además, la empresa necesita monitorizar el uso de recursos del sistema, como la memoria y el espacio en disco, y recibir alertas proactivas cuando estos recursos se están agotando. Esto es crucial para prevenir fallos del sistema que pueden afectar la productividad y la disponibilidad de servicios críticos.

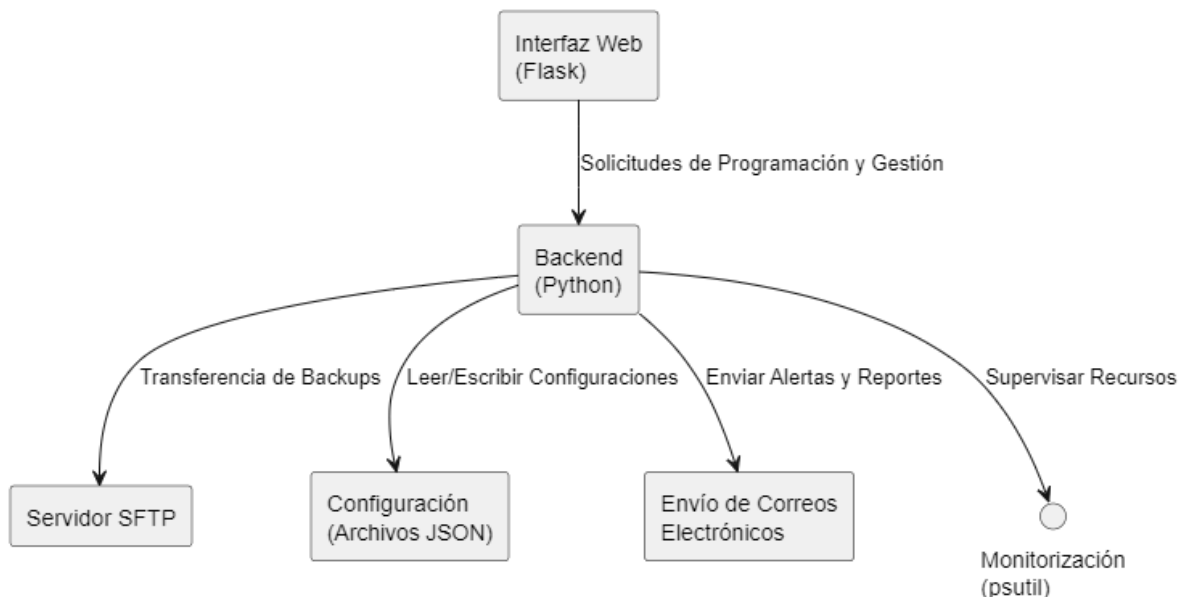
Después de mi periodo de prácticas en el departamento de IT en “*Celestica*” me he dado cuenta que un sistema de alertas bien configurado permitirá al equipo reaccionar rápidamente ante posibles problemas, antes de que se conviertan en incidentes mayores.



## 3. DISEÑO

### 3.1 DISEÑO GENERAL (Parte de Análisis)

El diseño general del proyecto se ha pensado para ofrecer una solución integral que aborde la automatización y monitorización de tareas administrativas en sistemas Linux. La estructura global del sistema está compuesta por varios módulos interconectados que trabajan en conjunto para cumplir con los objetivos establecidos. A continuación, se presenta una visión panorámica de los componentes principales y sus interacciones:



**Interfaz Web (Flask):** Proporciona una plataforma accesible para que los administradores puedan programar y gestionar tareas, visualizar el estado del sistema y acceder a informes.

**Backend (Python):** Gestiona la lógica del sistema, incluyendo la programación de tareas mediante APScheduler y la monitorización de recursos utilizando psutil.

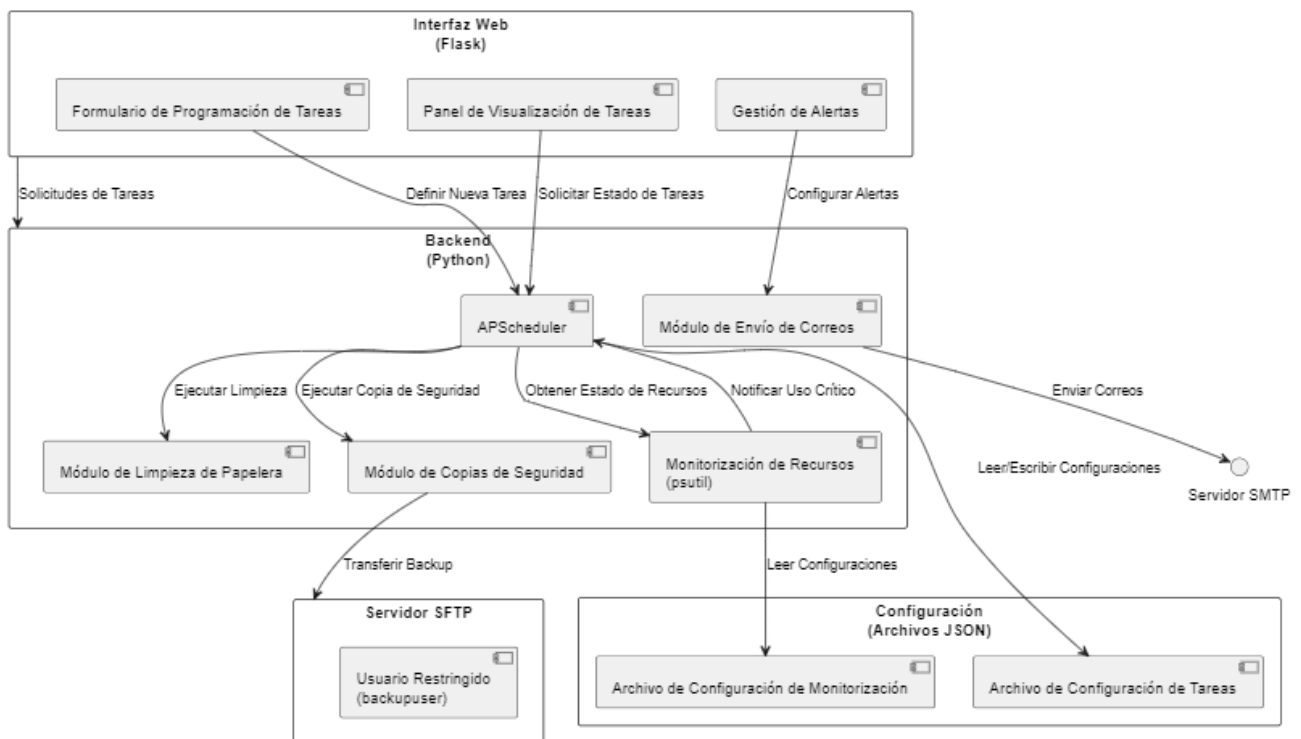
**Servidor SFTP:** Encargado de la transferencia segura de archivos de respaldo. Está configurado con un usuario específico, backupuser, con permisos restringidos.

**Configuración (Archivos JSON):** Almacena las configuraciones del sistema, como detalles de las tareas programadas y parámetros de monitorización.

**Módulo de Envío de Correos Electrónicos:** Gestiona el envío de alertas y reportes a los administradores en caso de eventos críticos o finalización de tareas.

### 3.2 DISEÑO DETALLADO

En esta sección se describen con mayor profundidad los componentes individuales del sistema, sus funcionalidades específicas y las tecnologías empleadas para su implementación.



#### Descripción del Diagrama:

##### Interfaz Web (Flask):

- **Formulario de Programación de Tareas:** Permite a los administradores definir nuevas tareas automatizadas.
- **Panel de Visualización de Tareas:** Muestra una lista de tareas programadas y su estado.

- **Gestión de Alertas:** Configuración y visualización de alertas sobre el uso de recursos críticos.

### Backend (Python):

- **APScheduler:** Responsable de la planificación y ejecución de las tareas programadas.
- **Módulo de Limpieza de Papelera:** Automatiza la limpieza periódica de la papelera de reciclaje.
- **Módulo de Copias de Seguridad:** Gestiona la creación y transferencia de copias de seguridad.
- **Monitorización de Recursos (psutil):** Supervisa continuamente el uso de recursos del sistema.
- 

### Servidor SFTP:

- **Usuario Restringido (backupuser):** Configurado para tener acceso limitado a un directorio específico para la transferencia de backups.

### Configuración (Archivos JSON):

- **Archivo de Configuración de Tareas:** Almacena detalles como nombre de la tarea, frecuencia y parámetros.
- **Archivo de Configuración de Monitorización:** Define umbrales y parámetros para la supervisión de recursos.

### Envío de Correos Electrónicos:

- **Servidor SMTP:** Configurado para el envío automatizado de correos electrónicos de alertas y reportes.

## 4. IMPLANTACIÓN

### 4.1 Entorno de Desarrollo utilizado

- **Sistema Operativo:** Se utilizó Ubuntu Server 24.04 LTS por su estabilidad y amplio soporte para herramientas de automatización.
- **Lenguaje de Programación:** Python 3.12, elegido por su versatilidad y la disponibilidad de bibliotecas especializadas que facilitan la automatización y monitorización del sistema.
- **IDE:** Visual Studio Code fue seleccionado como entorno de desarrollo integrado debido a su flexibilidad, extensibilidad y soporte para múltiples lenguajes y herramientas.
- **Control de Versiones:** Git se empleó para gestionar el código fuente, permitiendo un seguimiento eficiente de los cambios y facilitando la colaboración en equipo.

#### 4.1.1 Configuración del Entorno de Desarrollo:

- **Creación de un Entorno Virtual:** Se estableció un entorno virtual de Python utilizando venv para aislar las dependencias del proyecto y evitar conflictos con otras aplicaciones.
- **Instalación de Dependencias:** Se instalaron las bibliotecas necesarias, incluyendo Flask para el desarrollo de la interfaz web, APScheduler para la programación de tareas, psutil para la monitorización de recursos, y paramiko para la gestión de transferencias SFTP seguras.

### 4.2 Procesos Desarrollados

Durante la implantación del sistema, se desarrollaron diversos procesos que permiten la automatización y monitorización eficiente de tareas administrativas. A continuación, se describen los principales procesos implementados:

#### 4.2.1 Configuración Inicial del Sistema:

**Carga de Configuraciones:** Se implementó la carga de configuraciones desde archivos JSON, permitiendo una fácil modificación y actualización de parámetros sin necesidad de alterar el código fuente.

**Definición de Funciones de Tarea:** Se crearon funciones específicas para cada tipo de tarea automatizada, como limpieza de archivos temporales, generación de copias de seguridad y gestión de recursos del sistema.

**Creación del Bucle Principal:** Se desarrolló un bucle principal que coordina la ejecución de tareas programadas y la monitorización continua de recursos, asegurando una operación fluida y sincronizada del sistema.

#### 4.2.2 Programación de Tareas Automatizadas:

- **Definición y Configuración de Jobs:** Los administradores pueden definir nuevos trabajos a través de la interfaz web, especificando el tipo de tarea, la frecuencia de ejecución y los parámetros adicionales necesarios.
- **Ejecución Programada:** APScheduler gestiona la programación y ejecución de estas tareas en los intervalos establecidos, garantizando que se realicen de manera oportuna sin intervención manual.

## 5. RECURSOS

### 5.1 Herramientas hardware

El proyecto Job Scheduler se implementó en un entorno de desarrollo basado en el siguiente hardware:

#### **Servidor Principal:**

Se utilizó un portátil Lenovo dedicado a alojar el Job Scheduler y el Servidor SFTP con las siguientes especificaciones:

- Procesador: Intel Core i5 de 8ª generación
- Memoria RAM: 8 GB
- Almacenamiento: SSD de 256 GB
- Sistema operativo: Ubuntu 24.04 LTS

#### **Máquinas Virtuales:**

Se usaron dos máquinas virtuales configuradas en modo puente dentro del servidor principal.

#### **Máquina Virtual 1 (Job Scheduler):**

- **SO:** Ubuntu 24.04 LTS
- **IP:** 192.168.1.10
- **Recursos Asignados:** 2 núcleos de CPU, 4 GB de RAM, 50 GB de almacenamiento
- **Función:** Ejecutar la aplicación

#### **Máquina Virtual 2 (Servidor SFTP):**

- **SO:** Ubuntu Server 24.04 LTS:
- **IP:** 192.168.1.66
- **Recursos Asignados:** 2 núcleos de CPU, 2 GB de RAM, 100 GB de almacenamiento (para almacenar los backup)
- **Función:** Gestionar las transferencias seguras de backups mediante SFTP.

### 5.2 Herramientas software

Las herramientas de software utilizadas para el desarrollo e implementación del Job Scheduler incluyeron:

**Lenguaje de Programación:**

- **Python 3.12:**

**Función:** Implementar funcionalidades de programación de tareas, monitorización de recursos, y gestión de transferencias SFTP.

**Framework Web:**

- **Flask:**

**Función:** Proporcionar una plataforma accesible y fácil de usar para que los administradores gestionen las tareas y monitoreen el estado del sistema.

**Bibliotecas de Python:**

- **APScheduler:**

**Función:** Gestionar la planificación y ejecución de tareas automatizadas en intervalos definidos.

- **psutil:**

**Función:** Supervisar el uso de memoria, CPU y espacio en disco, permitiendo la detección de condiciones críticas.

- **Paramiko:**

**Función:** Facilitar las transferencias seguras de archivos mediante SFTP.

- **Schedule:**

**Función:** Complementar a APScheduler en la definición y gestión de tareas automatizadas.

**Entorno de Desarrollo Integrado (IDE):**

- **Visual Studio Code:**

**Función:** Facilitar la escritura, depuración y gestión del código del proyecto mediante extensiones y herramientas integradas.

**Servicios de Correo Electrónico:**

- **SMTP Server:**

**Función:** Enviar alertas y notificaciones a los administradores cuando se detectan condiciones críticas en el sistema.

### **5.3 Sistemas operativos y paquetes necesarios**

El sistema operativo utilizado para el desarrollo y la implementación del Job Scheduler fue Ubuntu Desktop 24.04 LTS y Ubuntu Server 24.04 LTS. Se seleccionó Ubuntu debido a su estabilidad, amplia disponibilidad de paquetes de software y su compatibilidad con Python y otras tecnologías utilizadas en el proyecto.

Los paquetes necesarios para la implementación del Job Scheduler se gestionaron mediante el sistema de gestión de paquetes de Python, pip. Se utilizó un entorno virtual de Python para gestionar las dependencias del proyecto de manera aislada.

### **5.4 Personal**

El proyecto Job Scheduler fue desarrollado por un único desarrollador, Marcos Zaragoza, estudiante de segundo curso del ciclo formativo de grado superior en Administración de Sistemas Informáticos en Red (ASIR). Marcos Zaragoza también se encargó del mantenimiento y la gestión continua del proyecto durante su desarrollo.



## 6. RESULTADOS Y PRUEBAS

### 6.1 Resultados Obtenidos

El proyecto JobScheduler ha alcanzado los objetivos principales establecidos al inicio del desarrollo. El sistema es capaz de definir, programar y ejecutar diversas tareas de manera automática en un entorno Linux, lo cual incluye la capacidad de realizar copias de seguridad, enviar correos electrónicos con información relevante, monitorear el uso de recursos del sistema y limpiar la papelera del usuario.

### 6.2 Análisis de Desempeño

El desempeño del JobScheduler ha sido evaluado en términos de eficiencia y uso de recursos del sistema. Las tareas programadas se ejecutan de acuerdo con los intervalos especificados, y el impacto en el rendimiento del sistema es mínimo. La monitorización del uso de memoria y disco demuestra que el sistema puede gestionar múltiples tareas sin causar sobrecarga. Estos fueron algunos de los aspectos clave del análisis de desempeño:

- **Ejecución de Tareas:** Las tareas se ejecutan puntualmente sin retrasos significativos. La programación de tareas diarias, horarias y por minutos ha demostrado ser precisa y confiable.
- **Uso de CPU y Memoria:** Durante la ejecución de tareas intensivas como la creación de copias de seguridad, el uso de la CPU se incrementa temporalmente, pero vuelve a niveles normales una vez completada la tarea.
- **Impacto en el Sistema:** La carga adicional generada por el Job Scheduler es baja, asegurando que el sistema continúe operando sin interrupciones o ralentizaciones perceptibles.

### 6.3 Pruebas Realizadas

Se llevaron a cabo varias pruebas para asegurar la funcionalidad y fiabilidad del Job Scheduler. Estas pruebas incluyeron:

#### 1. Pruebas de Funcionalidad:

- **Definición y programación de tareas:** Se verificó que el sistema permite definir y programar tareas correctamente, con diferentes intervalos y configuraciones.

- Ejecución de tareas: Se realizaron pruebas para asegurar que las tareas se ejecutan en los intervalos especificados y cumplen con su propósito ( por ejemplo: creación de backups, envío de correos, limpieza de la papelera).

## 2. Pruebas de Integración:

- Integración con SMTP: Se probó el envío de correos electrónicos a través de un servidor SMTP configurado, asegurando la entrega de mensajes.
- Acceso a sistemas de archivos: Se verificó el acceso y manipulación de archivos del sistema, especialmente en la creación de copias de seguridad y limpieza de la papelera.

## 3. Pruebas de Desempeño:

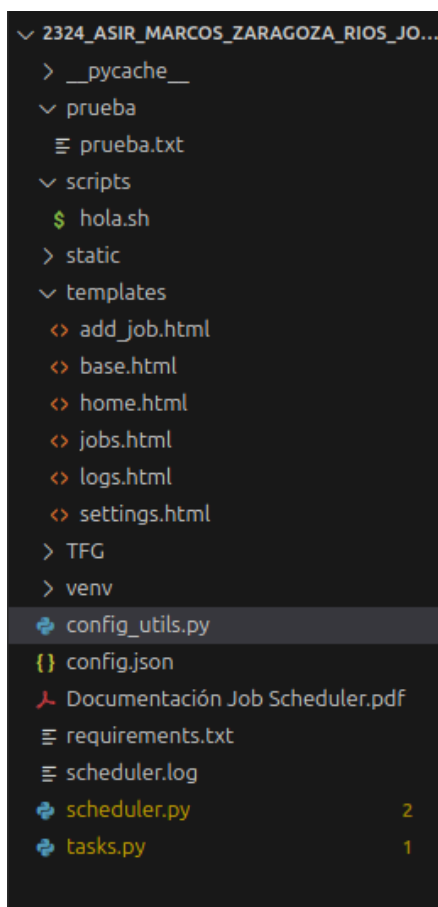
- Monitoreo de recursos: Se midió el uso de CPU y memoria durante la ejecución de tareas intensivas para evaluar el impacto en el sistema.
- Escalabilidad: Se probaron múltiples tareas simultáneamente para asegurar que el sistema puede manejar la carga sin degradación significativa en el rendimiento.

## 7. MANUAL DE USUARIO Y GUÍA DE INSTALACIÓN

### 7.1 Manual de Usuario

Este manual de usuario proporciona una guía detallada para utilizar el Job Scheduler. Se describen las funcionalidades principales, cómo configurarlo y cómo interpretar sus salidas.

#### 7.1.1 Estructura de Ficheros



El directorio principal del Job Scheduler contiene los 6 ficheros principales:

- **scheduler.py**

Es el script principal del Job Scheduler. Contiene la lógica para la programación y ejecución de tareas automatizadas, integrando las funcionalidades proporcionadas por las bibliotecas APScheduler y schedule.

- **tasks.py**

Este script define las tareas que serán programadas y ejecutadas por el Job Scheduler. Incluye funciones específicas para cada tipo de tarea, como limpieza de archivos temporales, generación de copias de seguridad, entre otras.

- **config.json**

Este archivo contiene la configuración principal del sistema, incluyendo parámetros esenciales para la operación del Job Scheduler, como la programación de tareas, credenciales de acceso y rutas de almacenamiento de backups.

Ejemplo de un fichero de configuración:

```
{ } config.json > [ ] jobs
1  {
2    "jobs": [
3      {
4        "id": "668afb5-4b20-4e16-9e58-5ff3868a4a27",
5        "task": "backup_and_transfer",
6        "interval": "daily",
7        "time": "18:11"
8      },
9      {
10       "id": "08068fa1-2075-410e-96bc-c3794ab4271c",
11       "task": "get_logged_in_users_and_send_email",
12       "interval": "minute",
13       "minutes": 2
14     },
15     {
16       "id": "e5e7daf8-83b5-4f2d-af09-4e174c105662",
17       "task": "clean_trash",
18       "interval": "hourly",
19       "minute": 4
20     },
21     {
22       "id": "37d89c9b-adf2-4e77-9870-28ef3c78e74f",
23       "task": "get_logged_in_users_and_send_email",
24       "interval": "daily",
25       "time": "22:56"
26     }
27   ],
28   "email": {
29     "smtp_server": "smtp.gmail.com",
30     "smtp_port": 587,
31     "username": null,
32     "password": null,
33     "from_addr": "marcoszaragoza2002@gmail.com",
34     "to_addrs": [
35       "tfgmarcosz@gmail.com"
36     ]
37   },
38   "daily_login_report": {
39     "hour": 23,
40     "minute": 59
41   },
42   "memory_alert_threshold": 1,
43   "last_execution": "2025-01-10 23:03:18"
44 }
```

## “Jobs”

Se definirán automáticamente la ejecución de tareas que añadamos.

### [Email]

En este apartado se define la configuración del servidor SMTP (Simple Mail Transfer Protocol) que es el que se encargará de mandar los avisos mediante correos electrónicos.

Aquí hay que configurar los siguientes 6 parámetros:

Configuración del servidor SMTP	
<i>smtp_server</i>	Servidor smtp que usaremos, en nuestro caso el de google mail
<i>smtp_port</i>	Este parámetro define el puerto que se utilizará para conectarse al servidor SMTP. Los puertos comunes son 25, 465 y 587:
<i>username</i>	Este parámetro especifica el nombre de usuario que se utilizará para autenticar la conexión con el servidor SMTP. Generalmente, es la dirección de correo electrónico del remitente.
<i>password</i>	Este parámetro define la contraseña asociada con el nombre de usuario especificado. Se utiliza para la autenticación con el servidor SMTP.
<i>from_addr</i>	Este parámetro indica la dirección de correo electrónico del remitente. Es la dirección que aparecerá en el campo "De" (From) del correo enviado.
<i>to_addrs</i>	Este parámetro es una lista de direcciones de correo electrónico de los destinatarios. Especifica a quiénes se enviarán los correos electrónicos. Puede contener múltiples direcciones de correo.

Los parámetros *daily\_login\_report* y *memory\_alert\_threshold* son parámetros que se configurarán automáticamente.

El primero se usa para el envío de un informe diario sobre los usuarios que han iniciado sesión en el sistema.

Y el segundo define el umbral de uso de memoria (en porcentaje) que, al ser superado, desencadenará el envío de una alerta por correo electrónico.

- **scripts/**

Este directorio alberga scripts los cuales puede ser utilizado para ejecutar tareas previamente programadas

- **requirements.txt**

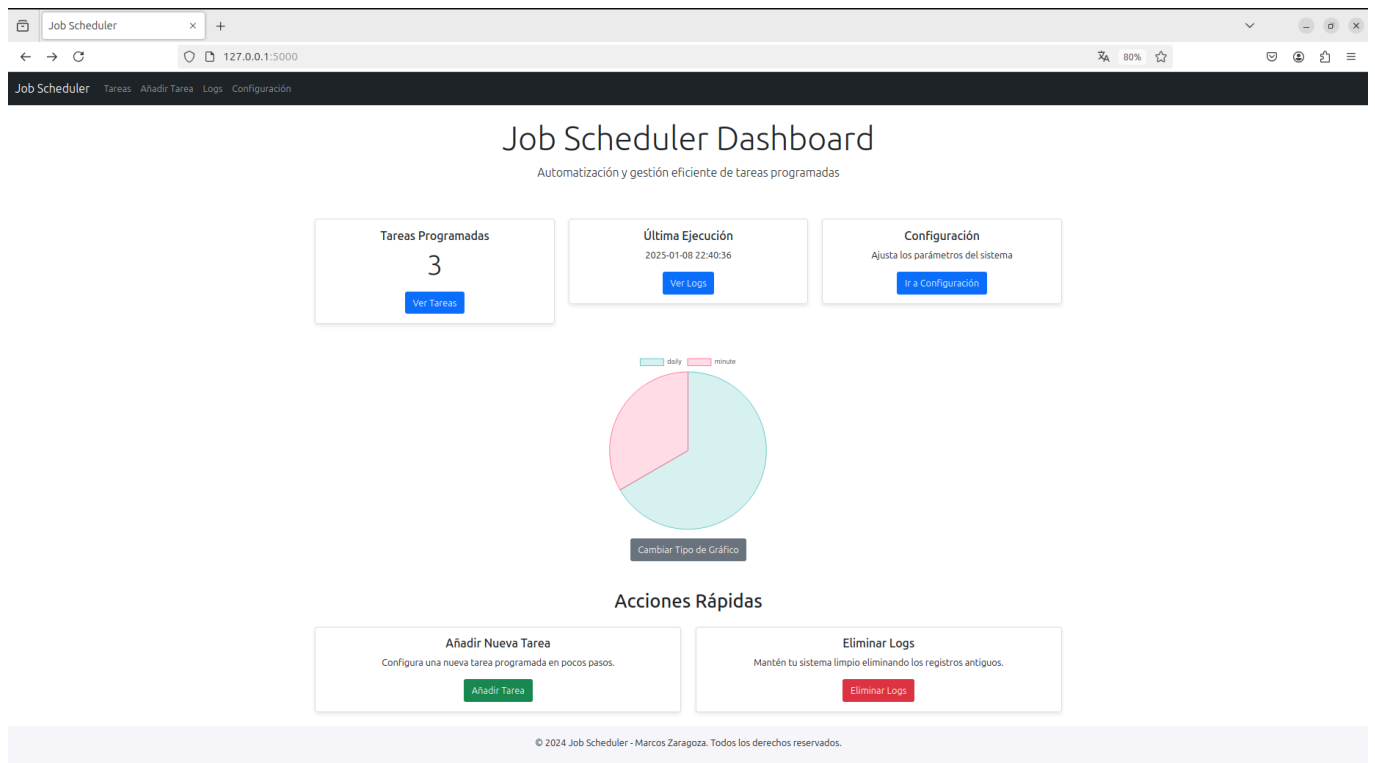
Lista de todas las dependencias de Python necesarias para ejecutar la aplicación. Permite la instalación sencilla de las bibliotecas requeridas mediante el uso del comando `pip install -r requirements.txt`.

### 7.1.2 Acceso a la Interfaz Web

Con el servicio arrancado o ejecutando manualmente la aplicación, podemos abrir nuestro navegador favorito e introducir la dirección ip del servidor seguida del puerto configurado para Flask, por ejemplo:

<http://192.168.1.10:5000>

### 7.1.3 Menú Principal (Dashboard)



El dashboard proporciona una visión general del estado actual del sistema, incluyendo:

**Tareas Programadas:** Número de tareas programadas en ejecución.

**Última Ejecución:** Mostrará la fecha de la última tarea ejecutada y un botón que nos redirigirá al fichero de registros de errores, por si queremos consultarlo en cualquier momento.

**Configuración:** Apartado para configurar los parámetros del sistema .

**Gráfico:** Una gráfica la cual mostrará una estadística de las tareas en ejecución y su intervalo.

**Añadir Nueva Tarea:** Apartado para añadir las tareas y establecer su tiempo de ejecución

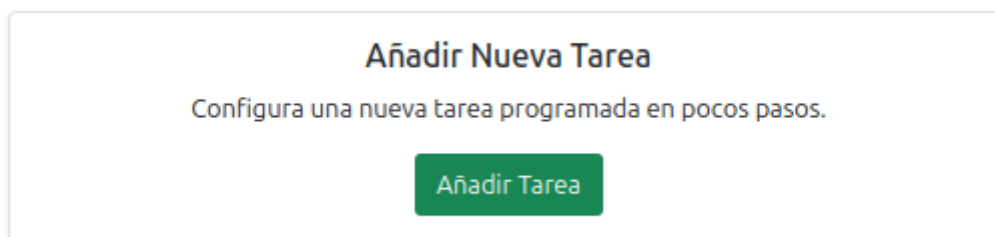
**Eliminar Logs:** Botón para eliminar los registros.

### 7.1.4 Programar Nueva Tarea

Para programar una nueva tarea automatizada:

#### 1. Navegar a "Añadir Tarea":

- Desde el menú principal, seleccione la opción "Añadir Tarea".



#### 2. Completar el Formulario de Tarea:

#### Añadir Nueva Tarea Programada

Tarea

Selecciona una tarea

Intervalo

Selecciona un intervalo

Ruta del Script (opcional)

/ruta/al/script.sh

Guardar Tarea Cancelar

- **Tarea:** Elige entre las diferentes tareas definidas por defecto:

Tarea

Selecciona una tarea

Selecciona una tarea

backup\_and\_transfer

get\_logged\_in\_users\_and\_send\_email

clean\_trash

Ruta del Script (opcional)

Al elegirla, nos mostrará una breve descripción de lo que hace esa tarea:

Tarea

backup\_and\_transfer

Realiza una copia de seguridad y la envía a través de un servidor seguro SFTP.



- **Intervalo:** Define el intervalo de tiempo (cada hora, diariamente, cada X minutos)

Intervalo

Selecciona un intervalo

Selecciona un intervalo

Diario

Cada Hora

Cada Minuto

Guardar Tarea Cancelar

- **Ruta del Script (opcional):** Este apartado sirve por si queremos añadir una tarea que no está definida mediante un fichero de script

### 3. Guardar la Tarea:

- Haga clic en el botón "Guardar Tarea" para programar la tarea. La tarea aparecerá en la lista de tareas programadas del dashboard.

## 7.1.5 Visualizar y Gestionar Tareas

Para gestionar las tareas existentes:

### 1. Acceder a la Lista de Tareas:

- Desde el menú principal, seleccione "Ver Tareas".

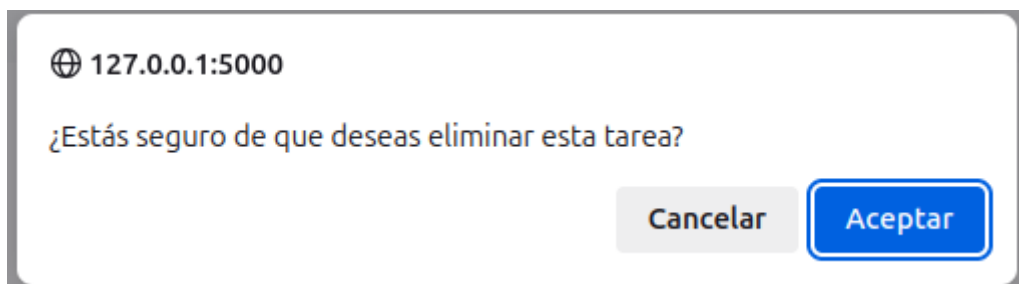


### 2. Opciones Disponibles:

- **Eliminar:** Elimina una tarea programada que ya no sea necesaria.

## Tareas Programadas

#	Tarea	Intervalo	Detalles	Acciones
1	backup_and_transfer	daily	Hora: 23:44	<button>Eliminar</button>
2	backup_and_transfer	daily	Hora: 18:11	<button>Eliminar</button>
3	get_logged_in_users_and_send_email	minute	Cada: 2 minutos	<button>Eliminar</button>

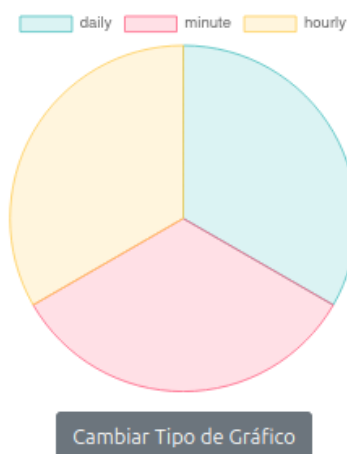
Añadir Nueva Tarea

Si la tarea se ha eliminado correctamente, nos mostrará el siguiente mensaje:

Tarea 'backup\_and\_transfer' eliminada correctamente.

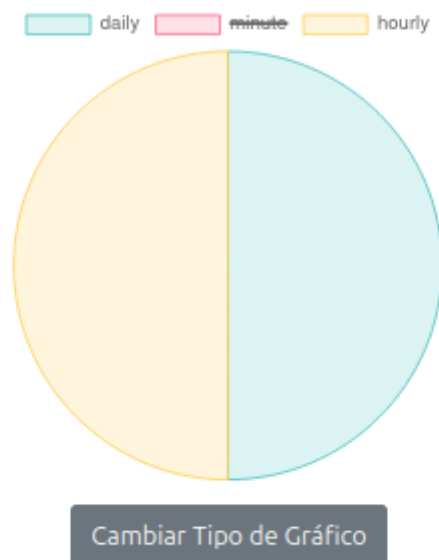
### 7.1.6 Monitorización de Tareas

Con las tareas creadas, podremos monitorizarlas:



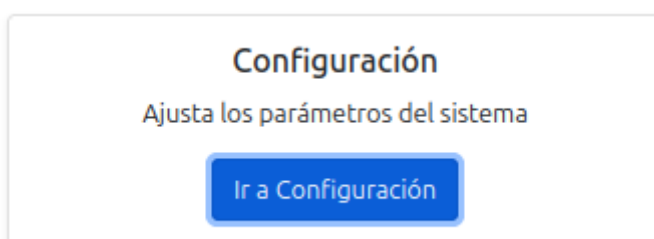


Eligiendo el tipo de gráfico que queramos ver en cada momento, y filtrando por intervalos de ejecución:



### 7.1.7 Configuración de Parámetros:

En el apartado configuración podremos ajustar algunos parámetros del sistema:



## Configuración del Sistema

Servidor SMTP

smtp.gmail.com

Puerto SMTP

587

Umbral de Memoria (%)

1

Guardar

© 2024 Job Scheduler - Marcos Zaragoza. Todos los derechos reservados.

Como el servidor SMTP y el Puerto.

El último define el umbral de uso de memoria (en porcentaje) que, al ser superado, desencadenará el envío de una alerta por correo electrónico.

El umbral se guarda automáticamente en config.json:

```
16     },  
17     "memory_alert_threshold": 15  
18 }
```

## 7.2 Tareas Definidas

#	Tarea
1	backup_and_transfer
2	get_logged_in_users_and_send_email
3	clean_trash

En este ejemplo se presentan tres jobs definidos,

- “*backup\_and\_transfer*”

Este job crea un backup con el intervalo de ejecución que establezcamos de la carpeta que hayamos definido en nuestro programa principal de tareas “tasks.py”, la comprime y la envía a través del protocolo de transferencia segura SFTP a un servidor que hemos configurado previamente.

**\*Importante\*** Deberemos de configurar un servidor SFTP y definirlo en nuestro fichero tasks.py además de la carpeta que queramos respaldar

```
# Parámetros de conexión SFTP
SFTP_HOST = '192.168.1.66'
SFTP_PORT = 22
SFTP_USERNAME = 'backupuser'
SFTP_PASSWORD = '1234'

# Ruta en el servidor SFTP donde se almacenarán los respaldos
SFTP_REMOTE_PATH = '/home/backupuser/backup/backup.zip'

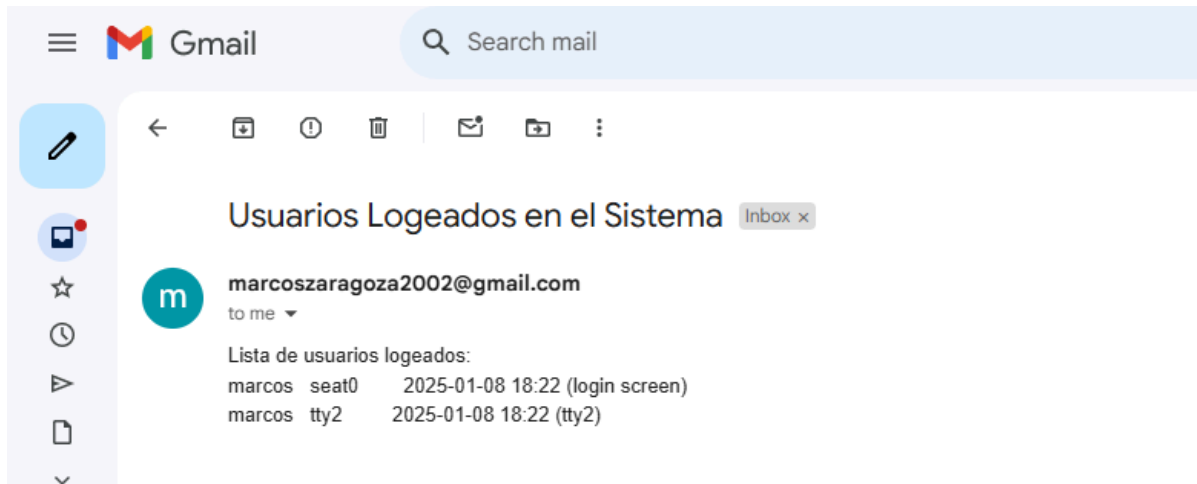
# Ruta local del archivo de respaldo
LOCAL_BACKUP_FILE = '/home/marcos/backup/backup.zip'
```

- “*get\_logged\_in\_users\_and\_send\_email*”

Esta tarea analiza los usuarios que han iniciado sesión en nuestro sistema y se envía un mail al administrador con la siguiente información:

[Nombre de usuario que ha iniciado sesión]	[Terminal/Sesión]	[Fecha y hora de inicio de sesión]
--	-------------------	------------------------------------

Ejemplo del mail que nos llega a nuestra bandeja de entrada:



Esta tarea está programada para que se ejecute cada 30 minutos, tiene como finalidad mantener nuestro sistema controlado y monitorizado 24/7 para detectar accesos no autorizados a nuestro sistema.

- *"clean\_trash"*

Esta tarea limpia la papelera de nuestro sistema, para eliminar archivos basura y liberar espacio, está programada cada hora y media.

## Interpretación de los Logs

### - scheduler.log

El archivo scheduler.log contiene registros detallados de todas las actividades del Job Scheduler. Cada entrada incluye una marca de tiempo, el nivel de log (INFO o ERROR) y un mensaje descriptivo. Utilice este archivo para solucionar problemas y verificar la correcta ejecución de las tareas.

```
13 2025-01-10 23:12:34,748 - INFO - Tarea 'backup_and_transfer' programada diariamente a 18:11
14 2025-01-10 23:12:34,748 - INFO - Tarea 'get_logged_in_users_and_send_email' programada cada 2 minutos
15 2025-01-10 23:12:34,749 - INFO - Tarea 'clean_trash' programada cada hora en el minuto 4
16 2025-01-10 23:12:34,749 - INFO - Tarea 'get_logged_in_users_and_send_email' programada diariamente a 22:56
17 2025-01-10 23:12:34,749 - INFO - Tarea 'get_logged_in_users_and_send_email' programada cada 1 minutos
18 2025-01-10 23:12:34,749 - INFO - Tarea 'clean_trash' programada cada 1 minutos
19 2025-01-10 23:12:34,749 - INFO - Tarea 'backup_and_transfer' programada diariamente a 23:12
20 2025-01-10 23:12:34,749 - INFO - Todas las tareas han sido programadas.
```

Este fichero ha sido crítico para la correcta programación de la aplicación, ya que gracias a él he detectado errores que hacían que mi programa no funcionara.

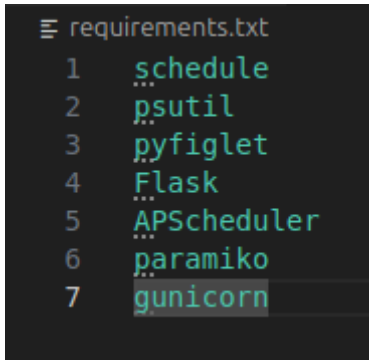
También es accesible través de la interfaz web:

## Logs del Sistema

```
2025-01-10 23:12:34,558 - INFO - Tarea 'backup_and_transfer' programada diariamente a 18:11
2025-01-10 23:12:34,558 - INFO - Tarea 'get_logged_in_users_and_send_email' programada cada 2 minutos
2025-01-10 23:12:34,559 - INFO - Tarea 'clean_trash' programada cada hora en el minuto 4
2025-01-10 23:12:34,559 - INFO - Tarea 'get_logged_in_users_and_send_email' programada diariamente a 22:56
2025-01-10 23:12:34,559 - INFO - Tarea 'get_logged_in_users_and_send_email' programada cada 1 minutos
2025-01-10 23:12:34,559 - INFO - Tarea 'clean_trash' programada cada 1 minutos
2025-01-10 23:12:34,559 - INFO - Tarea 'backup_and_transfer' programada diariamente a 23:12
2025-01-10 23:12:34,559 - INFO - Todas las tareas han sido programadas.
2025-01-10 23:12:34,588 - INFO - [31m[1mWARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.[0m
* Running on http://127.0.0.1:5000
2025-01-10 23:12:34,589 - INFO - [33mPress CTRL+C to quit[0m
2025-01-10 23:12:34,590 - INFO - * Restarting with stat
```

### - requirements.txt

Este fichero contiene las librerías necesarias para el funcionamiento de nuestro Job Scheduler, en la guía de instalación explicaremos cómo instalarlas.

A screenshot of a code editor with a dark background. The file name 'requirements.txt' is visible at the top. The code is as follows:

```
1 schedule
2 psutil
3 pyfiglet
4 Flask
5 APScheduler
6 paramiko
7 gunicorn
```

En este fichero no se han incluido todas la librerías que usa nuestro programa, ya que las siguientes vienen instaladas por defecto con la instalación de Python:

```
os
json
shutil
smtpplib
subprocess
time
datetime
logging
zipfile
email.mime.multipart
email.mime.text
email.mime.base.
```

En caso de no estar instaladas, habría que instalarlas manualmente con el comando:

```
pip install [librería]
```



## Solución de Problemas Comunes

Error al enviar correo: Verifique la configuración SMTP en config.json y asegúrese de que las credenciales y los servidores sean correctos.

Tarea no se ejecuta a la hora programada: Revise el archivo de log para identificar posibles errores y asegúrese de que el sistema esté en ejecución.

Uso de memoria alto sin alerta: Asegúrese de que el umbral de alerta esté correctamente configurado en config.json.

*Este manual de usuario debe proporcionar toda la información necesaria para utilizar eficazmente el Job Scheduler. Si tiene preguntas adicionales o necesita asistencia técnica, consulte la documentación adicional o comuníquese con el correo de soporte [tfgmarcosz@gmail.com](mailto:tfgmarcosz@gmail.com).*

## 7.2 Guía de instalación

Esta guía proporciona instrucciones detalladas para la instalación y configuración del Job Scheduler. Los archivos necesarios están alojados en un repositorio de GitHub, que debe ser clonado para completar el proceso de instalación.

### Requisitos Previos

1. Sistema Operativo: Este programa está diseñado para funcionar en sistemas basados en Unix, como Linux.
2. Python: Asegúrese de tener Python 3.12 instalado en su sistema.
3. Git: Necesita Git instalado para clonar el repositorio.

### Paso 1: Clonar el Repositorio de GitHub

Primero, debe clonar el repositorio que contiene el código del Job Scheduler. Abra una terminal y ejecute el siguiente comando:

```
git clone  
https://github.com/ieslavereda-projects/2324_ASIR_MARCOS_ZARAGOZA  
_RIOS_JOB_SCHEDULER.git
```

Esto creará una copia local del repositorio en su máquina.

### Paso 2: Navegar al Directorio del Proyecto

Cambie al directorio del proyecto recién clonado:

```
cd 2324_ASIR_MARCOS_ZARAGOZA_RIOS_JOB_SCHEDULER
```

### Paso 3: Crear y Activar un Entorno Virtual (opcional pero recomendado)

Para evitar conflictos de dependencias, se recomienda crear un entorno virtual para el proyecto:

```
python3 -m venv TFG
```

```
source TFG/bin/activate
```

#### Paso 4: Instalar Dependencias

Instale las dependencias necesarias utilizando pip:

```
pip install -r requirements.txt
```

#### Paso 5: Configurar el servidor SMTP

El servidor SMTP debe configurarse en el fichero de configuración config.json para definir los parámetros necesarios para el funcionamiento de envío de alerta del Job Scheduler.

En el Manual de usuario hay un apartado que explica todos los parámetros que pueden configurarse

Este es un ejemplo de configuración:

```
{
  "email": {
    "smtp_server": "smtp.example.com",
    "smtp_port": 587,
    "username": "usuario@example.com",
    "password": "contraseña_segura",
    "from_addr": "usuario@example.com",
    "to_addrs": ["destinatario1@example.com",
    "destinatario2@example.com"]
  },
}
```

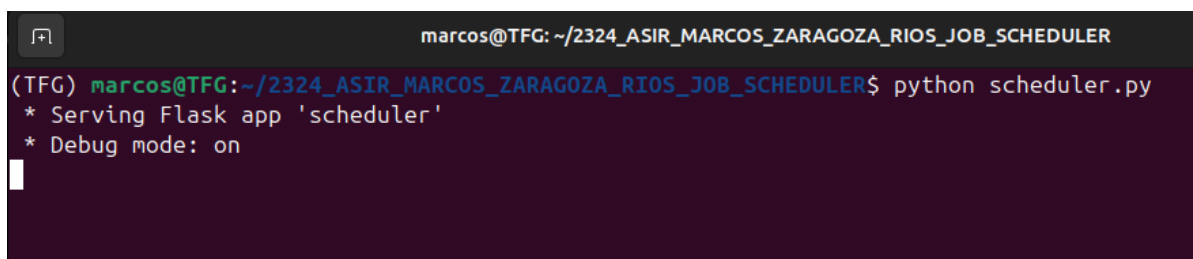
Asegúrese de modificar estos valores según sus necesidades.

## Paso 6: Ejecutar el Job Scheduler

Para ejecutar el Job Scheduler, utilice el siguiente comando:

```
python scheduler.py
```

(asegurate que el programa tenga los permisos necesarios para ejecutarse en su sistema)

A terminal window with a dark background. The title bar shows 'marcos@TFG: ~/2324\_ASIR\_MARCOS\_ZARAGOZA\_RIOS\_JOB\_SCHEDULER'. The prompt is '(TFG) marcos@TFG:~/2324\_ASIR\_MARCOS\_ZARAGOZA\_RIOS\_JOB\_SCHEDULER\$'. The command 'python scheduler.py' has been entered. The output shows two lines: '\* Serving Flask app 'scheduler'' and '\* Debug mode: on'. A cursor is visible on the line following the output.

```
marcos@TFG: ~/2324_ASIR_MARCOS_ZARAGOZA_RIOS_JOB_SCHEDULER
(TFG) marcos@TFG:~/2324_ASIR_MARCOS_ZARAGOZA_RIOS_JOB_SCHEDULER$ python scheduler.py
* Serving Flask app 'scheduler'
* Debug mode: on
```

Esto iniciará el programa y mostrará que flask se ha ejecutado, por lo que una vez ejecutado y sin errores podremos acceder a la Interfaz Web:

Abriremos nuestro navegador favorito e introducir la dirección ip del servidor seguida del puerto configurado para Flask, por ejemplo:

<http://192.168.1.10:5000>

## (Opcional) Paso 7: Configurar el Servicio para Ejecutarse en Segundo Plano

### 1. Crear el Archivo de Servicio:

```
sudo nano /etc/systemd/system/jobscheduler.service
```

Agrega la siguiente configuración

```

marcos@TFG: ~/2324_ASIR_MARCOS_ZARAGOZA_RIOS_JOB_SCHEDULER
GNU nano 7.2 /etc/systemd/system/jobscheduler.service
[Unit]
Description=Job Scheduler Service
After=network.target

[Service]
# El usuario que ejecuta la aplicación
User=marcos

# Ruta del proyecto
WorkingDirectory=/home/marcos/2324_ASIR_MARCOS_ZARAGOZA_RIOS_JOB_SCHEDULER

# Ejecutable de Python(entorno virtual) + archivo principal + definicion del puerto en el que se ejecuta el programa mediante parámetro
ExecStart=/home/marcos/TFG/bin/python /home/marcos/2324_ASIR_MARCOS_ZARAGOZA_RIOS_JOB_SCHEDULER/scheduler.py --port=5010

# Si algo falla se reiniciará automáticamente
Restart=always

[Install]
WantedBy=multi-user.target

^G Ayuda      ^O Guardar    ^W Buscar     ^K Cortar     ^T Ejecutar   ^C Ubicación  M-U Deshacer  M-A Poner marca
^X Salir      ^R Leer fich. ^\ Reemplazar ^U Pegar      ^J Justificar ^/_ Ir a línea  M-E Rehacer   M-6 Copiar

```

Recargar systemd y Habilitar el Servicio:

```

sudo systemctl daemon-reload
sudo systemctl start jobscheduler
sudo systemctl enable jobscheduler

```

Verificaremos el estado:

```

marcos@TFG: ~/2324_ASIR_MARCOS_ZARAGOZA_RIOS_JOB_SCHEDULER
(TFG) marcos@TFG:~/2324_ASIR_MARCOS_ZARAGOZA_RIOS_JOB_SCHEDULER$ sudo systemctl start jobscheduler.service
(TFG) marcos@TFG:~/2324_ASIR_MARCOS_ZARAGOZA_RIOS_JOB_SCHEDULER$ sudo systemctl status jobscheduler.service
● jobscheduler.service - Job Scheduler Service
   Loaded: loaded (/etc/systemd/system/jobscheduler.service; enabled; preset: enabled)
   Active: active (running) since Fri 2025-01-10 23:23:14 CET; 2s ago
     Main PID: 10520 (python)
        Tasks: 5 (limit: 4614)
      Memory: 54.0M (peak: 54.7M)
         CPU: 396ms
    CGroup: /system.slice/jobscheduler.service
            └─10520 /home/marcos/TFG/bin/python /home/marcos/2324_ASIR_MARCOS_ZARAGOZA_RIOS_JOB_SCHEDULER/sc>
              10522 /home/marcos/TFG/bin/python /home/marcos/2324_ASIR_MARCOS_ZARAGOZA_RIOS_JOB_SCHEDULER/sc>

ene 10 23:23:14 TFG systemd[1]: Started jobscheduler.service - Job Scheduler Service.
ene 10 23:23:14 TFG python[10520]: * Serving Flask app 'scheduler'
ene 10 23:23:14 TFG python[10520]: * Debug mode: on

(TFG) marcos@TFG:~/2324_ASIR_MARCOS_ZARAGOZA_RIOS_JOB_SCHEDULER$

```

Este comando mostrará el estado actual del servicio, confirmando que está activo y funcionando correctamente.

Siguiendo estos pasos, debería poder instalar y configurar el Job Scheduler correctamente. Si encuentra algún problema o tiene preguntas adicionales, no dude en consultar la documentación o contactar con el correo de soporte [tfgmarcosz@gmail.com](mailto:tfgmarcosz@gmail.com)

## 8. CONCLUSIONES

El desarrollo del Job Scheduler ha sido un proyecto integral que ha permitido aplicar y consolidar mis conocimientos adquiridos a lo largo de la formación en los dos años de ASIR. Este proyecto ha cumplido con los objetivos planteados inicialmente y ha demostrado ser una herramienta eficaz para la automatización de tareas en entornos Linux.

### 8.1 Alcance y Logros

El principal logro del JobScheduler ha sido la implementación exitosa de un sistema que permite definir, programar y ejecutar tareas de manera automatizada.

### 8.2 Desafíos, Problemas y Lecciones Aprendidas

Durante el desarrollo del Job Scheduler, se enfrentaron varios desafíos que proporcionaron valiosas lecciones:

- Integración de servicios externos: La configuración y uso de servidores SMTP para el envío de correos electrónicos requirió una comprensión profunda de la seguridad y la configuración de red. Este proceso encontró problemas como la configuración incorrecta del servidor y las credenciales, que fueron resueltos mediante documentación exhaustiva, tutoriales en youtube y pruebas repetidas.
- Gestión de recursos: Implementar tareas que monitorean y utilizan recursos del sistema sin causar una sobrecarga fue un aspecto crítico. Se encontraron problemas de rendimiento cuando varias tareas se ejecutaban simultáneamente, lo que llevó a optimizar el código y mejorar la gestión de recursos.
- Manejo de excepciones: La importancia de manejar adecuadamente las excepciones y errores para asegurar la fiabilidad del sistema fue una lección clave. Durante las pruebas, se identificaron y resolvieron varios errores no manejados que podrían haber causado fallos en la ejecución de tareas.
- Compatibilidad y Permisos: Se encontraron problemas de compatibilidad con diferentes versiones de Linux y permisos de usuario que impedían la ejecución de ciertas tareas. Estos problemas se resolvieron ajustando las configuraciones y probando en diferentes entornos.

### 8.3 Futuras Mejoras

Aunque el Job Scheduler ha cumplido con los objetivos principales, siempre hay margen para mejoras y expansiones futuras:

- Automatización de Tareas Basada en Inteligencia Artificial: Incorporar algoritmos de machine learning para predecir y optimizar la programación de tareas, adaptándose dinámicamente a las necesidades del sistema.
- Soporte para múltiples Sistemas Operativos: Extender la compatibilidad del Job Scheduler a otros sistemas operativos, como Windows y macOS.
- Notificaciones en tiempo real: Implementar un sistema de notificaciones en tiempo real para alertar a los usuarios sobre el estado de las tareas programadas y posibles fallos.

## 8.4 Conclusión Final

El Job Scheduler ha demostrado ser una solución efectiva y robusta para la automatización de tareas en entornos Linux. La implementación de este proyecto no solo ha proporcionado una herramienta útil para la gestión del sistema, sino que también ha permitido consolidar una serie de habilidades técnicas y de gestión de proyectos. El éxito de este proyecto sienta una base sólida para futuras exploraciones y desarrollos en el ámbito de la automatización y la gestión de sistemas.

## 9. REFERENCIAS

The Python Standard Library Documentation <https://docs.python.org/3/library/>

Schedule Library Documentation <https://schedule.readthedocs.io/en/stable/>

psutil Documentation <https://psutil.readthedocs.io/en/latest/>

schedule Documentation <https://schedule.readthedocs.io/en/stable/>

smtplib SMTP protocol client <https://docs.python.org/3/library/smtplib.html>

Python Email Library Documentation <https://docs.python.org/3/library/email.html>

Shutil High-level file operations <https://docs.python.org/3/library/shutil.html>

Real Python <https://realpython.com/>

Stack Overflow <https://stackoverflow.com/>

GitHub schedule library <https://github.com/dbader/schedule>

Scheduling Tasks Professionally in Python

<https://www.youtube.com/watch?v=yDPQfj4bZY8>

How To SCHEDULE Functions & Tasks In Python (FULL GUIDE)

<https://www.youtube.com/watch?v=FCPBG6NqMmQ>

Automatizar tareas con python

<https://www.iebschool.com/blog/automatizar-tareas-con-python-simplificando-tu-trabajo-diario-big-data/>

Documentación Paramiko: <https://docs.paramiko.org/en/stable/>

Documentación de Flask: <https://flask-es.readthedocs.io/>



## 10. ANEXOS

### Repositorio GitHub

Junto a esta memoria adjunto un enlace a un repositorio de GitHub, donde están todos los ficheros y configuraciones necesarios para el funcionamiento del Job Scheduler.

[https://github.com/ieslavereda-projects/2324\\_ASIR\\_MARCOS\\_ZARAGOZA\\_RIOS\\_JOB\\_SCHEDULER](https://github.com/ieslavereda-projects/2324_ASIR_MARCOS_ZARAGOZA_RIOS_JOB_SCHEDULER)