INFORMATION RISK & SECURITY INSTITUTE
CURSO DE SCRIPTING

SISTEMA AUTOMATIZADO DE GESTIÓN DE COMPRAS, FACTURAS, ENVIOS Y USUARIOS

Integrantes:
Josué Vásquez Latin
Sofia Alvarado
Criss Valdes
Fecha: 24 de julio de 2025

1. Introducción

En el contexto actual de la gestión administrativa, la automatización de procesos es una herramienta fundamental para aumentar la eficiencia, reducir errores humanos y garantizar la seguridad en el manejo de datos sensibles. Este proyecto tiene como objetivo diseñar e implementar un sistema automatizado que permita la generación, gestión y envío de compras, facturas, reportes y usuarios temporales, utilizando una combinación de tecnologías scripting (Python, Bash, PowerShell) coordinadas mediante herramientas de automatización de tareas programadas como cron en linux y Task Scheduler para sistemas windows.

La solución propuesta integra múltiples módulos especializados que interactúan de forma orquestada para cubrir todo el flujo del proceso administrativo digitalizado, desde la generación de datos hasta la entrega y registro de resultados, garantizando trazabilidad y control mediante logs y reportes automáticos.

2. Arquitectura Implementada

Arquitectura Modular en Capas + Separación de Responsabilidades por Dominio Funcional

El proyecto está organizado siguiendo una arquitectura modular en capas con separación funcional por dominio, lo que permite independencia tecnológica entre los distintos scripts (Python, Bash, PowerShell) y facilita una automatización ordenada y orquestada mediante cron o Task Scheduler.

Este diseño está inspirado en los principios de Clean Architecture y Buenas Prácticas DevOps para scripting cross-platform, incorporando conceptos básicos de Domain-Driven Design (DDD) para mantener cada funcionalidad aislada y fácilmente mantenible.

Resumen de la arquitectura:

- Modularidad**:** Cada funcionalidad reside en un submódulo específico dentro del proyecto (por ejemplo, core/compras, core/envios, infrastructure), asegurando la independencia y reutilización del código.
- Escalabilidad: La estructura permite agregar nuevos módulos o funcionalidades sin afectar los existentes.
- Reutilización: Las funciones y scripts están diseñados para ser reutilizables y separados por responsabilidad clara.
- Automatización ordenada: El directorio infrastructure/ contiene los scripts de orquestación, tareas programadas, generación de reportes y logs.
- Mantenibilidad: La separación clara entre datos, scripts, configuraciones y documentación facilita las labores de mantenimiento y mejora continua.
- Entrega profesional: Esta organización permite generar la documentación y evidencias completas (como este reporte PDF), integrando código, logs y capturas de pantalla de forma coherente.

Componentes Principales y Flujo

Módulo / Script	Lenguaje	Función principal
generador_compras.py	Python	Genera archivos CSV con datos de compras
generador_facturas.sh	Bash	Genera facturas y compila documentos LaTeX
enviador.py	Python	Envía archivos CSV por correo y limpia datos
usuarios.ps1	PowerShell	Crea usuarios temporales con contraseñas seguras
Automatización (cron/TS)	Bash/Windows	Orquesta la ejecución programada y generación de logs

Diagrama de Arquitectura

```
proyecto_facturacion_irsis_sisap/
│
├── core/                    ← Dominio principal del sistema
│   ├── compras/             ← Lógica de generación de compras (Python)
│   │   └── generador_compras.py
│   ├── facturacion/         ← Sustitución y generación PDF (Bash + LaTeX)
│   │   ├── generador_facturas.sh
│   │   └── plantilla_factura.tex
│   ├── envios/              ← Envío de correos (Python)
│   │   └── enviador.py
│   │   |___enviar_correo.py
│   └── usuarios/            ← Gestión de empleados temporales (PowerShell)
│       └── usuarios.ps1
│
├── infrastructure/          ← Automatización, tareas programadas, y ejecución cruzada
│   ├── cron/
│   │   └── cron_jobs.sh
│   ├── task_scheduler/
│   │   └── tareas_diarias.xml (si estás en Windows)
│   └── reportes/
│       └── resumen_diario.py
│
├── data/                    ← Datos generados o leídos (entradas y salidas)
│   ├── csv/
│   │   ├── compras_*.csv
│   │   ├── empleados_*.csv
│   │   └── pendientes_envio.csv
│   ├── facturas/
│   │   └── *.pdf
│   └── logs/
│       ├── log_diario.log
│       ├── log_envios.csv
│       └── log_usuarios.txt
├── docs/                    ← Reporte técnico, README, apéndices
│   ├── reporte_final_proyecto.pdf
│   ├── instrucciones_de_uso.pdf
│   └── capturas/
│       ├── correo_ok.png
│       └── usuario_ok.png
│
│
└── README.md                ← Instrucciones del sistema (técnicas y de uso)
```

3. Desarrollo Detallado

3.1 generador_compras.py

Este módulo está encargado de la generación automatizada de archivos CSV que simulan registros de compras con datos ficticios y realistas para pruebas y demostraciones. Utiliza la biblioteca Faker para generar datos coherentes y variados, y aplica una función para escapar caracteres especiales compatibles con LaTeX, asegurando la integridad en la generación de reportes posteriores.

Funcionamiento principal:

- **Generación de datos ficticios:** Se crean registros con campos como nombre, correo, teléfono, dirección, ciudad, cantidad, monto, tipo y estado de pago, dirección IP, fecha de emisión, timestamp y observaciones.
- **Escapado de caracteres LaTeX:** Se implementa la función latex_escape que transforma caracteres especiales y acentos para evitar errores al compilar reportes en LaTeX.
- **Creación dinámica de directorios y archivos:** El script verifica y crea la carpeta de salida (data/csv/) si no existe, y guarda los archivos CSV con un nombre basado en la fecha y hora actual para evitar sobreescrituras.
- **Registro en logs:** Cada generación de CSV queda registrada en un log específico con marca temporal, nombre del archivo y cantidad de registros generados, usando una función externa agregar_log.

Fragmento de código:

(Referencia Apéndice, sección generador_compras.py)

```
def generar_registro(fake, idx):
    # Campos generados con Faker y valores aleatorios controlados
    nombre = latex_escape(fake.name())
    correo = latex_escape(fake.email())
    # ...
    cantidad = randint(1, 10)
    monto = round(cantidad * randint(50, 500), 2)
    # ...
    return [str(idx), fecha_emision, nombre, correo, ..., observaciones]
```

El módulo permite generar un número configurable de registros (por defecto 10) mediante la función generar_csv_compras, que orquesta la creación del directorio, generación de datos y almacenamiento en disco. Salida esperada: Un archivo CSV con la estructura y datos detallados, listo para ser procesado por otros módulos del sistema. (Ver código en Apéndice).

3.2 generador_facturas.sh

Script Bash que automatiza la creación de facturas y compila los documentos en formato PDF utilizando LaTeX. Se diseñó para integrarse en sistemas Linux/Unix y se valida la correcta generación mediante logs. (Ver código en Apéndice).

3.3 enviador.py

Este script Python automatiza el envío de archivos CSV a destinatarios configurados mediante correo electrónico, incluye validaciones previas al envío y realiza limpieza de archivos procesados para evitar duplicidades.
Decisión técnica clave**:** Se implementaron validaciones estrictas para asegurar integridad de datos y se gestionan excepciones para robustez. (Ver código en Apéndice).

3.4 usuarios.ps1

PowerShell script encargado de crear usuarios temporales en el sistema, generando contraseñas seguras con alta entropía, registrando cada acción en logs específicos para auditoría y asegurando que no haya datos sensibles hardcodeados.
Evidencia: *(Capturas de pantalla y logs de creación de usuarios)*

3.5 Automatización con cron / Task Scheduler

La ejecución de los scripts se programa en horarios específicos para asegurar un flujo ordenado y continuo:

- generador_facturas.sh a la 1:00 AM
- enviador.py a la 2:00 AM
- Consolidación de logs y reportes al finalizar las tareas

Esta automatización garantiza la operación autónoma y la generación de reportes diarios.

4. Instrucciones para Ejecutar el Sistema

Requisitos

- Python 3.8+ con módulos: csv, smtplib, os, etc.
- Bash shell compatible (Linux/WSL)
- PowerShell 5.1+ (Windows)
- LaTeX instalado y configurado para compilación de documentos
- Configuración de cron en Linux o Task Scheduler en Windows
-

Pasos para ejecución manual

1. Clonar el repositorio y posicionarse en la carpeta raíz.
2. Ejecutar python generador_compras.py para generar los CSV.
3. Ejecutar ./generador_facturas.sh para compilar las facturas.
4. Ejecutar python enviador.py para enviar los archivos y limpiar CSV.
5. Ejecutar .\usuarios.ps1 -u para crear usuarios temporales.
6. Verificar logs en el directorio infrastructure/reportes/.

O ejecutar el main.py de forma directa y pasarle una de estas banderas -g, -f, -i, -e, -u

Automatización

Configurar cron o Task Scheduler para que ejecute:

- generador_facturas.sh a la 1:00 AM
- enviador.py a la 2:00 AM
- Consolidación y generación de reportes al finalizar

5. Conclusión

Este proyecto ha permitido desarrollar un sistema integral que automatiza procesos administrativos críticos, utilizando una arquitectura modular y principios de diseño profesional. La combinación de diferentes tecnologías scripting con una correcta orquestación garantiza eficiencia, seguridad y mantenibilidad.

Se destaca la importancia de aplicar buenas prácticas en la generación de contraseñas, manejo de errores, validación de datos y documentación, que han sido clave para el éxito del sistema. Además, el uso de LaTeX para reportes ofrece una presentación formal y profesional.

Como áreas de mejora futura se identifican la incorporación de interfaces gráficas y la integración con bases de datos para mayor robustez y escalabilidad.

6. Problemas Encontrados y Soluciones

Durante el desarrollo, se enfrentaron varios retos técnicos:

- **Manejo de rutas y datos sensibles:** Se resolvió evitando hardcodeo y utilizando variables de entorno.
- **Compatibilidad cross-platform:** Se diseñaron scripts específicos para cada plataforma y se orquestó mediante herramientas estándar como cron y Task Scheduler.
- **Generación de contraseñas seguras:** Se implementó una función específica en PowerShell que cumple con criterios de complejidad y aleatoriedad.
- **Compilación LaTeX automatizada:** Se optimizó el script para detectar y reportar errores en logs, facilitando la depuración.

Estas soluciones se documentaron cuidadosamente para asegurar trazabilidad y permitir futuras mejoras.

7. Apéndice: Código Fuente y Logs

Código fuente: generador_compras.py

```
import os
import csv
from faker import Faker
from random import randint, choice
from datetime import datetime
from core.utils.logger import agregar_log

script_dir = os.path.dirname(os.path.abspath(__file__))
proyecto_dir = os.path.abspath(os.path.join(script_dir, "../../"))
log_path_csv = os.path.join(proyecto_dir, "data", "logs",
"generacion_compras_csv.log")

def latex_escape(text):
    if not isinstance(text, str):
        text = str(text)
    replacements = {
```

```
        '\\': r'\textbackslash{}',
        '&': r'\&',
        '%': r'\%',
        '$': r'\$',
        '#': r'\#',
        '_': r'\_',
        '{': r'\{',
        '}': r'\}',
        '~': r'\textasciitilde{}',
        '^': r'\textasciicircum{}',
        'á': r"\'{a}",
        'é': r"\'{e}",
        'í': r"\'{i}",
        'ó': r"\'{o}",
        'ú': r"\'{u}",
        'Á': r"\'{A}",
        'É': r"\'{E}",
        'Í': r"\'{I}",
        'Ó': r"\'{O}",
        'Ú': r"\'{U}",
        'ñ': r'\~{n}',
        'Ñ': r'\~{N}',
    }
    for key, val in replacements.items():
        text = text.replace(key, val)
    return text

def crear_directorio_salida():
    BASE_DIR = os.path.dirname(os.path.abspath(__file__))
    output_dir = os.path.join(BASE_DIR, "../../data", "csv")
    os.makedirs(output_dir, exist_ok=True)
    return output_dir

def generar_registro(fake, idx):
    pagos = ["completo", "fraccionado"]
    estados = ["exitoso", "fallido"]


    nombre = latex_escape(fake.name())
    correo = latex_escape(fake.email())
    telefono = latex_escape(fake.phone_number())
    direccion = latex_escape(fake.street_address().replace(',', ''))
    ciudad = latex_escape(fake.city())
    cantidad = randint(1, 10)
    monto = round(cantidad * randint(50, 500), 2)
    pago = choice(pagos)
    estado_pago = choice(estados)
```

```
    ip = fake.ipv4()
    fecha_emision = fake.date()
    timestamp = datetime.now().isoformat()
    observaciones = latex_escape(choice(["Cliente frecuente", "Promoción aplicada",
""]))

    return [
        str(idx),
        fecha_emision,
        nombre,
        correo,
        telefono,
        direccion,
        ciudad,
        str(cantidad),
        str(monto),
        pago,
        estado_pago,
        ip,
        timestamp,
        observaciones
    ]

def generar_registros(num=10):
    fake = Faker('es-ES')
    registros = []
    for i in range(1, num+1):
        registros.append(generar_registro(fake, i))
    return registros

def guardar_csv(registros, output_dir):
    headers = [
        "id_transaccion", "fecha_emision", "nombre", "correo", "telefono",
        "direccion", "ciudad", "cantidad", "monto", "pago",
        "estado_pago", "ip", "timestamp", "observaciones"
    ]
    filename = f"compras_{datetime.now().strftime('%Y%m%d_%H%M%S')}.csv"
    filepath = os.path.join(output_dir, filename)

    with open(filepath, mode="w", newline="", encoding="utf-8") as f:
        writer = csv.writer(f)
        writer.writerow(headers)
        writer.writerows(registros)

     agregar_log(log_path_csv, [datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
"CSV generado", filename, f"Registros: {len(registros)}"])
```

```
    return filepath

def generar_csv_compras(num_registros=10):
    output_dir = crear_directorio_salida()
    registros = generar_registros(num_registros)
    filepath = guardar_csv(registros, output_dir)
    print(f"Archivo generado: {filepath}")
    return filepath
```

Código fuente: generador_facturas.sh

```
#!/bin/bash

export LANG=es_GT.UTF-8
export LC_ALL=es_GT.UTF-8

SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"

csv_dir="$SCRIPT_DIR/../../data/csv"
template="$SCRIPT_DIR/plantilla_facturas.tex"
output_dir="$SCRIPT_DIR/../../data/facturas"
log="$SCRIPT_DIR/../../data/logs/log_diario.log"
pendientes="$SCRIPT_DIR/../../data/csv/pendientes_envio.csv"

mkdir -p "$output_dir"
: > "$pendientes"
: > "$log"

echo "Plantilla existe? $( [ -f "$template" ] && echo 'Sí' || echo 'No' )"

shopt -s nullglob
csv_files=("$csv_dir"/compras_*.csv)
shopt -u nullglob

echo "Archivos CSV encontrados: ${#csv_files[@]}"
for f in "${csv_files[@]}"; do echo " - $f"; done

if [ ${#csv_files[@]} -eq 0 ]; then
    echo "No se encontraron archivos CSV en $csv_dir"
    exit 1
fi

# Función para escapar caracteres LaTeX
escape_latex() {
    echo "$1" | sed \
        -e 's/\\/\\textbackslash{}/g' \
        -e 's/_/\\_/g' \
```

```
        -e 's/&/\\&/g' \
        -e 's/%/\\%/g' \
        -e 's/#/\\#/g' \
        -e 's/{/\\{/g' \
        -e 's/}/\\}/g' \
        -e 's/\$/\\\$/g' \
        -e 's/\^/\\^/g' \
        -e 's/~/\\~/g' \
        -e 's/"/"/g'

}

for file in "${csv_files[@]}"; do
    echo "Procesando archivo: $file"

    tail -n +2 "$file" | while IFS=',' read -r id_transaccion fecha_emision nombre correo telefono direccion ciudad cantidad monto pago estado_pago ip timestamp observaciones; do

        nombre=$(escape_latex "$nombre")
        direccion=$(escape_latex "$direccion")
        ciudad=$(escape_latex "$ciudad")
        observaciones=$(escape_latex "$observaciones")
        correo=$(escape_latex "$correo")
        telefono=$(escape_latex "$telefono")
        pago=$(escape_latex "$pago")
        estado_pago=$(escape_latex "$estado_pago")

        nombre_archivo="factura_${id_transaccion}.tex"
        tex="$output_dir/$nombre_archivo"
        pdf="${tex/.tex/.pdf}"

        sed -e "s|{id_transaccion}|$id_transaccion|g" \
            -e "s|{fecha_emision}|$fecha_emision|g" \
            -e "s|{nombre}|$nombre|g" \
            -e "s|{correo}|$correo|g" \
            -e "s|{telefono}|$telefono|g" \
            -e "s|{direccion}|$direccion|g" \
            -e "s|{ciudad}|$ciudad|g" \
            -e "s|{cantidad}|$cantidad|g" \
            -e "s|{monto}|$monto|g" \
            -e "s|{pago}|$pago|g" \
            -e "s|{estado_pago}|$estado_pago|g" \
            -e "s|{ip}|$ip|g" \
            -e "s|{timestamp}|$timestamp|g" \
            -e "s|{observaciones}|$observaciones|g" \
            "$template" > "$tex"
```

```
        echo "DEBUG: Archivo .tex generado: $tex"
        ls -l "$tex"
        head -5 "$tex"

        pdflatex -interaction=nonstopmode -output-directory="$output_dir" "$tex" 2>&1
| tee -a "$log"

        if [ $? -eq 0 ]; then
            echo "Factura generada: $pdf"
            echo "$pdf,$correo" >> "$pendientes"
        else
            echo "Error al generar PDF para ID: $id_transaccion" >> "$log"
        fi

        # rm "$tex"  # Comentar para depurar
    done
done

echo "✅ Script completado."
```

Código fuente: enviador.py

```
import csv
import os
from core.utils.logger import agregar_log
from .enviar_correo import enviar_correo

EMAIL_DESTINO = "advinjosuev899@gmail.com"
ASUNTO = "Factura Electrónica"
CUERPO = """Estimado cliente,

Adjunto a este correo encontrará su factura en formato PDF.

Saludos cordiales,
Tu Empresa"""

script_dir = os.path.dirname(os.path.abspath(__file__))
proyecto_dir = os.path.abspath(os.path.join(script_dir, "../../"))
log_path = os.path.join(proyecto_dir, "data", "logs", "log_envios.log")

def registrar_envio(nombre_pdf, correo, estado):
    agregar_log(log_path, [nombre_pdf, correo, estado])

def limpiar_pendientes(pendientes):
```

```
            ruta_pendientes = os.path.join(proyecto_dir, "data", "csv",
"pendientes_envio.csv")
    with open(ruta_pendientes, "w", newline='', encoding='utf-8') as archivo:
        escritor = csv.writer(archivo)
        escritor.writerows(pendientes)

def corregir_ruta_windows(ruta):
    ruta = ruta.strip()
    if not ruta:
        return None

    # Soporte para rutas tipo /c/Users/... => C:\Users\...
    if ruta.startswith("/"):
        partes = ruta.lstrip("/").split("/", 1)
        if len(partes) == 2 and len(partes[0]) == 1 and partes[0].isalpha():
            letra_unidad = partes[0].upper()
            resto = partes[1].replace("/", "\\")
            ruta = f"{letra_unidad}:\\" + resto
        else:
            ruta = ruta.replace("/", "\\")
    else:
        ruta = ruta.replace("/", "\\")

    # Normalizar y resolver rutas relativas (../..)
    ruta = os.path.normpath(ruta)

    if ruta in ('\\', ''):
        return None

    return ruta

def procesar_csv(ruta_csv):
    if not os.path.exists(ruta_csv):
        print(f"Archivo CSV no encontrado: {ruta_csv}")
        return

    pendientes_restantes = []

    with open(ruta_csv, newline='', encoding='utf-8') as archivo:
        lector = csv.reader(archivo)
        for fila in lector:
            if len(fila) != 2:
                print(f"Línea inválida: {fila}")
                continue

            ruta_original = fila[0].strip()
            correo = fila[1].strip()
```

```
            ruta_pdf = corregir_ruta_windows(ruta_original)
            print(f"📂 Ruta corregida del PDF: {ruta_pdf}")

            if not ruta_pdf or not os.path.isfile(ruta_pdf):
                print(f"❌ Archivo no encontrado: {ruta_pdf}")
                pendientes_restantes.append(fila)
                continue

            print(f"📨 Enviando PDF: {ruta_pdf} a {correo}")
            # exito = enviar_correo(correo, ASUNTO, CUERPO, ruta_pdf)
            exito = enviar_correo(correo, ASUNTO, CUERPO, [ruta_pdf])
            estado = "exitoso" if exito else "fallido"
            registrar_envio(os.path.basename(ruta_pdf), correo, estado)

            if not exito:
                pendientes_restantes.append(fila)

    limpiar_pendientes(pendientes_restantes)
```

Código fuente: usuarios.ps1

```
param (
    [string]$CsvPath = ".\empleados.csv"
)

$rootPath = Split-Path -Parent $MyInvocation.MyCommand.Path
$rootPath = Split-Path -Parent $rootPath
$rootPath = Split-Path -Parent $rootPath  # Tres niveles arriba para llegar a raíz

$logDir = Join-Path $rootPath "data\logs"

# Crear carpeta de logs si no existe
if (-not (Test-Path $logDir)) {
    New-Item -ItemType Directory -Path $logDir -Force | Out-Null
}

# Función para generar contraseña segura
function Generar-Contrasenia {
    Add-Type -AssemblyName System.Web
    return [System.Web.Security.Membership]::GeneratePassword(12, 3)
}

# Procesar CSV
Import-Csv -Path $CsvPath | ForEach-Object {
```

```
    $nombre = $_.Nombre
    $apellido = $_.Apellido
    $correo = $_.Correo

    # Crear nombre de usuario (ej. jlopez)
    $usuario = ("{0}{1}" -f $nombre.Substring(0,1), $apellido).ToLower()

    # Generar contraseña
    $contrasenia = Generar-Contrasenia
    $securePassword = ConvertTo-SecureString $contrasenia -AsPlainText -Force

    # Verificar existencia
    if (Get-LocalUser -Name $usuario -ErrorAction SilentlyContinue) {
        Write-Output "⚠️ Usuario $usuario ya existe. Saltando..."
        return
    }

    try {
        # Crear usuario local
        New-LocalUser -Name $usuario `
                      -FullName "$nombre $apellido" `
                      -Password $securePassword `
                      -PasswordNeverExpires:$false `
                      -UserMayNotChangePassword:$false `
                      -Description "Usuario temporal creado automáticamente"

        # Agregar al grupo de administradores
        Add-LocalGroupMember -Group "Administradores" -Member $usuario

        # Guardar log individual
        $logFile = Join-Path $logDir "$usuario.log"
        $contenidoLog = @"
Usuario: $usuario
Nombre completo: $nombre $apellido
Correo: $correo
Contraseña: $contrasenia
Fecha de creación: $(Get-Date -Format "yyyy-MM-dd HH:mm:ss")
"@
        $contenidoLog | Out-File -FilePath $logFile -Encoding utf8

        Write-Output "✅ Usuario $usuario creado correctamente."

    } catch {
        Write-Output ("❌ Error al crear {0}: {1}" -f $usuario, $_)

    }
```

}

Fragmento de log diario (log_diario.log)

```
This is pdfTeX, Version 3.141592653-2.6-1.40.25 (MiKTeX 24.1) (preloaded format=pdflatex.fmt)
 restricted \write18 enabled.
entering extended mode

(C:/Users/Josue Vasquez/Desktop/curso_scripting/IRSI-SISAP/core/facturacion/../../data/facturas/factura_1.tex
LaTeX2e <2023-11-01> patch level 1
L3 programming layer <2024-01-04>

(C:\Users\Josue Vasquez\AppData\Local\Programs\MiKTeX\tex/latex/base\article.cls
Document Class: article 2023/05/17 v1.4n Standard LaTeX document class

(C:\Users\Josue Vasquez\AppData\Local\Programs\MiKTeX\tex/latex/base\size10.clo
))
(C:\Users\Josue Vasquez\AppData\Local\Programs\MiKTeX\tex/latex/base\inputenc.sty)
(C:\Users\Josue Vasquez\AppData\Local\Programs\MiKTeX\tex/latex/geometry\geometry.sty
```

Fragmento de log de envíos (log_envios.csv)

```
# LOG DE EVENTOS GENERADO EL 2025-07-23 21:20:05
[2025-07-23 21:20:05] factura_1.pdf | certificacioncriberseguridad@gmail.com | exitoso
[2025-07-23 21:20:09] factura_2.pdf | advinjosuev899@gmail.com | exitoso
[2025-07-23 21:20:11] factura_3.pdf | 17010331@galileo.edu | exitoso
```

Capturas de pantalla

Las capturas de pantalla se encuentran en el repositorio de github https://github.com/Josue19688/IRSI-SISAP esto con el fin de no saturar el documento, asi mismo el codigo correspondiente y sus captuas en docs/capturas