

# MATERIA – SISTEMAS DISTRIBUIDOS

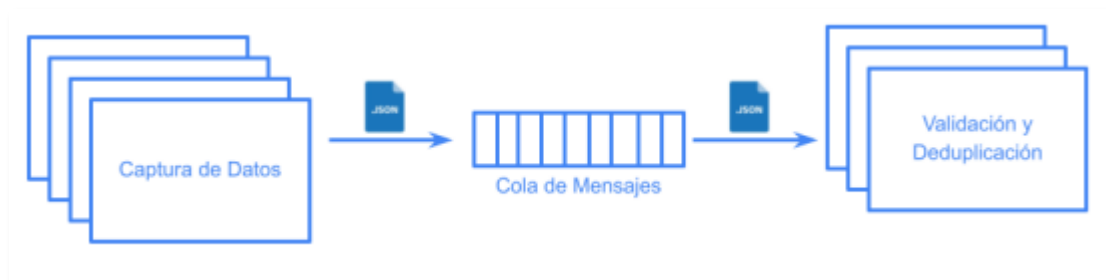
## TAREA-02: Comunicación indirecta

### GRUPO DE TRABAJO:

DANNY MORA MORÁN  
CHRISTIAN FRANCISCO GOMEZ  
CRISTOPHER LARA  
JORGE RIGOBERTO VILLANUEVA

### Descripción general

Con esta tarea se inicia el desarrollo del proyecto integrador. El objetivo de esta tarea es construir un prototipo funcional de los 2 primeros módulos y la implementación de la comunicación entre ellos a través de una cola de mensajes.



### Requerimientos mínimos

1. El módulo de Cola de Mensajes debe ser funcional. Es decir que debe permitir la comunicación efectiva entre los módulos de Captura de Datos y de Validación/Deduplicación.
2. El “Módulo de Captura de Datos” será una aplicación independiente que generará de forma automática los formularios del censo llenos, para lo cual puede llenarlos de forma aleatoria. Cada vez que genere un nuevo formulario, deberá conectarse por medio de la red IP a la cola de mensajes y depositar el formulario codificado en formato JSON.
3. El módulo de Validación/Deduplicación se encargará de obtener los formularios en formato JSON desde la cola de mensajes. Para esta entrega bastará con que obtenga los formularios y realice la validación. La deduplicación no será requerida para esta entrega debido a que el módulo de almacenamiento aún no está disponible.

### Indicaciones sobre la entrega

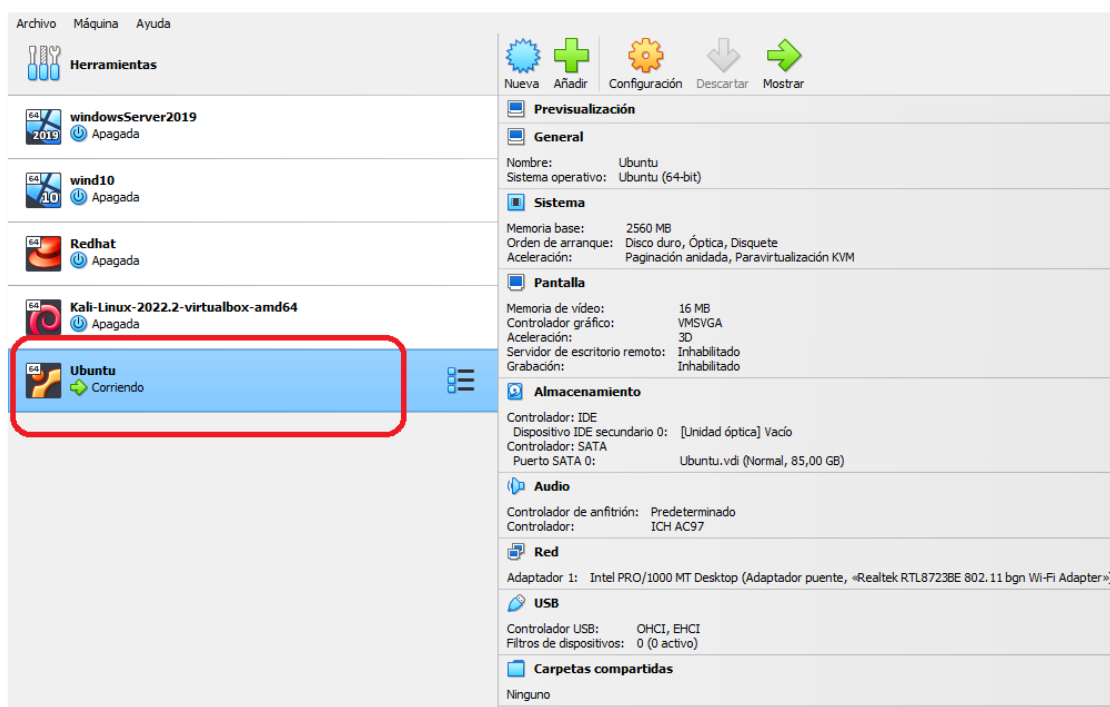
- Se debe entregar un reporte indicando el diseño arquitectónico de los módulos implementados, así como una explicación de las configuraciones utilizadas para la cola de mensaje.
- Para el desarrollo del proyecto deben utilizar un repositorio en GitHub. Puede ser un repositorio privado compartido con los integrantes del grupo y con el docente (usuario

ebozag). Recuerden que en el repositorio solo se sube el código fuente, no se suben los archivos compilados o ejecutables.

- El reporte debe incluir los nombres de los integrantes del grupo, así como la URL del repositorio.
- El reporte también debe mostrar capturas del funcionamiento, con una descripción clara de lo que muestra cada imagen.
- El reporte debe ser subido en un archivo PDF.
- No subir en el Blackboard archivos ZIP, ni código fuente, ni en otros formatos. Únicamente el archivo del reporte en formato PDF.
- El formato del reporte es libre, pero debe mostrar profesionalismo y cuidado en el desarrollo, además, las capturas de pantalla deben ser claras e indicar claramente lo que se está mostrando en cada pantalla.
- No se aceptarán entregas tardías, por lo que se recomienda realizar la tarea con tiempo y subirla al Blackboard lo antes posible.
- Las dudas sobre el trabajo se pueden publicar en el foro.
- También es válido publicar en el foro referencias a los sitios donde encuentren instrucciones que les ayuden a completar la tarea, y de esta manera colaborar con sus compañeros del curso.
- El docente puede solicitar demostraciones del funcionamiento durante las sesiones sincrónicas, además de una explicación del código o del procedimiento realizado

## Procedimiento de instalación

- 1) Se utiliza una máquina virtual con sistema operativo - kernel Ubuntu Linux



## 2) Se instaló plataforma de contenedores DOCKER

```
root@danny:~# apt install docker-ce docker-ce-cli containerd.io
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  docker-buildx-plugin docker-ce-rootless-extras docker-compose-plugin libltdl7 libsllp0 pigz slirp4netns
Suggested packages:
  aufs-tools cgroupfs-mount | cgroup-lite
The following NEW packages will be installed:
  containerd.io docker-buildx-plugin docker-ce docker-ce-cli docker-ce-rootless-extras docker-compose-plugin libltdl7 libsllp0 pigz slirp4netns
0 upgraded, 10 newly installed, 0 to remove and 54 not upgraded.
Need to get 111 MB of archives.
After this operation, 402 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

A continuación, se detalla la versión de Docker que se instaló.

```
root@danny:/home/test/sistemasdistribuidos# docker version
Client: Docker Engine - Community
Version:      24.0.2
API version:  1.43
Go version:   gol.20.4
Git commit:   cb74dfc
Built:        Thu May 25 21:51:00 2023
OS/Arch:      linux/amd64
Context:      default
```

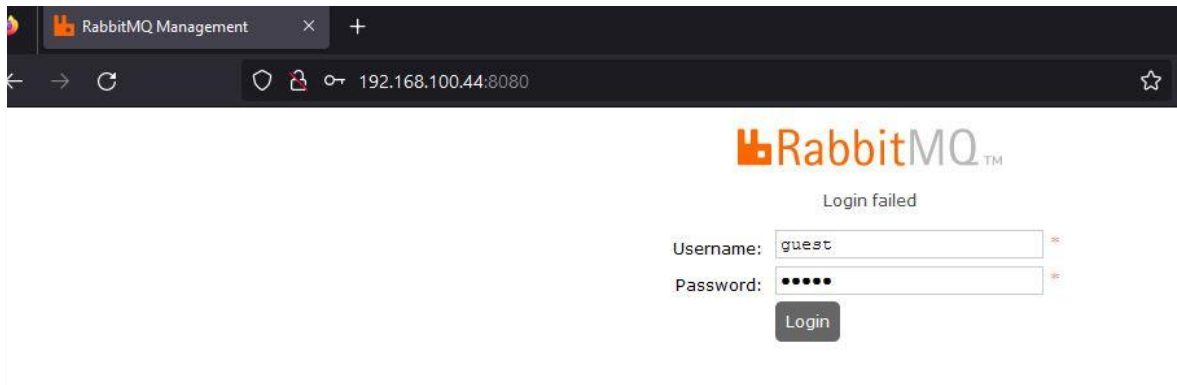
- 3) Se instaló imagen contenedora de RABBITMQ, el cual es un software que implementa el protocolo mensajería de capa de aplicación AMQP (Advanced Message Queuing Protocol), el cual está enfocado en la comunicación de mensajes asíncronos con garantía de entrega, a través de confirmaciones de recepción de mensajes desde el broker al productor y desde los consumidores al broker.

```
root@danny:~# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
hello-world   latest    9c7a54a9a43c   6 weeks ago   13.3kB

root@danny:~# docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES

root@danny:~#
root@danny:~#
root@danny:~#
root@danny:~# docker pull rabbitmq
Using default tag: latest
latest: Pulling from library/rabbitmq
3f94e4e483ea: Downloading [=====>] 19.31MB/30.43MB
db610c556cbb: Download complete
046b2545f902: Download complete
396c21d32d40: Downloading [=====>] 5.38MB/55.17MB
685bcbcad0b8: Download complete
c414cb9e1350: Downloading [=====>] 1.475MB/20.43MB
be93a5592724: Waiting
8733cec7a1ec: Waiting
9d03529088ec: Waiting
c37a463dc99c: Waiting
```

La administración de RabbitMq es vía web a través del puerto TCP/8080



- 4) Se instaló una imagen contenedora de un servicio web en el cual se instaló un servidor web Flask

```
root@danny:/home/test/sistemasdistribuidos# docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
httpd         2.4       ad303d7f80f9   8 days ago    168MB
rabbitmq      3-management fa6f2153e5c0   2 weeks ago   256MB
hello-world   latest    9c7a54a9a43c   7 weeks ago   13.3kB
root@danny:/home/test/sistemasdistribuidos#
```

- 5) Se instalaron paquetes y una serie de librerías necesarias, se adjunta un ejemplo de unos paquetes instalados.

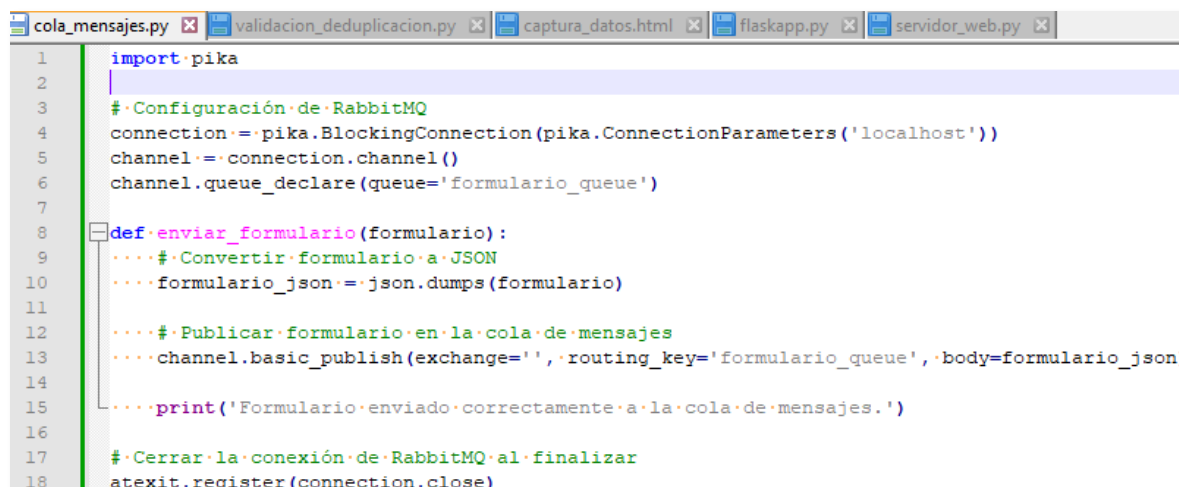
```
pip install pika flask
```

## Explicación de la arquitectura implementada.

Se detalla la implementación de un módulo de cola de mensajes y un módulo de validación/deduplicación en Python, utilizando RabbitMQ para la comunicación entre los módulos.

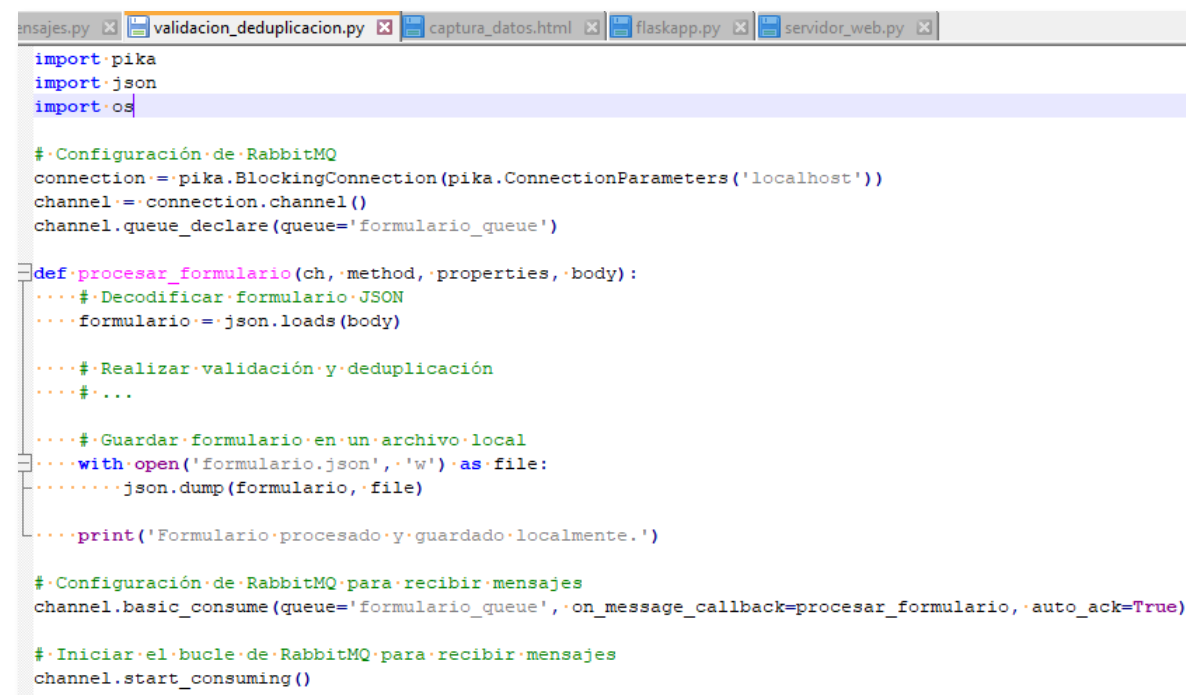
También se incluye una aplicación web en HTML para el módulo de captura de datos.

- A continuación, se muestra el código para el módulo de Cola de mensajes.



```
1 import pika
2
3 # Configuración de RabbitMQ
4 connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
5 channel = connection.channel()
6 channel.queue_declare(queue='formulario_queue')
7
8 def enviar_formulario(formulario):
9     # Convertir formulario a JSON
10     formulario_json = json.dumps(formulario)
11
12     # Publicar formulario en la cola de mensajes
13     channel.basic_publish(exchange='', routing_key='formulario_queue', body=formulario_json)
14
15     print('Formulario enviado correctamente a la cola de mensajes.')
16
17 # Cerrar la conexión de RabbitMQ al finalizar
18 atexit.register(connection.close)
```

- A continuación, se muestra el código validación/deduplicación



```
import pika
import json
import os

# Configuración de RabbitMQ
connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.queue_declare(queue='formulario_queue')

def procesar_formulario(ch, method, properties, body):
    # Decodificar formulario JSON
    formulario = json.loads(body)

    # Realizar validación y deduplicación
    # ...

    # Guardar formulario en un archivo local
    with open('formulario.json', 'w') as file:
        json.dump(formulario, file)

    print('Formulario procesado y guardado localmente.')

# Configuración de RabbitMQ para recibir mensajes
channel.basic_consume(queue='formulario_queue', on_message_callback=procesar_formulario, auto_ack=True)

# Iniciar el bucle de RabbitMQ para recibir mensajes
channel.start_consuming()
```

- A continuación, se muestra el código html de la interfaz web para el módulo de captura de datos (HTML)

```
mensajes.py x validacion_deduplicacion.py x captura_datos.html x flaskapp.py x
<!DOCTYPE html>
<html>
<head>
<title>Captura de Datos</title>
</head>
<body>
<h1>Captura de Datos</h1>
<form action="/generar_formulario" method="POST">
<button type="submit">Generar formulario</button>
</form>
</body>
</html>
```

- A continuación, se detalla el código en Python usado por el servidor web FLASK.

```
mensajes.py x validacion_deduplicacion.py x captura_datos.html x flaskapp.py x servidor_web.py x
from flask import Flask, render_template, request
from cola_mensajes import enviar_formulario

app = Flask(__name__)

@app.route('/generar_formulario', methods=['POST'])
def generar_formulario():
    # Generar formulario aleatorio
    formulario = {
        'nombre': 'John Doe',
        'edad': 30,
        'direccion': '123 Main St',
        # Agrega más campos según tus necesidades
    }

    # Enviar formulario a la cola de mensajes
    enviar_formulario(formulario)

    return 'Formulario generado y enviado correctamente.'

if __name__ == '__main__':
    app.run(host='192.168.100.44', port=8003, debug=True)
```

### Procedimiento para ejecución de las pruebas.

Se guardó cada parte del código en archivos separados: validacion\_deduplicacion.py, captura\_datos.html, cola\_mensajes.py y servidor\_web.py.

A continuación, sigue estos pasos para ejecutar el sistema:

1. Se ejecutó en primera instancia el módulo de validación/deduplicación ejecutando el archivo validacion\_deduplicacion.py. Este módulo estuvo esperando recibir formularios en formato JSON desde la cola de mensajes.
2. Configuración de un servidor web Flask para servir el archivo HTML captura\_datos.html. Se creó un archivo llamado servidor\_web.py y se le copió el código correspondiente. Posteriormente en el servidor web se ejecutó python servidor\_web.py.
3. Se accedió a la interfaz web a través de un navegador visitando <http://192.168.100.44:8003> y luego se dio clic en el botón "Generar formulario". Esto envió una solicitud POST al servidor web.
4. El controlador de la ruta /generar\_formulario invocó la función enviar\_formulario del módulo de cola de mensajes, pasando el formulario generado como argumento. Esto envió el formulario a la cola de mensajes.
5. El módulo de validación/deduplicación recibió el formulario desde la cola de mensajes, y posteriormente lo procesó y guardó localmente en formato JSON.

### Evidencias de las pruebas realizadas.

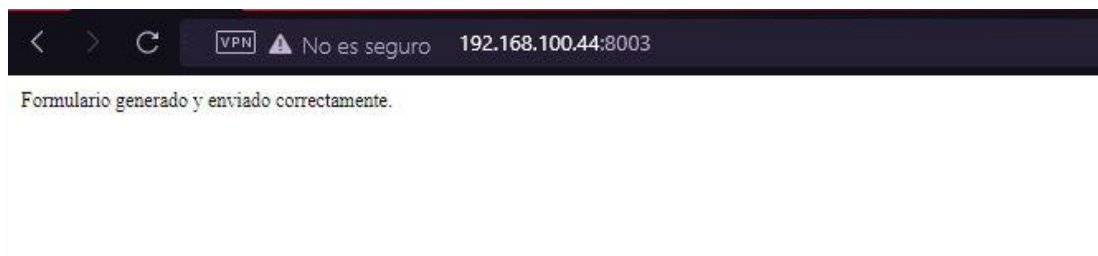
- a) Se ejecutó archivo validación\_deduplicación.py

```
root@danny:/home/test/sistemasdistribuidos# python3 validacion_deduplicacion.py
```

- b) Se ejecutó el servicio web a través del archivo captura\_datos.html

```
root@danny:/home/test/sistemasdistribuidos#  
root@danny:/home/test/sistemasdistribuidos# python3 servidor_web.py  
* Serving Flask app 'servidor_web'  
* Debug mode: on  
WARNING: This is a development server. Do not use it in a production deployment.  
Use a production WSGI server instead.  
* Running on http://127.0.0.1:8001  
Press CTRL+C to quit  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 111-028-891
```

- c) Se accede a la interfaz web y se genera el formulario



- d) Se accede a la consola de RabbitMQ y se validan las colas de mensajes durante el proceso de consultas.

**RabbitMQ** RabbitMQ 3.12.0 Erlang 25.3.2.2

Overview **Connections** Channels Exchanges Queues Admin

Connections

▼ All connections (3)

Pagination

Page 1 of 1 - Filter:  ☐ Regex ?

Name	User name	State	SSL / TLS	Protocol	Channels	From client	To client
172.17.0.1:36226	guest	running	-	AMQP 0-9-1	1	2 B/s	0 B/s
172.17.0.1:52712	guest	running	-	AMQP 0-9-1	1	0 B/s	0 B/s
172.17.0.1:52716	guest	running	-	AMQP 0-9-1	1	0 B/s	0 B/s

HTTP API Documentation Tutorials New releases Commercial edition Commercial support Google Group Discord Slack Plugins GitHub

**RabbitMQ** RabbitMQ 3.12.0 Erlang 25.3.2.2

Overview **Connections** Channels Exchanges **Queues** Admin

Queued messages last minute ?

6.0 k  
4.0 k  
2.0 k  
0.0 k

19:48:30 19:48:40 19:48:50 19:49:00 19:49:10 19:49:20

Ready 5,848  
Unacked 0  
Total 5,848

Message rates last minute ?

1.0 /s  
0.0 /s

19:48:30 19:48:40 19:48:50 19:49:00 19:49:10 19:49:20

Publish 0.00/s

Details

Features queue storage version: 1

State idle

Consumers 0

Consumer capacity ? 0%

Messages ?

Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
5,848	5,848	0	1	0	5,847

Message body bytes ?

343 KiB	343 KiB	0 B	60 B	0 B	343 KiB

Process memory ? 16 KiB

**RabbitMQ** RabbitMQ 3.12.0 Erlang 25.3.2.2

Overview **Connections** Channels Exchanges Queues Admin

Connection 172.17.0.1:47916 -> 172.17.0.2:5072

▼ Overview

Data rates last minute

2 B/s  
1 B/s  
0 B/s

19:47:50 19:48:00 19:48:10 19:48:20 19:48:30 19:48:40

From client 0 B/s  
To client 2 B/s

Details

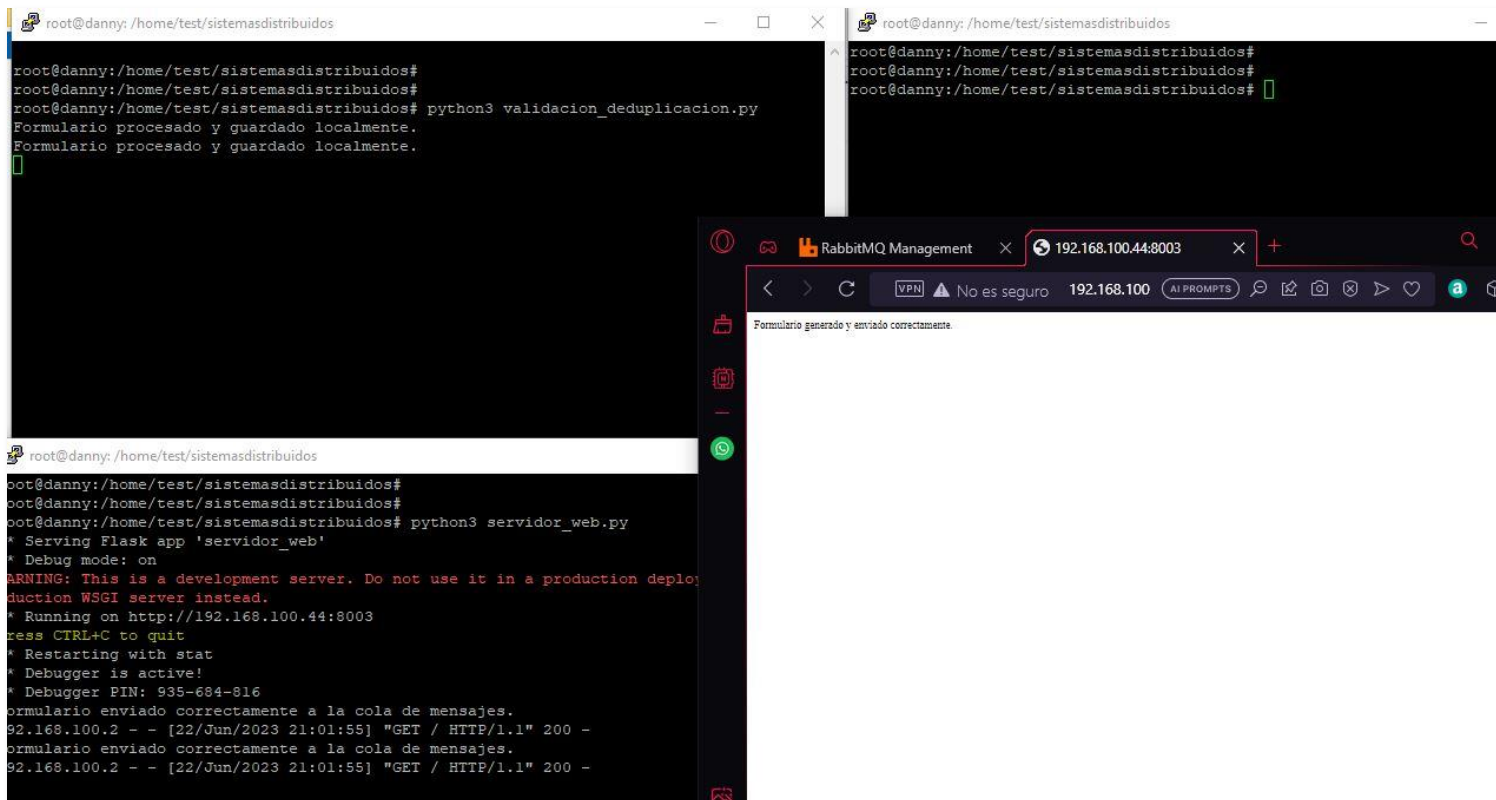


e) Se genera y se almacena formulario

```
root@danny:/home/test/sistemasdistribuidos# cat formulario.json
{"nombre": "John Doe", "edad": 30, "direccion": "123 Main St"}root@danny:/home/test/siste
```

```
-rw-r--r-- 1 root root 62 Jun 22 20:42 formulario.json
```

f) Evidencia general



A continuación, se adjunta enlace del repositorio en GitHub, en el que se encuentran los códigos utilizados en esta tarea.

[https://github.com/diem1285/TAREA-2-Comunicacion\\_indirecta.git](https://github.com/diem1285/TAREA-2-Comunicacion_indirecta.git)