INFORME DE LABORATORIO

(formato estudiante)

INTEGRANTE (s): ARACAYO MAMANI JHON MARCO	**NOTA:**	

INFORMACIÓN BÁSICA					
TÍTULO DE LA PRÁCTICA:	**Integración de Sistemas y Flujos de Trabajo**				
NÚMERO DE PRÁCTICA:	*8*	**AÑO LECTIVO:**	*2024*	**NRO. SEMESTRE:**	*VI*
FECHA DE PRESENTACIÓN	*22/06/24*				

SOLUCIÓN Y RESULTADOS

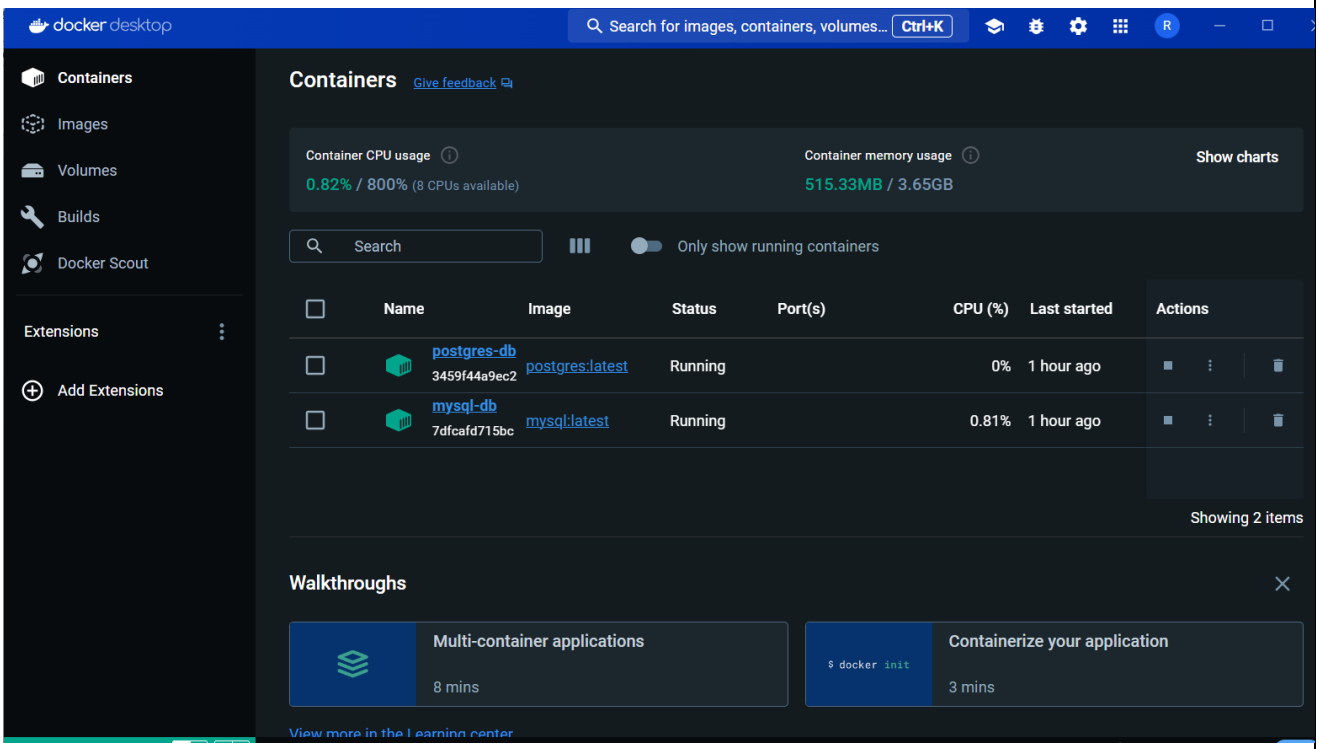
Parte 1: Configuración del Entorno (30 minutos)

1.- Instalacion del docker

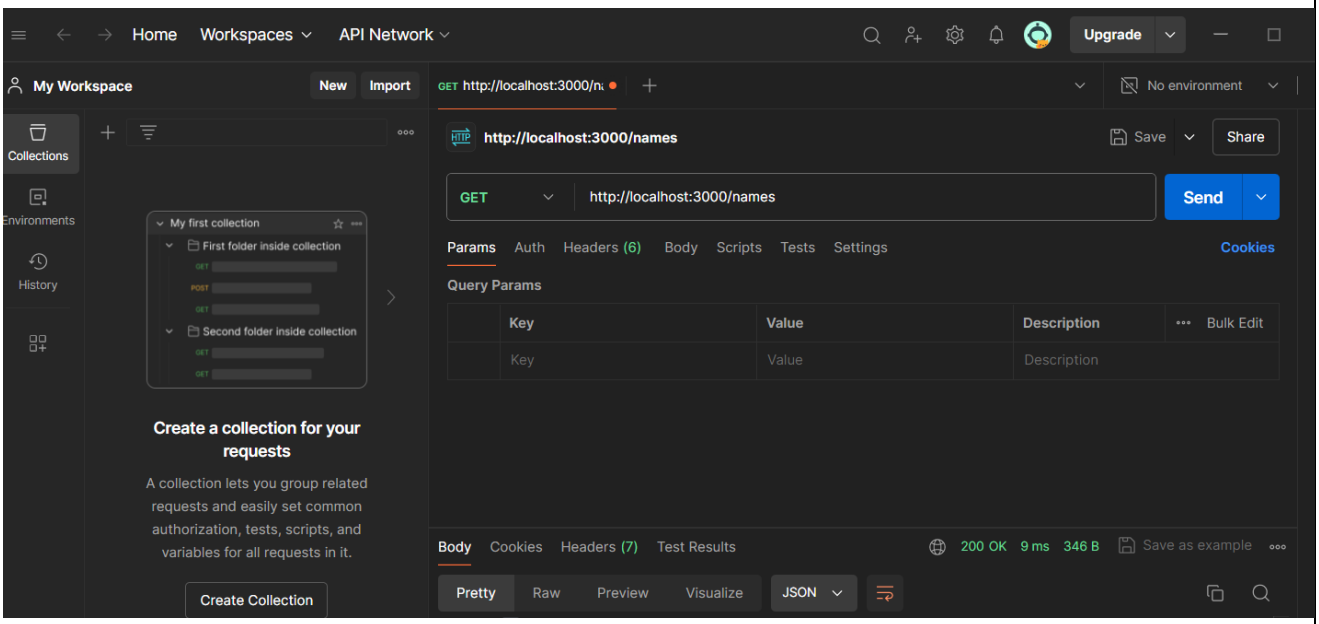
Instalamos el docker correctamente y lo inicializacmos con los comandos correspondientes:

docker run --name mysql-db -e MYSQL_ROOT_PASSWORD=root -d mysql:latest
docker run --name postgres-db -e POSTGRES_PASSWORD=root -d postgres:latest

I.



2.- Instalación de Postman:



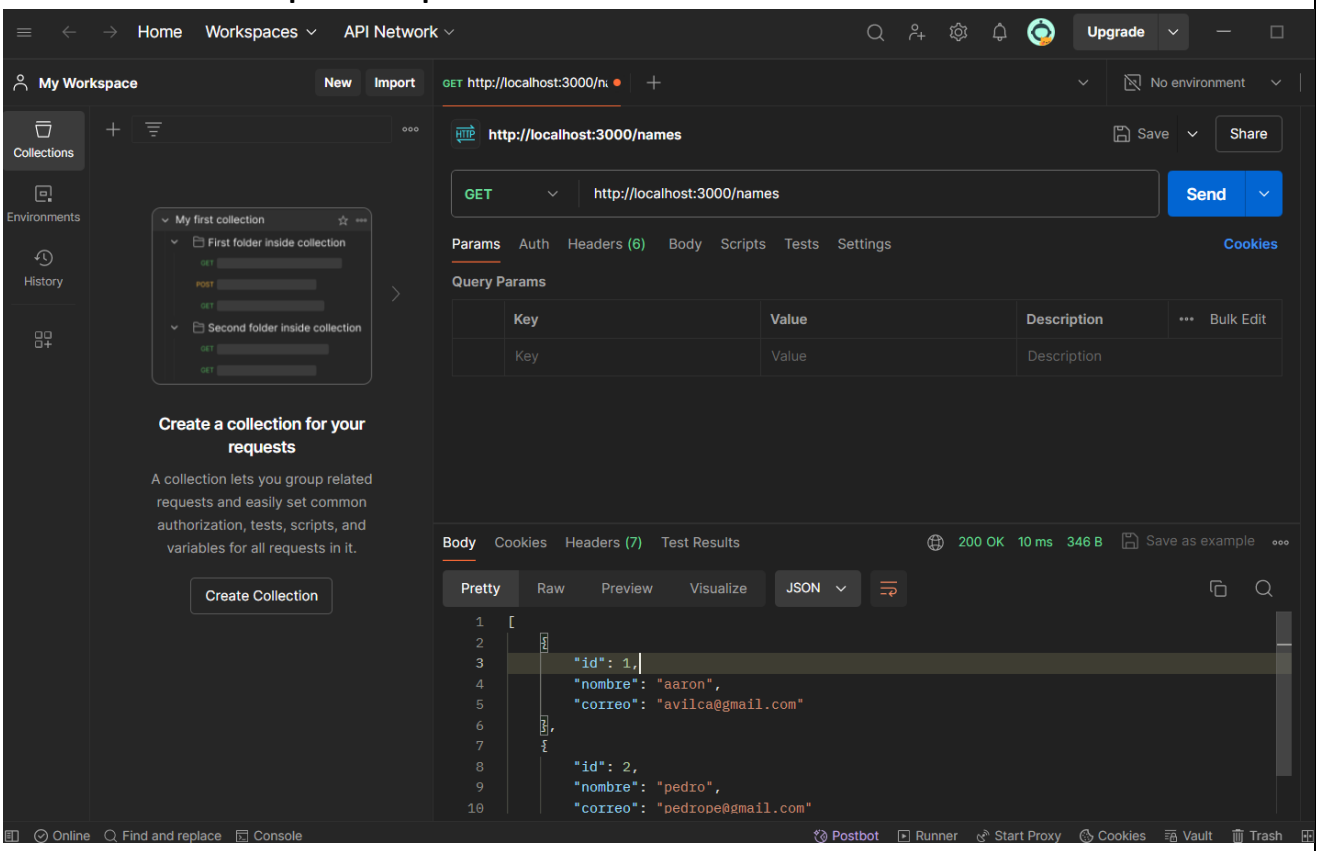
II. Creación y Consumo de APIs (

En esta occasion usaremos node.js para crear una API basica que pueda interactuar con la base de datos, iniciando dando los datos de mi base de datos.

```
const express = require('express');
const mysql = require('mysql');

const app = express();

// Configurar la conexión a la base de datos MySQL
const db = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: '',
    database: 'usuarios'
});

db.connect((err) => {
    if (err) {
        console.error('Error connecting to MySQL:', err);
        return;
    }
    console.log('Connected to MySQL database');
});

// Definir una ruta para obtener datos desde la base de datos
app.get('/names', (req, res) => {
    db.query('SELECT * FROM names', (err, results) => {
        if (err) {
            console.error('Error executing query:', err);
            res.status(500).json({ error: 'Failed to retrieve data', details:
err.message });
            return;
        }
        res.json(results);  // Devolver los resultados en formato JSON
    });
});

app.listen(3000, () => {  // Iniciar el servidor en el puerto 3000
    console.log('Server started on port 3000');
});
```

III. Probamos la API en el postman con nuestros datos ingresados:

Corroboramos con el postman que nuestra API funcione correctamente:



IV. Implementacion de un flujo de trabajo basico:

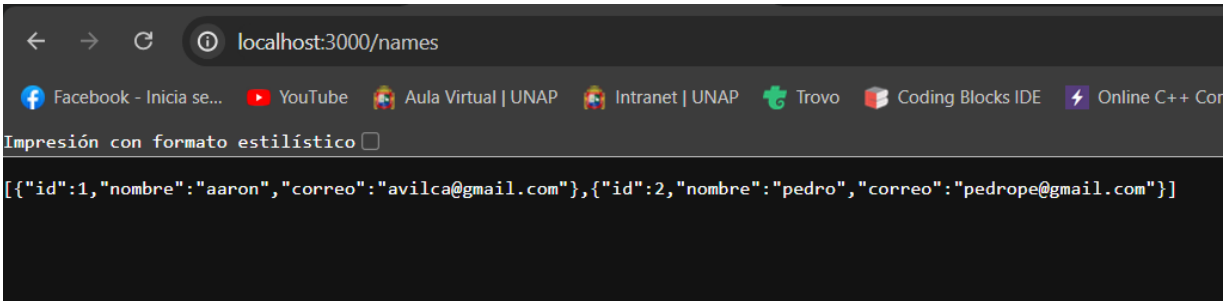
- Ejemplo de codigo de automatizacion de tareas

```
import requests
import csv
import time

while True:  # Bucle infinito para ejecutar la tarea repetidamente
    try:
        response = requests.get('http://localhost:3000/names')  # Hacer una solicitud
GET a la API
        response.raise_for_status()  # Verificar si la solicitud fue exitosa (código de
estado 200)
        try:
            data = response.json()  # Obtener los datos en formato JSON
        except requests.exceptions.JSONDecodeError:
            print("Error: No se pudo decodificar la respuesta JSON.")
            data = []

        if data:  # Verificar si data no está vacío
```

```
            with open('data.csv', mode='w', newline='') as file:  # Abrir un archivo
CSV en modo escritura
                writer = csv.writer(file)  # Crear un escritor de CSV
                # Escribir los encabezados de las columnas
                writer.writerow(['id', 'nombre', 'correo'])
                # Iterar sobre los datos obtenidos y escribir cada fila en el archivo
CSV
                for row in data:
                    writer.writerow([row['id'], row['nombre'], row['correo']])
            print("Datos guardados correctamente en data.csv")
        else:
            print("No se recibieron datos válidos de la API.")
    except requests.exceptions.RequestException as e:
        print(f"Error en la solicitud HTTP: {e}")

    time.sleep(3600)  # Esperar una hora antes de la siguiente ejecución
```

V. Evaluacion y documentacion

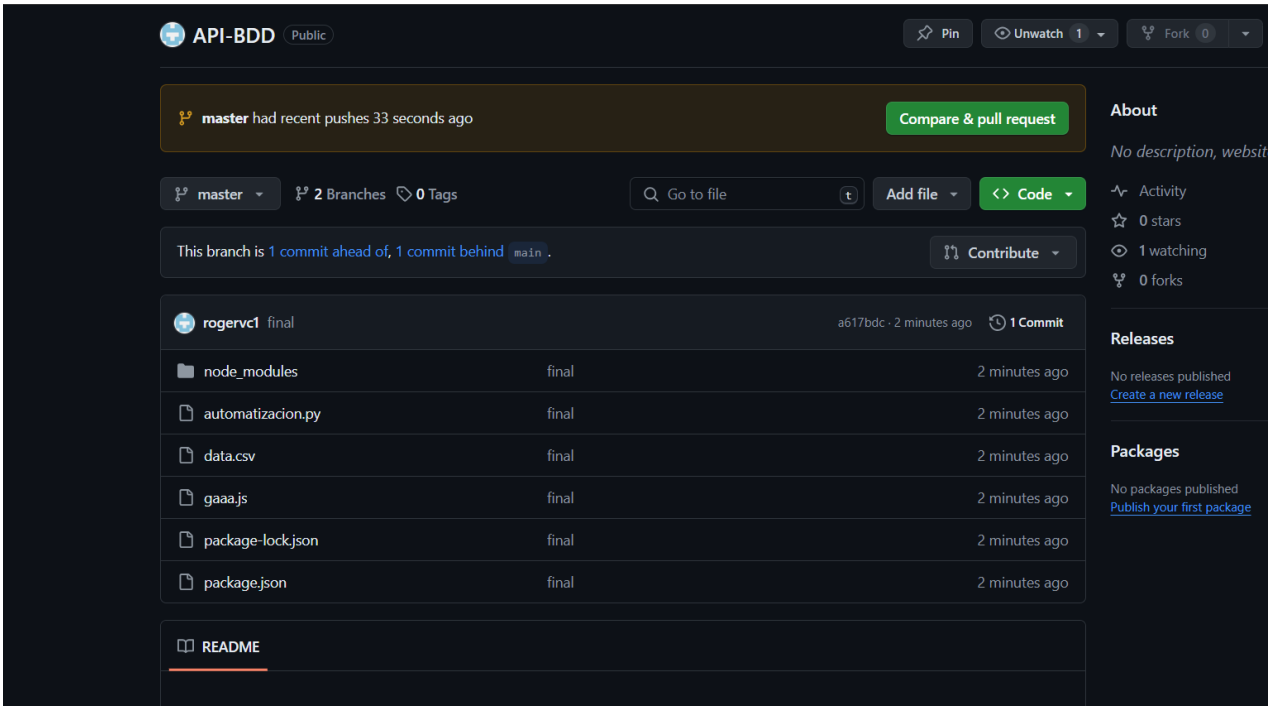
Gracias al postman sabemos que nuestra API funciona correctamente, ahora ejecutaremos el codigo python de automatizacion para crear un archivo .CSV que guarde los datos que obtenga de la API:



VI.

Link del repositorio del proyecto:
https://github.com/tamoil13/practica8.git

VII.



REFERENCIAS Y BIBLIOGRAFÍA