INFORME DE LABORATORIO

(formato estudiante)

INTEGRANTE (s): Mayta Yujra Efrain Saul	NOTA:	

INFORMACIÓN BÁSICA					
TÍTULO DE LA PRÁCTICA:	Integración de Sistemas y Flujos de Trabajo				
NÚMERO DE PRÁCTICA:	*8*	AÑO LECTIVO:		NRO. SEMESTRE:	*VI*
FECHA DE PRESENTACIÓN	*20/06/2024*				

SOLUCIÓN Y RESULTADOS

I. Introducción y objetivos

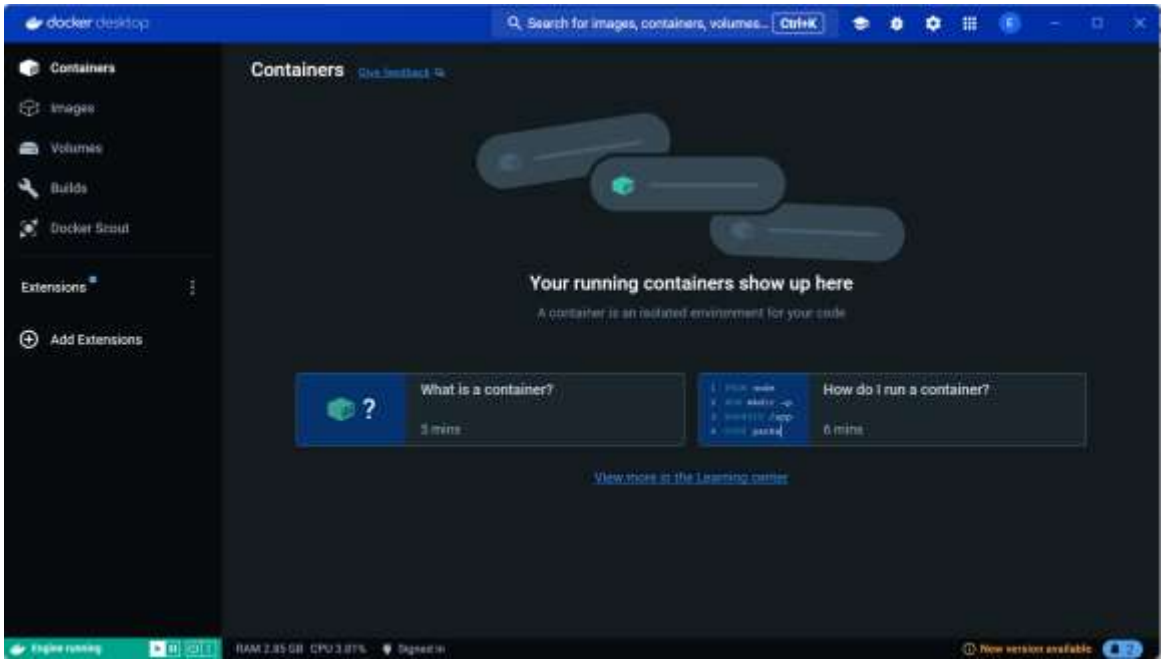
En este proyecto, hemos implementado una API RESTful que interactúa con una base de datos MySQL utilizando Docker para la configuración del entorno. También se incluye un script de automatización que consulta la API a intervalos regulares y guarda los datos obtenidos en un archivo CSV. El objetivo principal es demostrar la capacidad de configurar un entorno de desarrollo utilizando Docker, crear una API para interactuar con una base de datos y automatizar el flujo de trabajo de recolección de datos.

Objetivos:

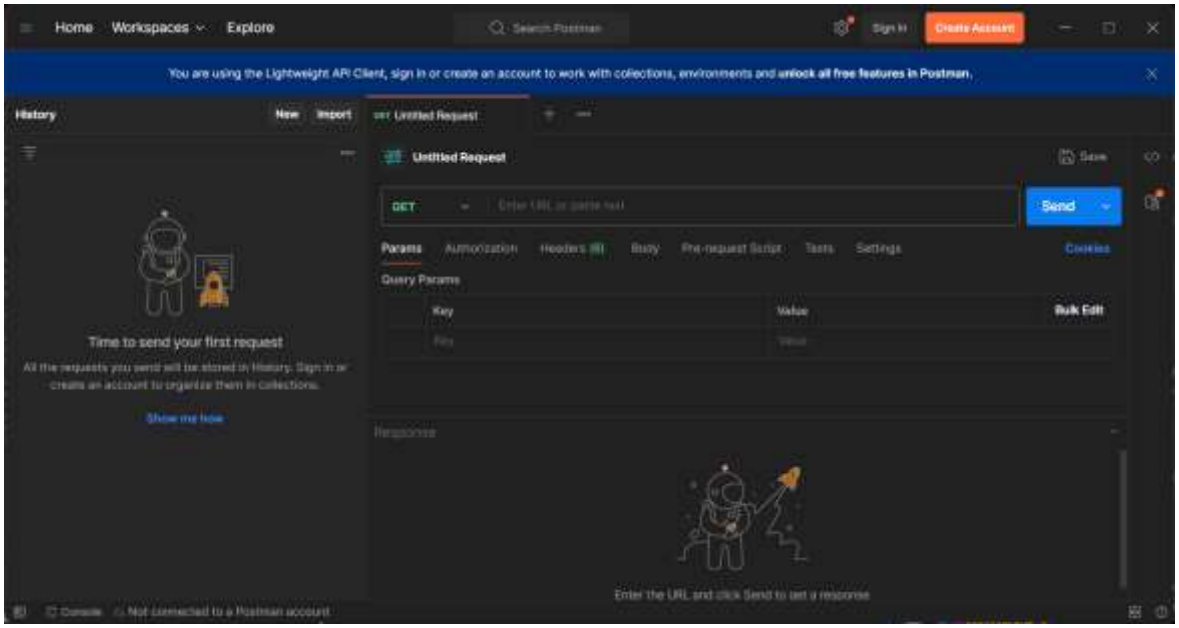
- Configurar Docker en un entorno Windows.
- Levantar una instancia de MySQL utilizando Docker.
- Crear una API RESTful básica usando Flask (Python) y Express (Node.js).
- Probar la API utilizando Postman.
- Implementar un script en Python para automatizar la consulta a la API y el almacenamiento de datos en un archivo CSV.
- Documentar el proceso completo de configuración, implementación y pruebas.

II. Procedimientos detallados

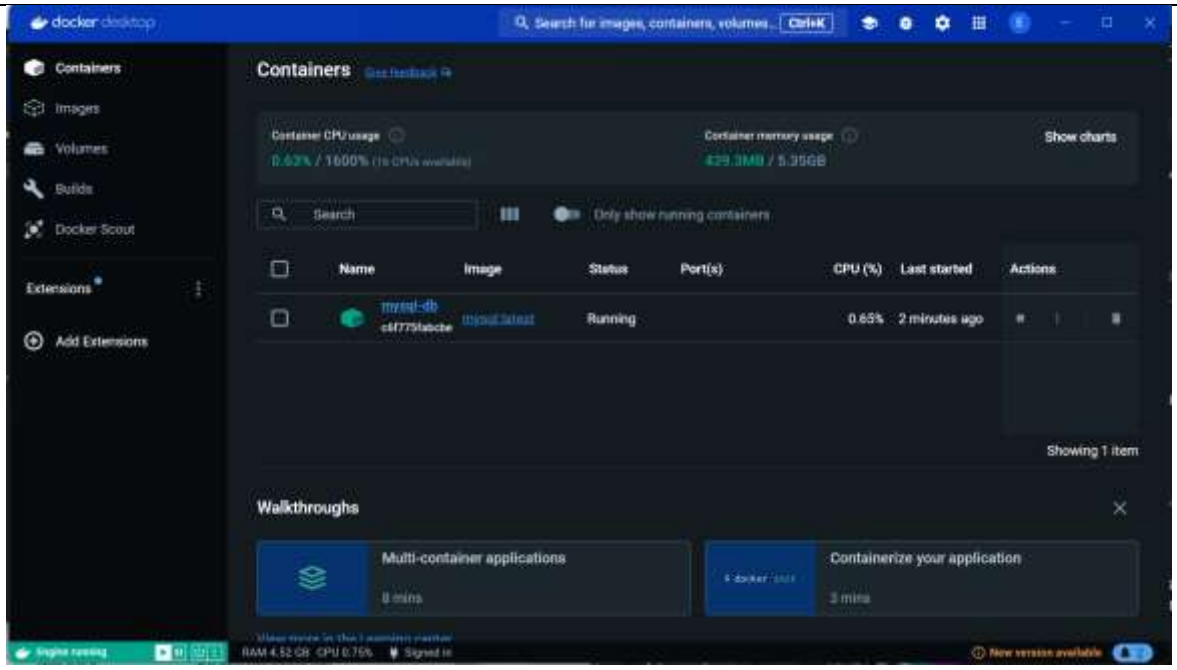
1. Instalación de Docker en Windows.

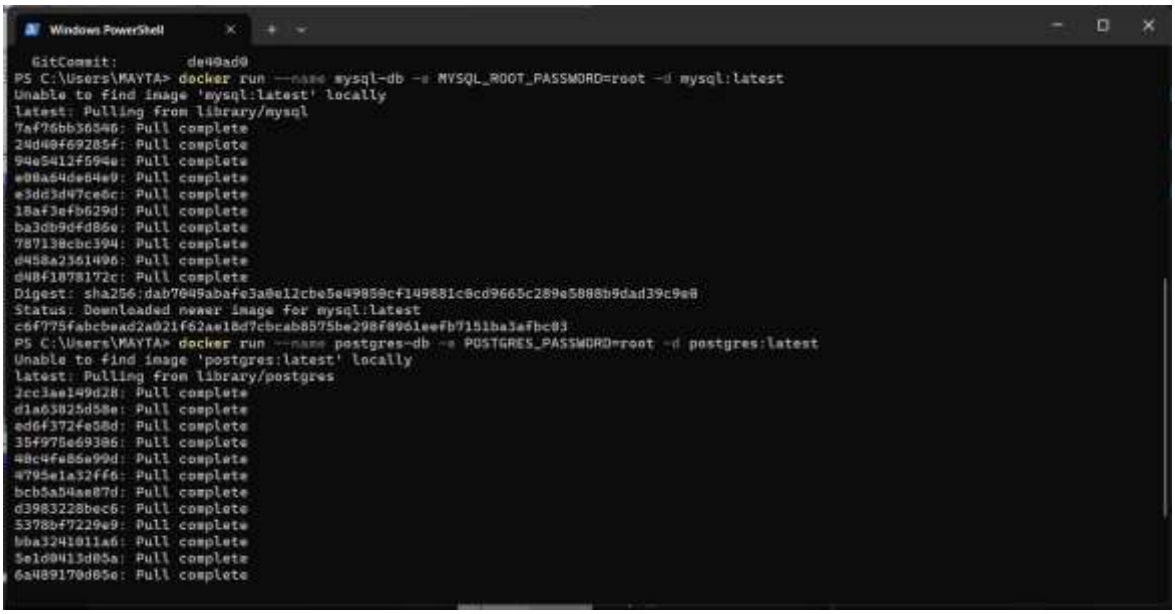


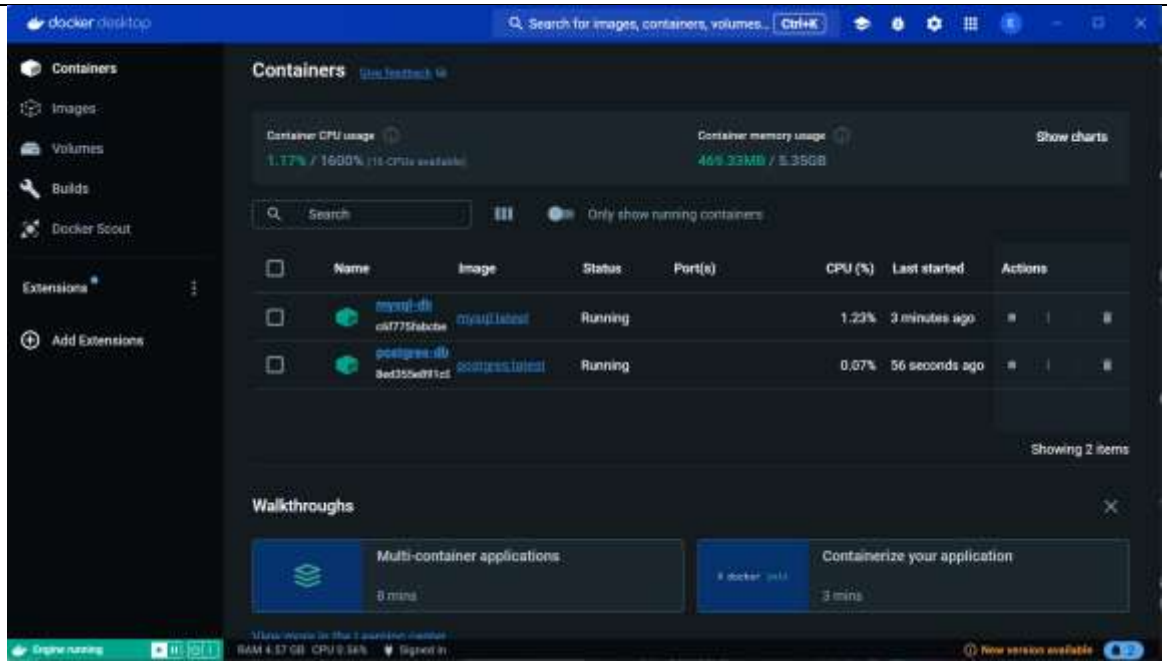
2. **Instalación de Postman.**



3. **Configuración de MySQL con Docker:**
   docker run --name mysql-db -e MYSQL_ROOT_PASSWORD=root -d mysql:latest


docker desktop
Containers
Images
Volumes
Builds
Docker Scout
Extensions
Add Extensions
Containers
Container CPU usage
Container memory usage
Show charts
Search
Only show running containers
Name
Image
Status
Port(s)
CPU (%)
Last started
Actions
mysql-db
Running
0.65%
2 minutes ago
Showing 1 item
Walkthroughs
Multi-container applications
8 mins
Containerize your application
3 mins


Windows PowerShell
GitCommit: de40ad0
PS C:\Users\MAYTA> docker run --name mysql-db -e MYSQL_ROOT_PASSWORD=root -d mysql:latest
Unable to find image 'mysql:latest' locally
latest: Pulling from library/mysql
7af76bb36546: Pull complete
24d40f69285f: Pull complete
94e5412f594e: Pull complete
e80a54de64e9: Pull complete
e3dd3d47ce6c: Pull complete
18af3efb629d: Pull complete
ba3db9dfd86e: Pull complete
787138cbc394: Pull complete
d458a2361496: Pull complete
d48f1878172c: Pull complete
Digest: sha256:dab7049abafe3a0e12cbe5e49850cf149881c0cd9665c289e5888b9dad39c9e8
Status: Downloaded newer image for mysql:latest
c6f775fabcbead2a021f62ae18d7cbcab8575be298f8961eefb7151ba3afbc03
PS C:\Users\MAYTA> docker run --name postgres-db -e POSTGRES_PASSWORD=root -d postgres:latest
Unable to find image 'postgres:latest' locally
latest: Pulling from library/postgres
2cc3ae149d28: Pull complete
d1a63825d58e: Pull complete
ed6f372fe58d: Pull complete
35f975e69386: Pull complete
48c4fe86e99d: Pull complete
4795e1a32ff6: Pull complete
bcb5a54ae87d: Pull complete
d3983228bec6: Pull complete
5378bf7229e9: Pull complete
bba3241011a6: Pull complete
5e1d0413d05a: Pull complete
6a489170d85e: Pull complete



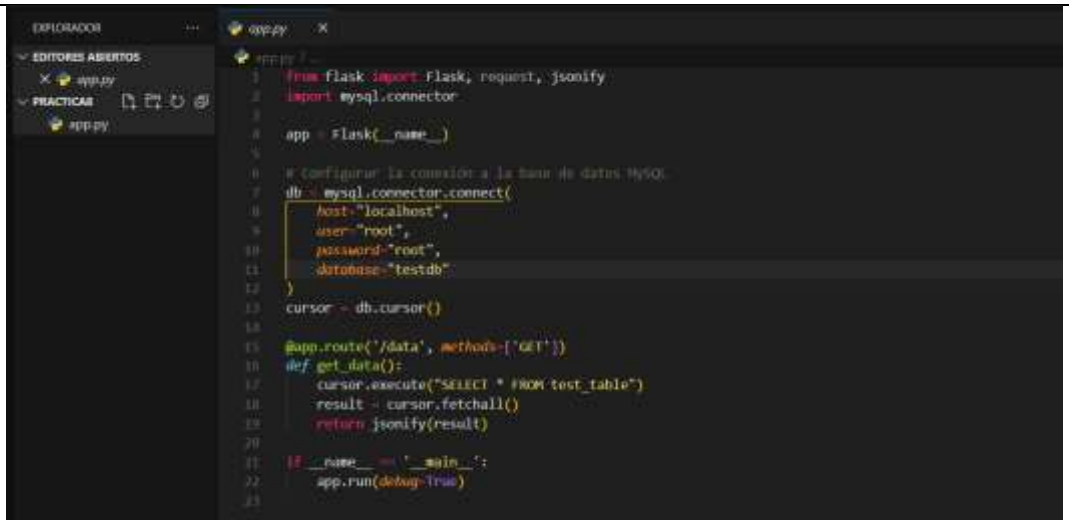
4. Crear una API RESTful básica:

1. Instalar Flask y MySQL Connector:

```
pip install flask mysql-connector-python
```

2. Crear un archivo `app.py` con el siguiente contenido:

```python
from flask import Flask, request, jsonify
import mysql.connector

app = Flask(__name__)

# Configurar la conexión a la base de datos MySQL
db = mysql.connector.connect(
    host="localhost",
    user="root",
    password="root",
    database="testdb"
)
cursor = db.cursor()

@app.route('/data', methods=['GET'])
def get_data():
    cursor.execute("SELECT * FROM test_table")
    result = cursor.fetchall()
    return jsonify(result)

if __name__ == '__main__':
    app.run(debug=True)
```



3. Ejecutar el script de Flask:
python app.py

Express (Node.js):

1. Instalar Node.js y npm desde Node.js.
2. Crear un proyecto de Node.js e instalar las dependencias:

```
npm init -y
npm install express mysql
```

```
C:\Users\MAYTA>npm init -y
Wrote to C:\Users\MAYTA\package.json:

{
  "dependencies": {
    "react-facebook-login": "^4.1.1"
  },
  "name": "mayta",
  "version": "1.0.0",
  "main": "index.js",
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}



C:\Users\MAYTA>npm install express mysql

added 76 packages in 11s

12 packages are looking for funding
  run `npm fund` for details
npm notice
npm notice New minor version of npm available! 10.2.0 -> 10.8.1
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.8.1
npm notice Run npm install -g npm@10.8.1 to update!
npm notice
```

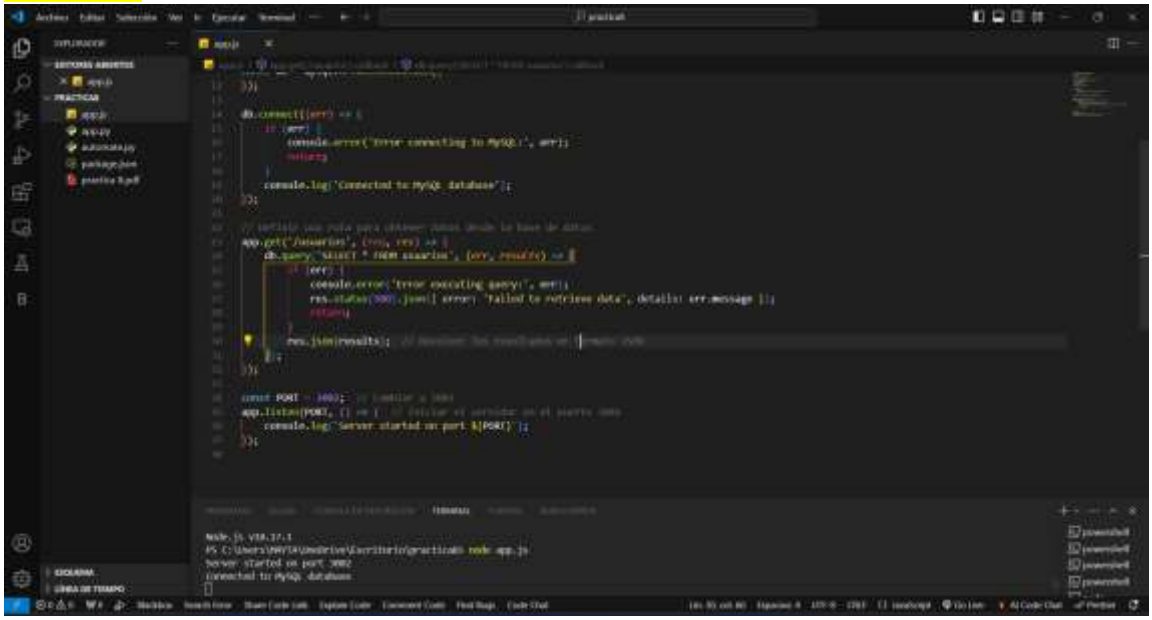
3. Crear un archivo `app.js` con el siguiente contenido:

```
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: 'laboratorio'
});

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

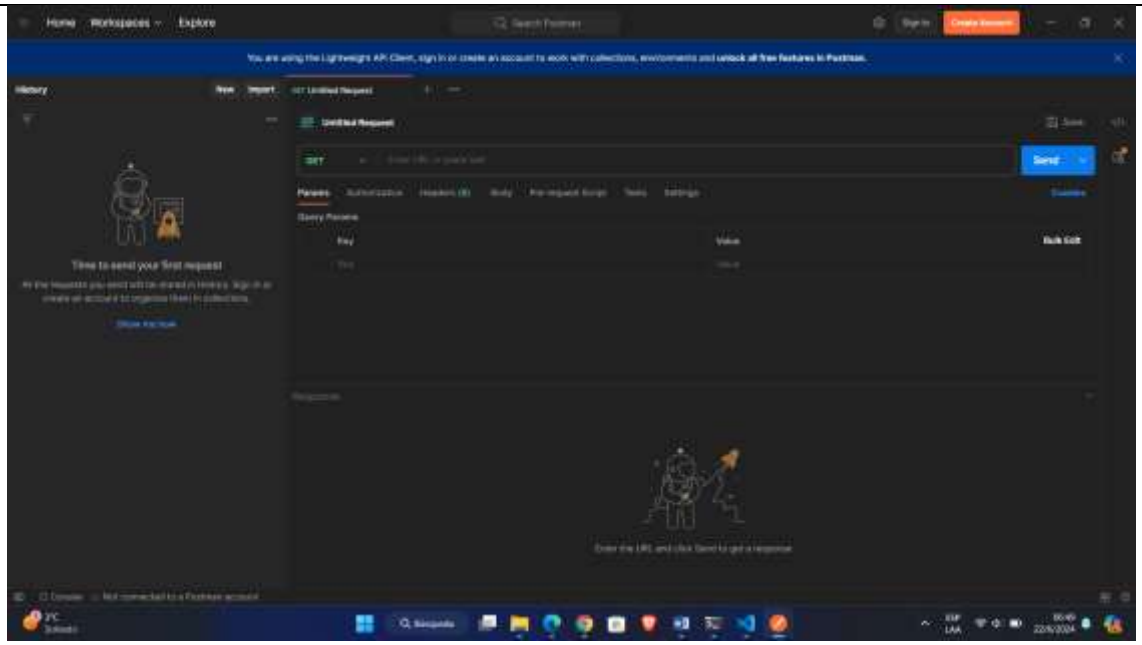
```
// Definir una ruta para obtener datos desde la base de datos
app.get('/usuarios', (req, res) => {
    db.query('SELECT * FROM usuarios', (err, results) => {
        if (err) {
            console.error('Error executing query:', err);
            res.status(500).json({ error: 'Failed to retrieve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

const PORT = 3002;  // Cambiar a 3001
app.listen(PORT, () => {  // Iniciar el servidor en el puerto 3001
    console.log(`Server started on port ${PORT}`);
});
```

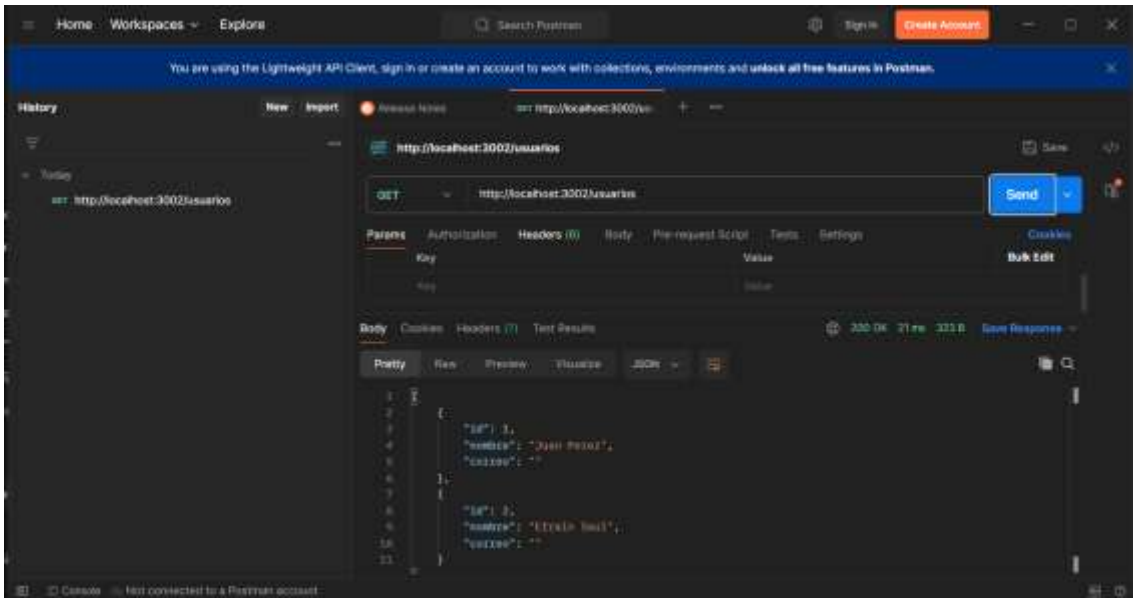
4. Ejecutar el script de Node.js:

node app.js



5. Probar la API con Postman:





6. Automatización de tareas:

Crear un archivo automate.py con el siguiente contenido:

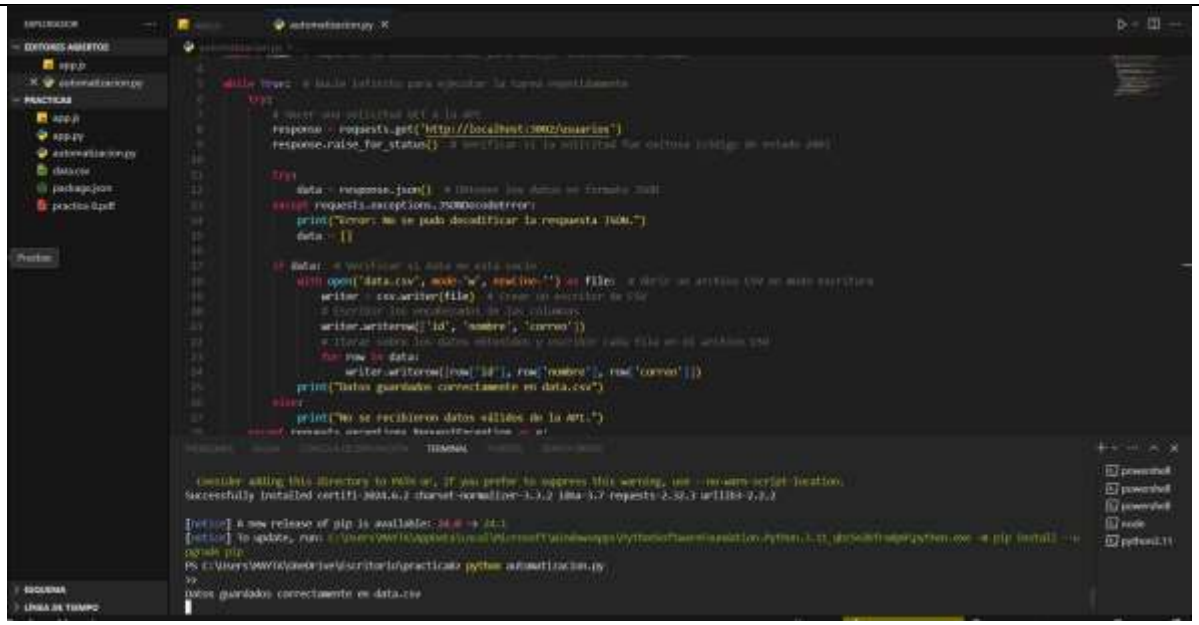
```
import requests  # Importar la biblioteca requests para hacer solicitudes HTTP
import csv  # Importar la biblioteca csv para trabajar con archivos CSV
import time  # Importar la biblioteca time para manejar intervalos de tiempo

while True:  # Bucle infinito para ejecutar la tarea repetidamente
    try:
```

```
        # Hacer una solicitud GET a la API
        response = requests.get('http://localhost:3002/usuarios')
        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
```

Ejecutar el script de automatización:

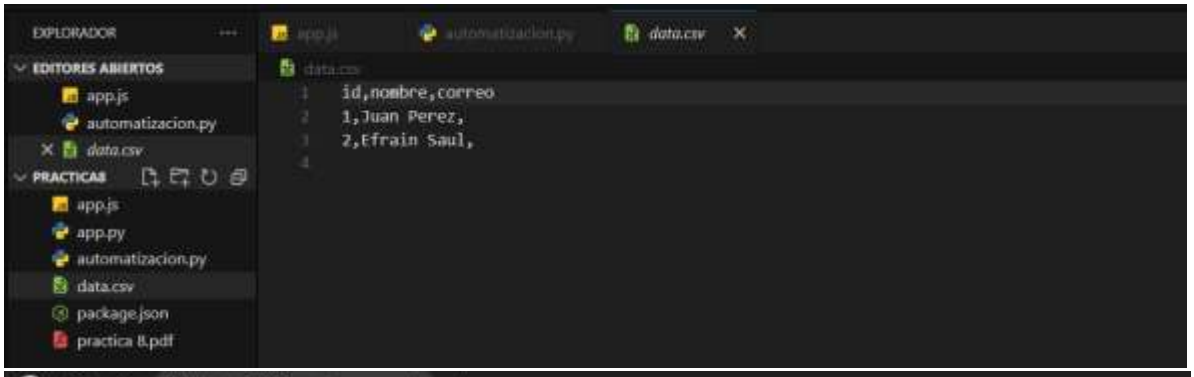
python automate.py

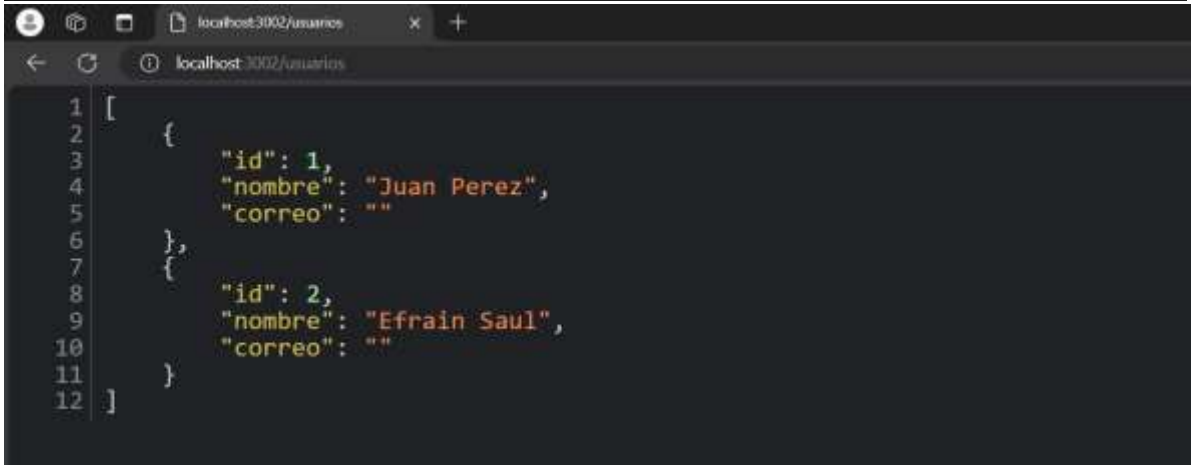
```
PS C:\Users\MAYTA\OneDrive\Escritorio\practica8> python automatizacion.py
>>
Datos guardados correctamente en data.csv
```



7. Evaluación y documentación

Gracias al postman sabemos que nuestra API funciona correctamente, y ahora al ejecutar el código Python de automatización para crear un archivo .CSV que guarde los datos que obtenga de la API:





III. Resultados y análisis

Principios de diseño de sistemas:

1. **Principio de consistencia:** Los elementos del sistema deben comportarse de la misma manera en situaciones similares, lo que ayuda a los usuarios a predecir cómo funcionarán las funciones.
2. **Principio de visibilidad:** Las opciones y funciones más importantes deben ser fácilmente accesibles y visibles para los usuarios.
3. **Principio de retroalimentación:** El sistema debe proporcionar retroalimentación clara y perceptible en respuesta a las acciones del usuario.
4. **Principio de jerarquía de la información:** La información debe presentarse de manera organizada, con elementos más importantes o relevantes más prominentes que otros.
5. **Principio de simplicidad:** El diseño debe ser lo más simple y claro posible, evitando la complejidad innecesaria.
6. **Principio de tolerancia al error:** El sistema debe ser capaz de manejar errores humanos de manera efectiva, minimizando sus consecuencias negativas.

Usabilidad en línea:

1. **Facilidad de navegación:** Los usuarios deben poder moverse fácilmente por el sitio web o la aplicación.
2. **Claridad en la estructura:** La estructura del sitio o la aplicación debe ser clara y fácil de entender.
3. **Rapidez de carga:** Los tiempos de carga deben ser rápidos para evitar la frustración del usuario.
4. **Diseño receptivo:** El diseño debe adaptarse a diferentes dispositivos y tamaños de pantalla.
5. **Legibilidad del contenido:** El contenido debe ser fácil de leer y comprender, con un tamaño de fuente adecuado y un contraste suficiente.

Ejemplos concretos:

1. **Buenas prácticas:** Sitios web como Google, Amazon y Airbnb son conocidos por su diseño intuitivo, navegación clara y facilidad de uso.

2. **Malas prácticas:** Sitios web con menús confusos, botones poco claros o sobrecarga de información pueden ser ejemplos de mal diseño. Un ejemplo clásico es el sitio web del gobierno de los Estados Unidos antes de su rediseño, que era criticado por su falta de usabilidad y claridad.

IV. Conclusiones y recomendaciones

Conclusiones:

- La configuración de Docker y el levantamiento de una instancia de MySQL fueron realizados con éxito en un entorno Windows.
- Las APIs desarrolladas en Flask y Express pudieron interactuar correctamente con la base de datos.
- Postman se utilizó eficazmente para probar las APIs y verificar los datos devueltos.
- El script de automatización en Python funcionó correctamente, recolectando y almacenando datos en un archivo CSV.

Recomendaciones:

- Implementar medidas de seguridad en las APIs, como autenticación y autorización.
- Utilizar un sistema de gestión de versiones como Git para controlar los cambios en el código fuente.
- Ampliar la funcionalidad de las APIs para incluir más operaciones CRUD (Crear, Leer, Actualizar, Eliminar).
- Considerar el uso de herramientas de monitoreo para supervisar el rendimiento de las APIs y la base de datos.

V. Repositorio con el código de la API y el script de automatización

https://github.com/MaytaYujraEfrain/Efrain_API_RESTful.git

REFERENCIAS Y BIBLIOGRAFÍA

https://docs.docker.com/desktop/install/windows-install/
https://www.postman.com/
https://nodejs.org/en/download/package-manager