TOR IP Management API

This project is a Flask-based REST API designed to fetch, filter, and manage TOR IPs from external sources. The application includes authentication and role-based access control (RBAC) to secure its endpoints.

Features

1. Fetch TOR IPs from external sources and store them in a database.
2. Filter TOR IPs by excluding certain IPs stored in an exclusion list.
3. Authentication using JWT with role-based permissions (`admin` and `user`).
4. Dockerized application with PostgreSQL as the database.

Table of Contents

Prerequisites

Before starting the project, ensure you have the following tools installed in your environment:

1. **Python 3.9 or higher**
   - Verify the installation:

   ```
   python --version
   ```

2. **Docker**
   - To install Docker, follow the instructions on the official site: Docker Installation.
   - Verify the installation:

   ```
   docker --version
   ```

3. **Docker Compose**
   - Docker Compose is typically included with Docker Desktop. Verify its availability:

   ```
   docker-compose --version
   ```

4. **Git**
   - To clone the repository, ensure Git is installed.
   - Verify the installation:

   ```
   git --version
   ```

Project Dependencies

The application dependencies are listed in the `requirements.txt` file. They will be automatically installed by Docker during the build process.

Setup Instructions

1. Clone the Repository

```
git clone https://github.com/lucasspinelli/tor-ip-Management-api.git
cd tor-ip-Management-api
```

2. Configure Environment Variables

The application uses the following environment variables, which are already set in the `docker-compose.yml` file:

- `POSTGRES_DB=excluded_ips_db`
- `POSTGRES_USER=postgres`
- `POSTGRES_PASSWORD=1234`
- `POSTGRES_HOST=db`
- `POSTGRES_PORT=5432`
- `SECRET_KEY=your-very-secure-secret-key`

If you want to modify them, edit the `docker-compose.yml` or provide them in a `.env` file.

3. Build and Run the Docker Containers

Run the following command to build and start the application:

```
docker-compose up --build
```

The application will be available at:

- **API Base URL:** `http://localhost:5000`

4. Access the Logs

Logs are stored in the `/logs` directory within the container and can be accessed locally if mounted as a volume. Use:

```
docker exec -it <container_name> cat /logs/api.log
```

Authentication and Roles

Users (Hardcoded)

The application uses two predefined users with different roles:

Username	Password	Role
`admin_user`	`admin123`	Admin
`normal_user`	`user123`	User

How to Authenticate

To interact with the API, you must first authenticate by obtaining a JWT token via the `/auth/login` endpoint. The token must then be included in the `Authorization` header for subsequent requests.

Example Login Request

I'll use curl to show the requests, but you can use Postman or Insomnia as well

```
curl -X POST http://localhost:5000/auth/login -H "Content-Type: application/json" -d '{"username": "admin_user", "password": "admin1
```

Example Response

```
{
   "token": "<jwt-token>"
}
```

Role-Based Permissions

- **Admin (`admin`):** Can access all endpoints, including adding and removing IPs from the exclusion list.
- **User (`user`):** Can only access `GET /api/tor-ips` and `GET /api/filtered-tor-ips`.

API Endpoints

1. POST /auth/login

- **Description:** Authenticate and generate a JWT token.
- **Request:**

```
{
    "username": "admin_user",
    "password": "admin123"
}
```

- **Response:**

```
{
    "token": "<jwt-token>"
}
```

---

2. GET /api/tor-ips

- **Description:** Retrieve all TOR IPs stored in the database.
- **Authorization Required:** Yes (Admin or User).
- **Headers:**

```
Authorization: Bearer <jwt-token>
```

- **Response Example:**

```
{
    "tor_ips": [
        "192.168.0.1",
        "10.0.0.2"
    ]
}
```

---

3. GET /api/filtered-tor-ips

- **Description:** Retrieve all TOR IPs from the database, excluding those in the exclusion list.
- **Authorization Required:** Yes (Admin or User).
- **Headers:**

```
Authorization: Bearer <jwt-token>
```

- **Response Example:**

```
{
    "filtered_tor_ips": [
        "192.168.0.3"
    ]
}
```

---

4. POST /api/excluded-ips

- **Description:** Add an IP or list of IPs to the exclusion list.
- **Authorization Required:** Yes (Admin only).
- **Headers:**

```
Authorization: Bearer <jwt-token>
```

- **Request Body Example (Single IP):**

```
{
    "ip": "192.168.0.1"
}
```

- **Request Body Example (Multiple IPs):**

```
{
    "ips": ["192.168.0.1", "10.0.0.1"]
}
```

- **Response Example:**

```
{
    "results": [
        {"ip": "192.168.0.1", "status": "success"},
        {"ip": "10.0.0.1", "status": "success"}
    ]
}
```

---

5. DELETE /api/excluded-ips

- **Description:** Remove an IP from the exclusion list.
- **Authorization Required:** Yes (Admin only).
- **Headers:**

```
Authorization: Bearer <jwt-token>
```

- **Request Body Example:**

```
{
    "ip": "192.168.0.1"
}
```

- **Response Example:**

```
{
    "message": "IP 192.168.0.1 successfully removed from exclusion list."
}
```

Run Tests

To run static analysis and verify the security of the code, use the **Bandit** tool:

```
pip install bandit
```

```
bandit -r app/ -o bandit_report.txt -f txt
```

This will generate a file `bandit_report.txt` with a summary of any potential vulnerabilities.

Logging

- All logs are saved in `/logs/api.log` and include:
  - Requests to endpoints.
  - Errors during operations (e.g., database issues, invalid tokens).
  - IP fetching from external sources.

Example of logs:

```
 2024-11-20 00:00:00, INFO - Fetched 100 TOR IPs from external sources.
2024-11-20 00:01:00, ERROR - Failed to insert IP 192.168.0.1: Duplicate entry.
```

Possible Errors and Troubleshooting

Here are some common issues you might encounter while setting up or running the application, along with their solutions:

1. psycopg2.OperationalError: connection to server at "db" (172.x.x.x), port 5432 failed

- **Cause:** This typically means the PostgreSQL container is not ready yet.
- **Solution:**
  - Verify if the database container is running: `docker ps` .
  - Check logs for PostgreSQL: `docker logs <postgres_container_name>` .
  - Ensure the healthcheck in `docker-compose.yml` is correctly configured.

---

2. Error: pg_config executable not found during psycopg2 installation

- **Cause:** This happens if the PostgreSQL development libraries are missing in the environment.
- **Solution:** Use the `psycopg2-binary` package, which is already included in this project. Ensure the requirements file is correctly installed.

---

3. JWT Issues: Unauthorized or Token has expired

- **Cause:** Either the token is missing, invalid, or expired.
- **Solution:**
  - Ensure the `Authorization` header is correctly set:

    ```
    Authorization: Bearer <jwt-token>
    ```

  - If the token is expired, generate a new one by logging in again via `/auth/login` .

---

4. docker-compose: command not found

- **Cause:** Docker Compose is not installed or not in your PATH.
- **Solution:**
  - Install Docker Compose following the official guide: Docker Compose Installation.
  - Verify installation with:

    ```
    docker-compose --version
    ```

---

5. Application Not Starting or Connection Refused

- **Cause:** This could be due to multiple reasons:
  1. The Flask app is not running.
  2. Ports are not correctly exposed.
  - **Solution:**
    - Verify the application container logs:

      ```
      docker logs <api_container_name>
      ```

    - Ensure the port `5000` is accessible and correctly mapped in `docker-compose.yml` .

---

6. Issues with Logs Not Being Written

- **Cause:** The `/logs` directory may not have been created or mounted properly.
- **Solution:**
  - Ensure the `/logs` directory exists in the container and has write permissions.
  - Check the logger configuration in `app/logging_config.py` .

If you encounter an error not listed here, check the logs for more details:

```
docker logs <container_name>
```