Full Stack Developer Internship Assignment

Design and Implement a Dynamic SMS Management Web Application

---

Scenario Overview

Your task is to **build a web-based dashboard** that dynamically manages and monitors the SMS system running on a Linux server. The system consists of **multiple Python programs (5-6 programs)** that trigger SMS messages to **multiple countries - telecom operators pairs** using **phone numbers**. Once an SMS is triggered, the programs communicate with an **SMS Gateway API** to verify message delivery and submit the status back if the message is received.

The **system manages over 100+ country-operator pairs**, with the goal of **sending 10 SMS per minute per country**, irrespective of how many operators belong to that country. These country-operator pairs are dynamic and must be managed based on **real-time SMS success rates**. Some pairs are designated as **high-priority** and must always remain active regardless of their success rate.

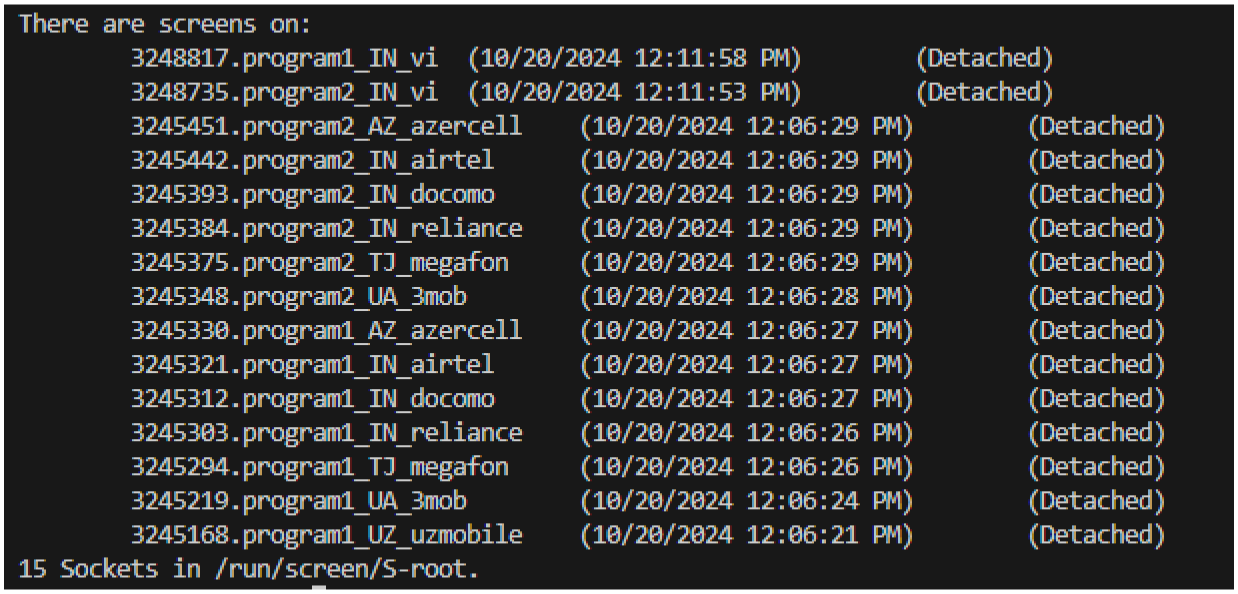
Each **Python program runs independently** using **screen sessions**, with each screen session handling one or more country-operator pairs. Your job is to develop a **dynamic management system** that enables:

- **Control over program execution** (start/stop/restart sessions)
- **Monitoring SMS performance metrics in real-time**
- **Adding, updating, and prioritizing country-operator pairs**
- **Automatic alerts for critical failures or low success rates**

Sample List of country-operator pairs

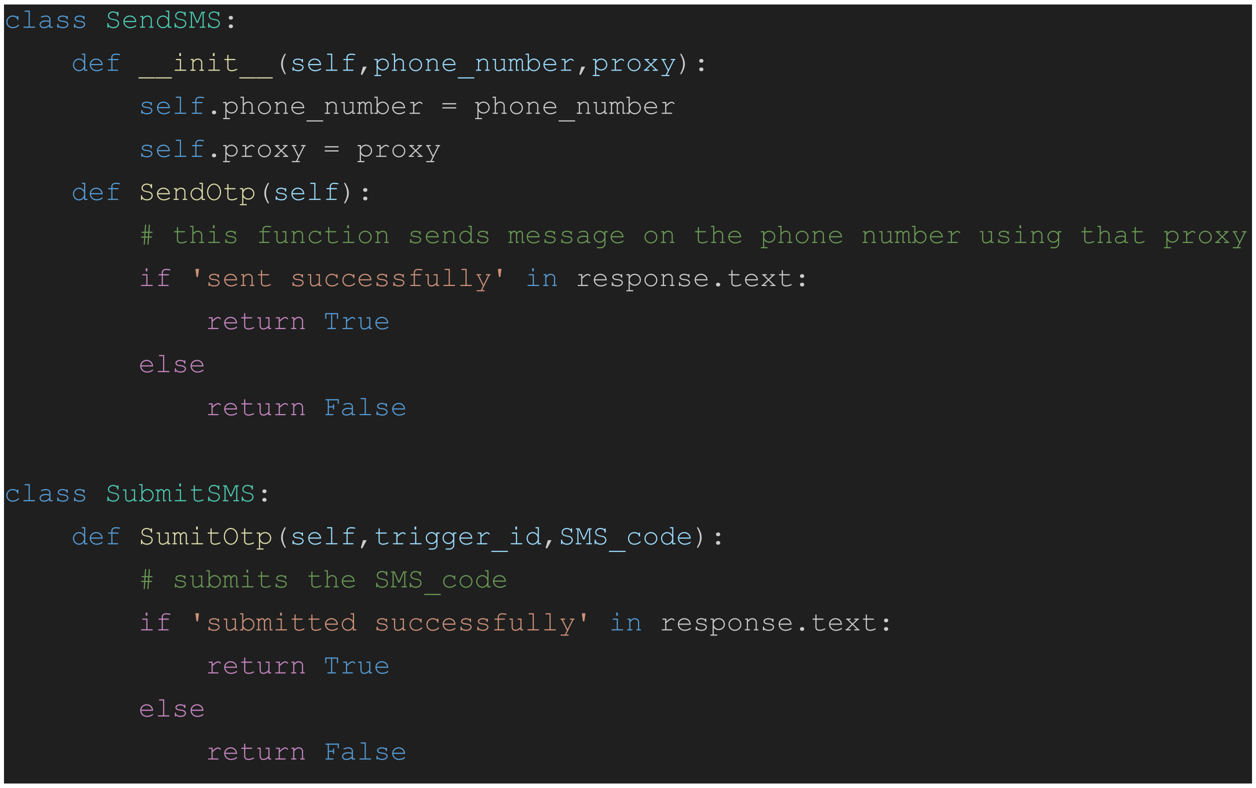
1. Uzbekistan - UzMobile
2. Ukraine - 3Mob
3. Tajikistan - MegaFon - TT Mobile
4. India - Reliance - West Bengal
5. India - TATA DOCOMO - Maharashtra & Goa
6. India - Vi India - Maharashtra & Goa
7. India - AirTel - Gujarat

Sample of country-operator pairs running on the linux system using screens. Two programs such as program1.py and program2.py are running here with multiple country-operator pairs.



```
There are screens on:
        3248817.program1_IN_vi  (10/20/2024 12:11:58 PM)        (Detached)
        3248735.program2_IN_vi  (10/20/2024 12:11:53 PM)        (Detached)
        3245451.program2_AZ_azercell    (10/20/2024 12:06:29 PM)        (Detached)
        3245442.program2_IN_airtel      (10/20/2024 12:06:29 PM)        (Detached)
        3245393.program2_IN_docomo      (10/20/2024 12:06:29 PM)        (Detached)
        3245384.program2_IN_reliance    (10/20/2024 12:06:29 PM)        (Detached)
        3245375.program2_TJ_megafon     (10/20/2024 12:06:29 PM)        (Detached)
        3245348.program2_UA_3mob        (10/20/2024 12:06:28 PM)        (Detached)
        3245330.program1_AZ_azercell    (10/20/2024 12:06:27 PM)        (Detached)
        3245321.program1_IN_airtel      (10/20/2024 12:06:27 PM)        (Detached)
        3245312.program1_IN_docomo      (10/20/2024 12:06:27 PM)        (Detached)
        3245303.program1_IN_reliance    (10/20/2024 12:06:26 PM)        (Detached)
        3245294.program1_TJ_megafon     (10/20/2024 12:06:26 PM)        (Detached)
        3245219.program1_UA_3mob        (10/20/2024 12:06:24 PM)        (Detached)
        3245168.program1_UZ_uzmobile    (10/20/2024 12:06:21 PM)        (Detached)
15 Sockets in /run/screen/S-root.
```

Sample Structure of the Python program:



```
class SendSMS:
    def __init__(self,phone_number,proxy):
        self.phone_number = phone_number
        self.proxy = proxy
    def SendOtp(self):
        # this function sends message on the phone number using that proxy
        if 'sent successfully' in response.text:
            return True
        else
            return False

class SubmitSMS:
    def SumitOtp(self,trigger_id,SMS_code):
        # submits the SMS_code
        if 'submitted successfully' in response.text:
            return True
        else
            return False
```

Assignment Tasks

You are required to build a **full-stack web application** that fulfills the following requirements:

---

1. Backend Development:

Develop a **backend API** using **Python (Flask/FastAPI)** with the following functionality:

1. **Process Management:**
   - Manage the **screen sessions** (start/stop/restart programs) programmatically.
   - Each session should correspond to a country-operator pair (e.g., `program1_UZ_uzmobile`).
2. **Real-Time Metrics:**
   - Expose API endpoints to **retrieve real-time metrics** like SMS sent, success rates, and errors per country.
   - Implement **rate limiting logic** to ensure only **10 SMS per minute per country** are sent.
3. **Country-Operator Management:**
   - Provide **CRUD operations** to dynamically add, remove, or update **country-operator pairs**.
   - Allow setting **high-priority pairs**, which cannot be stopped even if their success rate drops.
4. **Alerts and Notifications:**
   - Implement **alerting logic** to send **notifications via Telegram** when:
     - A critical success rate drop occurs.
     - A program fails or crashes.
   - Use tools like **Telegram bots** for notifications.
5. **Authentication and Authorization:**
   - Implement **JWT-based authentication** to secure the API.
   - Only authenticated users should be able to control sessions and update configurations.

---

2. Frontend Development:

Develop a **user-friendly dashboard** using **ReactJS** (or any frontend framework) with the following pages:

1. **Dashboard:**

    - Display real-time metrics using **charts and tables** (e.g., SMS sent, success rates, failures).
    - Implement **WebSocket integration** (optional) to push real-time data without refreshing the page.
2. **Program Control:**
    - Provide controls to **start, stop, or restart** SMS programs via buttons.
    - Display the **current status** of each session (active/inactive).
3. **Country-Operator Management:**
    - Allow the addition, removal, or update of country-operator pairs dynamically.
    - Display **prioritized pairs** in a separate section.
4. **Login Page:**
    - Implement a **secure login page** with JWT authentication.

---

3. Database Design:

Design a **two-database structure** for optimal performance:

1. **MongoDB:**
    - Store program configurations (e.g., active pairs, high-priority status, session details).
    - Enable dynamic updates to configurations from the frontend.
2. **MySQL:**
    - Store **SMS metrics** (e.g., SMS sent, success rates, failures).
    - Use a schema that supports querying performance trends over time.

---

4. Monitoring and Alerts:

1. **Prometheus Integration:**
    - Monitor key metrics like SMS sent, success rates, and failures using **Prometheus**.
    - Create **Grafana dashboards** to visualize the data.
2. **Alerts Setup:**
    - Set up **Prometheus AlertManager** to send notifications when:
        - SMS success rates fall below a threshold.
        - A program fails or crashes.

Bonus Tasks (Optional):

- **WebSocket Integration:** Push real-time updates to the frontend using WebSockets.

- **Advanced Analytics:** Implement charts to show **trends in success rates** over time.
- **Containerization:** Use **Docker** to containerize the application for easier deployment.

Deliverables:

1. **Architecture Diagram:**
   - Provide a high-level architecture diagram showing the interaction between the frontend, backend, database, monitoring, and screen sessions.
2. **Code and Repository:**
   - Submit your **GitHub repository** containing:
     - Frontend and backend code.
     - Configuration files for databases and monitoring tools.
     - Scripts for starting and stopping sessions using screens.
     - Documentation with setup instructions (optional).

Evaluation Criteria:

1. **System Design and Functionality:**
   - Does the solution meet the requirements?
   - Is the architecture well thought out and scalable?
2. **Code Quality:**
   - Is the code organized, efficient, and maintainable?
   - Are the APIs well-structured and properly documented?
3. **Frontend Usability:**
   - Is the interface intuitive and easy to navigate?
   - Are metrics displayed accurately in real-time?
4. **API and Database Integration:**
   - Are the backend APIs functional and smoothly integrated with MongoDB and MySQL?
5. **Monitoring and Alerts:**
   - Are alerts configured correctly and sent as expected?
   - Are metrics displayed in a meaningful way on the Grafana dashboard?
6. **Authentication and Security:**
   - Is the application secured with JWT-based authentication?
   - Are sensitive operations restricted to authorized users?

Submission Guidelines:

- **Deadline:** 31-10-2024
- Submit your GitHub repository link along with:
    - Architecture diagram
    - Documentation

Conclusion:

This assignment reflects **real-world challenges** in **full-stack development** and will help assess your ability to design, build, and manage complex web applications. It covers key areas like **backend automation, database integration, real-time monitoring**, and **dynamic configurations**.

We look forward to reviewing your submission and evaluating your ability to contribute effectively to our dynamic team!
