Simple Testing Practice.

You are tasked with testing the **Medical Monitoring App**, a simulation of a medical system that monitors a patient's vital signs (heart rate, blood pressure, and oxygen saturation). The app integrates with a wearable device, generates alerts if the readings are outside safe thresholds, and handles emergency contacts.

The Simulation can be found in this link :

https://github.com/mohamed-halemo/Testing-of-a-Simulated-Medical-Monitoring-App

The **Medical Monitoring App** consists of the following components:

1- **Biosignal Classes**: Monitor and validate individual signals such as:
   - **Heart Rate**: Safe range: 60–100 bpm.
   - **Blood Pressure**: Safe range: 80–120 mmHg.
   - **Oxygen Saturation**: Safe range: 95–100%.

2- **Wearable Device Class**: Simulates the device that aggregates readings from biosignal classes and provides overall status updates.
3- **Emergency Service Class**: Handles contacting emergency services (hospital and ambulance) when vital signs exceed safe thresholds.
4- **Medical Monitoring App Class**: Combines the device and emergency service components into an application for real-time patient monitoring and alert generation.

Your Task

You will design and implement tests for this system to ensure it behaves correctly, integrates components seamlessly, and handles performance demands effectively. Below are the specific test levels to address:

1- **Unit Testing**: Test each class independently to verify its behavior.
2- **Integration Testing**: Test the interaction between components.
3- **System Testing**: Simulate real-world workflows involving all components.
4- **Performance Testing**: load test the app to evaluate how it handles high-frequency updates and large volumes of data.

Requirements

1. **Test Coverage**:

   Write tests for **all main functionalities** of the app, ensuring no critical paths are left untested.

2. **Testing Framework**:

   Use a testing framework such as **unittest** & **pytest** to implement and execute your tests.

3. **Test Cases**:

   Include detailed assertions to verify the expected outcomes of each test use AAA technique.

   Handle edge cases such as negative input values and borderline readings.

4. **Documentation**:

   Document your test cases, including:

   The purpose of the test.

   The steps or setup required to execute the test.

   Expected results.

5. **Performance Testing**:

   mock data to simulate rapid updates and heavy workloads. Measure and

   document response times and identify potential bottlenecks.

Submission:

1. .zip folder named after one of you STUDENT NAME_ID.ZIP
2. a .txt file containing your names and ID's
3. Test execution results pass/fail outcomes and performance metrics (screen shots and .txt file containing any details you want to provide or comments.
4. Word Document containing documentation of your test cases as showed below.

Test Case Documentation Example

Test Case ID

HR-001

Test Title

Verify Heart Rate Alert for Values Above Safe Threshold

Test Objective

Ensure the HeartRate class triggers an alert when the heart rate exceeds the upper safe limit (100 bpm).

Preconditions

1. The HeartRate class must be initialized.
2. A method update_value() must exist in the HeartRate class to update the heart rate.

Test Steps

1. Instantiate the HeartRate class.
2. Use the update_value() method to set a heart rate of 110 bpm.
3. Verify if the alert attribute or method returns a message indicating an unsafe heart rate.

Expected Results

The alert method should return the message: "Alert: Heart rate exceeds safe threshold".

Actual Results

(the results you get should be added here after test is executed)