Dog Activity Tracker - Developer Requirements Document

Project Overview: Develop a dog activity tracker using ESP32 Arduino library (version 2.0.14), LittleFS for file system management, and TensorFlow Lite for activity classification. The tracker will monitor a dog's activities, store the data efficiently, and sync with a mobile app.

Hardware:

- ESP32-S3 microcontroller
- QMI8658 accelerometer+gyroscope
- LiPo battery
- Charging circuit
- LCD Screen

Software Dependencies:

- ESP32 Arduino Core (v2.0.14)
- LittleFS
- TensorFlow Lite for ESP32 (version TBD based on compatibility)
- BLE library
- ArduinoOTA library

Milestone 1: Core Functionality and Sensor Simulation (we are using SensorLib)

1.1 Accelerometer Simulation

- Create a dummy accelerometer module that simulates the QMI8658 behavior
- Implement the following functions:
    - begin(): Initialize the simulated sensor
    - readFromFifo(IMUdata* acc, int accCount, IMUdata* gyr, int gyrCount): Simulate reading from FIFO
    - configWakeOnMotion(): Configure wake-on-motion settings
    - setWakeupMotionEventCallBack(callback): Set callback for motion events

Example code structure for accelerometer simulation:

```
class SimulatedQMI8658 {
```

```
private:
  bool motionDetected;
  void (*wakeupCallback)();

public:
  SimulatedQMI8658() : motionDetected(false), wakeupCallback(nullptr) {}

  bool begin() {
    // Simulate sensor initialization
    return true;
  }

  bool readFromFifo(IMUdata* acc, int accCount, IMUdata* gyr, int gyrCount) {
    // Generate random accelerometer and gyroscope data
    for (int i = 0; i < accCount; i++) {
      acc[i].x = random(-32768, 32767) / 1000.0;
      acc[i].y = random(-32768, 32767) / 1000.0;
      acc[i].z = random(-32768, 32767) / 1000.0;
    }
    for (int i = 0; i < gyrCount; i++) {
      gyr[i].x = random(-32768, 32767) / 100.0;
      gyr[i].y = random(-32768, 32767) / 100.0;
      gyr[i].z = random(-32768, 32767) / 100.0;
    }
    return true;
  }

  void configWakeOnMotion() {
    // Simulate wake-on-motion configuration
  }

  void setWakeupMotionEventCallBack(void (*callback)()) {
    wakeupCallback = callback;
  }

  // Simulate motion detection (call this periodically in your main loop)
  void simulateMotion() {
    if (random(100) < 5 && !motionDetected) { // 5% chance of motion detection
      motionDetected = true;
      if (wakeupCallback) {
        wakeupCallback();
      }
    } else {
      motionDetected = false;
    }
  }
};
```

1.2 Wake-up Mechanism

- Implement deep sleep functionality using ESP32's sleep modes
- Use ESP32's RTC memory to store wake-up time and counters
- Implement wake-on-motion using the simulated accelerometer
- Wake up every 2 minutes to update time when inactive

1.3 Time Management

- Implement a TimeManager class to handle time-related functions
- Use ESP32's RTC for timekeeping

- Implement NTP synchronization when Wi-Fi is available

1.4 Basic Activity Classification

- Implement a simple activity classifier using predefined thresholds
- Classify activities into Resting, Walking, Running, and Playing

1.5 Data Storage Structure

- Design and implement an efficient data storage structure using LittleFS
- Store activity data in 10-minute intervals
- Data format: timestamp, activity durations (in seconds) for each activity type

Example LittleFS data structure:

```
/data/
  ├── YYYYMMDD.dat
  ├── YYYYMMDD.dat
  └── ...
```

File format (binary):

[timestamp: uint32][activity_0: uint16][activity_1: uint16][activity_2: uint16]...