Posture Pal - Frontend Requirements Documentation

1. User Interface (UI) Requirements

Splash and Registration Screens:

- **Splash Screen**: Display the Posture Pal logo and a friendly mascot.
- **Registration and Login**: Simple form allowing users to either **Register** or **Login**.

Main Dashboard (Home):

- The dashboard should display:
  - ➢ **Current posture status** with a graphical representation (circular indicator) of posture accuracy (e.g., 95% good posture).
  - ➢ **Real-time feedback** on posture alerts (notifications, vibration, sound).
  - ➢ **Device temperature monitoring**: Add an icon or temperature reading to show the current temperature of the device ensuring the user is aware if it gets too hot.
  - ➢ **Average angle** of posture in degrees.

2. Statistics Section:

- Show detailed graphs or percentages for:
  - ➢ **Posture history** over a week (e.g., percentage of good posture).
  - ➢ **Time spent standing** vs. sitting.
- This section should be customizable to view data by day, week, or month.

3. Learning Section:

- Provide users with information on improving posture through:
  - ➢ **Tutorials and exercises** such as yoga for posture and gym workouts.
  - ➢ **Health tips** related to posture benefits.

4. Settings:

- Allow users to:
  - ➢ **Toggle device settings** for sound, vibration, and power.
  - ➢ **Monitor temperature** with a dedicated display to show device temperature.
  - ➢ **Customize appearance**, such as theme (dark/light mode) and font size.
  - ➢ **Calibrate device** for accurate posture readings.
  - ➢ **Set goals**, export data or delete the account.

- The UI should follow **material design guidelines** to maintain consistency with Android's native interface.
- A **responsive design** is required to support different screen sizes and orientations for better user experience across devices.

2. User Interaction Requirements

- Users must be able to:
    - ➢ **Adjust alert preferences** (switch between sound and vibration).
    - ➢ **View posture history** with timestamps of slouching events.
    - ➢ **Pair the device** with the app through Bluetooth or Wi-Fi.
    - ➢ **Receive real-time alerts** when slouching is detected.
- The app should provide **immediate feedback** through push notifications, vibrations, or sound when posture deviations occur.

3. Data Management Requirements

- The app must store:
    - ➢ **Posture history data** (e.g., timestamps, slouching events) in a user-friendly format with graphs or tables showing past performance.
- The app should **fetch data from AWS** (where posture history and user data are stored) and display it within the app in real-time.
- Integration with **PubNub** is required for **real-time communication** between the wearable device and the mobile app enabling seamless posture monitoring.

4. Performance Requirements

- The frontend must be **optimized for smooth performance** ensuring no lag in receiving real-time alerts.
- Ensure that **push notifications** and other real-time alerts are delivered **without delay**.
- The app should remain **responsive** across devices with **minimal load times** when accessing posture history or performing actions.

5. Security Requirements

- Users should be able to **log in securely** to their accounts with posture history and personal data fetched from the AWS backend.
- The frontend must ensure **data privacy** by using secure communication protocols (e.g., **HTTPS**) to prevent unauthorized access to posture data.

6. Accessibility Requirements

- The app must support **multiple feedback modes** (visual, auditory,vibrations and notifications) to cater to users with different accessibility needs.

- Ensure **text readability** with appropriate font sizes and clear color contrast across all UI components.

7. Pros and Cons of Kotlin

Pros:

- **Concise Code**: Kotlin enables writing shorter, cleaner code reducing boilerplate and improving productivity.
- **Null Safety**: Kotlin's null safety feature reduces the risk of null pointer exceptions.
- **Interoperability**: Fully interoperable with Java allowing easy integration with existing libraries and frameworks.
- **Coroutines Support**: Kotlin's coroutines facilitate writing efficient asynchronous code useful for real time posture monitoring.

Cons:

- **Learning Curve**: For developers familiar with Java learning Kotlin's syntax may take time.
- **Compilation Speed**: Kotlin can have slower compilation times compared to Java.
- **Smaller Community**: Kotlin's community while growing is still smaller than Java's which can limit resources.

8. Extended Services in the App

PubNub (Real-Time Communication):

- **Purpose**: Manages real-time data transmission between the wearable device and the mobile app for posture alerts.
- **Integration**: PubNub will send posture updates and data from the ESP8266 microcontroller to the app.
- **Benefit**: Enables low latency, real time messaging for immediate posture correction alerts.

AWS (Cloud Data Storage):

- **Purpose**: Stores user accounts and posture history in the cloud.
- **Integration**: The mobile app fetches and stores data using AWS services ensuring scalability and security.
- **Benefit**: Reliable cloud storage and fast data retrieval for posture history and account information.

DHT22 Temperature Sensor:

- **Purpose**: Monitors the device's temperature to prevent overheating.
- **Integration**: Temperature data is transmitted to the app via the ESP8266, allowing real-time monitoring.

9. Frontend Integration with Other Parts of the Project

- **Wearable Device (ESP8266 + BNO055 Sensor)**:
  - ➢ The **mobile app** integrates with the wearable device by receiving real-time posture data through **PubNub**. The BNO055 sensor tracks posture and the ESP8266 sends this data to the app.
- **AWS (Backend)**:
  - ➢ The app interacts with **AWS** to store and retrieve user posture history and account information providing seamless data synchronization between the wearable device and the app.
- **PubNub (Real-Time Data Transmission)**:
  - ➢ **PubNub** handles the communication between the wearable device and the app ensuring posture data is received in real time and alerts are triggered without delay.