Topics

JSON Nachrichtenaustausch standardisieren

Problem: Sowohl in Android als auch in Python muss ein Adapter gehalten werden, der die gleichen JSONs baut.

Viel Aufwand, muss kongruent sein.

Lösungsvorschlag: Rest-Applikation die für die JSONs kontaktiert wird.

Nachteile: Zusätzliche Latenz Vorteile: Einheitliches Nachrichtenmodell mit einzelnem Definitionsgrund. Mischmasch wird verhindert.

Button Toggle implementieren

Lieber eine Anfrage die 10 mal toggelt oder 10 Anfragen die jweils Toggeln.

Falls letzteres: Wie synchronisieren? QOS 2 bedingt sehr zeitversetze Übertragung. Anfragen synchronisiert entgegennehmen? Kann der Message Listener?

Mitgenommener Input

Generelle Tips:

- Binäres Protokoll von PC zu Middleware. Sonst wird ein JSON Parser in der Lib benötigt. Vorteil: K
- Mehr Inhalt Wenn ein Satz Sinn ergibt wenn man ihn negiert ist das kein Guter Satz. Beispiel: Wir wollen umfassende Beispiele für Anwendungsbeispiele. Warum sollte man keine umfassenden Beispiele wollen?
- Immer im Präsenz schreiben. Kein Futur. Immer aus der allwissenden Erzählerperspektive schreiben.
- Alle Kapitel schonmal rausschreiben für die Gliederung. Am Besten auch schonmal hinten raus viele Kapitel wo Themen reinpassen.
- C Lib auch gerne schonmal vorschreiben
- Vorschlag allgemeine Gliederung:

    1. Einführung
    2. Anforderungen und Aufgabenstellungen
    3. Architektur
    4. Funktionsaufrufe
    5. Sensoren
    6. Nachrichtenformate

        1. MQTT :

        - Generell nochmal über MQTT schreiben. Was is n Broker was is ne Message was ist QOS.

    2. UDP

        - Warum UDP
        - wie läuft der Verbindungsablauf ab, welche Ports

    7. Fazit

- Lösung Request Queue: Immer zwei Werte pro Sensor speichern. Bei Anfrage vergleichen ob sich Sensorwert geändert hat. Falls nein time.sleep(10ms)
- Ruhig mehr Arbeit schreiben, weniger Code. Demo läuft. Großer Schritt.

Einführung

- Viel mehr Zielsetzung, weniger Motivation.
- Android Anwendung raus. Kommt später dann bei Architektur.
- Erklären was wo steht. Vorwärtsverweise nur hier in Ordnung.

Experimente / Aufgaben

- Einführende Sätze: Warum braucht man Aufgaben was soll in ihnen gemacht werden. Was soll erreicht werden.
- Aufgaben ruhig bisschen länger und komplexer.
- Erweitern um Anforderungen. Am Besten später. Erst Experimente beschreiben, dann Anforderungen rausdestilieren, beschreiben warum Sie gebraucht werden. Beispiel: QOS2 = Blöd weil > 1s Latenz und dann in Kapitel 3 wieder auf die Anforderungen verweisen wie Sie implementiert wurden.
- Wo soll was angezeigt werden? Wenn jetzt gewürfelt wird, auf welchem Device soll das Ergebnis ausgegeben werden? Wie ist der Ablauf generell? Wie wird eine Aufgabe implementiert?
- Nicht zu technisch werden. Ganz dumm beschreiben was gemacht werden soll, nichts zur Umsetzung. Kommt dann später im Arch. Kapitel oder noch später.
- API komplett raus.

Architektur

- "Library in C" -> "Programmiersprachenunabhängige Bedienung"
- Sequenzdiagramm zum Nachrichtenablauf.
- Beschleunigsungssensordaten wirklich g abziehen? Lieber die Entwickler selber rechnen lassen. Nicht zu viel Arbeit abnehmen.

Sonstiges

- Keine REST App für Requests. Abfrage dauert schon lang genug. Lieber schnell in UDP binäres Protokoll implementieren.
- Toggle nicht synchron bei QOS2. Im Zweifel lieber drauf verzichten. Nochmal Testen im Lab beim IOT Wlan aus kurzer Distanz und niedrigerem QOS.
- Bei Sensoren vielleicht Batch-Transfer zurück? Beispiel: Accellordaten nehmen, packen als Pack weitersenden, End of Transfer angeben. Client kann damit dann machen was er will.
    - Frequenzen übertragen als Modell? Mehrere Frequenzen rüberschicken?
- TK Inter für Server damit Settings agegeben werden können. Vielleicht erweitern um Liveview der Datenstruktur.

Für nächstes mal

- Eher über technische Problemchen reden, weniger über die Arbeit. Natürlich trotzdem dran weiterarbeiten.