Ausarbeitung – LLM-Light-Control

1. Gewählte Architektur und Begründung

Wir haben uns für eine **Event-Driven-Architecture** entschieden, wie in der Aufgabenstellung gefordert. Die Kommunikation zwischen Frontend/Backend (Producer) und der Lampensteuerung (Consumer) erfolgt asynchron über **RabbitMQ** als Message Broker.

Vorteile dieser Architektur:

- **Entkopplung:** Producer und Consumer sind unabhängig voneinander und können getrennt entwickelt, getestet und betrieben werden.
- **Skalierbarkeit:** Weitere Producer oder Consumer können einfach hinzugefügt werden (z.B. mehrere Lampen oder Steuerungs-UIs).
- **Fehlertoleranz:** Nachrichten gehen nicht verloren, wenn ein Teil des Systems kurzzeitig nicht erreichbar ist.
- **Flexibilität:** Die Architektur kann leicht auf andere IoT-Geräte oder zusätzliche Funktionen erweitert werden.

Warum RabbitMQ?

RabbitMQ wurde als Message Broker gewählt, da es folgende Vorteile bietet:

- **Bewährte Technologie:** RabbitMQ ist eine etablierte Lösung mit einer großen Community und wird in vielfach eingesetzt.
- **Unterstützung für AMQP:** RabbitMQ basiert auf dem Advanced Message Queuing Protocol (AMQP), das speziell für zuverlässige und skalierbare Nachrichtenübermittlung entwickelt wurde.
- **Einfaches Management:** RabbitMQ bietet standardmäßig ein benutzerfreundliches Web-UI zur Verwaltung an.
- **Hohe Zuverlässigkeit:** Nachrichten können theoretisch persistent gespeichert werden, sodass sie auch bei Systemausfällen nicht verloren gehen.
- **Skalierbarkeit:** RabbitMQ könnte leicht horizontal skaliert werden, um größere Lasten zu bewältigen.
- **Kompatibilität:** RabbitMQ lässt sich problemlos mit Node.js und anderen Technologien integrieren, die in unserem Projekt verwendet werden.

Warum keine Alternativen wie Kafka oder Redis?

- **Apache Kafka:** Kafka ist für sehr große Datenmengen und Echtzeit-Streaming optimiert. Es ist daher für unser Projekt überdimensioniert, da wir keine hohen Durchsatzanforderungen haben.
- **Redis:** Redis ist ein schneller In-Memory-Datenspeicher, der auch als Message Broker verwendet werden kann. Allerdings fehlen Redis einige Funktionen wie komplexe Routing-Mechanismen und persistente Speicherung.

---

2. Funktionsweise der Anwendung

Komponenten

- **Producer (Frontend + Backend):**
  - Das Frontend (HTML/JS) bietet eine Web-Oberfläche zur Steuerung der Lampe (An/Aus, Helligkeit, Farbe, Morsecode).
  - Das Backend (Node.js/Express) nimmt die Steuerbefehle entgegen und sendet sie als Nachrichten an RabbitMQ.
- **Consumer (Lampensteuerung):**
  - Ein Node.js-Skript empfängt die Nachrichten von RabbitMQ.
  - Die Steuerung der TP-Link Tapo Bulb erfolgt über die Bibliothek `tplink-bulbs`.
  - Statusänderungen werden per WebSocket an eine Statusanzeige im Browser übertragen.
- **RabbitMQ:**
  - Vermittelt die Nachrichten zwischen Producer und Consumer.
  - Die Queues und Nachrichten können über das Web-UI (http://localhost:15672, Login: guest/guest) eingesehen werden.

Ablauf

1. **Nutzerinteraktion:**
   Der Nutzer steuert die Lampe über das Web-Frontend (z.B. Licht einschalten, Helligkeit ändern).
2. **Backend:**
   Das Backend nimmt die Befehle entgegen und sendet sie als JSON-Nachricht an die RabbitMQ-Queue `lamp-commands`.
3. **Consumer:**
   Der Consumer liest die Nachrichten aus der Queue und führt die gewünschten Aktionen an der Lampe aus.
   Ist keine echte Lampe verbunden, läuft ein Demo-Modus (nur Konsolen-Logs).
4. **Statusanzeige:**
   Der aktuelle Lampenstatus wird per WebSocket an eine Statusseite im Browser übertragen und dort live angezeigt.

---

3. Matrikelnummern

- Leon: 6550883
- Luca: 6949663
- Monika: 1396397