Ausarbeitung: Smart Bulb Steuerung mit Event-Driven Architecture

Gewählte Architektur: Event-Driven Architecture (EDA)

Für die Umsetzung unserer Anwendung haben wir uns für eine *Event-Driven Architecture* entschieden. Diese Architektur wurde nicht nur vorgegeben, sondern eignet sich auch ideal für lose gekoppelte, skalierbare und asynchrone Systeme. Die Kommunikation erfolgt über ein Message-Queue-System (RabbitMQ), bei dem ein *Producer* Ereignisse erzeugt und ein *Consumer* diese verarbeitet. Ebenfalls erleichterte die für das Assignment vorgegebene Vorlage unsere Arbeit.

Vorteile der EDA in unserem Szenario:

- **Entkopplung:** Producer (API) und Consumer (Smart Bulb Steuerung) sind unabhängig.
- **Asynchronität:** Befehle müssen nicht in Echtzeit verarbeitet werden.
- **Skalierbarkeit:** Beide Komponenten können unabhängig skaliert oder erweitert werden.
- **Erweiterbarkeit:** Weitere Event-Handler (z. B. Logging, Automationen) können leicht integriert werden.

Funktionsweise der Anwendung

Die Anwendung besteht aus zwei Hauptkomponenten:

1. Producer (API)

Der Producer basiert auf Node.js mit Express und stellt REST-Endpunkte zur Verfügung. Jeder Aufruf eines Endpunkts sendet ein JSON-Objekt als Nachricht an die RabbitMQ-Queue `lamp-command`.

Beispiele:

- `POST /power` mit `{ "value": "on" }` → Lampe einschalten
- `POST /color` mit `{ "value": "red" }` → Farbe ändern
- `POST /brightness` mit `{ "value": 80 }` → Helligkeit setzen
- `POST /morse` mit `{ "value": "SOS" }` → Morsecode blinken

Zusätzlich wird eine kleine Benutzeroberfläche über die Datei `index.html` bereitgestellt.

index.html

Diese Datei dient als einfache grafische Steuerung für die Lampe über Buttons und Eingabefelder im Browser. Sie ruft die REST-API-Endpunkte direkt auf:

- Buttons zum Ein- und Ausschalten der Lampe (`/power`)
- Eingabefeld zur Farbänderung (`/color`)
- Eingabefeld für Helligkeit (`/brightness`)
- Morsecode-Textfeld zur Umsetzung in Blinksignale (`/morse`)

Die Antworten des Servers werden live angezeigt. Damit eignet sich diese Datei ideal zum Testen der API ohne zusätzliche Tools.

2. Consumer (Smart Bulb Steuerung)

Der Consumer verbindet sich mit einer TP-Link Smart Bulb über die `tplink-bulbs`-Bibliothek. Gleichzeitig lauscht er auf Nachrichten aus der RabbitMQ-Queue und führt je nach Befehl passende Aktionen aus:

- **`on` / `off`** → Schaltet die Lampe ein/aus
- **`brightness`** → Setzt die Helligkeit (0–100)
- **`color`** → Ändert die Lampenfarbe
- **`morse`** → Wandelt Text in Morsecode um und blinkt die Lampe entsprechend

Der aktuelle Zustand der Lampe wird zusätzlich in Echtzeit über einen WebSocket an Clients übertragen.

status.html

Diese Datei visualisiert in Echtzeit den aktuellen Zustand der Lampe:

- Anzeige ob Lampe **an/aus** ist
- Anzeige von **Helligkeit**, **Farbe** und **Verbindungsstatus**
- WebSocket-Verbindung zur Live-Aktualisierung
- Anzeige des Zeitpunkts der letzten Änderung

Die Statusseite eignet sich ideal als Dashboard oder Kontrollmonitor.

Morsecode-Funktion

Die Morse-Funktion wandelt Textzeichen in Morsecode um und blinkt die Lampe:

- `.` **(Punkt)** → kurzes Blinken
- `-` **(Strich)** → langes Blinken
- Buchstaben- und Wortpausen werden zeitlich berücksichtigt

Hilfsprogramm: display_all_devices.js

Um die verfügbaren TP-Link Geräte zu ermitteln, wurde das Skript `display_all_devices.js` erstellt. Es authentifiziert sich über Umgebungsvariablen (`TPLINK_USERNAME` , `TPLINK_PASSWORD`) bei der TP-Link Cloud und listet alle gefundenen Tapo-Lampen im Terminal auf.

Dies war vor allem in der Initialisierungs- und Debugging-Phase hilfreich, um Geräte-IDs und deren Eigenschaften zu prüfen.

Übersicht der Kommunikation

```
 [Benutzer]
↓ (HTTP-POST)
[Producer / REST API / index.html]
↓ (JSON)
[RabbitMQ Queue: lamp-command]
↓ (Event)
[Consumer / Smart Bulb Steuerung]
→ [TP-Link Bulb]
→ [WebSocket → status.html: Status an Clients]
```

Fazit

Durch den Einsatz von Event-Driven Architecture konnten wir eine flexible, entkoppelte und einfach erweiterbare Smart-Bulb-Steuerung umsetzen. Die Trennung zwischen API und Geräteansteuerung erlaubt sauberes Design, schnelle Fehlerbehebung und spätere Erweiterbarkeit. Die Dateien `index.html` und `status.html` vereinfachen sowohl die Bedienung als auch die Beobachtung des Systems erheblich.