Kommunikationstechnik Teil 3 Webservices

DI (FH) Andreas Pötscher

HTL Litec

Client-Server Modell

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

*Die **Client-Server-Architektur** ist das fundamentale Konzept der modernen Netzwerkkommunikation und basiert auf einer klaren Rollenverteilung zwischen zwei Partnern. Der Server ist dabei der passive Teil der auf eine Verbindung vom Client wartet. Während der Client aktiv die Verbindung zum Server aufbaut.*

Client-Server Modell

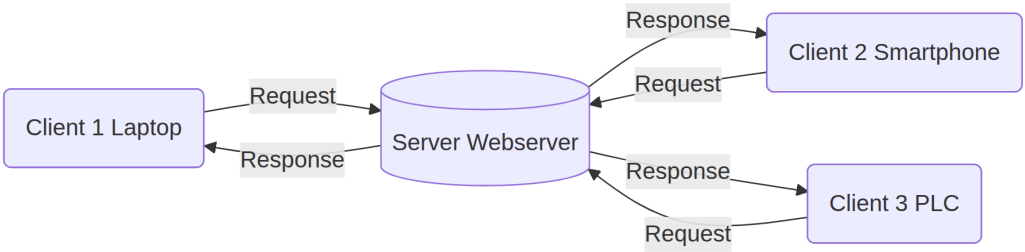

Figure 1: Ein Server mit mehreren verbundenen Clients

Client-Server Modell

Damit das Client-Server-Modell funktioniert, muss der Server für den Client über das Netzwerk erreichbar sein. Hierbei unterscheidet man zwei Szenarien:

1. **Lokale Erreichbarkeit**: Der Server befindet sich im selben Netzwerk wie der Client

Client-Server Modell

Damit das Client-Server-Modell funktioniert, muss der Server für den Client über das Netzwerk erreichbar sein. Hierbei unterscheidet man zwei Szenarien:

1. **Lokale Erreichbarkeit**: Der Server befindet sich im selben Netzwerk wie der Client
2. **Erreichbarkeit über das Internet**: Der Server steht weltweit zur Verfügung.

Beispiel Öffnen einer Website

1. Der Benutzer tippt in seinem Browser die Adresse *www.litec.ac.at* in die Adressleiste.

Beispiel Öffnen einer Website

1. Der Benutzer tippt in seinem Browser die Adresse *www.litec.ac.at* in die Adressleiste.
2. Der Browser baut eine Verbindung zum Webserver auf (meistens mit TCP/IP) und sendet einen Request.

Beispiel Öffnen einer Website

1. Der Benutzer tippt in seinem Browser die Adresse *www.litec.ac.at* in die Adressleiste.
2. Der Browser baut eine Verbindung zum Webserver auf (meistens mit TCP/IP) und sendet einen Request.
3. Der Server sendet einen Response, der die Website enthält, zurück da den Browser.

Beispiel Öffnen einer Website

1. Der Benutzer tippt in seinem Browser die Adresse *www.litec.ac.at* in die Adressleiste.
2. Der Browser baut eine Verbindung zum Webserver auf (meistens mit TCP/IP) und sendet einen Request.
3. Der Server sendet einen Response, der die Website enthält, zurück da den Browser.
4. Der Browser zeigt die Website an.

Beispiel Abfragen eines Webservice

1. Eine Dashboard-Software aktualisiert Wetterdaten einmal in der Stunde.

Beispiel Abfragen eines Webservice

1. Eine Dashboard-Software aktualisiert Wetterdaten einmal in der Stunde.
2. Die Software baut einer Verbindng zum Server auf (meistens mit TCP/IP) und sendet einen Request.

Beispiel Abfragen eines Webservice

1. Eine Dashboard-Software aktualisiert Wetterdaten einmal in der Stunde.
2. Die Software baut einer Verbindng zum Server auf (meistens mit TCP/IP) und sendet einen Request.
3. Der Server sendet einen Response, der die Wetterdaten als json String enthält, zurück an die Software.

Beispiel Abfragen eines Webservice

1. Eine Dashboard-Software aktualisiert Wetterdaten einmal in der Stunde.
2. Die Software baut einer Verbindng zum Server auf (meistens mit TCP/IP) und sendet einen Request.
3. Der Server sendet einen Response, der die Wetterdaten als json String enthält, zurück an die Software.
4. Die Wetterdaten werden am Dashboard angezeigt.

HTTP (Hypertext Transfer Protokoll)

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

Für die Kommunikation vom Geräten in einem Netzwerk wird sehr häufig das HTTP Protokoll verwendet. Eine HTTP-Nachricht ist das formale Paket, das zwischen Client und Server geschnürt wird. Da HTTP ein textbasiertes Protokoll ist, folgt der Aufbau einer strengen, für Menschen lesbaren Struktur, die in Request (Anfrage) und Response (Antwort) unterteilt wird.

Beispiel Node-Red

1. Mit Node-Red einen HTTP Server aufbauen und Website mit dem Browser ansehen.
2. Daten aus dem Request auf der Website anzeigen.
3. Mit Node-Red einen HTTP Client aufbauen.
4. Mit Node-Red einen TCP Server aufbauen und mit dem Browser darauf verbinden.

HTTP Request

Ein HTTP Reqest wird vom Client an den Server in einem lesbaren Format (als Text) gesendet. Beispielsweise:

```
> GET /news HTTP/1.1
> Host: www.litec.ac.at
> User-Agent: curl/8.5.0
> Accept: */*
```

Der Request besteht dabei aus drei Teilen:

HTTP Request

Request start line: `GET /news HTTP/1.1`

- Die Methode die Verwendet wird z.B. `GET`
- Den Pfad zur gewünschten Unter-Seite z.B. `/news`. Wenn die Hauptseite aufgerufen werden soll dann muss / verwendet werden.
- Die Protokollversion z.B. `HTTP/1.1`

Request Headers:

- Enthält Parameter in der Form `Name: Wert`
- Die Adresse die aufgerufen werden soll `Host: www.litec.ac.at`
- Software den Reqeust absetzt. `User-Agent: curl/8.5.0`
- Dateien die im Response akzetiert werden. `Accept: */*`
- Danneben gibt es noch mehr optionale Parameter

Request Body:

- Optional für bestimmte Methoden (z.B. POST).
- Enthält die Daten die übertragen werden.

HTTP Methoden

- **GET:** Holt ein Dokument vom Server. *Beispiel*: Laden einer Website.

HTTP Methoden

- **GET:** Holt ein Dokument vom Server. *Beispiel*: Laden einer Website.
- **HEAD:** Holt den Header eines Dokuments vom Server. *Beispiel*: Vorladen einer Website

HTTP Methoden

- **GET:** Holt ein Dokument vom Server. *Beispiel*: Laden einer Website.
- **HEAD:** Holt den Header eines Dokuments vom Server. *Beispiel*: Vorladen einer Website
- **POST:** Sendet Daten zum Server. *Beispiel*: Daten werden auf einer Website eingegeben und zum Server gesendet.

HTTP Methoden

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

- **GET:** Holt ein Dokument vom Server. *Beispiel*: Laden einer Website.
- **HEAD:** Holt den Header eines Dokuments vom Server. *Beispiel*: Vorladen einer Website
- **POST:** Sendet Daten zum Server. *Beispiel*: Daten werden auf einer Website eingegeben und zum Server gesendet.
- **PUT:** Sendet eine Datei zum Server. *Beispiel*: Hochladen von neuem Content einer Website.

HTTP Methoden

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

- **GET:** Holt ein Dokument vom Server. *Beispiel*: Laden einer Website.
- **HEAD:** Holt den Header eines Dokuments vom Server. *Beispiel*: Vorladen einer Website
- **POST:** Sendet Daten zum Server. *Beispiel*: Daten werden auf einer Website eingegeben und zum Server gesendet.
- **PUT:** Sendet eine Datei zum Server. *Beispiel*: Hochladen von neuem Content einer Website.
- **TRACE** Verfolgt eine Message zum Server. *Beispiel*: Testen einer Verbindung.

HTTP Methoden

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

- **GET:** Holt ein Dokument vom Server. *Beispiel*: Laden einer Website.
- **HEAD:** Holt den Header eines Dokuments vom Server. *Beispiel*: Vorladen einer Website
- **POST:** Sendet Daten zum Server. *Beispiel*: Daten werden auf einer Website eingegeben und zum Server gesendet.
- **PUT:** Sendet eine Datei zum Server. *Beispiel*: Hochladen von neuem Content einer Website.
- **TRACE** Verfolgt eine Message zum Server. *Beispiel*: Testen einer Verbindung.
- **OPTIONS** Anfrage nach den Options eines Servers. *Beispiel:* Anfragen welche Methoden der Server anbietet.

HTTP Methoden

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

- **GET:** Holt ein Dokument vom Server. *Beispiel*: Laden einer Website.
- **HEAD:** Holt den Header eines Dokuments vom Server. *Beispiel*: Vorladen einer Website
- **POST:** Sendet Daten zum Server. *Beispiel*: Daten werden auf einer Website eingegeben und zum Server gesendet.
- **PUT:** Sendet eine Datei zum Server. *Beispiel*: Hochladen von neuem Content einer Website.
- **TRACE** Verfolgt eine Message zum Server. *Beispiel*: Testen einer Verbindung.
- **OPTIONS** Anfrage nach den Options eines Servers. *Beispiel:* Anfragen welche Methoden der Server anbietet.
- **DELETE** Löscht ein Dokument vom Server. *Beispiel*: Löschen einer Artikels aus einem Online Shop.

Beispiel HTTP Server mit Node Red

- Mit dem HTTP In Node wird ein GET Request eingelesen und mit dem HTTP Out Node wieder zurückgegeben.
- Der Debug Node muss auf `complete msg object` gestellt werden. Dann wird der HTTP Request angezeigt.
- Mit `Ihr Betriebssystem: {{req.headers.sec-ch-ua-platform}}` im Template Node kann das Betriebssystem des Client auf einer Website angezeigt werden.

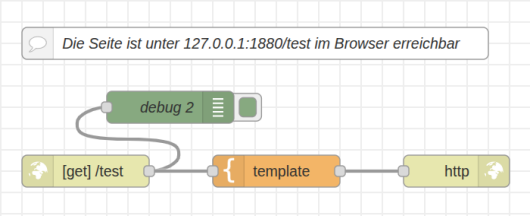

Figure 2: HTTP Server mit Node Red

Beispiel HTTP Client mit Node Red

- Mit dem HTTP Request Node eine öffentliche HTML Seite abfragen.
- Mit dem HTTP Request Node ein Webservice abfragen (z.B. https://jsonplaceholder.typicode.com/todos/1)

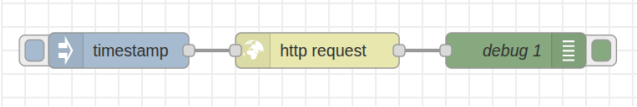

Figure 3: HTTP Client mit Node Red

Response

Die Antwort (Response) wird vom Server an den Client zurückgesendet. Sie bestätigt den Erhalt der Anfrage und liefert die gewünschten Daten oder eine Fehlermeldung. Beispielsweise:

```
< HTTP/1.1 200 OK
< Content-Type: text/html; charset=UTF-8
< Content-Length: 1542
< Server: Apache/2.4.58
<
< <!DOCTYPE html><html>... (Daten)
```

Der Response besteht ebenfalls aus drei Teilen:

Response

Status line: `HTTP/1.1 200 OK`

- Die verwendete Protokollversion, z. B. `HTTP/1.1`.
- Der Status-Code, eine dreistellige Zahl, die das Ergebnis angibt, z. B. 200.
- Die Status-Message, eine kurze Textbeschreibung zum Code, z. B. `OK`.

Response Headers:

- Enthält Metadaten zur Antwort in der Form `Name: Wert`.
- Der Datentyp der Antwort: Content-Type: `text/html` (zeigt an, dass eine Webseite folgt).
- Die Größe der Daten im Body: Content-Length: `1542` (in Bytes).
- Informationen über die Server-Software: `Server: Apache/2.4.58`.
- Weitere optionale Parameter (z. B. Datum, Cache-Einstellungen).

Response Body

- Enthält die eigentliche Nutzlast (Payload).
- Das kann der HTML-Code einer Webseite, ein Bild oder ein Datensatz im JSON-Format (z. B. Sensorwerte der SPS) sein.
- Der Body ist durch eine Leerzeile von den Headern getrennt.

Response Codes

Der Server teilt dem Client über einen dreistelligen Code mit, wie die Anfrage verarbeitet wurde. Man unterscheidet fünf Kategorien:

- **1xx Information:** Die Anfrage wurde empfangen, die Bearbeitung dauert an. *Beispiel:* 100.

Response Codes

Der Server teilt dem Client über einen dreistelligen Code mit, wie die Anfrage verarbeitet wurde. Man unterscheidet fünf Kategorien:

- **1xx Information:** Die Anfrage wurde empfangen, die Bearbeitung dauert an. *Beispiel:* 100.
- **2xx: Erfolg:** Die Anfrage wurde erfolgreich bearbeitet. *Beispiel:* `200 OK` Die Standardantwort für erfolgreiche GET-Anfragen, `201 Created` Die Anfrage war erfolgreich und eine neue Ressource wurde erstellt (oft nach POST/PUT).

Response Codes

Der Server teilt dem Client über einen dreistelligen Code mit, wie die Anfrage verarbeitet wurde. Man unterscheidet fünf Kategorien:

- **1xx Information:** Die Anfrage wurde empfangen, die Bearbeitung dauert an. *Beispiel:* 100.
- **2xx: Erfolg:** Die Anfrage wurde erfolgreich bearbeitet. *Beispiel:* `200 OK` Die Standardantwort für erfolgreiche GET-Anfragen, `201 Created` Die Anfrage war erfolgreich und eine neue Ressource wurde erstellt (oft nach POST/PUT).
- **3xx Umleitung:** Weitere Schritte sind erforderlich, um die Anfrage abzuschließen.*Beispiel:* `301 Moved Permanently` Die angeforderte Seite/Datei befindet sich nun dauerhaft unter einer anderen Adresse.

Response Codes

Der Server teilt dem Client über einen dreistelligen Code mit, wie die Anfrage verarbeitet wurde. Man unterscheidet fünf Kategorien:

- **4xx Client Fehler:** Die Anfrage war fehlerhaft oder kann nicht ausgeführt werden. *Beispiel:* `400 Bad Request` Die Anfrage war syntaktisch falsch (z. B. Fehler im JSON-Body). `401 Unauthorized` Der Zugriff erfordert eine Authentifizierung (z. B. Login fehlt).

Response Codes

Der Server teilt dem Client über einen dreistelligen Code mit, wie die Anfrage verarbeitet wurde. Man unterscheidet fünf Kategorien:

- **4xx Client Fehler:** Die Anfrage war fehlerhaft oder kann nicht ausgeführt werden. *Beispiel:* `400 Bad Request` Die Anfrage war syntaktisch falsch (z. B. Fehler im JSON-Body). `401 Unauthorized` Der Zugriff erfordert eine Authentifizierung (z. B. Login fehlt).
- **5xx** Der Server hat die Anfrage eigentlich verstanden, konnte sie aber nicht verarbeiten. *Beispiel:* `500 Internal Server Error` Ein allgemeiner Fehler im Server (z. B. Absturz des SPS-Webservice-Skripts). `503 Service Unavailable` Der Server ist überlastet oder wird gerade gewartet.

URL (Uniform Resource Locator)

Eine URL (Uniform Resource Locator) ist die eindeutige Webadresse, die als Adresse einer Ressource (Webseite, Bild, Datei) im Internet dient. Z.B.

`http://www.litec.ac.at/fileadmin/user_upload/00_TS_Anmeldeformular.dotx`

URL (Uniform Resource Locator)

LINZER TECHNIKUM
HTL Paul-Hahn-Straße

Die Adresse besteht aus 3 Teilen.

- **Scheme:** Das Protokoll das verwendet wird. `http://`
- **Host:** Ort an dem der Server liegt. `www.litec.ac.at` Kann auch nur eine IP-Adresse sein. Z.B. `192.168.88.1`
- **Path:** Pfad zur gewünschten Resource. `/fileadmin/user_upload/00_TS_Anmeldeformular.dotx`

Query-Parameter

Parameter können direkt an die URL (Uniform resource locator) angehängt werden. Das wird oft für Filter oder IDs genutzt. Diese werden mit einem Fragezeichen getrennt an die URL angehängt und bestehen aus Key-Value Pairs `URL?key1=value1&key2=value2`*. Der Server kann diese dann als Parameter weiter verwenden und dementsprechend für die Antwort nutzen.*

Beispiel, es soll über ein Webservice die Temperatur von Sensor 42 in der Einheit celsius abgefragt werden:

```
/api/sensor?id=42&unit=celsius
```

Verschlüsselung

Um die Sicherheit der Datenübertragung zu gewährleisten, wird HTTP heute fast ausschließlich mit TLS (Transport Layer Security) kombiniert, was als HTTPS bezeichnet wird.

- Das Problem bei HTTP: Daten werden im Klartext übertragen. In einem Netzwerk könnte jeder Teilnehmer (z. B. im Firmen-WLAN) Passwörter oder sensible Sensordaten mitlesen (Sniffing).

Verschlüsselung

Um die Sicherheit der Datenübertragung zu gewährleisten, wird HTTP heute fast ausschließlich mit TLS (Transport Layer Security) kombiniert, was als HTTPS bezeichnet wird.

- Das Problem bei HTTP: Daten werden im Klartext übertragen. In einem Netzwerk könnte jeder Teilnehmer (z. B. im Firmen-WLAN) Passwörter oder sensible Sensordaten mitlesen (Sniffing).
- Die Lösung HTTPS: Bevor die eigentlichen HTTP-Daten fließen, bauen Client und Server einen verschlüsselten Tunnel auf. Dabei authentifiziert sich der Server mit einem Zertifikat, um sicherzustellen, dass der Client wirklich mit der echten SPS oder dem echten Server spricht (Schutz vor Man-in-the-Middle-Angriffen).

Verschlüsselung

Um die Sicherheit der Datenübertragung zu gewährleisten, wird HTTP heute fast ausschließlich mit TLS (Transport Layer Security) kombiniert, was als HTTPS bezeichnet wird.

- Das Problem bei HTTP: Daten werden im Klartext übertragen. In einem Netzwerk könnte jeder Teilnehmer (z. B. im Firmen-WLAN) Passwörter oder sensible Sensordaten mitlesen (Sniffing).
- Die Lösung HTTPS: Bevor die eigentlichen HTTP-Daten fließen, bauen Client und Server einen verschlüsselten Tunnel auf. Dabei authentifiziert sich der Server mit einem Zertifikat, um sicherzustellen, dass der Client wirklich mit der echten SPS oder dem echten Server spricht (Schutz vor Man-in-the-Middle-Angriffen).
- Auswirkung auf die Nachricht: Die Anatomie der Nachricht (Request/Response) bleibt identisch, aber der gesamte Inhalt wird für Außenstehende unlesbar chiffriert.

REST

REST (Representational State Transfer) ist kein eigenes Protokoll, sondern eine Art „Regelwerk“, wie man HTTP effizient nutzt, um mit Ressourcen (z. B. Sensoren, Motoren oder Datenbanken) zu kommunizieren. Eine API (Schnittstelle), die diesen Regeln folgt, nennt man RESTful API.

Eigenschaften von REST

- **Ressourcenorientierung**: Alles (ein Sensor, ein Messwert, ein Benutzer) ist eine Ressource und hat eine eindeutige Adresse, die URL (z. B. `/sensoren/temperatur`).

Eigenschaften von REST

- **Ressourcenorientierung**: Alles (ein Sensor, ein Messwert, ein Benutzer) ist eine Ressource und hat eine eindeutige Adresse, die URL (z. B. `/sensoren/temperatur`).
- **Standard-Methoden**: REST nutzt die HTTP-Methoden genau für das, wofür sie gedacht sind (CRUD-Prinzip):

Eigenschaften von REST

- **Ressourcenorientierung**: Alles (ein Sensor, ein Messwert, ein Benutzer) ist eine Ressource und hat eine eindeutige Adresse, die URL (z. B. `/sensoren/temperatur`).
- **Standard-Methoden**: REST nutzt die HTTP-Methoden genau für das, wofür sie gedacht sind (CRUD-Prinzip):
- **C**reate = POST

Eigenschaften von REST

- **Ressourcenorientierung**: Alles (ein Sensor, ein Messwert, ein Benutzer) ist eine Ressource und hat eine eindeutige Adresse, die URL (z. B. `/sensoren/temperatur`).
- **Standard-Methoden**: REST nutzt die HTTP-Methoden genau für das, wofür sie gedacht sind (CRUD-Prinzip):
- **C**reate = POST
- **R**ead = GET

Eigenschaften von REST

- **Ressourcenorientierung**: Alles (ein Sensor, ein Messwert, ein Benutzer) ist eine Ressource und hat eine eindeutige Adresse, die URL (z. B. `/sensoren/temperatur`).
- **Standard-Methoden**: REST nutzt die HTTP-Methoden genau für das, wofür sie gedacht sind (CRUD-Prinzip):
- **C**reate = POST
- **R**ead = GET
- **U**pdate = PUT

Eigenschaften von REST

- **Ressourcenorientierung**: Alles (ein Sensor, ein Messwert, ein Benutzer) ist eine Ressource und hat eine eindeutige Adresse, die URL (z. B. `/sensoren/temperatur`).
- **Standard-Methoden**: REST nutzt die HTTP-Methoden genau für das, wofür sie gedacht sind (CRUD-Prinzip):
- **C**reate = POST
- **R**ead = GET
- **U**pdate = PUT
- **D**elete = DELETE

Eigenschaften von REST

- **Ressourcenorientierung**: Alles (ein Sensor, ein Messwert, ein Benutzer) ist eine Ressource und hat eine eindeutige Adresse, die URL (z. B. `/sensoren/temperatur`).
- **Standard-Methoden**: REST nutzt die HTTP-Methoden genau für das, wofür sie gedacht sind (CRUD-Prinzip):
- **C**reate = POST
- **R**ead = GET
- **U**pdate = PUT
- **D**elete = DELETE
- **Zustandslosigkeit**: Wie HTTP selbst ist auch REST zustandslos. Jede Anfrage enthält alle Informationen, die der Server braucht.

Eigenschaften von REST

- **Ressourcenorientierung**: Alles (ein Sensor, ein Messwert, ein Benutzer) ist eine Ressource und hat eine eindeutige Adresse, die URL (z. B. `/sensoren/temperatur`).
- **Standard-Methoden**: REST nutzt die HTTP-Methoden genau für das, wofür sie gedacht sind (CRUD-Prinzip):
- **C**reate = POST
- **R**ead = GET
- **U**pdate = PUT
- **D**elete = DELETE
- **Zustandslosigkeit**: Wie HTTP selbst ist auch REST zustandslos. Jede Anfrage enthält alle Informationen, die der Server braucht.
- **Repräsentationen**: Eine Ressource kann in verschiedenen Formaten gesendet werden. In der modernen Technik ist JSON (JavaScript Object Notation) der absolute Standard, da es leichtgewichtig und für Mensch und Maschine gut lesbar ist.

Beispiel einer REST Kommunikation

- **Anfrage:** GET /maschinen/foerderband/speed
- **Antwort (JSON)**: {“value”: 15.5, “unit”: “m/s”}

Node Red Beispiel REST

- In Node RED kann eine LED mit einem Button ein- und ausgeschaltet werden.
- Der Zustand wird noch in einer Context Variable gespeichert.

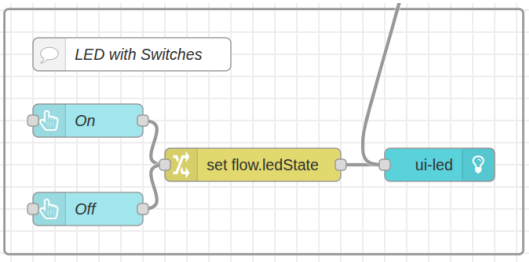

Figure 4: REST Server

Node Red Beispiel REST

- Als erstes kann der Zustand der LED über die REST Schnittstelle abgefragt werden.
- Dazu wird ein HTTP GET Request verwendet.
- Über die URL `127.0.0.1:1880/led`kann der Zustand der LED im json Format `{"ledState":1}` abgefragt werden.

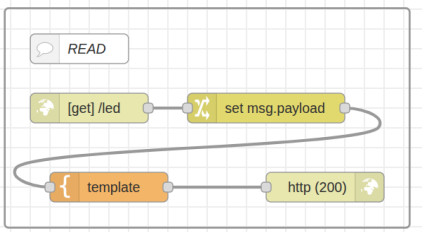

Figure 5: REST Server

Node Red Beispiel REST

- Beim REST Client kann der Status der LED jetzt abegefragt und angezeigt werden.
- Dazu wird eine HTTP Request Node verwendet. Aus dem Objekt muss noch der Wert für die LED ausgelesen werden.

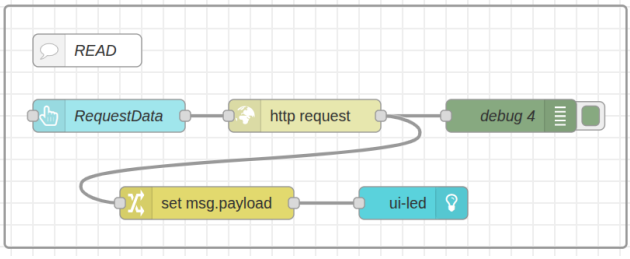

Figure 6: REST Client

Node Red Beispiel REST

- Damit die LED über einen HTTP Request geschaltet werden kann wird ein HTTP In Node mit HTTP PUT verwendet.
- Diesem wird von aussen im json Format der neue Status der LED übergeben.

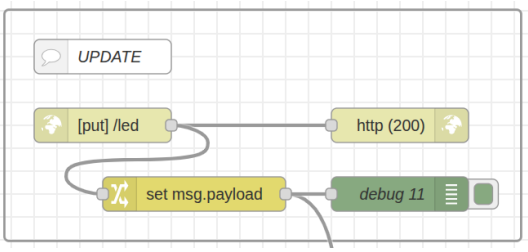

Figure 7: REST Server

- Der REST Client kann die LED jetzt über einen HTTP Put Request schalten.
- Die Buttons müssen die Daten im json Format `{"ledState":1}` versenden.

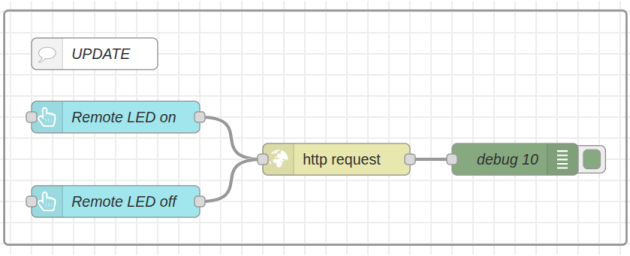

Figure 8: REST Client