JSON - Nachricht:

Request:

```
{
  "operation" : "register" | "unregister" | "list" | "ring-info" | "client-info"
  "payload" : {...}
}
```

Response:

```
{
  status : "ok/error",
  payload : ...,
  message : ...,
}
```

Entries

pro Knoten:

- int id → fortlaufend (für Token Ring)
- String name
- String ip
- int port
- Type type → enum: CLIENT, ROBOT

Operationen:

register (name, ip, port, type)

- Client/Robot meldet sich an
- Registry prüft, ob Name bereits existiert
  - → Wenn ja → Fehler
  - → Wenn nein:
    - neue ID
    - Eintrag im Speicher ablegen
    - Bestätigung an Client senden

- Request:

```
{ "operation" : "register",
  "payload": {
      "name" : "client1",
      "ip" : "127.00.1",
      "port": 1000,
      "type": "CLIENT"
  }}
```

- Response:

```
{
  "status" : "ok" ,
  "id" : 1 ,
}
```

```
{
  "status" : "error",
  "message" : "Name bereits registriert"
}
```

unregister(name)

- entfernt Eintrag im Registry
- wenn Name nicht existiert → Fehler

- Request:

```
{
"operation": "unregister",
"payload": {
        "name": "client1"
}
}
```

- Response:

```
{
  "status": "ok"
}
```

```
{
  "status": "error",
  "message": "Eintrag nicht gefunden"
}
```

list(type)

- gibt alle Einträge des gewünschten Typs zurück

- Request:

```
{"operation": "list",
 "payload": {
      "type": "ROBOT"
}}
```

- Response:

```
{
  "status": "ok",
  "entries": [
      { "id": 1, "name": "arm1", "ip": "127.0.0.1", "port": 9000,
        "type": "robot"},
      ...
]}
```

```
{
  "status": "error",
  "message": "Keine Einträge gefunden"
}
```

ring-info(clientId)

- liefert für einen Client Vorgänger- und Nachfolger-ID

- Request:

```
{
  operation: "ring-info"
  payload: {
      "clientId": 2
  }
}
```

- Response:

```
{
  status : ok ,
  payload:{
      prev : 1,
      next : 3
  }
}
```

```
{
  status: error,
  message : "Client-ID nicht vorhanden
}
```

client-info(clientId)

- liefert IP + Port eines Clients (für Token-Weitergabe)

- Request:

```
{
  operation : client-info
  payload : 3
}
```

- Response:

```
{
  status : ok
  payload: {
      ip : "127.0.0.1",
      port: 8003
  }
}
```

```
{ status : error
  message : Client mit ID "3" nicht gefunden
}
```

Robot

Move:

- Bewegt den Roboter auf "75" % in allen Achsen

```
{
  operation : move
  payload : 75
}
```

Shutdown:

- Fährt den Roboter herunter und meldet ihn ab

Token-Übergabe

Token-Nachricht (Client <-> Client):

```
{
  operation : token,
  payload: {
      currentHolderId :"Client1"
  }
}
```

- empfangende Client erhält Zugriff auf den Roboterarm
- Nach Ausführung eines Befehls wird das Token weitergegeben
- Falls kein Befehl vorliegt, wird das Token automatisch weitergeleitet

Token-Ring-Design:
- Token wird nicht vom Registry-Service verwaltet
- Registry stellt nur IDs und Topologieinfos bereit
- Clients kommunizieren Peer-to-Peer
- AtomicBoolean stellt Token-Zustand thread sicher dar

Fehlerverhalten

- doppelte Namen
  - Fehler bei Registry
- ungültige Operation
  - "unknown_operation"
- ungültiger Typ
- Nicht existierender Client
  - Fehler bei Client_Info
- Netzwerkfehler
  - lokale Fehlermeldung, System bleibt stabil
- ungültiger Eintrag
- ungültige ID

Nebenläufigkeit

- Thread pro Verbindung
  - pro Client-Connection
  - einfach + gut verständlich

Persistenz

- in HashMap speichern
  - Arbeitsspeicher