

Chronolite

Dokumentation der Referenzimplementierung
Chronolite-App · Chronolite Data Hub · IoT Controller

Stand: 15. Januar 2025

Chronolite — Referenzimplementierung

Referenzimplementierung der zentralen Komponenten des Forschungsprojekts **Chronolite**: der **Chronolite-App**, des **Chronolite Data Hub** und eines **IoT Controllers** zur Ansteuerung der Raumbeleuchtung.

Ziel des vom Bundesministerium für Digitales und Staatsmodernisierung (BMDs) geförderten Projekts Chronolite war es, chronobiologisch wirksame, vernetzte Beleuchtung für den Mobilitätssektor (Flugzeug, Transitbereich, Auto, Bahn) nutzbar zu machen. Diese Referenzimplementierung veröffentlicht zum Projektabschluss den grundlegenden Aufbau und die Wirkprinzipien von App und Hub in einer Form, die jede*r Interessierte lokal ausführen und nachvollziehen kann.

Einordnung: Referenz vs. Projektssystem

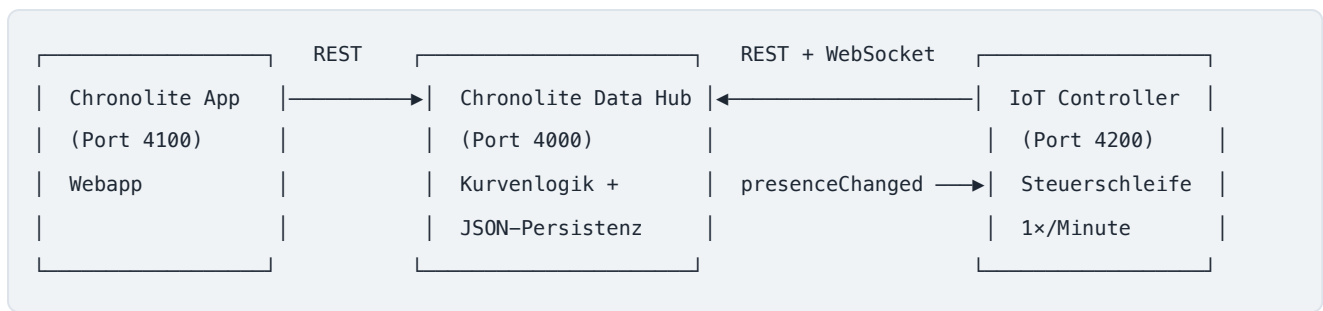
Im Förderprojekt kam eine **komplexere, hochskalierbare Variante** des Systems zum Einsatz: eine cloudbasierte, serverlose Plattform mit verwalteten Datenbanken und nativen iOS-/Android-Apps. Für die Veröffentlichung wurde das System bewusst **auf Einfachheit und Nachvollziehbarkeit hin vereinfacht**:

Aspekt	Projektsystem	Referenzimplementierung
App	Native iOS-/Android-App	Lokale Webapp (Vite + React)
Data Hub	Serverlose Cloud-Plattform (hochskalierbar)	Ein lokaler Express-Prozess
Persistenz	Verwaltete Cloud-Datenbanken	Lesbare JSON-Datei
Raumerkennung	QR-Code, GPS, BLE-Raumbeacons	Manuelle Auswahl aus 3 Demo-Räumen
Raumverwaltung	Registrierung über API/Portal	Statische Liste
Lichtsensorik	Reale/infrastrukturgebundene/virtuelle Sensoren	Einfaches virtuelles Tageslichtmodell (Mock)
Leuchtenansteuerung	IoT-Protokolle (z. B. DALI/KNX, Kabinensysteme)	Formatierte Konsolenausgabe

Die App ist hier **bewusst keine native Mobile-App**: Native Apps erfordern ein komplexes lokales Entwickler-Setup (Xcode/Android Studio, Signierung, Emulatoren) und müssten kontinuierlich an neue OS- und Toolchain-Versionen angepasst werden — beides widerspricht dem Ziel einer dauerhaft einfach ausführbaren Referenz. Die Wirkprinzipien (anonyme Registrierung, Raumzuordnung, Präferenzen, Routinen, Hub-Anbindung) sind identisch umgesetzt.

Die Referenzimplementierung hat **keinerlei Abhängigkeiten zu externen oder verwalteten Diensten** — alles läuft lokal mit Node.js.

Komponenten



Komponente	Pfad	Port	Beschreibung
Chronolite App	<code>apps/app</code>	4100	Anonyme Registrierung, Raumauswahl, Präferenzen, Routinen, Kurven-Visualisierung
Chronolite Data Hub	<code>apps/hub</code>	4000	REST-API + WebSocket, Präferenz-/Routinen-Speicher, Kurvengenerierung
IoT Controller	<code>apps/iot-controller</code>	4200	Hört auf Anwesenheitsänderungen, berechnet minütlich die Lichtwerte
Kurven-Bibliothek	<code>packages/curves</code>	—	Polynom-Kurvengenerierung („Curvis“) und Personalisierungs-Pipeline

Schnellstart

Voraussetzungen: [Node.js](#) ≥ 20 und [pnpm](#) ≥ 9.

```
pnpm install  # Abhängigkeiten installieren
pnpm dev     # Hub, IoT Controller und App gemeinsam starten
```

Danach die App im Browser öffnen: <http://localhost:4100>

Ausführliche Anleitung inkl. Demo-Walkthrough: <docs/INSTALLATION.md>

Demo in 2 Minuten

1. Die App öffnet sich, registriert sich anonym am Hub und speichert die erhaltene ID im Local Storage des Browsers.
2. Unter **Persönliche Präferenzen** Chronotyp (MSFsc), Geburtsjahr und Lichtfarb-Präferenz speichern.
3. Unter **Routinen** z. B. „20:00 — Lesen — Entspannung“ anlegen.
4. Den Raum **Flughafen-Gate** betreten.
5. Im Terminal beobachten, wie der IoT Controller über den WebSocket benachrichtigt wird, die personalisierten Kurven lädt und minütlich neue Lichtwerte (Dimmwert + Farbtemperatur) „an die Beleuchtung übergibt“ (Konsolenausgabe).
6. Den Raum verlassen → der Controller fällt auf die Basiskurven zurück.

Dokumentation

- [docs/INSTALLATION.md](#) — Installation, Start, Troubleshooting
- [docs/ARCHITEKTUR.md](#) — Systemaufbau, Datenfluss, Datenschutz, Vereinfachungen
- [docs/ALGORITHMEN.md](#) — Kurvengenerierung (Tagesverlauf) und Steuerungslogik, mit Diagrammen
- [docs/API.md](#) — REST- und WebSocket-API des Data Hub mit Beispielen

Entwicklung

```
pnpm lint      # Biome (Linter + Formatter)
pnpm test      # Unit-Tests der Kurven-Bibliothek (Vitest)
```

Alle Abhängigkeiten sind **exakt auf die im Januar 2025 verfügbaren Versionen fixiert**, damit die Referenzimplementierung reproduzierbar bleibt.

Lizenz

MIT

Installation & Start

Diese Anleitung führt Schritt für Schritt durch Installation und Start der Chronolite-Referenzimplementierung. Es werden **keine externen Dienste, Konten oder Cloud-Ressourcen** benötigt — alles läuft lokal.

1. Voraussetzungen

Werkzeug	Version	Prüfen mit
Node.js	≥ 20 (LTS)	<code>node --version</code>
pnpm	≥ 9	<code>pnpm --version</code>

pnpm lässt sich am einfachsten über Corepack aktivieren, das bei Node.js mitgeliefert wird:

```
corepack enable
```

2. Installation

Im Wurzelverzeichnis des Repositories:

```
pnpm install
```

Das installiert die Abhängigkeiten aller Workspace-Pakete. Alle Versionen sind exakt fixiert (Stand Januar 2025), sodass die Installation reproduzierbar ist.

3. Start

Variante A: Alles auf einmal

```
pnpm dev
```

Startet Data Hub, IoT Controller und App gemeinsam (farblich getrennte Log-Ausgaben: `hub`, `controller`, `app`).

Variante B: Jede Komponente in einem eigenen Terminal

```
# Terminal 1 – Chronolite Data Hub (zuerst starten)
pnpm --filter @chronolite/hub start

# Terminal 2 – IoT Controller (Standard-Raum: gate)
pnpm --filter @chronolite/iot-controller start

# Terminal 3 – Chronolite App
pnpm --filter @chronolite/app dev
```

Der IoT Controller lässt sich per Umgebungsvariable auf einen anderen Raum setzen:

```
ROOM_ID=train pnpm --filter @chronolite/iot-controller start
```

Feste Ports

Komponente	URL
Chronolite Data Hub (REST)	http://localhost:4000
Chronolite Data Hub (WebSocket)	ws://localhost:4000/ws
Chronolite App	http://localhost:4100
IoT Controller (Status)	http://localhost:4200/status

4. Demo-Walkthrough

1. **App öffnen:** <http://localhost:4100>. Die App registriert sich automatisch anonym am Hub; die erhaltene ID wird oben angezeigt und im Local Storage des Browsers gespeichert.
2. **Präferenzen speichern:** Chronotyp als MSFsc-Wert eintragen (den eigenen Chronotyp bestimmen: [Was ist dein Chronotyp?](#)), Geburtsjahr und Lichtfarb-Präferenz setzen, speichern.
3. **Routine anlegen:** z. B. `20:00 / Lesen / Entspannung`.
4. **Raum betreten:** „Flughafen-Gate“ auswählen. Die App meldet die Anwesenheit an den Hub; der Hub benachrichtigt den IoT Controller über den WebSocket.
5. **Controller beobachten:** Im Terminal des Controllers erscheint

```
[controller:gate] Presence changed (1 present) – refreshing curves
[controller:gate] Curves updated – personalised for first user <id>
[controller:gate] 14:31:26 | target mEDI 395 lx | daylight 452 lx | >>> set luminaires to 0 % @ 563
```

Die letzte Zeile ist die (in der Referenz als Konsolenausgabe umgesetzte) Übergabe der Sollwerte an die Raumbeleuchtung — einmal pro Minute neu berechnet.

6. **Kurven ansehen:** In der App zeigt das Diagramm „Tageskurven im aktuellen Raum“ die beiden vom Hub gelieferten Polynomkurven (Ziel-mEDI und CCT-Präferenz, 0–24 Uhr). Präferenzen oder Routinen ändern und beobachten, wie sich die Kurven verschieben.
7. **Raum verlassen:** Der Controller erhält erneut eine Benachrichtigung und fällt auf die Basiskurven zurück.
8. **Status prüfen (optional):** <http://localhost:4200/status> zeigt den letzten Steuerungs-Tick als JSON.

5. Tests & Linting

```
pnpm test    # Unit-Tests der Kurven-Bibliothek (Vitest)
pnpm lint    # Biome-Linter und -Formatter
```

6. Troubleshooting

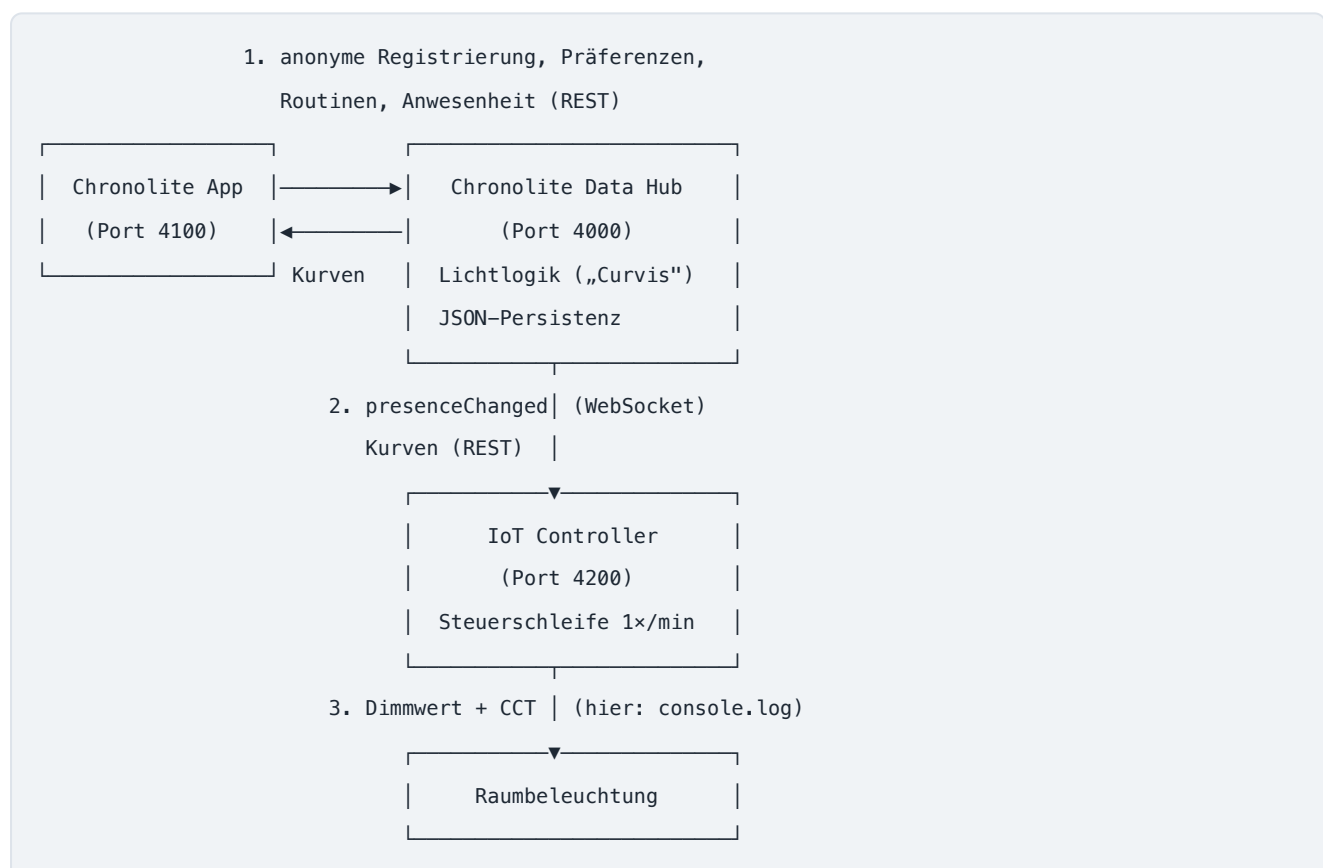
Problem	Ursache / Lösung
App zeigt „Der Chronolite Data Hub ist nicht erreichbar“	Hub zuerst starten (Port 4000), dann die App neu laden.
<code>EADDRINUSE</code> beim Start	Einer der festen Ports (4000/4100/4200) ist belegt. Den blockierenden Prozess beenden (<code>lsof -i :4000</code>).
Controller loggt „Could not fetch curves“	Hub läuft nicht oder <code>HUB_URL</code> zeigt auf eine falsche Adresse.
Gespeicherte Daten zurücksetzen	Hub stoppen und <code>apps/hub/data/hub-data.json</code> löschen. Die App registriert sich beim nächsten Laden automatisch neu.
pnpm nicht gefunden	<code>corepack enable</code> ausführen oder pnpm gemäß https://pnpm.io/installation installieren.

Architektur

Überblick

Das Chronolite-Ökosystem verbindet drei Rollen zu einem geschlossenen Regelkreis für chronobiologisch wirksame Beleuchtung:

1. **Chronolite App** — das nutzerzentrierte Frontend. Hier geben Nutzer*innen ihre Präferenzen (Chronotyp, Geburtsjahr, Lichtfarb-Präferenz) und Routinen ein und melden ihren Aufenthaltsort (Raum).
2. **Chronolite Data Hub** — die zentrale, datensparsame Plattform. Sie speichert die anonymen Profile, verwaltet die Anwesenheit in Räumen und erzeugt daraus mit der Chronolite-Lichtlogik personalisierte **Tageskurven** (Subsystem „Curvis“).
3. **IoT Controller** — die lokale Steuerungsinstanz eines Raums. Er abonniert Anwesenheitsänderungen seines Raums, holt sich die aktuellen Kurven und rechnet sie minütlich in konkrete Stellwerte (Dimmwert, Farbtemperatur) für die Raumbeleuchtung um.



Datenfluss einer typischen Sitzung

1. Die App registriert sich anonym (`POST /api/users`) und erhält eine zufällige ID, die sie im Local Storage ablegt.

2. Die Nutzerin hinterlegt Präferenzen (`PUT /api/users/:id/preferences`) und Routinen (`POST /api/users/:id/routines`).
3. Sie betritt einen Raum (`POST /api/rooms/:roomId/presence`). Im realen System wird dieser Schritt durch QR-Code-Scan, GPS-Geofencing oder BLE-Raumbeacons ausgelöst; die Referenzimplementierung nutzt eine manuelle Auswahl aus drei Demo-Räumen.
4. Der Hub benachrichtigt alle WebSocket-Abonnenten des Raums (`{"type": "presenceChanged", ...}`).
5. Der IoT Controller des Raums lädt daraufhin die aktuellen Kurven (`GET /api/rooms/:roomId/curves`). Der Hub personalisiert sie für die **erste anwesende Person** (siehe „Bewusste Vereinfachungen“).
6. Die Steuerschleife des Controllers wertet die Kurven einmal pro Minute an der aktuellen Tagesminute aus, zieht den (gemockten) Tageslichtanteil ab und übergibt Dimmwert und Farbtemperatur an die Beleuchtung — in der Referenz als formatierte Konsolenausgabe.
7. Beim Verlassen des Raums wiederholt sich Schritt 4–6; ohne anwesende Person liefert der Hub die Basiskurven.

Komponenten im Detail

`packages/curves` — geteilte Kurven-Bibliothek

Die fachliche Kernlogik ist bewusst in ein eigenes, von allen drei Anwendungen genutztes Paket ausgelagert:

- `polygen.js` — erzeugt aus Stützpunkten `[Tagesminute, Wert]` stückweise Polynomfunktionen (Newton-Interpolation). Kurven werden als Polynom-Koeffizienten über die API transportiert: kompakt, auflösungsunabhängig und von Controllern mit wenigen Multiplikationen auswertbar.
- `baseCurves.js` — die Basiskurven (Ziel-mEDI und CCT-Präferenz) für eine Referenzperson.
- `adapt.js` — die Personalisierungs-Pipeline (Chronotyp, Alter, Jahreszeit, Routinen, Lichtfarb-Präferenz), siehe [ALGORITHMEN.md](#).

`apps/hub` — Chronolite Data Hub

Ein einzelner Express-Prozess mit REST-API und WebSocket-Endpunkt (vollständige API: [API.md](#)). Die Persistenz ist eine einzelne, menschenlesbare JSON-Datei (`apps/hub/data/hub-data.json`), die nach jeder Änderung geschrieben wird.

`apps/app` — Chronolite App

Eine bewusst schlichte React-Single-Page-App (kein Router, kein State-Management-Framework). Sie visualisiert zusätzlich die vom Hub gelieferten Kurven mit Chart.js — so wird das Wirkprinzip der Personalisierung unmittelbar sichtbar.

`apps/iot-controller` — IoT Controller

Ein Prozess pro Raum (`R00M_ID` -Umgebungsvariable). Verbindungsverluste zum Hub werden automatisch überbrückt (Reconnect + erneutes Abonnieren + Kurven-Resync). Ein kleiner Status-Endpunkt (`GET :4200/status`) macht den letzten Steuerungs-Tick inspizierbar.

Datenschutz (Privacy by Design)

Der Hub kennt **keinerlei personenbezogene Daten**: keine Namen, keine Konten, keine Kontaktdaten. Apps registrieren sich anonym und erhalten eine zufällige UUID; alle Präferenzen und Routinen sind ausschließlich an diese ID gebunden. Die Raumzuordnung speichert nur „ID X ist in Raum Y“ und wird beim Verlassen wieder gelöscht. Dieses Prinzip stammt unverändert aus dem Projektsystem und ist eines der zentralen Wirkprinzipien, die diese Referenz demonstriert.

Bewusste Vereinfachungen gegenüber dem Projektsystem

Im Förderprojekt lief eine komplexere, hochskalierbare Variante (serverlose Cloud-Plattform, native Apps, reale Sensorik und Leuchten-Gateways). Für die Nachvollziehbarkeit wurde hier vereinfacht:

Thema	Vereinfachung	Im Projektsystem
Skalierung	Ein Prozess, JSON-Datei	Serverlose Cloud-Architektur, verwaltete Datenbanken
App	Lokale Webapp	Native iOS-/Android-App
Raumerkennung	Manuelle Auswahl (3 Demo-Räume)	QR-Code, GPS, BLE-Raumbeacons
Raumverwaltung	Statische Liste im Code	Registrierung über API/Portal
Chronotyp-Erhebung	Direkte MSFsc-Eingabe mit Verweis auf externen Test	Vollständiger MCTQ-Fragebogen in der App
Mehrpersonen-Räume	Kurven der ersten Person; leerer Raum → Basiskurven. Das Zusammenführen der Präferenzen mehrerer Personen in einem Raum wurde im Rahmen des Projekts noch nicht definiert.	offen (Forschungsfrage)
Helligkeitsmessung	Virtueller Tageslicht-Mock	Sensor-Hierarchie (individuell / infrastrukturegebunden / virtuell)
Leuchtenansteuerung	<code>console.log</code>	IoT-Protokolle (z. B. DALI/KNX, Kabinenbeleuchtungssysteme)
Authentifizierung	keine (lokaler Demo-Betrieb)	abgesicherte APIs

Erweiterungspunkte

- **Reale Leuchten:** Im Controller die Ausgabe in `controlTick()` durch eine Anbindung an das jeweilige Lichtsteuerungssystem ersetzen.

- **Reale Sensorik:** `apps/iot-controller/src/sensor.js` gegen echte Messwerte oder ein besseres virtuelles Sensormodell austauschen.
- **Automatische Raumerkennung:** In der App die manuelle Raumauswahl durch QR-Scan/GPS/BLE ersetzen; die Hub-API bleibt unverändert.
- **Präferenz-Fusion:** Die „erste Person“-Regel im Hub (`GET /api/rooms/:roomId/curves`) durch eine Strategie zum Zusammenführen mehrerer Profile ersetzen.

Algorithmen der Chronolite-Lichtlogik

Dieses Dokument beschreibt die beiden zentralen Abläufe der Referenzimplementierung:

1. die **Generierung des personalisierten Tagesverlaufs** (zwei Kurven) im Data Hub und
2. die **Steuerungsschleife** im IoT Controller, die daraus konkrete Lichtwerte macht.

Grundbegriffe

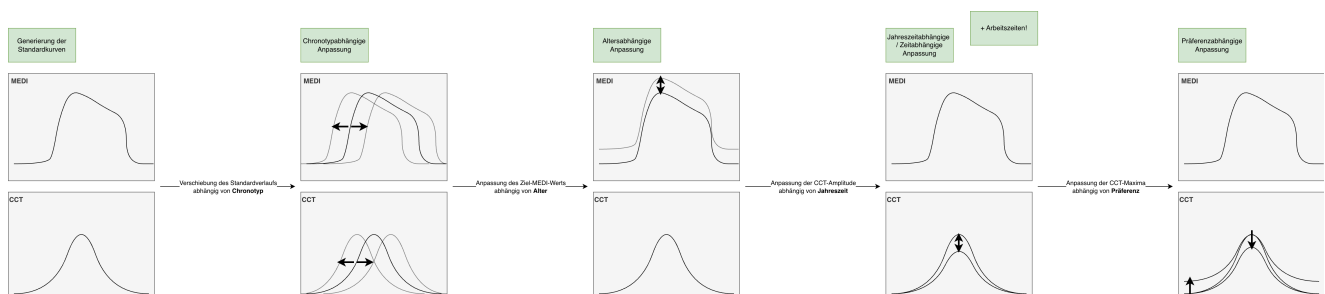
- **mEDI** (melanopisch äquivalente Tageslicht-Beleuchtungsstärke, lx am Auge, nach DIN/TS 5031-100): Maß für die biologische Wirksamkeit von Licht auf die innere Uhr. Tagsüber sollte sie hoch sein (stabilisiert den zirkadianen Rhythmus, fördert Wachheit), abends niedrig (ermöglicht Melatoninausschüttung und Einschlafen).
- **CCT** (Correlated Colour Temperature, Kelvin): die Farbtemperatur des Lichts — warmweiß (≈ 2700 K) bis tageslichtweiß (≈ 6500 K).
- **Chronotyp (MSFsc)**: individuelles Timing der inneren Uhr, ermittelt per Munich Chronotype Questionnaire (MCTQ) als „Mitte des Schlafs an freien Tagen, schlafkorrigiert“ in Uhr-Stunden. MSFsc ≈ 4 Uhr ist der Durchschnitt; kleinere Werte = Frühtypen („Lerchen“), größere = Spättypen („Eulen“).

1. Generierung des Tagesverlaufs (Data Hub, `packages/curves`)

Der Hub liefert pro Raum zwei Kurven für den aktuellen Tag, jeweils als **stückweise Polynomfunktionen** über die Tagesminuten 0–1439 (0–24 Uhr):

- **Ziel-mEDI-Kurve** — wie viel biologisch wirksames Licht die Person zu jeder Tageszeit erhalten soll, und
- **CCT-Präferenz-Kurve** — welche Farbtemperatur zu jeder Tageszeit passend ist.

Der Ablauf folgt diesem Schema:



(editierbare draw.io-Quellen liegen im Repository unter docs/assets/)

Schritt 1 — Basiskurven laden

Ausgangspunkt sind die Standardkurven einer Referenzperson (durchschnittlicher Chronotyp, ca. 30 Jahre, keine besonderen Präferenzen), definiert als Stützpunkte in

`packages/curves/src/baseCurves.js` :

- mEDI: nachts nahe 0 lx, steiler Anstieg am Morgen, Tagesplateau ≈ 250 lx, Abfall am Abend auf melatoninfeindliche Werte.
- CCT: warmweiß (2700 K) nachts und abends, glockenförmiger Anstieg auf ≈ 6000 K zur Mittagszeit.

Schritt 2 – Chronotypabhängige Verschiebung (X-Achse)

Beide Kurven werden entlang der Zeitachse verschoben: Für einen Spättyp (z. B. MSFsc = 6) beginnt der „Lichttag“ zwei Stunden später, für einen Frühtyp entsprechend früher. Verschiebung = $(\text{MSFsc} - 4, 0) \times 60$ Minuten; die Tagesgrenzen (0:00/23:59) werden mit den Randwerten neu verankert. →

`shiftByChronotype()` in `packages/curves/src/adapt.js`

Schritt 3 – Altersabhängige Anpassung des Ziel-mEDI

Mit zunehmendem Alter sinkt die Transmission der Augenlinse; für dieselbe biologische Wirkung wird mehr Licht benötigt. Die gesamte mEDI-Kurve wird daher **durchgehend angehoben**: +1 % pro Lebensjahr über 30, gedeckelt beim Zweifachen des Basiswerts. → `adjustMediForAge()`

Schritt 4 – Jahreszeitabhängige Anpassung der CCT-Amplitude

Die Amplitude der CCT-Kurve (Abstand zur warmen Basislinie 2700 K) wird über den Tag des Jahres skaliert: im Sommer ausgeprägter (Faktor $\approx 1,1$), im Winter flacher und wärmer (Faktor $\approx 0,85$), dazwischen kosinusförmig interpoliert. → `adjustCctAmplitudeForSeason()`

Schritt 5 – Anpassung anhand der Routinen

Jede Routine verändert die CCT-Kurve lokal in einem Fenster von ± 90 Minuten um ihre Uhrzeit:

- **Aktivität** (z. B. Arbeiten, Sport) → Farbtemperatur wird um bis zu 600 K **angehoben** (aktivierend),
- **Entspannung** (z. B. Lesen, Abendruhe) → Farbtemperatur wird um bis zu 600 K **abgesenkt** (beruhigend).

Der Effekt läuft zu den Fensterrändern linear aus; außerhalb bleibt die Kurve unverändert. →

`applyRoutinesToCct()`

Schritt 6 – Präferenzabhängige Anpassung der CCT-Maxima

Die persönliche Lichtfarb-Präferenz (-1 = „Ich mag sehr kaltes/blauess Licht nicht“ bis $+1$ = „Ich mag kühles Licht“) skaliert die Kurve oberhalb der warmen Basislinie so um, dass das Maximum auf den bevorzugten Wert abgesenkt (bis -1200 K) bzw. angehoben (bis $+500$ K) wird — die Kurvenform bleibt erhalten, physikalische Grenzen (2700–6500 K) werden eingehalten. → `applyCctPreference()`

Ausgabe über die API

Die angepassten Stützpunkte werden per Newton-Interpolation in stückweise Polynome überführt (`computeCurve()` in `packages/curves/src/polygen.js`) und über `GET /api/rooms/:roomId/curves` ausgegeben:

```
{
  "roomId": "gate",
  "date": "2025-01-15",
  "basedOnUser": "4a30...",
  "medi": { "segments": [{ "coeffs": [1.36, 0, 0], "start": 0, "end": 420 }, "..."] },
  "cct": { "segments": ["..."] }
}
```

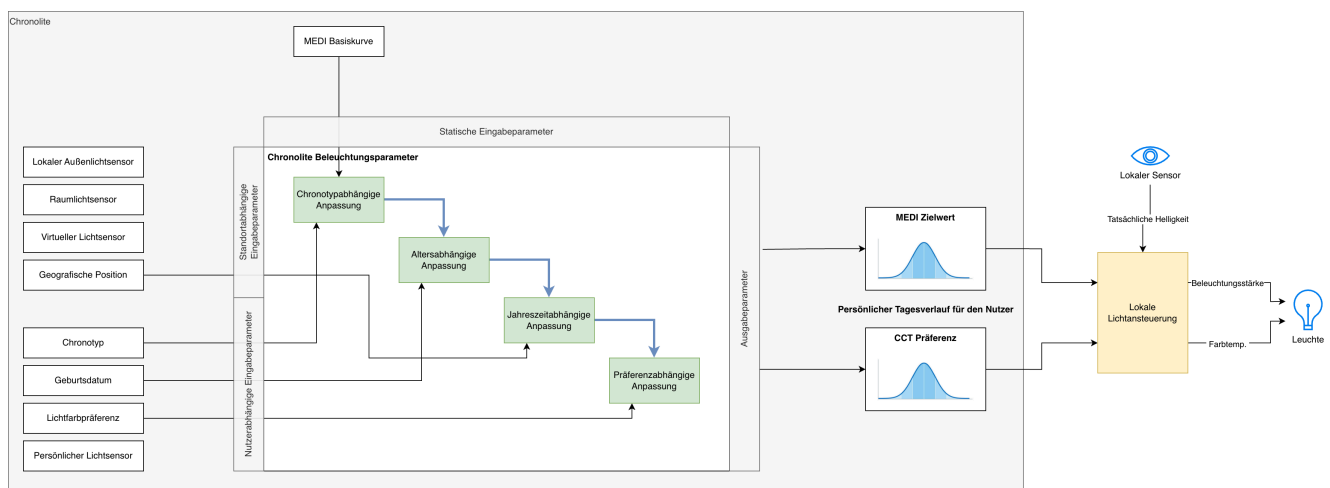
Jedes Segment ist ein Polynom $f(x) = \sum coeffs[i] \cdot x^i$, gültig für `start ≤ x ≤ end` (x = Tagesminute).

Wessen Kurven bekommt der Raum?

Die Kurven werden für die **erste im Raum anwesende Person** personalisiert. Ist niemand (mit hinterlegten Präferenzen) anwesend, liefert der Hub die Basiskurven. Das Zusammenführen der Präferenzen **mehrerer** Personen in einem Raum wurde im Rahmen des Projekts noch nicht definiert und ist bewusst als Erweiterungspunkt offen gelassen.

2. Steuerungsschleife (IoT Controller)

Die Umsetzung der Kurven in konkrete Lichtwerte folgt diesem Schema:



(editierbare draw.io-Quellen liegen im Repository unter docs/assets/)

Der Controller (`apps/iot-controller/src/controller.js`) arbeitet ereignis- und zeitgesteuert:

- **Ereignis:** Über den WebSocket des Hubs abonniert er Anwesenheitsänderungen seines Raums. Bei jeder Änderung (Betreten/Verlassen) lädt er die Kurven neu und wendet sie sofort an.

- **Zeit:** Eine Dauerschleife berechnet **einmal pro Minute** die aktuellen Lichtwerte:

1. **Aktuelle Kurven auswerten:** Ziel-mEDI und Ziel-CCT an der aktuellen Tagesminute (`evaluateSegments()`).
2. **Tatsächliche Helligkeit im Raum abfragen:** In der Referenzimplementierung ein virtueller Sensor (`sensor.js`), der den Tageslichtanteil aus einem einfachen Sonnenstandsmodell und einem raumabhängigen Tageslichtfaktor schätzt. Im Projektsystem stehen hier reale, infrastrukturegebundene oder virtuelle Sensoren.
3. **Stellwerte berechnen und übergeben:** Das Kunstlicht muss nur die Lücke zwischen Ziel-mEDI und vorhandenem Tageslicht füllen:

```
fehlendes mEDI = max(0, Ziel-mEDI - Tageslicht-mEDI)
Dimmwert       = fehlendes mEDI / Leuchten-Maximalbeitrag × 100 %
Farbtemperatur = Ziel-CCT
```

Die Übergabe an die Beleuchtung erfolgt in der Referenzimplementierung als formatierte Konsolenausgabe; in einer realen Installation stünde hier die Anbindung an das Lichtsteuerungssystem des Raums (z. B. DALI/KNX-Gateways oder Kabinenbeleuchtungssysteme):

```
[controller:gate] 14:31 | target mEDI 395 lx | daylight 452 lx | >>> set luminaires to 0 % @ 5635 K
```

Der letzte Steuerungs-Tick ist jederzeit unter `GET http://localhost:4200/status` abrufbar.

API des Chronolite Data Hub

Basis-URL: `http://localhost:4000` — WebSocket: `ws://localhost:4000/ws`

Alle Endpunkte sprechen JSON. Fehler werden als `{ "error": "..." }` mit passendem HTTP-Statuscode (400/404) beantwortet. Der Hub kennt keine personenbezogenen Daten; sämtliche Nutzerdaten hängen an anonymen, zufällig vergebenen IDs.

Nutzer

`POST /api/users` — anonyme Registrierung

Registriert eine neue App-Instanz und liefert die anonyme Nutzer-ID, die die App lokal (Local Storage) speichert.

```
curl -X POST http://localhost:4000/api/users
```

```
{ "userId": "4a3093d0-65b8-4872-8d19-7be536dbea7e" }
```

Präferenzen

`PUT /api/users/:userId/preferences` — Präferenzen speichern

Feld	Typ	Bedeutung
<code>chronotype</code>	number (0–12)	Chronotyp als MSFsc-Wert in Uhr-Stunden (per MCTQ ermittelt)
<code>birthYear</code>	number	Geburtsjahr; der Hub leitet daraus das Alter ab
<code>cctPreference</code>	number (–1...+1)	Lichtfarb-Präferenz: –1 „mag sehr kaltes/blaues Licht nicht“, 0 neutral, +1 „mag kühles Licht“

```
curl -X PUT http://localhost:4000/api/users/$USER_ID/preferences \
-H 'Content-Type: application/json' \
-d '{"chronotype": 6, "birthYear": 1960, "cctPreference": -0.5}'
```

`GET /api/users/:userId/preferences` — Präferenzen abrufen

Liefert die gespeicherten Präferenzen oder `null`, wenn noch keine gesetzt wurden.

Routinen

POST /api/users/:userId/routines — Routine anlegen

Feld	Typ	Bedeutung
time	string "HH:MM"	Uhrzeit der Routine
label	string	Tätigkeit (frei wählbar)
category	"activity" "relaxation"	Aktivität → lokal kühleres Licht, Entspannung → lokal wärmeres Licht

```
curl -X POST http://localhost:4000/api/users/$USER_ID/routines \  
-H 'Content-Type: application/json' \  
-d '{"time": "20:00", "label": "Lesen", "category": "relaxation"}'
```

```
{ "id": "c01fcf0f-...", "time": "20:00", "label": "Lesen", "category": "relaxation" }
```

GET /api/users/:userId/routines — Routinen auflisten

DELETE /api/users/:userId/routines/:routineId — Routine löschen (204)

Räume & Anwesenheit

GET /api/rooms — Raumliste

Die Referenzimplementierung nutzt eine statische Liste von drei Demo-Räumen an einem Flughafen (im realen System werden Räume über eine API bzw. ein Portal registriert).

```
curl http://localhost:4000/api/rooms
```

```
[  
  { "id": "gate", "name": "Flughafen-Gate", "description": "...", "occupantCount": 0 },  
  { "id": "cubicle", "name": "Ruhe-Cubicle", "description": "...", "occupantCount": 0 },  
  { "id": "train", "name": "Bahnwaggon", "description": "...", "occupantCount": 0 }  
]
```

POST /api/rooms/:roomId/presence — Anwesenheit melden

Meldet, dass eine Nutzer-ID den Raum betreten hat (im realen Einsatz ausgelöst durch QR-Scan, GPS oder BLE-Beacon). Eine ID kann nur in einem Raum gleichzeitig anwesend sein; ein Betreten verlässt implizit alle anderen Räume.

```
curl -X POST http://localhost:4000/api/rooms/gate/presence \
-H 'Content-Type: application/json' \
-d '{"userId": \"$USER_ID\"}'
```

DELETE /api/rooms/:roomId/presence/:userId — Verlassen melden (204)

Beide Presence-Endpunkte lösen eine `presenceChanged`-Nachricht an alle WebSocket-Abonnenten des Raums aus.

Beleuchtungskurven

GET /api/rooms/:roomId/curves?date=YYYY-MM-DD — Tageskurven abrufen

Liefert die beiden Kurven des Raums für den angegebenen Tag (Standard: heute) als stückweise Polynomfunktionen — personalisiert für die **erste** anwesende Person mit Präferenzen, sonst die Basiskurven (Details: [ALGORITHMEN.md](#)).

```
curl "http://localhost:4000/api/rooms/gate/curves?date=2025-01-15"
```

```
{
  "roomId": "gate",
  "date": "2025-01-15",
  "basedOnUser": "4a3093d0-...",
  "occupantCount": 1,
  "medi": {
    "segments": [
      { "coeffs": [1.36, 0, 0], "start": 0, "end": 420 },
      { "coeffs": [57.99, -0.607, 0.00112], "start": 420, "end": 510 }
    ]
  },
  "cct": { "segments": ["..."] }
}
```

Auswertung eines Segments: $wert = \sum coeffs[i] \cdot x^i$ für die Tagesminute $start \leq x \leq end$ (0 = 00:00, 1439 = 23:59). Die Bibliothek `@chronolite/curves` stellt dafür `evaluateSegments(segments, minuteOfDay)` bereit.

WebSocket-Protokoll

Endpunkt: `ws://localhost:4000/ws` — alle Nachrichten sind JSON.

Richtung	Nachricht	Bedeutung
Client → Hub	<code>{ "type": "subscribe", "roomId": "gate" }</code>	Anwesenheitsänderungen eines Raums abonnieren
Hub → Client	<code>{ "type": "subscribed", "roomId": "gate" }</code>	Bestätigung des Abonnements
Hub → Client	<code>{ "type": "presenceChanged", "roomId": "gate", "occupantCount": 1 }</code>	Anwesenheit hat sich geändert → Kurven neu laden
Hub → Client	<code>{ "type": "error", "error": "..." }</code>	Ungültige Nachricht oder unbekannter Raum

Beispiel mit `wscat` (oder einem beliebigen WebSocket-Client):

```
$ wscat -c ws://localhost:4000/ws
> {"type":"subscribe","roomId":"gate"}
< {"type":"subscribed","roomId":"gate"}
< {"type":"presenceChanged","roomId":"gate","occupantCount":1}
```

Über das Projekt

Diese Dokumentation gehört zur Referenzimplementierung des Forschungsprojekts **Chronolite**, in dem chronobiologisch wirksame, vernetzte Beleuchtung für den Mobilitätssektor entwickelt und erprobt wurde. Weitere Informationen zum Projekt und zu den beteiligten Partnern finden Sie unter <https://www.chronolite.de/>.

Ein Förderprojekt des BMDV

Das Bundesministerium für Verkehr (BMV) fördert verkehrsträgerübergreifend die Entwicklung und Erprobung innovativer Technologien, welche intelligente Mobilität unterstützen oder ermöglichen. Das Projekt Chronolite ist eines davon. Es wird von der jetlite GmbH aus Hamburg koordiniert und hat ein Gesamtvolumen von ca. 4 Mio. Euro (davon 79,65% Förderanteil durch BMDV). Die Zuwendungen sind Teil der „Innovationsoffensive Künstliche Intelligenz in der Mobilität“, mit der bislang 11 KI-Projekte mit 48 Millionen Euro durch den Bund gefördert werden. Weitere Infos finden Sie auf der Seite des [BMDV](#).

Gefördert durch:



aufgrund eines Beschlusses
des Deutschen Bundestages