Astronomische Beobachtungsliste

Entwickler-Prompt-Dokumentation

Version 32.0 · Stand: 30. April 2026

Einleitung

Dieses Dokument beschreibt alle Entwickler-Prompts, die zur iterativen Entwicklung der Astronomischen Beobachtungsliste verwendet wurden. Jeder Prompt ist aus der Perspektive eines Anwendungsentwicklers formuliert, der ein LLM als Entwicklungspartner einsetzt.

Die Prompts sind chronologisch nach Entwicklungsphasen gegliedert. Sie können als Vorlage dienen, um die Anwendung in einem neuen LLM-Kontext neu aufzubauen oder gezielt Funktionsbereiche zu erweitern.

Legende

- [INIT] — Erstaufbau, Grundstruktur
- [FEATURE] — Neue Funktion hinzufügen
- [REFACTOR] — Umbau / Modularisierung
- [BUGFIX] — Fehlerkorrektur
- [CONFIG] — Konfigurationsdaten ergänzen
- [INTEGRATION] — Externe API / Dienst anbinden
- [DOCS] — Dokumentation erstellen

Phase 1 — Grundstruktur und Katalogverwaltung

P-01 *[INIT]* Grundgerüst: Einzelne HTML-Datei mit Dark-Theme

Prompt

```
Erstelle eine einzelne HTML-Datei "beobachtungsliste.html" für eine
astronomische Beobachtungsliste. Anforderungen:

## Design
- Dark Theme: Hintergrund #0a0d1a, Akzentfarbe #60a5fa (blau)
- Monospace-Schrift (font-family: "Courier New") für Zahlenwerte
- Responsiv, kein externes CSS-Framework

## Felder (Eingabe)
- Beobachtungsort: Breitengrad, Längengrad, UTC-Offset
- Datum (date input)
- Minimale Höhe über Horizont (Slider, 0-60°)

## Tabelle (Ausgabe)
```

```
- Spalten: Name, Typ, Mag, RA, Dec, Aufgang, Kulmination,
  Untergang, Max. Höhe
- Sortierbar durch Klick auf Spaltenköpfe
- Filter: "Nur sichtbare Objekte"

## Daten
- Vorläufig: 5 Beispielobjekte hardcodiert als JS-Array
- Objekt-Felder: name, type, ra (hh:mm:ss), dec (°'"),
  v_mag, size, constellation
```

⚠ Hinweise & bekannte Fallstricke
→ Noch keine externe Bibliothek — alles in einer Datei
→ RA/Dec-Parser muss hh:mm:ss und dd°mm'ss" verarbeiten

P-02 *[FEATURE]* Astronomische Berechnungen: Aufgang, Kulmination, Untergang

Kontext / Voraussetzungen
· P-01 abgeschlossen — HTML-Grundgerüst vorhanden

Prompt

```
Implementiere in der bestehenden HTML-Datei die astronomischen
Berechnungen für Aufgang, Kulmination und Untergang.

## Algorithmus
- Julianisches Datum aus Gregorianischem Datum
- Lokale Sternzeit (LST) für Längengrad
- Höhe über Horizont: sin(h) = sin(φ)·sin(δ) + cos(φ)·cos(δ)·cos(H)
  φ = Breitengrad, δ = Deklination, H = Stundenwinkel
- Aufgang/Untergang: numerische Suche (15-Minuten-Schritte)
- Kulmination: Objekt im Meridian (H = 0)

## Ausgabe
- Lokalzeit (UTC + Offset)
- Objekte unter Minimalhöhe als "nicht sichtbar" kennzeichnen
- Bürgerliche Dämmerung (Sonne bei -6°) berechnen
```

⚠ Hinweise & bekannte Fallstricke
→ Keine externe Bibliothek — alle Formeln selbst implementieren
→ Genauigkeit ±1 Minute ist ausreichend

P-03 *[FEATURE]* JSON-Import mit Feld-Mapping-Dialog

Kontext / Voraussetzungen
· P-02 abgeschlossen

Prompt

```
Erweitere die App um einen JSON-Import für eigene Sternkataloge.

## Import-Dialog
- Datei-Upload (input type="file", .json)
- Nach dem Laden: Feld-Mapping-Dialog
```

```
  → Benutzer ordnet JSON-Felder den App-Feldern zu
  → App-Felder: name, ra, dec, v_mag, size, type, constellation
- Vorschau der ersten 3 Datensätze im Dialog

## Fehlerbehandlung
- JSON-Syntaxfehler: Zeilenummer + Kontextausschnitt anzeigen
- Ungültige Koordinaten: pro Objekt melden, Rest importieren
- Koordinaten-Formate: "hh:mm:ss", "hh mm ss", Dezimalgrad

## Speicherung
- IndexedDB (nicht localStorage — unbegrenzte Kapazität)
- Mehrere Kataloge verwaltbar: Name, Datum, Objektanzahl
- Aktiver Katalog per Dropdown wählbar
```

⚠ Hinweise & bekannte Fallstricke
→ IndexedDB-API ist async — alle Datenbankzugriffe mit await
→ Feld-Mapping in localStorage speichern (Wiederverwendung)
→ Parser-Selbsttest beim Start: 38 Testfälle für Koordinatenformate

P-04 *[FEATURE]* Gespeicherte Beobachtungsorte und Notizen

Kontext / Voraussetzungen
· P-03 abgeschlossen

Prompt

```
Füge zwei Komfort-Features hinzu:

## Gespeicherte Orte
- Bis zu 20 Orte speichern (Name, Lat, Lon, UTC)
- Dropdown-Auswahl lädt Koordinaten in die Felder
- Ortssuche via Nominatim/OpenStreetMap:
  GET https://nominatim.openstreetmap.org/search?q=...&format=json
- Löschen einzelner Orte

## Beobachtungsnotizen
- Pro Objekt: freier Textbereich (Textarea)
- Notiz-Icon in der Tabellenzeile (📝)
- Notiz-Modal: öffnet sich beim Klick auf das Icon
- Speicherung in localStorage: Schlüssel = Objektname

## Fotografiert-Markierung
- Checkbox pro Zeile
- Fotografierte Objekte: andere Zeilenfarbe
- Filter: "Nicht fotografierte ausblenden"
```

Phase 2 — Visualisierung, Export und Mondberechnung

P-05 *[FEATURE]* Höhenkurven-Chart (Canvas) und Export

Kontext / Voraussetzungen

· P-04 abgeschlossen

Prompt

```
Implementiere zwei neue Funktionen:

## Höhenkurven-Chart
- HTML5 Canvas, keine externe Bibliothek
- X-Achse: Zeit von Dämmerung bis Dämmerung (stündlich)
- Y-Achse: Höhe 0–90°
- Je Objekt eine farbige Kurve
- Dämmerungslinien (bürgerlich, nautisch, astronomisch)
- Horizontlinie bei konfigurierbarer Mindesthöhe
- Legende mit Farb-Zuordnung

## Export
- AsiAir CSV: name,ra_h,ra_m,ra_s,dec_d,dec_m,dec_s,mag,type
- N.I.N.A. CSV: Name,RA,Dec,Magnitude,DSO Type
- SkySafari .skylist: XML-Format
- Voyager RoboClip CSV
- Slew-Reihenfolge: Nearest-Neighbour-Optimierung
```

⚠ Hinweise & bekannte Fallstricke
→ Canvas: window.devicePixelRatio für Retina-Displays berücksichtigen
→ Slew-Algorithmus: Startpunkt = aktuell höchstes Objekt

P-06 *[FEATURE]* Mondberechnung nach Jean Meeus

Kontext / Voraussetzungen
· P-05 abgeschlossen

Prompt

```
Implementiere die Mondberechnung nach Jean Meeus "Astronomical
Algorithms", Kapitel 47.

## Berechnungen
- Mondposition: RA, Dec (ekliptikale Länge → äquatoriale Koord.)
- Mondphase: Beleuchtungsgrad in %, Symbol (🌑🌒🌓🌔🌕🌖🌗🌘)
- Mondaufgang / Monduntergang (wie Sternaufgang)
- Winkelabstand Mond–Objekt: Haversine-Formel

## Integration
- Mondkurve im Höhenkurven-Chart (gelb gestrichelt)
- Neue Tabellenspalte "Mond∠": Winkelabstand farbkodiert
  < 15°: rot, 15–30°: orange, > 30°: grün
- Statuszeile: Mondphase + Beleuchtung + Aufgang/Untergang
- Sortierung nach Mondabstand
```

⚠ Hinweise & bekannte Fallstricke
→ Meeus Kap. 47 enthält 60 Terme für Länge und 7 für Breite
→ Genauigkeit: ±0.1° ist ausreichend für Beobachtungsplanung

P-07 *[FEATURE]* Rotlicht-Modus (Night Vision)

Kontext / Voraussetzungen
· P-06 abgeschlossen

Prompt

```
Implementiere einen Rotlicht-Modus für die Nutzung am Teleskop.

## Anforderungen
- CSS filter: sepia(1) saturate(5) hue-rotate(300deg) brightness(0.6)
- Gilt für das gesamte Dokument (html-Element)
- Kein Aufblitzen beim Umschalten (transition: filter 0.3s)
- Zustand in localStorage speichern
- Toggle-Button im Header:  /
- NIGHT VISION Badge im Header sichtbar wenn aktiv
```

Phase 3 — Externe Datenquellen

P-08 *[INTEGRATION]* JPL Horizons — Kometen & Asteroiden

Kontext / Voraussetzungen
· P-07 abgeschlossen

Prompt

```
Integriere JPL Horizons für aktuelle Kometen und Asteroiden.

## API
- Endpoint: https://ssd.jpl.nasa.gov/api/horizons.api
  Typ-Abfrage: ?format=json&COMMAND='DES%3DCOMET'
- Positionsabfrage (sbwobs): standortabhängig
  SITE_COORD = Lon,Lat,0 (Grad)

## CORS
- JPL erlaubt kein direktes fetch() vom Browser
- Proxy: heartsome.de/astronomie/ephemworker/cgi/cors-anywhere.php/
- URL: PROXY + encodeURIComponent(JPL_URL)

## UI
- Neuer Toolbar-Bereich "☄ JPL Horizons"
- Typ-Filter: Kometen / Asteroiden / Beide
- Eigener Tabellenabschnitt mit Erdabstand (AU), galakt. Breite
- Collapse-Button für den Abschnitt
```

⚠ Hinweise & bekannte Fallstricke
→ CORS-Proxy: Accept-Encoding: identity setzen (kein gzip)
→ JPL antwortet mit JSONP — .replace(/^.+?\(*/, "") nötig
→ Rate-Limiting: max. 1 Anfrage/Sekunde

P-09 *[INTEGRATION]* Supernovae (Rochester Astronomy) und NEA-Vorbeiflüge

Kontext / Voraussetzungen
· P-08 abgeschlossen

Prompt

```
Füge zwei weitere externe Datenquellen hinzu:

## Supernovae
- Quelle: http://www.rochesterastronomy.org/snimages/snactive.html
- HTML scrapen (via Proxy), Tabelle parsen
- Felder: Name, Galaxie, RA, Dec, Datum, Magnitude, Typ (Ia/II/Ib/Ic)
- Typ-Badges farbkodiert

## NEA — Near Earth Approaches
- Zweistufig:
  1. JPL CAD API: https://ssd-api.jpl.nasa.gov/cad.api
     Parameter: date-min=heute, date-max=+14d, dist-max=0.1
  2. Für jedes NEA: Horizons-API zum Vorbeiflugdatum
     → RA, Dec, Distanz in AU, Relativgeschwindigkeit
- Toolbar: Zeitfenster (7/14/30 Tage), Max. Distanz (AU), Gruppe
- Farbkodierung nach Distanz: < 0.01 AU rot, < 0.05 gelb
```

⚠ Hinweise & bekannte Fallstricke
→ Beide APIs über heartsome.de Proxy
→ NEA-Abfragen parallel mit Promise.all()
→ Rochester HTML hat kein charset — UTF-8 annehmen

P-10 *[INTEGRATION]* AAVSO VSX — Veränderliche Sterne

Kontext / Voraussetzungen
· P-09 abgeschlossen

Prompt

```
Integriere AAVSO VSX für veränderliche Sterne.

## API
- Endpoint: https://vsx.aavso.org/index.php?view=api.list
- Parameter: ra (Dezimalgrad!), dec (%2B für +), radius, tomag, format=json
- WICHTIG: ra ist in Dezimalgrad (0-360), NICHT in Stunden
- WICHTIG: Feldname im JSON ist "Declination2000", nicht "Decl2000"
- Suchzentrum: LST des Standorts als RA (Meridian)

## UI
- Toolbar: Radius (3-20°), Max. Magnitude (10-14), Typ-Filter
- 17 Variablentyp-Badges farbkodiert:
  M/SR/L: violett, DCEP/CEP: blau, RR/RRAB: cyan,
  EA/EB/EW: orange, N/NR/UG: rot
- Nächstes Maximum berechnen aus Epoche + Periode:
  Epochen < 2400000 = reduziertes HJD → +2400000 addieren
  jdRef ist bereits volles JD — KEIN +2400000.5 Offset!
- Farbampel: ≤3d rot, ≤14d gelb, >14d grün
- VSX↗-Link pro Objekt
```

⚠ Hinweise & bekannte Fallstricke
→ URL manuell bauen: %2B für + in Deklinationen (URLSearchParams kodiert + als Leerzeichen)

→ MaxMag kann Einheit enthalten: "9.0 V" → parseFloat() funktioniert trotzdem
→ HJD-Berechnung: nur 1× testen — Fehler erzeugt Jahreszahlen im Jahr 8000+

P-11 *[INTEGRATION]* SIMBAD Detailinfo via TAP API

Kontext / Voraussetzungen
· P-10 abgeschlossen

Prompt

```
Füge SIMBAD-Detailinfo per Klick auf Objektnamen hinzu.

## API
- Endpoint: https://simbad.cds.unistra.fr/simbad/sim-tap/sync
- Methode: POST (nicht GET — URL wird sonst zu lang)
- Format: application/x-www-form-urlencoded
  Body: REQUEST=doQuery&LANG=ADQL&FORMAT=json&QUERY=...

## ADQL-Query (WICHTIGE Syntax-Regeln für SIMBAD)
- TOP statt LIMIT: "SELECT TOP 1 ..." (SQL-92, kein MySQL)
- Explizite Tabellenpräfixe: "basic.oid = ident.oidref"
- Korrekt: SELECT TOP 1 main_id, ra, dec, otype FROM basic
           JOIN ident ON basic.oid = ident.oidref
           WHERE ident.id = 'NGC 7293'

## CORS-Proxy
- Lokal: python simbad_proxy.py (Port 11113) — bevorzugt
- Fallback: heartsome.de Proxy
- Accept-Encoding: identity — verhindert gzip-Probleme mit PHP-Proxy

## Angezeigt im Modal
- Hauptbezeichnung, Objekttyp (deutsch), Koordinaten J2000
- Helligkeiten B/V/R, Parallaxe + Entfernung in pc
- Radialgeschwindigkeit, Spektraltyp, Alternativnamen
```

⚠ Hinweise & bekannte Fallstricke
→ SIMBAD-Proxy simbad_proxy.py muss do_GET und do_POST implementieren
→ Epoche-Format prüfen: RA/Dec im JSON ist in Grad, nicht Stunden
→ Objekt-Typ-Map: 40+ SIMBAD-Kürzel → deutsche Bezeichnung

Phase 4 — Modularisierung und Architektur

P-12 *[REFACTOR]* Aufteilung in 14 JavaScript-Module

Kontext / Voraussetzungen
· P-11 abgeschlossen — monolithische HTML-Datei ~3000 Zeilen

Prompt

```
Refaktoriere die monolithische HTML-Datei in eigenständige
JavaScript-Module im Unterordner js/

## Modulstruktur
```

```
js/version.js       — APP_VERSION, APP_DATE
js/astro-math.js    — RA/Dec-Parser, JD, LST, Höhenberechnung
js/moon.js          — Mondberechnung (Meeus)
js/ui.js            — Modals, Rotlicht, Collapse, Notizen
js/export.js        — AsiAir/NINA/SkySafari/Voyager CSV
js/chart.js         — Höhenkurven-Canvas
js/jpl.js           — JPL Horizons, NEA, Supernovae, VSX, SIMBAD
js/builtin-catalogs.js — Messier, Caldwell, Herschel 400
js/catalog.js       — IndexedDB, Import, Mapping, Parser
js/location.js      — Orte, Nominatim
js/equipment-config.js — Optiken, Sensoren, Montierungen
js/fov-calculator.js — FOV-Modal-UI
js/alpaca-client.js — ASCOM Alpaca alle 8 Geräteklassen
js/compute.js       — berechne(), renderTable(), DOMContentLoaded

## Ladereihenfolge in HTML
// Wichtig: Abhängigkeiten zuerst laden
<script src="js/version.js"></script>
<script src="js/astro-math.js"></script>
... (wie oben sortiert)
<script src="js/compute.js"></script>
```

⚠ Hinweise & bekannte Fallstricke
→ Keine ES6-Imports verwenden — alle Module als globale Skripte laden
→ Alle Funktionen als globale Variablen/Funktionen deklarieren
→ Auf zirkuläre Abhängigkeiten achten

P-13 *[FEATURE]* Eingebettete Kataloge: Messier, Caldwell, Herschel 400

Kontext / Voraussetzungen
· P-12 abgeschlossen

Prompt

```
Erstelle builtin-catalogs.js mit drei eingebetteten Katalogen.

## Datenformat (je Objekt)
{
  Name: "M 31",
  ra: "00:42:44.4",
  dec: "+41:16:09",
  v_mag: 3.4,
  "Size arcsec": "10800x6300",
  "Object type": "Galaxy",
  Constellation: "And",
  common_names: "Andromeda Galaxy"
}

## Registry
const BUILTIN_KATALOGE = {
  "__messier__":    { name: "Messier (110)", data: KATALOG_MESSIER },
  "__caldwell__":   { name: "Caldwell (109)", data: KATALOG_CALDWELL },
  "__herschel400__":{ name: "Herschel 400",  data: KATALOG_HERSCHEL400 },
};
```

```
## Schutz
- IDs mit __ Präfix/Suffix identifizieren eingebettete Kataloge
- Löschen-Button im UI deaktiviert für eingebettete Kataloge
```

⚠ Hinweise & bekannte Fallstricke
→ Apostroph in common_names (z.B. "Bode's Galaxy") → doppelte Anführungszeichen verwenden
→ Herschel 400 enthält 318 Objekte (nicht 400 — überlappend mit Messier/Caldwell)

Phase 5 — Ausrüstungskonfiguration und FOV-Rechner

P-14 *[FEATURE]* equipment-config.js: Optiken, Sensoren, Montierungen

Kontext / Voraussetzungen
· P-13 abgeschlossen

Prompt

```
Erstelle equipment-config.js als reines Daten- und
Berechnungsmodul (keine DOM-Abhängigkeiten).

## Datenstruktur
const EQUIPMENT_CONFIG = {
  optics:  { id: { name, ap, fl, fixedCamera? } },
  sensors: { id: { name, pixel, qe, dark, sensorW, sensorH } },
  mounts:  { id: { name, type, payload, period, goto } },
  controllers: { id: { name, protocol, defaultHost,
                       alpacaPort, cors, setup[] } },
};

## Optiken (80+)
- Smart Scopes: Seestar S50/S30, Dwarf II/III, Celestron Origin
- Astrographen: ASA, Celestron, ZWO, TS-Optics, Sky-Watcher...
- fixedCamera: Sensor-ID bei Smart Scopes (fest verbaut)

## Sensoren (40+)
- ZWO ungekühlt, ZWO Pro, DSLR/DSLM, weitere

## Berechnungsfunktionen
function fovImageScale(focalMm, pixelMicron)
  → Abbildungsmaßstab in Bogensekunden/Pixel
function fovBildfeld(fl, px, sensorW, sensorH)
  → { wArcmin, hArcmin, wArcsec, hArcsec }
function fovPassung(fovW, fovH, objArcsec)
  → "gut"|"eng"|"knapp"|"zu_gross"|"unbekannt"
function fovMaxBelichtung(mode, fl, px, polErr, dec, allowedPx, trkErr)
  → Sekunden
```

⚠ Hinweise & bekannte Fallstricke
→ Smart-Scope fixedCamera: Sensor-Dropdown im UI sperren wenn gesetzt
→ Montierungen: 30+ inkl. ZWO AM3/5, Sky-Watcher HEQ5/EQ6-R, iOptron CEM-Serie
→ Controller: StellarMate, ASIAIR, OmniSim, Windows ASCOM mit cors-Flag

P-15 *[FEATURE]* FOV-Rechner Modal und Tabellen-Integration

Kontext / Voraussetzungen
· P-14 abgeschlossen

Prompt

```
Erstelle fov-calculator.js und integriere den FOV-Rechner.

## Modal (📐-Button im Header)
- Optik-Dropdown mit <optgroup> pro Kategorie
- Sensor-Dropdown (gesperrt bei Smart Scopes)
- Belichtungsparameter: Modus, Deklination, Polfehler,
  Trackingfehler, erlaubte Drift in Pixel
- Ergebnisanzeige: Bildfeld, Abbildungsmaßstab, f/, max. Belichtung

## Tabellen-Integration
- Neue Spalte "FOV" in jeder Objektzeile
- Klasse "fov-zelle" + data-obj-size Attribut
- Passung-Badge: ✓ passt (grün), → klein (blau),
  ⚠ eng (gelb), ✗ groß (rot)
- Badge aktualisiert sich ohne renderTable() neu aufzurufen
- Header-Badge: zeigt aktuelles Bildfeld "45.3×30.2"

## Persistenz
- Gewählte Optik + Sensor in localStorage (Key: pn_equipment_v1)
```

⚠ Hinweise & bekannte Fallstricke
→ fovPassung(): Objekt füllt >70% FOV = knapp/⚠ eng, >100% = zu_groß
→ Header-Badge per id="fov-badge-text" aktualisieren

Phase 6 — ASCOM Alpaca Teleskopsteuerung

P-16 *[INTEGRATION]* alpaca-client.js: Basis-Teleskopsteuerung

Kontext / Voraussetzungen
· P-15 abgeschlossen

Prompt

```
Erstelle alpaca-client.js mit ASCOM Alpaca HTTP-Client.

## Klassenstruktur
class AlpacaDevice       — Basis: _get(), _put(), connect()
class AlpacaTelescope    — extends AlpacaDevice
class AlpacaCamera       — extends AlpacaDevice
class AlpacaFocuser      — extends AlpacaDevice
class AlpacaFilterWheel  — extends AlpacaDevice
class AlpacaRotator      — extends AlpacaDevice
class AlpacaSwitch       — extends AlpacaDevice
class AlpacaObservingConditions — extends AlpacaDevice
class AlpacaDome         — extends AlpacaDevice
class AlpacaUI           — DOM-Steuerung für das Modal
```

```
## API-Aufruf-Muster
GET  http://{host}:{port}/api/v1/{device}/{n}/{property}
PUT  http://{host}:{port}/api/v1/{device}/{n}/{action}
     Body: application/x-www-form-urlencoded
     ClientID=1&ClientTransactionID=N&{params}

## Polling
- AlpacaTelescope: ra, dec, alt, az, slewing, tracking alle 1.5s
- AlpacaCamera: camerastate, percentcompleted alle 2s
- AlpacaFocuser: position, ismoving alle 2s

## AlpacaLog
- Level 0-5: 0=still, 2=Standard, 5=Polling
- window.AlpacaLog für Browser-Konsole zugänglich
```

⚠ Hinweise & bekannte Fallstricke
→ Fehlerbehandlung: json?.ErrorNumber null-safe, Kleinschreibung als Fallback
→ 8-Sekunden-Timeout auf allen Anfragen
→ Optionale Endpoints (gain, offset etc.) im _onError ignorieren

P-17 *[FEATURE]* GoTo per Zeilen-Klick mit dreistufigem Feedback

Kontext / Voraussetzungen
· P-16 abgeschlossen

Prompt

```
Implementiere GoTo-Steuerung durch Klick auf Tabellenzeilen.

## Trigger
- body erhält Klasse "alpaca-bereit" wenn Teleskop verbunden
- CSS: body.alpaca-bereit .obj-row { cursor: crosshair }
- Klick auf Zeile ruft gotoMitStatus(raDeg, decDeg, name, trEl)
- Ausnahme: Klick auf Checkbox, Button, Icon → kein GoTo

## Drei Phasen
// Phase 1: Slew starten
Badge: "🔭 GoTo NGC 7293 …" (blau, Spinner)

// Phase 2: Polling-Loop (alle 500ms)
Badge: "→ RA 22h30m · Alt 45° · 67% · 12s"
Fortschritt = 1 - (RestAbstand / StartAbstand)

// Phase 3: Angekommen
Badge: "✓ NGC 7293 erreicht · Δ 12″ · 23s"
Verschwindet nach 4 Sekunden

## Hilfsfunktionen
function _slewDistArcsec(raIst, decIst, raSoll, decSoll)
function _slewFortschritt(raStart, decStart, raIst, decIst, raSoll, decSoll)
```

⚠ Hinweise & bekannte Fallstricke
→ Schwebendes Badge: position:fixed, z-index:9999, Bildschirmmitte

→ Zeile blau während Slewing, grün bei Ankunft, rot bei Fehler

P-18 *[FEATURE]* Erweitertes Tab-Panel: Kamera, Fokus, Filter, Wetter

Kontext / Voraussetzungen
· P-17 abgeschlossen

Prompt

```
Erweitere das Alpaca-Modal zu einem 5-Tab-Interface.

## Tab-Struktur
🔭 Teleskop — RA/Dec/Alt/Az, Tracking, Park, Jog
📷 Kamera   — Status, Belichtung, Kühler, Fortschrittsbalken
🔧 Fokus    — Position, absolute/relative Steuerung, TempComp
🎨 Filter   — Filterwahl per Button, Fokus-Offsets
⛅ Wetter   — 9 Sensorkacheln

## Aktivitäts-Indikatoren
// Tab-Dot: pulsierender Punkt wenn Tab aktiv ist
.tab-dot.aktiv { animation: dot-pulse .7s infinite alternate; }

// Activity-Banner unter den Tabs
#alpaca-activity-banner.slewing  { border-left: 3px solid #3b82f6 }
#alpaca-activity-banner.exposing { border-left: 3px solid #22c55e }
#alpaca-activity-banner.focusing { border-left: 3px solid #8b5cf6 }
#alpaca-activity-banner.filter   { border-left: 3px solid #f59e0b }

## Methoden in AlpacaUI
_setTabDot(tabId, state)      — "aktiv"|"fertig"|"fehler"|""
_setActivityBanner(type, text, abortFn)
_abortAktiveAktion()          — Stop-Button Handler

## Robustheit
- setBinning() + setGain() in try/catch (optional, nicht alle Treiber)
- _camWasExposing Flag: Banner genau einmal löschen
```

⚠ Hinweise & bekannte Fallstricke
→ Auto-Erkennung nach Connect: _discoverAndConnectDevices() für alle Geräte
→ Kamera: gain-Feld ausgrauen wenn Treiber Gain nicht unterstützt

P-19 *[CONFIG]* Steuergerät-Profile und CORS-Proxy

Kontext / Voraussetzungen
· P-18 abgeschlossen

Prompt

```
Ergänze equipment-config.js um Controller-Profile und
erstelle alpaca_cors_proxy.py.

## Controller-Profile
```

```
EQUIPMENT_CONFIG.controllers = {
  sm_pro:      { name: "StellarMate Pro",
                 defaultHost: "stellarmate.local",
                 alpacaPort: 11111, cors: true },
  asiair_plus: { name: "ASIAIR Plus",
                 defaultHost: "asiair.local",
                 alpacaPort: 4700,  cors: false },
  omnisim:     { name: "OmniSimulator",
                 defaultHost: "localhost",
                 alpacaPort: 32323, cors: true },
};

## alpaca_cors_proxy.py
- Python http.server, Port 11111
- Auto-Discovery: testet Ports 32323, 11111, 11112, 4700
- OPTIONS-Preflight direkt beantworten
- HTTP-400 von Alpaca-Treiber als JSON {ErrorNumber:1025} zurückgeben
- Polling-Requests (ra, dec, alt, az) im Log unterdrücken
- Argumente: python alpaca_cors_proxy.py [ziel_port] [proxy_port]
```

⚠ Hinweise & bekannte Fallstricke
→ StellarMate: indi_alpaca_server nötig — ist bereits in indi-bin enthalten
→ Start: indi_setprop "INDI Alpaca Server.SERVER_CONTROL.START=On"
→ StellarMate antwortet ohne CORS-Header → Proxy auch dort nötig

Phase 7 — Hilfsskripte und Dokumentation

P-20 *[FEATURE]* simbad_proxy.py — CORS-Proxy für SIMBAD

Kontext / Voraussetzungen
· P-19 abgeschlossen

Prompt

```
Erstelle simbad_proxy.py als dedizierten CORS-Proxy für SIMBAD.

## Anforderungen
- Python http.server, Port 11113
- Unterstützt GET und POST (do_GET und do_POST implementieren!)
- Ziel: https://simbad.cds.unistra.fr
- Kein Accept-Encoding Header weiterleiten
  (verhindert gzip → PHP-Proxy-Probleme)

## Fehlerbehandlung
- urllib.error.HTTPError separat fangen
- SIMBAD gibt bei ADQL-Fehlern XML zurück:
  <INFO name="QUERY_STATUS" value="ERROR">...
- XML-Fehlermeldung extrahieren und als JSON zurückgeben:
  { "ErrorNumber": 400, "ErrorMessage": "..." }

## Debug-Logging
- Request-URL und ersten 200 Zeichen der Antwort ausgeben
- Hilft bei ADQL-Fehlerdiagnose
```

⚠ Hinweise & bekannte Fallstricke
→ do_POST fehlt → Python antwortet mit 501 Not Implemented
→ SIMBAD TAP: POST-Body als application/x-www-form-urlencoded

P-21 *[DOCS]* CHANGELOG.md und Anforderungsdokumentation

Kontext / Voraussetzungen
· App vollständig implementiert — v32 erreicht

Prompt

```
Erstelle zwei Dokumentationsdateien:

## CHANGELOG.md
- Markdown, chronologisch absteigend (neueste Version zuerst)
- Je Version: Versionsnummer, Datum, geänderte Dateien
- Abschnitte: ### Neu, ### Behoben, ### Geändert
- Tabellen für komplexe Inhalte (Modulliste, Log-Level)
- Bekannte Fallstricke (z.B. HJD-Berechnung, ADQL-Syntax)

## anforderungen.docx
- Word-Dokument mit professionellem Layout
- REQn für funktionale Anforderungen (n = 1, 2, 3, ...)
- RQn für nicht-funktionale Anforderungen
- Jede Anforderung: ID, Kurzname, SOPHIST-Satz
- SOPHIST: "Das System MUSS/SOLL/KANN/DARF NICHT [Tätigkeit]."
- Gliederung: Katalog, Berechnungen, Standort, Filter,
  Export, Notizen, Externe Quellen, FOV, Alpaca, UI,
  Technik, Datenhaltung, Zuverlässigkeit, Wartbarkeit
```

P-22 *[DOCS]* Prompt-Dokumentation (dieses Dokument)

Kontext / Voraussetzungen
· anforderungen.docx erstellt

Prompt

```
Erstelle eine Prompt-Dokumentation als Word-Dokument.

## Zweck
- Nachvollziehbarkeit: wie wurde die App entwickelt?
- Vorlage für Neuentwicklung in einem anderen LLM
- Aus Sicht eines Anwendungsentwicklers formuliert

## Struktur je Prompt
- Nummer (P-01, P-02, ...)
- Phase: [INIT][FEATURE][REFACTOR][BUGFIX][CONFIG][INTEGRATION][DOCS]
- Titel
- Kontext / Voraussetzungen
- Prompt-Text (exakt wie eingegeben)
- Hinweise & bekannte Fallstricke

## Phasen-Gliederung
```

```
1. Grundstruktur und Katalogverwaltung  (P-01 bis P-04)
2. Visualisierung und Export            (P-05 bis P-07)
3. Externe Datenquellen                 (P-08 bis P-11)
4. Modularisierung                      (P-12 bis P-13)
5. Ausrüstung und FOV                   (P-14 bis P-15)
6. ASCOM Alpaca Teleskopsteuerung        (P-16 bis P-19)
7. Hilfsskripte und Dokumentation       (P-20 bis P-22)
```

Allgemeine Entwicklungshinweise

Technologie-Entscheidungen

- Kein JavaScript-Framework (React, Vue, Angular) — natives JS ausreichend
- Keine externen CSS-Frameworks — eigenes Dark-Theme in <style>
- Module als globale Skripte (<script src>) statt ES6-Imports — kein Build-Schritt nötig
- IndexedDB für Katalogdaten (unbegrenzt), localStorage für Einstellungen
- Alle astronomischen Berechnungen selbst implementiert — keine Astro-Bibliothek

Prompting-Strategie

Folgende Prinzipien haben sich bei der LLM-gestützten Entwicklung bewährt:

- Einen Funktionsbereich pro Prompt — nie mehrere unabhängige Features gleichzeitig
- Technische Rahmenbedingungen explizit nennen (z.B. "kein Framework", "eine Datei")
- API-Eigenheiten vorab beschreiben (z.B. VSX RA in Grad nicht Stunden)
- Bekannte Fallstricke im Prompt erwähnen — erspart Bugfix-Iterationen
- Nach jedem Prompt: Syntaxcheck (node -e "new Function(code)") vor Auslieferung
- Modul-Schnittstellen (Funktionsnamen, Parametertypen) explizit vorgeben
- CORS-Probleme früh adressieren — betrifft alle externen APIs

Wichtige API-Eigenheiten

API	Eigenheit	Lösung
AAVSO VSX	RA in Dezimalgrad, nicht Stunden	"RA2000" × 15 ist falsch
AAVSO VSX	Feldname "Declination2000", nicht "Decl2000"	JSON-Schlüssel exakt prüfen
AAVSO VSX	Epochen < 2400000 = reduziertes HJD	+ 2400000 addieren
SIMBAD TAP	LIMIT nicht erlaubt	SELECT TOP n verwenden
SIMBAD TAP	JOIN ohne Tabellenpräfix schlägt fehl	basic.oid = ident.oidref
SIMBAD TAP	Antwortet mit gzip wenn Accept-Encoding fehlt	Accept-Encoding: identity
JPL Horizons	Kein CORS-Header	Proxy nötig
heartsome Proxy	Leitet gzip weiter → PHP-Dekodierungsfehler	Accept-Encoding: identity im fetch()
Alpaca	GET für Properties, PUT für Aktionen	PUT-Body: URLSearchParams

API	Eigenheit	Lösung
Alpaca Camera	Gain/Offset optional — wirft Exception	setBinning/setGain in try/catch