ASTRO-ENGINE: Swiss Ephemeris Doğum Haritası Hesaplama API'si — Proje Şartnamesi

BU DÖKÜMAN NEDİR?

Bu döküman, Swiss Ephemeris kütüphanesi kullanarak profesyonel seviyede bir doğum haritası (natal chart) hesaplama API'si oluşturmak için kapsamlı bir teknik şartnamedir. Adım adım talimatları, kod örneklerini, dikkat edilmesi gereken kritik noktaları ve test prosedürlerini içerir.

Hedef: AstrologyAPI.com gibi üçüncü parti servislerin yerine geçecek, astro.com ile birebir eşleşen hassasiyette, açık kaynak (AGPL) bir hesaplama motoru ve REST API oluşturmak.

Teknoloji Stack'i:

- Runtime: Node.js (v18+)
- Hesaplama: `sweph` npm paketi (Swiss Ephemeris Node.js binding)
- Timezone: `luxon` (IANA timezone database)
- HTTP Server: Express.js
- Lisans: AGPL-3.0 (açık kaynak)

ADIM 1: PROJE KURULUMU

1.1 Proje Dizini Oluştur

```bash
mkdir astro-engine
cd astro-engine
npm init -y
```

1.2 Bağımlılıkları Kur

```bash
npm install sweph luxon express cors
npm install --save-dev nodemon
```

KRİTİK NOT — `sweph` kurulumu hakkında:

- `sweph` paketi C kodu derliyor (node-gyp üzerinden native N-API addon)
- Mac'te **Xcode Command Line Tools** gerekli: `xcode-select --install`
- Eğer kurulum hatası verirse: `npm install -g node-gyp` ardından tekrar dene

- `sweph` paketi Swiss Ephemeris 2.10.3b versiyonuna bağlı, en güncel ve en iyi maintained Node.js binding'dir (timotejroiko/sweph)
- Alternatif `swisseph` paketi (mivion) eski callback-based API kullanıyor, onu KULLANMA

1.3 Ephemeris Veri Dosyalarını İndir

Swiss Ephemeris yüksek hassasiyet için `.se1` dosyalarına ihtiyaç duyar. Bunlar olmadan da çalışır (Moshier fallback, 0.1 arcsec hassasiyet) ama tam hassasiyet için (0.001 arcsec) bu dosyalar gerekli.

```bash
mkdir ephe
cd ephe

# Ana gezegen dosyaları (1800-2400 AD)
curl -LO https://raw.githubusercontent.com/aloistr/swisseph/master/ephe/sepl_18.se1
curl -LO https://raw.githubusercontent.com/aloistr/swisseph/master/ephe/semo_18.se1
curl -LO https://raw.githubusercontent.com/aloistr/swisseph/master/ephe/seas_18.se1

cd ..
```

Dosya açıklamaları:

- `sepl_18.se1` — Gezegen pozisyonları (Güneş, Merkür, Venüs, Mars, Jüpiter, Satürn, Uranüs, Neptün, Plüton)
- `semo_18.se1` — Ay pozisyonları
- `seas_18.se1` — Asteroidler + Chiron
- Toplam boyut: ~2MB
- Kapsam: 1800-2400 yılları arası (modern doğum haritaları için fazlasıyla yeterli)

1.4 Proje Yapısı

Aşağıdaki dosya/klasör yapısını oluştur:

```
astro-engine/
├── ephe/                  # Ephemeris veri dosyaları
│   ├── sepl_18.se1
│   ├── semo_18.se1
│   └── seas_18.se1
├── src/
│   ├── calculator.js      # Ana hesaplama motoru
│   ├── timezone.js        # Timezone dönüşüm modülü
│   ├── aspects.js         # Aspekt hesaplama modülü
│   ├── dignities.js       # Gezegen dignite (onur/düşüş) modülü
│   ├── utils.js           # Yardımcı fonksiyonlar
│   └── constants.js       # Sabitler ve konfigürasyon
├── server.js              # Express API sunucusu
├── test.js                # Test ve doğrulama scripti
├── package.json
├── .gitignore
├── LICENSE                # AGPL-3.0
└── README.md
```

1.5 package.json Güncelle

```
json
```

```
{
  "name": "astro-engine",
  "version": "1.0.0",
  "description": "High-precision natal chart calculation engine powered by Swiss Ephemeris",
  "main": "server.js",
  "type": "module",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js",
    "test": "node test.js"
  },
  "keywords": ["astrology", "natal-chart", "swiss-ephemeris", "horoscope"],
  "license": "AGPL-3.0",
  "dependencies": {
    "sweph": "^2.10.3",
    "luxon": "^3.0.0",
    "express": "^4.18.0",
    "cors": "^2.8.5"
  },
  "devDependencies": {
    "nodemon": "^3.0.0"
  }
}
```

1.6 .gitignore Oluştur

```
node_modules/
.DS_Store
.env
```

NOT: `ephe/` klasörünü .gitignore'a EKLEME. Ephemeris dosyaları repoyla birlikte dağıtılmalı ki başkaları da kullanabilsin.

ADIM 2: SABİTLER VE KONFİGÜRASYON (src/constants.js)

javascript

```
// src/constants.js

// Burç isimleri — index 0'dan başlar (0° = Koç başlangıcı)
export const SIGNS = [
  'Aries', 'Taurus', 'Gemini', 'Cancer', 'Leo', 'Virgo',
  'Libra', 'Scorpio', 'Sagittarius', 'Capricorn', 'Aquarius', 'Pisces'
];

// Türkçe burç isimleri (opsiyonel, locale desteği için)
export const SIGNS_TR = [
  'Koç', 'Boğa', 'İkizler', 'Yengeç', 'Aslan', 'Başak',
  'Terazi', 'Akrep', 'Yay', 'Oğlak', 'Kova', 'Balık'
];

// Swiss Ephemeris gök cismi ID'leri
export const CELESTIAL_BODIES = [
  { id: 0,  name: 'Sun',        trName: 'Güneş' },
  { id: 1,  name: 'Moon',       trName: 'Ay' },
  { id: 2,  name: 'Mercury',    trName: 'Merkür' },
  { id: 3,  name: 'Venus',      trName: 'Venüs' },
  { id: 4,  name: 'Mars',       trName: 'Mars' },
  { id: 5,  name: 'Jupiter',    trName: 'Jüpiter' },
  { id: 6,  name: 'Saturn',     trName: 'Satürn' },
  { id: 7,  name: 'Uranus',     trName: 'Uranüs' },
  { id: 8,  name: 'Neptune',    trName: 'Neptün' },
  { id: 9,  name: 'Pluto',      trName: 'Plüton' },
  { id: 15, name: 'Chiron',     trName: 'Chiron' },
  { id: 11, name: 'True Node',  trName: 'Kuzey Ay Düğümü' },
  { id: 12, name: 'Lilith',     trName: 'Lilith' },
];

// Aspekt tanımları
// orb = tolerans derecesi (Güneş ve Ay için %25 genişletilir)
export const ASPECTS = [
  { name: 'Conjunction',  angle: 0,   orb: 8,   symbol: '☌', trName: 'Kavuşum' },
  { name: 'Opposition',   angle: 180, orb: 8,   symbol: '☍', trName: 'Karşıt' },
  { name: 'Trine',        angle: 120, orb: 7,   symbol: '△', trName: 'Üçgen' },
  { name: 'Square',       angle: 90,  orb: 7,   symbol: '□', trName: 'Kare' },
  { name: 'Sextile',      angle: 60,  orb: 5,   symbol: '⚹', trName: 'Altıgen' },
  { name: 'Quincunx',     angle: 150, orb: 2.5, symbol: '⚻', trName: 'Quincunx' },
  { name: 'Semi-sextile', angle: 30,  orb: 1.5, symbol: '⚺', trName: 'Yarı Altıgen' },
];

// Desteklenen ev sistemleri
// Swiss Ephemeris tek karakter kodları kullanır
export const HOUSE_SYSTEMS = {
```

```
  'P': { name: 'Placidus',      description: 'En yaygın Batı sistemi (varsayılan)' },
  'K': { name: 'Koch',         description: 'Placidus\'a benzer, bazı Avrupa astrologları tercih eder' },
  'W': { name: 'Whole Sign',   description: 'En eski sistem, Hellenistik astroloji. Tüm enlemlerde çalışır' },
  'E': { name: 'Equal',        description: 'Her ev 30°, ASC\'den başlar' },
  'B': { name: 'Alcabitius',   description: 'Orta Çağ Arap astrolojisi' },
  'R': { name: 'Regiomontanus', description: 'Horary astrolojide tercih edilir' },
  'O': { name: 'Porphyry',     description: 'En basit quadrant sistemi' },
  'C': { name: 'Campanus',     description: 'Mekan bazlı bölünme' },
};

// Element ve modalite sınıflandırması
export const ELEMENTS = {
  Fire:  ['Aries', 'Leo', 'Sagittarius'],
  Earth: ['Taurus', 'Virgo', 'Capricorn'],
  Air:   ['Gemini', 'Libra', 'Aquarius'],
  Water: ['Cancer', 'Scorpio', 'Pisces'],
};

export const MODALITIES = {
  Cardinal: ['Aries', 'Cancer', 'Libra', 'Capricorn'],
  Fixed:    ['Taurus', 'Leo', 'Scorpio', 'Aquarius'],
  Mutable:  ['Gemini', 'Virgo', 'Sagittarius', 'Pisces'],
};
```

ADIM 3: TIMEZONE DÖNÜŞÜM MODÜLÜ (src/timezone.js)

BU MODÜL KRİTİK ÖNEMDEDİR. Doğum haritası hatalarının büyük çoğunluğu timezone dönüşümünden kaynaklanır. 1 saatlik hata = yanlış yükselen burç.

javascript

```
// src/timezone.js
import { DateTime } from 'luxon';

/**
 * Doğum zamanını yerel saatten UTC'ye çevirir.
 *
 * KRİTİK NOTLAR:
 * - Ham UTC offset (örn. "+3") KULLANMA, her zaman IANA timezone ID kullan
 * - IANA ID'ler tarihsel DST değişikliklerini kodlar
 * - Türkiye örneği: 2016 öncesi UTC+2/+3 (DST), 2016 sonrası sabit UTC+3
 *
 * @param {number} year - Doğum yılı
 * @param {number} month - Doğum ayı (1-12, JavaScript'in 0-based Date'inden farklı!)
 * @param {number} day - Doğum günü
 * @param {number} hour - Doğum saati (0-23)
 * @param {number} minute - Doğum dakikası (0-59)
 * @param {string} timezone - IANA timezone ID (örn: "Europe/Istanbul", "America/New_York")
 * @returns {object} UTC bilgileri ve uyarılar
 */
export function birthTimeToUTC(year, month, day, hour, minute, timezone) {
  // IANA timezone doğrulama
  const testZone = DateTime.now().setZone(timezone);
  if (!testZone.isValid) {
    throw new Error(`Geçersiz timezone: "${timezone}". IANA formatı kullanın (örn: "Europe/Istanbul")`);
  }

  // Yerel zamanı oluştur
  const local = DateTime.fromObject(
    { year, month, day, hour, minute, second: 0 },
    { zone: timezone }
  );

  // DST spring-forward gap kontrolü
  // Örnek: Saat 02:00'den 03:00'e atladığında 02:30 geçersizdir
  if (!local.isValid) {
    throw new Error(
      `Geçersiz zaman: ${year}-${month}-${day} ${hour}:${minute} "${timezone}" timezone'unda mevcut değil. ` +
      `Muhtemelen yaz saati geçişi nedeniyle bu saat atlanmış. ` +
      `Sebep: ${local.invalidReason}. Açıklama: ${local.invalidExplanation}`
    );
  }

  // UTC'ye dönüştür
  const utc = local.toUTC();

  // Uyarılar oluştur
```

```
const warnings = [];

if (year < 1970) {
  warnings.push(
    '1970 öncesi timezone verileri güvenilir olmayabilir. ' +
    'astro.com atlas ile çapraz kontrol önerilir.'
  );
}

if (year < 1883 && timezone.startsWith('America/')) {
  warnings.push(
    '1883 öncesi ABD\'de standart zaman dilimleri yoktu. ' +
    'Yerel güneş saati (LMT) kullanılıyor olabilir.'
  );
}

// DST fall-back ambiguity kontrolü (aynı saat iki kez yaşanır)
// Luxon bu durumda ilk oluşumu (DST saatini) tercih eder
const oneHourLater = local.plus({ hours: 1 });
if (local.offset !== oneHourLater.offset && local.isInDST) {
  warnings.push(
    'Bu zaman DST geçiş saatine denk geliyor. ' +
    'Aynı saat iki kez yaşanmış olabilir (DST ve standart). ' +
    'DST versiyonu kullanıldı.'
  );
}

return {
  utcYear: utc.year,
  utcMonth: utc.month,
  utcDay: utc.day,
  utcHour: utc.hour,
  utcMinute: utc.minute,
  utcSecond: utc.second,
  // Ondalıklı saat (Swiss Ephemeris'in julday fonksiyonu için)
  utcDecimalHour: utc.hour + utc.minute / 60 + utc.second / 3600,
  // Offset bilgisi (dakika cinsinden)
  offsetMinutes: local.offset,
  offsetHours: local.offset / 60,
  // DST durumu
  isDST: local.isInDST,
  // Orijinal girdi (debug için)
  originalInput: {
    localTime: `${year}-${String(month).padStart(2, '0')}-${String(day).padStart(2, '0')}T${String(hour).padStart(2, '0')}:${S
    timezone,
    utcTime: utc.toISO(),
  },
```

```
    // Uyarılar
    warnings,
  };
}

/**
 * Koordinatlardan IANA timezone tahmin et.
 * NOT: Bu basit bir yaklaşım. Production'da Google Time Zone API veya
 * timezonefinder gibi bir servis kullanmak daha doğru olur.
 *
 * Bilinen önemli timezone'lar:
 * - Türkiye: "Europe/Istanbul" (tüm Türkiye için geçerli, 2016'dan beri UTC+3 sabit)
 * - Doğu ABD: "America/New_York"
 * - Batı ABD: "America/Los_Angeles"
 * - Brezilya: "America/Sao_Paulo"
 * - Hindistan: "Asia/Kolkata"
 */
export function getTimezoneInfo(timezone) {
  const now = DateTime.now().setZone(timezone);
  return {
    timezone,
    currentOffset: now.offset / 60,
    isDST: now.isInDST,
    abbreviation: now.offsetNameShort,
    longName: now.offsetNameLong,
  };
}
```

ADIM 4: ANA HESAPLAMA MOTORU (src/calculator.js)

javascript

```
// src/calculator.js
import * as swe from 'sweph';
import path from 'path';
import { fileURLToPath } from 'url';
import { CELESTIAL_BODIES, SIGNS, HOUSE_SYSTEMS } from './constants.js';
import { birthTimeToUTC } from './timezone.js';
import { calculateAspects } from './aspects.js';
import { getDignity } from './dignities.js';
import {
  longitudeToSign,
  determineMoonPhase,
  calculatePartOfFortune,
  getElementDistribution,
  getModalityDistribution,
  determineHemisphereEmphasis,
  findPlanetInHouse,
} from './utils.js';

// __dirname ES module equivalent
const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

// Ephemeris dosya yolunu ayarla
// KRİTİK: Bu fonksiyon TÜM hesaplamalardan önce çağrılmalı
// Dahili initialization yapar, null geçilse bile çağrılması şart
const ephePath = path.join(__dirname, '..', 'ephe');
swe.set_ephe_path(ephePath);

/**
 * Tam doğum haritası hesapla.
 *
 * @param {object} params - Doğum bilgileri
 * @param {number} params.year - Doğum yılı
 * @param {number} params.month - Doğum ayı (1-12)
 * @param {number} params.day - Doğum günü
 * @param {number} params.hour - Doğum saati (0-23)
 * @param {number} params.minute - Doğum dakikası (0-59)
 * @param {number} params.latitude - Doğum yeri enlemi (kuzey pozitif)
 * @param {number} params.longitude - Doğum yeri boylamı (doğu pozitif, BATI NEGATİF)
 * @param {string} params.timezone - IANA timezone (örn: "Europe/Istanbul")
 * @param {string} [params.houseSystem='P'] - Ev sistemi kodu (varsayılan: Placidus)
 * @returns {object} Tam doğum haritası verisi
 */
export function calculateNatalChart({
  year, month, day, hour, minute,
  latitude, longitude, timezone,
```

```
  houseSystem = 'P'
}) {
  // ========== GİRDİ DOĞRULAMA ==========

  if (!HOUSE_SYSTEMS[houseSystem]) {
    throw new Error(
      `Geçersiz ev sistemi: "${houseSystem}". ` +
      `Desteklenen sistemler: ${Object.keys(HOUSE_SYSTEMS).join(', ')}`
    );
  }

  if (latitude < -90 || latitude > 90) {
    throw new Error(`Geçersiz enlem: ${latitude}. -90 ile 90 arasında olmalı.`);
  }

  if (longitude < -180 || longitude > 180) {
    throw new Error(`Geçersiz boylam: ${longitude}. -180 ile 180 arasında olmalı.`);
  }

  if (month < 1 || month > 12) {
    throw new Error(`Geçersiz ay: ${month}. 1-12 arasında olmalı.`);
  }

  if (day < 1 || day > 31) {
    throw new Error(`Geçersiz gün: ${day}. 1-31 arasında olmalı.`);
  }

  if (hour < 0 || hour > 23) {
    throw new Error(`Geçersiz saat: ${hour}. 0-23 arasında olmalı.`);
  }

  if (minute < 0 || minute > 59) {
    throw new Error(`Geçersiz dakika: ${minute}. 0-59 arasında olmalı.`);
  }

  // ========== TIMEZONE DÖNÜŞÜMÜ ==========

  const utcData = birthTimeToUTC(year, month, day, hour, minute, timezone);

  // ========== JULIAN DAY HESABI ==========

  // utc_to_jd hem jd_et (Ephemeris Time) hem jd_ut (Universal Time) döndürür
  // KRİTİK: julday() yerine utc_to_jd() kullan — ΔT dönüşümünü otomatik yapar
  const jdResult = swe.utc_to_jd(
    utcData.utcYear,
    utcData.utcMonth,
    utcData.utcDay,
```

```
    utcData.utcHour,
    utcData.utcMinute,
    utcData.utcSecond,
    swe.constants.SE_GREG_CAL
);

// jdResult.data[0] = JD in ET (Ephemeris Time) — gezegen hesapları için
// jdResult.data[1] = JD in UT (Universal Time) — ev hesapları için
const jd_et = jdResult.data[0];
const jd_ut = jdResult.data[1];

// ========== GEZEGEN POZİSYONLARI ==========

// SEFLG_SWIEPH: Swiss Ephemeris dosyalarını kullan (yoksa Moshier'e fallback)
// SEFLG_SPEED: Hız verisi de döndür (retrograd tespiti için şart)
const calcFlags = swe.constants.SEFLG_SWIEPH | swe.constants.SEFLG_SPEED;

const planets = CELESTIAL_BODIES.map(body => {
  const result = swe.calc_ut(jd_et, body.id, calcFlags);

  // result.flag kontrol et — input flag'den farklıysa Moshier fallback olmuş olabilir
  const usedMoshier = (result.flag & swe.constants.SEFLG_SWIEPH) === 0;

  // result.data dizisi:
  // [0] = ekliptik boylam (0-360°)
  // [1] = ekliptik enlem
  // [2] = mesafe (AU)
  // [3] = boylam hızı (°/gün) — negatif = retrograd
  // [4] = enlem hızı
  // [5] = mesafe hızı
  const lon = result.data[0];
  const lat = result.data[1];
  const distance = result.data[2];
  const speedLon = result.data[3];

  const signData = longitudeToSign(lon);

  return {
    id: body.id,
    name: body.name,
    trName: body.trName,
    longitude: roundTo(lon, 6),
    latitude: roundTo(lat, 6),
    distance: roundTo(distance, 6),
    speed: roundTo(speedLon, 6),
    sign: signData.sign,
    signIndex: signData.signIndex,
```

```
    degree: signData.degree,
    minute: signData.minute,
    second: signData.second,
    isRetrograde: speedLon < 0,
    dignity: getDignity(body.name, signData.sign),
    formattedPosition: `${signData.degree}°${String(signData.minute).padStart(2, '0')}'${String(signData.second).padStart(2
    usedMoshierFallback: usedMoshier,
  };
});

// Güney Düğümü hesapla (Kuzey Düğümü + 180°)
const northNode = planets.find(p => p.name === 'True Node');
if (northNode) {
  const southLon = (northNode.longitude + 180) % 360;
  const southSignData = longitudeToSign(southLon);
  planets.push({
    id: -1,
    name: 'South Node',
    trName: 'Güney Ay Düğümü',
    longitude: roundTo(southLon, 6),
    latitude: roundTo(-northNode.latitude, 6),
    distance: northNode.distance,
    speed: northNode.speed,
    sign: southSignData.sign,
    signIndex: southSignData.signIndex,
    degree: southSignData.degree,
    minute: southSignData.minute,
    second: southSignData.second,
    isRetrograde: northNode.isRetrograde,
    dignity: null,
    formattedPosition: `${southSignData.degree}°${String(southSignData.minute).padStart(2, '0')}'${String(southSignData.s
    usedMoshierFallback: false,
  });
}

// ========== EV HESABI ==========

// KRİTİK: houses fonksiyonu jd_ut (Universal Time) alır, jd_et DEĞİL!
// KRİTİK: Swiss Ephemeris batı boylamı için NEGATİF değer bekler
const housesResult = swe.houses(jd_ut, latitude, longitude, houseSystem);

// housesResult.data.cusps: cusps[0] boş, cusps[1]-cusps[12] ev sınırları
// housesResult.data.points: [ASC, MC, ARMC, Vertex, EquatorialASC, co-ASC Koch, co-ASC Munkasey, PolarASC]
const cusps = housesResult.data.cusps;
const points = housesResult.data.points;

const ascendant = points[0];
```

```
const midheaven = points[1];
const armc = points[2]; // Sidereal time (ARMC)
const vertex = points[3];

// Polar enlem uyarısı (Placidus ve Koch ~66.5° üstünde başarısız olur)
const warnings = [...utcData.warnings];

if (['P', 'K'].includes(houseSystem) && Math.abs(latitude) > 66) {
  warnings.push(
    `${HOUSE_SYSTEMS[houseSystem].name} ev sistemi ${Math.abs(latitude)}° enlemde güvenilir olmayabilir. ` +
    `Kutup bölgelerinde Whole Sign ("W") sistemi önerilir. ` +
    `Swiss Ephemeris otomatik olarak Porphyry'e geçmiş olabilir.`
  );
}

// Ev cusp'larını detaylı formata çevir
const houses = [];
for (let i = 1; i <= 12; i++) {
  const cuspData = longitudeToSign(cusps[i]);
  houses.push({
    house: i,
    cusp: roundTo(cusps[i], 6),
    sign: cuspData.sign,
    degree: cuspData.degree,
    minute: cuspData.minute,
    formattedCusp: `${cuspData.degree}°${String(cuspData.minute).padStart(2, '0')}' ${cuspData.sign}`,
  });
}

// Ascendant ve MC detaylı
const ascData = longitudeToSign(ascendant);
const mcData = longitudeToSign(midheaven);
const vtxData = longitudeToSign(vertex);

// Her gezegenin hangi evde olduğunu bul
const planetsWithHouses = planets.map(planet => ({
  ...planet,
  house: findPlanetInHouse(planet.longitude, cusps),
}));

// ========== ASPEKTLER ==========

// Ascendant ve MC'yi de aspekt hesabına dahil et
const aspectBodies = [
  ...planetsWithHouses,
  { name: 'Ascendant', trName: 'Yükselen', longitude: ascendant, speed: 0 },
  { name: 'Midheaven', trName: 'Gökyüzü Ortası', longitude: midheaven, speed: 0 },
```

```
];

const aspects = calculateAspects(aspectBodies);

// ========== CHART ANALİZİ ==========

const sun = planetsWithHouses.find(p => p.name === 'Sun');
const moon = planetsWithHouses.find(p => p.name === 'Moon');

// Gündüz/gece haritası tespiti (Güneş ufkun üstünde mi?)
const isDayChart = sun ? isAboveHorizon(sun.longitude, ascendant) : true;

// Part of Fortune
const partOfFortune = calculatePartOfFortune(ascendant, sun?.longitude, moon?.longitude, isDayChart);
const pofData = longitudeToSign(partOfFortune);

// Ay fazı
const moonPhase = moon && sun ? determineMoonPhase(sun.longitude, moon.longitude) : null;

// Element ve modalite dağılımı
const elementDist = getElementDistribution(planetsWithHouses);
const modalityDist = getModalityDistribution(planetsWithHouses);

// Hemisfer vurgusu
const hemisphereEmphasis = determineHemisphereEmphasis(planetsWithHouses, ascendant, midheaven);

// Stellium tespiti (aynı burçta 3+ gezegen)
const stelliums = findStelliums(planetsWithHouses);

// ========== SONUÇ ==========

return {
  input: {
    localTime: utcData.originalInput.localTime,
    timezone: timezone,
    utcTime: utcData.originalInput.utcTime,
    coordinates: { latitude, longitude },
    offsetHours: utcData.offsetHours,
    isDST: utcData.isDST,
  },
  planets: planetsWithHouses,
  houses: {
    system: houseSystem,
    systemName: HOUSE_SYSTEMS[houseSystem].name,
    cusps: houses,
    ascendant: {
      longitude: roundTo(ascendant, 6),
```

```
      sign: ascData.sign,
      degree: ascData.degree,
      minute: ascData.minute,
      formatted: `${ascData.degree}°${String(ascData.minute).padStart(2, '0')}' ${ascData.sign}`,
    },
    midheaven: {
      longitude: roundTo(midheaven, 6),
      sign: mcData.sign,
      degree: mcData.degree,
      minute: mcData.minute,
      formatted: `${mcData.degree}°${String(mcData.minute).padStart(2, '0')}' ${mcData.sign}`,
    },
    vertex: {
      longitude: roundTo(vertex, 6),
      sign: vtxData.sign,
      degree: vtxData.degree,
      minute: vtxData.minute,
    },
  },
  aspects,
  analysis: {
    sunSign: sun?.sign || null,
    moonSign: moon?.sign || null,
    risingSign: ascData.sign,
    isDayChart,
    moonPhase,
    partOfFortune: {
      longitude: roundTo(partOfFortune, 6),
      sign: pofData.sign,
      degree: pofData.degree,
      minute: pofData.minute,
      formatted: `${pofData.degree}°${String(pofData.minute).padStart(2, '0')}' ${pofData.sign}`,
    },
    elements: elementDist,
    modalities: modalityDist,
    hemispheres: hemisphereEmphasis,
    stelliums,
  },
  meta: {
    julianDayET: roundTo(jd_et, 8),
    julianDayUT: roundTo(jd_ut, 8),
    siderealTime: roundTo(armc, 6),
    deltaT: roundTo((jd_et - jd_ut) * 86400, 2), // saniye cinsinden
    ephemerisMode: planetsWithHouses.some(p => p.usedMoshierFallback) ? 'Moshier (fallback)' : 'Swiss Ephemeris',
    engine: 'sweph (Swiss Ephemeris Node.js binding)',
    version: '1.0.0',
    warnings,
```

```
    },
  };
}

// ========== YARDIMCI FONKSİYONLAR ==========

function roundTo(num, decimals) {
  return Math.round(num * Math.pow(10, decimals)) / Math.pow(10, decimals);
}

function isAboveHorizon(planetLon, ascLon) {
  const desc = (ascLon + 180) % 360;
  if (ascLon < desc) {
    return planetLon >= ascLon && planetLon < desc ? false : true;
  } else {
    return planetLon >= ascLon || planetLon < desc ? false : true;
  }
}

function findStelliums(planets) {
  // Güneş, Ay ve geleneksel gezegenler (Chiron, Node'lar hariç)
  const mainPlanets = planets.filter(p =>
    ['Sun','Moon','Mercury','Venus','Mars','Jupiter','Saturn','Uranus','Neptune','Pluto'].includes(p.name)
  );

  const signCounts = {};
  mainPlanets.forEach(p => {
    if (!signCounts[p.sign]) signCounts[p.sign] = [];
    signCounts[p.sign].push(p.name);
  });

  const stelliums = [];
  for (const [sign, names] of Object.entries(signCounts)) {
    if (names.length >= 3) {
      stelliums.push({ sign, planets: names, count: names.length });
    }
  }
  return stelliums;
}
```

ADIM 5: ASPEKT HESAPLAMA MODÜLÜ (src/aspects.js)

javascript

```
// src/aspects.js
import { ASPECTS } from './constants.js';

/**
 * Tüm gök cisimleri arasındaki aspektleri hesapla.
 *
 * @param {Array} bodies - Gezegen ve nokta listesi (her biri {name, longitude, speed} içermeli)
 * @returns {Array} Aspekt listesi
 */
export function calculateAspects(bodies) {
  const aspects = [];

  for (let i = 0; i < bodies.length; i++) {
    for (let j = i + 1; j < bodies.length; j++) {
      const body1 = bodies[i];
      const body2 = bodies[j];

      // Node'lar arası aspekt ve South Node aspektlerini atla (çoğu astrolog dahil etmez)
      if (
        (body1.name === 'True Node' && body2.name === 'South Node') ||
        (body1.name === 'South Node' && body2.name === 'True Node')
      ) continue;

      // İki boylam arasındaki açı (0-180°)
      let diff = Math.abs(body1.longitude - body2.longitude);
      if (diff > 180) diff = 360 - diff;

      for (const aspect of ASPECTS) {
        // Güneş veya Ay içeren aspektlerde orb'u %25 genişlet
        let effectiveOrb = aspect.orb;
        const luminaries = ['Sun', 'Moon'];
        if (luminaries.includes(body1.name) || luminaries.includes(body2.name)) {
          effectiveOrb *= 1.25;
        }

        // ASC/MC aspektlerinde orb'u biraz daralt
        const angles = ['Ascendant', 'Midheaven'];
        if (angles.includes(body1.name) || angles.includes(body2.name)) {
          effectiveOrb *= 0.75;
        }

        const deviation = Math.abs(diff - aspect.angle);

        if (deviation <= effectiveOrb) {
          // Applying (yaklaşan) vs Separating (uzaklaşan) tespiti
          const isApplying = determineApplying(body1, body2, aspect.angle);
```

```
          aspects.push({
            planet1: body1.name,
            planet1Tr: body1.trName,
            planet2: body2.name,
            planet2Tr: body2.trName,
            type: aspect.name,
            typeTr: aspect.trName,
            symbol: aspect.symbol,
            exactAngle: aspect.angle,
            actualAngle: roundTo(diff, 2),
            orb: roundTo(deviation, 2),
            maxOrb: roundTo(effectiveOrb, 2),
            isApplying,
            strength: calculateAspectStrength(deviation, effectiveOrb),
          });

          // Her çift için sadece bir aspekt (en dar orb'lu)
          break;
        }
      }
    }
  }

  // Orb'a göre sırala (en güçlü aspektler önce)
  return aspects.sort((a, b) => a.orb - b.orb);
}

/**
 * Aspektin yaklaşan mı (applying) uzaklaşan mı (separating) olduğunu belirle.
 * Daha hızlı gezegen yavaş gezegene doğru hareket ediyorsa = applying
 */
function determineApplying(body1, body2, aspectAngle) {
  // Her iki cismin de hız verisi yoksa belirlenemez
  if (body1.speed === undefined || body2.speed === undefined) return null;
  if (body1.speed === 0 && body2.speed === 0) return null;

  // Hızlı gezegen = daha büyük mutlak hız
  const speed1 = Math.abs(body1.speed);
  const speed2 = Math.abs(body2.speed);

  // Göreceli hareket: hızlı gezegen yavaşa yaklaşıyor mu?
  const relativeSpeed = body1.speed - body2.speed;
  let currentDiff = body1.longitude - body2.longitude;

  // 0-360 aralığında normalize et
  if (currentDiff < 0) currentDiff += 360;
```

```
  if (currentDiff > 180) currentDiff = 360 - currentDiff;

  // Bir sonraki "adımda" açı aspekt açısına yaklaşacak mı?
  const futureBody1 = body1.longitude + body1.speed * 0.01;
  const futureBody2 = body2.longitude + body2.speed * 0.01;
  let futureDiff = Math.abs(futureBody1 - futureBody2);
  if (futureDiff > 180) futureDiff = 360 - futureDiff;

  const currentDeviation = Math.abs(currentDiff - aspectAngle);
  const futureDeviation = Math.abs(futureDiff - aspectAngle);

  return futureDeviation < currentDeviation;
}

/**
 * Aspekt gücünü 0-100 arasında hesapla.
 * Orb 0° = 100 (tam aspekt), orb = maxOrb = 0 (aspekt sınırında)
 */
function calculateAspectStrength(deviation, maxOrb) {
  if (maxOrb === 0) return 100;
  const strength = Math.round((1 - deviation / maxOrb) * 100);
  return Math.max(0, Math.min(100, strength));
}

function roundTo(num, decimals) {
  return Math.round(num * Math.pow(10, decimals)) / Math.pow(10, decimals);
}
```

ADIM 6: DİGNİTE MODÜLÜ (src/dignities.js)

javascript

```
// src/dignities.js

/**
 * Geleneksel gezegen dignite (onur/düşüş) tablosu.
 * Domicile = Gezegen kendi burcunda (en güçlü)
 * Exaltation = Yüceltildiği burç
 * Detriment = Zıt burç (zayıf)
 * Fall = Düşüş burcu (en zayıf)
 * Peregrine = Hiçbiri değilse
 */
const DIGNITY_TABLE = {
  Sun:     { domicile: ['Leo'],              exaltation: ['Aries'],      detriment: ['Aquarius'],            fall: ['Libra'] },
  Moon:    { domicile: ['Cancer'],           exaltation: ['Taurus'],     detriment: ['Capricorn'],           fall: ['Scorpio'] },
  Mercury: { domicile: ['Gemini', 'Virgo'],  exaltation: ['Virgo'],      detriment: ['Sagittarius', 'Pisces'], fall: ['Pisces'] },
  Venus:   { domicile: ['Taurus', 'Libra'],  exaltation: ['Pisces'],     detriment: ['Aries', 'Scorpio'],    fall: ['Virgo'] },
  Mars:    { domicile: ['Aries', 'Scorpio'], exaltation: ['Capricorn'], detriment: ['Libra', 'Taurus'],     fall: ['Cancer'] },
  Jupiter: { domicile: ['Sagittarius', 'Pisces'], exaltation: ['Cancer'], detriment: ['Gemini', 'Virgo'],    fall: ['Capricorn'] },
  Saturn:  { domicile: ['Capricorn', 'Aquarius'], exaltation: ['Libra'], detriment: ['Cancer', 'Leo'],       fall: ['Aries'] },
  Uranus:  { domicile: ['Aquarius'],         exaltation: ['Scorpio'],    detriment: ['Leo'],                 fall: ['Taurus'] },
  Neptune: { domicile: ['Pisces'],           exaltation: ['Cancer'],     detriment: ['Virgo'],               fall: ['Capricorn'] },
  Pluto:   { domicile: ['Scorpio'],          exaltation: ['Aries'],      detriment: ['Taurus'],              fall: ['Libra'] },
};

/**
 * Gezegenin bulunduğu burçtaki dignite durumunu döndürür.
 *
 * @param {string} planetName - Gezegen adı (İngilizce)
 * @param {string} sign - Burç adı (İngilizce)
 * @returns {string|null} 'domicile' | 'exaltation' | 'detriment' | 'fall' | 'peregrine' | null
 */
export function getDignity(planetName, sign) {
  const table = DIGNITY_TABLE[planetName];
  if (!table) return null; // Chiron, Node'lar vs. için dignite yok

  if (table.domicile.includes(sign))    return 'domicile';
  if (table.exaltation.includes(sign))  return 'exaltation';
  if (table.detriment.includes(sign))   return 'detriment';
  if (table.fall.includes(sign))        return 'fall';
  return 'peregrine';
}

/**
 * Türkçe dignite karşılığı
 */
export function getDignityTr(dignity) {
  const map = {
```

```
    domicile: 'Hane (Güçlü)',
    exaltation: 'Yücelme',
    detriment: 'Sürgün (Zayıf)',
    fall: 'Düşüş',
    peregrine: 'Peregrine (Nötr)',
  };
  return map[dignity] || null;
}
```

ADIM 7: YARDIMCI FONKSİYONLAR (src/utils.js)

javascript

```
// src/utils.js
import { SIGNS, ELEMENTS, MODALITIES } from './constants.js';

/**
 * Ekliptik boylamı burç, derece, dakika, saniyeye çevirir.
 *
 * @param {number} longitude - 0-360° ekliptik boylam
 * @returns {object} {sign, signIndex, degree, minute, second}
 */
export function longitudeToSign(longitude) {
  // Negatif veya 360+ değerleri normalize et
  let lon = longitude % 360;
  if (lon < 0) lon += 360;

  const signIndex = Math.floor(lon / 30);
  const posInSign = lon % 30; // 0-30° arası
  const degree = Math.floor(posInSign);
  const fractional = (posInSign - degree) * 60;
  const minute = Math.floor(fractional);
  const second = Math.round((fractional - minute) * 60);

  return {
    sign: SIGNS[signIndex],
    signIndex,
    degree,
    minute,
    second: second >= 60 ? 59 : second, // Yuvarlama taşmasını önle
  };
}

/**
 * Ay fazını belirle.
 * Güneş-Ay açısına göre 8 faz döndürür.
 */
export function determineMoonPhase(sunLon, moonLon) {
  let angle = moonLon - sunLon;
  if (angle < 0) angle += 360;

  if (angle < 22.5)  return { phase: 'New Moon',        phaseTr: 'Yeni Ay',          angle: roundTo(angle, 2) };
  if (angle < 67.5)  return { phase: 'Waxing Crescent', phaseTr: 'Hilal (Büyüyen)',  angle: roundTo(angle, 2) };
  if (angle < 112.5) return { phase: 'First Quarter',   phaseTr: 'İlk Dördün',       angle: roundTo(angle, 2) };
  if (angle < 157.5) return { phase: 'Waxing Gibbous', phaseTr: 'Şişkin Ay (Büyüyen)', angle: roundTo(angle, 2) };
  if (angle < 202.5) return { phase: 'Full Moon',      phaseTr: 'Dolunay',          angle: roundTo(angle, 2) };
  if (angle < 247.5) return { phase: 'Waning Gibbous', phaseTr: 'Şişkin Ay (Küçülen)', angle: roundTo(angle, 2) };
  if (angle < 292.5) return { phase: 'Last Quarter',   phaseTr: 'Son Dördün',       angle: roundTo(angle, 2) };
  if (angle < 337.5) return { phase: 'Waning Crescent', phaseTr: 'Hilal (Küçülen)', angle: roundTo(angle, 2) };
```

```
  return { phase: 'New Moon', phaseTr: 'Yeni Ay', angle: roundTo(angle, 2) };
}

/**
 * Part of Fortune (Fortuna Noktası) hesapla.
 * Gündüz: ASC + Moon - Sun
 * Gece: ASC + Sun - Moon
 */
export function calculatePartOfFortune(ascLon, sunLon, moonLon, isDayChart) {
  if (sunLon === undefined || moonLon === undefined) return 0;

  let pof;
  if (isDayChart) {
    pof = ascLon + moonLon - sunLon;
  } else {
    pof = ascLon + sunLon - moonLon;
  }

  // 0-360 aralığına normalize
  pof = pof % 360;
  if (pof < 0) pof += 360;
  return pof;
}

/**
 * Element dağılımını hesapla (Ateş/Toprak/Hava/Su)
 */
export function getElementDistribution(planets) {
  const mainPlanets = planets.filter(p =>
    ['Sun','Moon','Mercury','Venus','Mars','Jupiter','Saturn','Uranus','Neptune','Pluto'].includes(p.name)
  );

  const dist = { Fire: 0, Earth: 0, Air: 0, Water: 0 };
  mainPlanets.forEach(p => {
    for (const [element, signs] of Object.entries(ELEMENTS)) {
      if (signs.includes(p.sign)) {
        dist[element]++;
        break;
      }
    }
  });

  // En baskın element
  const dominant = Object.entries(dist).sort((a, b) => b[1] - a[1])[0];
  return { ...dist, dominant: dominant[0] };
}
```

```
/**
 * Modalite dağılımını hesapla (Kardinal/Sabit/Değişken)
 */
export function getModalityDistribution(planets) {
  const mainPlanets = planets.filter(p =>
    ['Sun','Moon','Mercury','Venus','Mars','Jupiter','Saturn','Uranus','Neptune','Pluto'].includes(p.name)
  );

  const dist = { Cardinal: 0, Fixed: 0, Mutable: 0 };
  mainPlanets.forEach(p => {
    for (const [modality, signs] of Object.entries(MODALITIES)) {
      if (signs.includes(p.sign)) {
        dist[modality]++;
        break;
      }
    }
  });

  const dominant = Object.entries(dist).sort((a, b) => b[1] - a[1])[0];
  return { ...dist, dominant: dominant[0] };
}

/**
 * Hemisfer vurgusunu hesapla.
 * Kuzey/Güney (ASC-DSC eksenine göre) ve Doğu/Batı (MC-IC eksenine göre)
 */
export function determineHemisphereEmphasis(planets, ascLon, mcLon) {
  const mainPlanets = planets.filter(p =>
    ['Sun','Moon','Mercury','Venus','Mars','Jupiter','Saturn','Uranus','Neptune','Pluto'].includes(p.name)
  );

  const descLon = (ascLon + 180) % 360;
  const icLon = (mcLon + 180) % 360;

  let southern = 0, northern = 0, eastern = 0, western = 0;

  mainPlanets.forEach(p => {
    // Güney yarıküre = evler 7-12 (ufkun üstü)
    // Kuzey yarıküre = evler 1-6 (ufkun altı)
    if (p.house >= 7) southern++;
    else northern++;

    // Doğu yarıküre = evler 10-3 (ASC tarafı)
    // Batı yarıküre = evler 4-9 (DSC tarafı)
    if ([10, 11, 12, 1, 2, 3].includes(p.house)) eastern++;
    else western++;
  });
```

```
  return { southern, northern, eastern, western };
}

/**
 * Gezegenin hangi evde olduğunu bul.
 * Ev sınırları (cusps) arasında gezegenin boylamını ara.
 */
export function findPlanetInHouse(planetLon, cusps) {
  // cusps[1] - cusps[12] (index 0 boş)
  for (let i = 1; i <= 12; i++) {
    const nextHouse = i === 12 ? 1 : i + 1;
    const start = cusps[i];
    const end = cusps[nextHouse];

    // 0°/360° geçişini handle et
    if (start <= end) {
      if (planetLon >= start && planetLon < end) return i;
    } else {
      // Burç sınırı geçişi (örn: 350° - 10°)
      if (planetLon >= start || planetLon < end) return i;
    }
  }

  return 1; // Fallback (olmamalı ama güvenlik için)
}

function roundTo(num, decimals) {
  return Math.round(num * Math.pow(10, decimals)) / Math.pow(10, decimals);
}
```

ADIM 8: EXPRESS API SUNUCUSU (server.js)

javascript

```
// server.js
import express from 'express';
import cors from 'cors';
import { calculateNatalChart } from './src/calculator.js';

const app = express();
const PORT = process.env.PORT || 3000;

app.use(cors());
app.use(express.json());

// ========== SAĞLIK KONTROLÜ ==========
app.get('/health', (req, res) => {
  res.json({ status: 'ok', engine: 'astro-engine', version: '1.0.0' });
});

// ========== DOĞUM HARİTASI HESAPLA ==========
app.post('/api/natal-chart', (req, res) => {
  try {
    const {
      year, month, day, hour, minute,
      latitude, longitude, timezone,
      houseSystem
    } = req.body;

    // Zorunlu alan kontrolü
    const required = { year, month, day, hour, minute, latitude, longitude, timezone };
    const missing = Object.entries(required)
      .filter(([key, val]) => val === undefined || val === null)
      .map(([key]) => key);

    if (missing.length > 0) {
      return res.status(400).json({
        error: 'Eksik alanlar',
        missing,
        example: {
          year: 1990, month: 7, day: 15, hour: 14, minute: 30,
          latitude: 41.01, longitude: 28.97,
          timezone: 'Europe/Istanbul',
          houseSystem: 'P'
        }
      });
    }

    const chart = calculateNatalChart({
      year: parseInt(year),
```

```
      month: parseInt(month),
      day: parseInt(day),
      hour: parseInt(hour),
      minute: parseInt(minute),
      latitude: parseFloat(latitude),
      longitude: parseFloat(longitude),
      timezone,
      houseSystem: houseSystem || 'P',
    });

    res.json(chart);

  } catch (error) {
    console.error('Hesaplama hatası:', error.message);
    res.status(400).json({
      error: error.message,
      hint: 'Timezone için IANA formatı kullanın (örn: "Europe/Istanbul", "America/New_York")'
    });
  }
});

// ========== DESTEKLENEN EV SİSTEMLERİ ==========
app.get('/api/house-systems', (req, res) => {
  const { HOUSE_SYSTEMS } = require('./src/constants.js');
  res.json(HOUSE_SYSTEMS);
});

// ========== SUNUCUYU BAŞLAT ==========
app.listen(PORT, () => {
  console.log(`🔭 Astro Engine çalışıyor: http://localhost:${PORT}`);
  console.log(`📡 API endpoint: POST http://localhost:${PORT}/api/natal-chart`);
  console.log(`❤️  Sağlık kontrolü: GET http://localhost:${PORT}/health`);
});
```

ADIM 9: TEST VE DOĞRULAMA (test.js)

Bu script bilinen doğum haritalarını hesaplayıp sonuçları gösterir. Manuel olarak astro.com ile karşılaştırman gerekiyor.

```
javascript
```

```
// test.js
import { calculateNatalChart } from './src/calculator.js';

console.log('='.repeat(70));
console.log('ASTRO-ENGINE TEST SÜİTİ');
console.log('Sonuçları astro.com ile karşılaştır: https://www.astro.com/cgi/chart.cgi');
console.log('='.repeat(70));

// ========== TEST 1: İstanbul, 2020 (UTC+3 sabit, DST yok) ==========
console.log('\n📌 TEST 1: İstanbul, 25 Ocak 2020, 15:35');
console.log('-'.repeat(50));
try {
  const test1 = calculateNatalChart({
    year: 2020, month: 1, day: 25, hour: 15, minute: 35,
    latitude: 41.0082, longitude: 28.9784,
    timezone: 'Europe/Istanbul',
    houseSystem: 'P',
  });

  console.log(`Güneş: ${test1.planets.find(p => p.name === 'Sun').formattedPosition}`);
  console.log(`Ay: ${test1.planets.find(p => p.name === 'Moon').formattedPosition}`);
  console.log(`Yükselen: ${test1.houses.ascendant.formatted}`);
  console.log(`MC: ${test1.houses.midheaven.formatted}`);
  console.log(`Uyarılar: ${test1.meta.warnings.length > 0 ? test1.meta.warnings.join('; ') : 'Yok'}`);
  console.log(`Ephemeris modu: ${test1.meta.ephemerisMode}`);
} catch (e) {
  console.error('HATA:', e.message);
}

// ========== TEST 2: İstanbul, 1990 (DST aktif dönem, yaz) ==========
console.log('\n📌 TEST 2: İstanbul, 15 Temmuz 1990, 14:30 (DST döneminde)');
console.log('-'.repeat(50));
try {
  const test2 = calculateNatalChart({
    year: 1990, month: 7, day: 15, hour: 14, minute: 30,
    latitude: 41.0082, longitude: 28.9784,
    timezone: 'Europe/Istanbul',
    houseSystem: 'P',
  });

  console.log(`Güneş: ${test2.planets.find(p => p.name === 'Sun').formattedPosition}`);
  console.log(`Ay: ${test2.planets.find(p => p.name === 'Moon').formattedPosition}`);
  console.log(`Yükselen: ${test2.houses.ascendant.formatted}`);
  console.log(`DST aktif mi: ${test2.input.isDST}`);
  console.log(`UTC offset: ${test2.input.offsetHours} saat`);
} catch (e) {
```

```
  console.error('HATA:', e.message);
}

// ========== TEST 3: New York, 1985 ==========
console.log('\n📌 TEST 3: New York, 4 Mart 1985, 09:15');
console.log('-'.repeat(50));
try {
  const test3 = calculateNatalChart({
    year: 1985, month: 3, day: 4, hour: 9, minute: 15,
    latitude: 40.7128, longitude: -74.0060, // BATI BOYLAM = NEGATİF
    timezone: 'America/New_York',
    houseSystem: 'P',
  });

  console.log(`Güneş: ${test3.planets.find(p => p.name === 'Sun').formattedPosition}`);
  console.log(`Ay: ${test3.planets.find(p => p.name === 'Moon').formattedPosition}`);
  console.log(`Yükselen: ${test3.houses.ascendant.formatted}`);
} catch (e) {
  console.error('HATA:', e.message);
}

// ========== TEST 4: Yüksek enlem testi (Reykjavik, İzlanda) ==========
console.log('\n📌 TEST 4: Reykjavik, 21 Haziran 2000, 12:00 (yüksek enlem)');
console.log('-'.repeat(50));
try {
  const test4 = calculateNatalChart({
    year: 2000, month: 6, day: 21, hour: 12, minute: 0,
    latitude: 64.1466, longitude: -21.9426,
    timezone: 'Atlantic/Reykjavik',
    houseSystem: 'P',
  });

  console.log(`Yükselen: ${test4.houses.ascendant.formatted}`);
  console.log(`Uyarılar: ${test4.meta.warnings.length > 0 ? test4.meta.warnings.join('; ') : 'Yok'}`);
} catch (e) {
  console.error('HATA:', e.message);
}

// ========== TEST 5: Tam çıktı örneği ==========
console.log('\n📌 TEST 5: Tam JSON çıktısı (İstanbul, 15 Temmuz 1990)');
console.log('-'.repeat(50));
try {
  const fullChart = calculateNatalChart({
    year: 1990, month: 7, day: 15, hour: 14, minute: 30,
    latitude: 41.0082, longitude: 28.9784,
    timezone: 'Europe/Istanbul',
  });
```

```
// Gezegen pozisyonları tablosu
console.log('\nGezegen Pozisyonları:');
fullChart.planets.forEach(p => {
  const retro = p.isRetrograde ? ' ℞' : '';
  const dignity = p.dignity ? ` [${p.dignity}]` : '';
  console.log(`  ${p.trName.padEnd(20)} ${p.formattedPosition.padEnd(25)} Ev ${p.house}${retro}${dignity}`);
});

// Ev sınırları
console.log('\nEv Sınırları:');
fullChart.houses.cusps.forEach(h => {
  console.log(`  Ev ${String(h.house).padStart(2)}: ${h.formattedCusp}`);
});

// Aspektler (ilk 10)
console.log('\nAspektler (ilk 10):');
fullChart.aspects.slice(0, 10).forEach(a => {
  const applying = a.isApplying ? 'yaklaşan' : 'uzaklaşan';
  console.log(`  ${a.planet1} ${a.symbol} ${a.planet2}  ${a.type} (orb: ${a.orb}°, ${applying}, güç: ${a.strength}%)`);
});

// Analiz
console.log('\nAnaliz:');
console.log(`  Güneş Burcu: ${fullChart.analysis.sunSign}`);
console.log(`  Ay Burcu: ${fullChart.analysis.moonSign}`);
console.log(`  Yükselen: ${fullChart.analysis.risingSign}`);
console.log(`  Gündüz Haritası: ${fullChart.analysis.isDayChart}`);
console.log(`  Ay Fazı: ${fullChart.analysis.moonPhase?.phaseTr}`);
console.log(`  Part of Fortune: ${fullChart.analysis.partOfFortune.formatted}`);
console.log(`  Baskın Element: ${fullChart.analysis.elements.dominant}`);
console.log(`  Baskın Modalite: ${fullChart.analysis.modalities.dominant}`);

if (fullChart.analysis.stelliums.length > 0) {
  fullChart.analysis.stelliums.forEach(s => {
    console.log(`  Stellium: ${s.sign} (${s.planets.join(', ')})`);
  });
}

} catch (e) {
  console.error('HATA:', e.message);
}

console.log('\n' + '='.repeat(70));
console.log('Testler tamamlandı. Sonuçları astro.com ile karşılaştırmayı unutma!');
console.log('='.repeat(70));
```

ADIM 10: DOĞRULAMA PROSEDÜRÜ

Testleri çalıştırdıktan sonra sonuçları astro.com ile karşılaştır:

1. https://www.astro.com/cgi/chart.cgi adresine git
2. Aynı doğum bilgilerini gir
3. Şu değerleri karşılaştır (önem sırasına göre):

Karşılaştırma	Kabul Edilebilir Fark	Neden
Gezegen boylamları	≤ 1 arc-dakika (0°01')	Aynı ephemeris, birebir eşleşmeli
Ascendant	≤ 1 arc-dakika	Timezone hatası varsa burada görünür
MC	≤ 1 arc-dakika	ASC ile birlikte kontrol
Ay pozisyonu	≤ 1 arc-dakika	En hızlı cisim, zaman hatalarına en duyarlı
Ev sınırları	≤ 2 arc-dakika	Ev sistemine göre değişebilir

Eğer farklar bundan büyükse:

- İlk kontrol: Timezone dönüşümü doğru mu? (UTC saatini karşılaştır)
- İkinci kontrol: `jd_et` vs `jd_ut` doğru yerde mi kullanılıyor?
- Üçüncü kontrol: Boylam negatif/pozitif doğru mu? (Batı = negatif)

ÖNEMLİ NOTLAR VE UYARILAR

sweph API Hakkında Kritik Bilgiler

1. **`set_ephe_path()` HER ZAMAN çağrılmalı** — hesaplama öncesi. Null geçilse bile dahili initialization yapar.
2. **`calc_ut()` fonksiyonu `jd_et` alır** — isim "ut" dese de Ephemeris Time bekler. Bu Swiss Ephemeris'in kafa karıştıran bir naming convention'ı.
3. **`houses()` fonksiyonu `jd_ut` alır** — Universal Time. jd_et ile jd_ut'yi karıştırma!
4. **`SEFLG_SPEED` flag'i şart** — retrograd tespiti için hız verisi lazım. Bu flag olmadan `result.data[3]` her zaman 0 döner.
5. **Batı boylamları NEGATİF** — New York: -74.01, Londra: -0.12, İstanbul: 28.97 (doğu, pozitif)
6. **Month 1-based** — Ocak = 1, Şubat = 2... JavaScript Date'in 0-based convention'ından farklı!

7. **Return flag kontrolü** — `result.flag` input flag'den farklıysa Moshier fallback olmuş. Ephemeris dosyaları bulunamıyor olabilir.

Bilinen Sınırlamalar

- **Chiron**: Sadece 675 CE - 4650 CE arası güvenilir (modern kullanım için sorun değil)
- **Placidus/Koch**: ~66.5° üstü enlemlerde çalışmaz (otomatik Porphyry fallback)
- **Pre-1970 timezone**: IANA veritabanı sınırlı olabilir, uyarı üretilir
- **Thread safety**: `sweph` global state kullanır, worker_threads'de dikkatli ol

Gelecek Geliştirmeler (bu şartnamede yok)

- Sinastri (iki kişi karşılaştırma)
- Transit hesabı
- Solar/Lunar return
- Progresyon
- Vedik astroloji (sidereal zodiac) desteği
- SVG/Canvas doğum haritası görseli

PROJE ÇALIŞTIRMA

```bash
# Geliştirme modunda başlat
npm run dev

# Testleri çalıştır
npm test

# API'yi test et (curl ile)
curl -X POST http://localhost:3000/api/natal-chart \
  -H "Content-Type: application/json" \
  -d '{
    "year": 1990, "month": 7, "day": 15,
    "hour": 14, "minute": 30,
    "latitude": 41.0082, "longitude": 28.9784,
    "timezone": "Europe/Istanbul"
  }'
```

LİSANS

Bu proje AGPL-3.0 lisansı altındadır. Swiss Ephemeris, Astrodienst AG tarafından geliştirilmiştir. Ticari kapalı kaynak kullanım için Astrodienst'ten profesyonel lisans satın alınmalıdır (https://www.astro.com/swisseph/).