Astara

Mobile Planetarium

Complete Technical Documentation

Version: 1.0

Date: January 2026

Classification: RESTRICTED

Organization: Astara Development Team

Project: Astara Development Team

This document contains proprietary information

Table of Contents

Part I: Introduction

Documentation Index

Version: 1.0 Date: January 2026 Classification: RESTRICTED Organization: Astara Development Team

Overview

Astara is an advanced mobile planetarium application. Built on the Stellarium Web Engine, it provides real-time sky visualization with sensor-based interaction, augmented reality overlays, and comprehensive offline astronomical data.

Key Features

- Gyroscope Mode - Point your device to identify stars
- AR Camera Overlay - Stars overlaid on live camera feed
- Direction Tracking - Visual guide to locate objects
- Offline Operation - 60,000+ objects bundled locally
- Astronomical Calendar - Moon phases, eclipses, conjunctions
- Multi-Culture Support - IAU, Indian, Chinese sky cultures

Documentation Map

Design Documents

Document	Description	Status
HLD.md	High-Level Design - System a	Complete

LLD.md	Low-Level Design - Detailed	Complete

User Guides

Document	Description	Status
USER_MANUAL.md	Complete user guide for oper	Complete
setup.md	Build and installation instr	Complete

Architecture & Technical

Document	Description	Status
architecture.md	System architecture overview	Complete
frontend.md	Vue.js frontend guide	Complete
upstream.md	Relationship with Stellarium	Complete

Engine Documentation

Document	Description	Status
internals.md	Stellarium Web Engine intern	Complete

Quick Links

For Users

- Getting Started → USER_MANUAL.md
- Features → USER_MANUAL.md
- Troubleshooting → USER_MANUAL.md

For Developers

- Architecture → HLD.md
- Build Setup → setup.md
- Code Structure → LLD.md
- Services → LLD.md

For Administrators

- Installation → setup.md
- Configuration → LLD.md

Document Summary

HLD.md - High-Level Design

The High-Level Design document provides:

- Executive summary and purpose
- Three-layer architecture (Native, Web, Engine)
- Technology stack details
- Component design overview
- Data flow diagrams
- Deployment architecture
- Security considerations
- Performance optimization strategies

LLD.md - Low-Level Design

The Low-Level Design document covers:

- Complete project structure
- Engine object system (C/WASM)
- Module registration and rendering pipeline
- Frontend Vue.js architecture
- Service layer implementations (Gyroscope, Search, Astronomy)

- Vuex state management
- Component specifications with code
- Algorithm specifications
- Data format specifications
- Build system configuration
- Testing specifications

USER_MANUAL.md - User Manual

The User Manual includes:

- Introduction and system requirements
- Installation guide
- Quick start guide
- User interface explanation
- Core features (Gyroscope, AR, Constellations, Time)
- Search and navigation
- Settings and configuration
- Astronomical calendar
- Advanced features (DSO overlays, Satellites)
- Troubleshooting guide
- Reference appendices

Technology Stack Summary

Layer	Technology	Purpose
Engine	C + WebAssembly	Astronomical calculations
Frontend	Vue.js 2 + Vuex	User interface
Mobile	Capacitor 4	Android wrapper
Sensors	Device APIs	Gyroscope, Camera
Data	JSON + Binary	Star catalogs

Version History

Version	Date	Changes
1.0	January 2026	Initial documentation releas

Contributing

For documentation updates:

- Follow existing document structure
- Use markdown formatting consistently
- Update INDEX.md when adding documents
- Keep technical accuracy with code

Contact

For questions or issues regarding Astara documentation, contact the development team through official IAF channels.

Astara Development Team Astara Development Team

Part II: Design Documents

High-Level Design (HLD)

Version: 1.0 Date: January 2026 Classification: RESTRICTED Organization: Astara Development Team

1. Executive Summary

Astara is an advanced mobile planetarium application. Built on the Stellarium Web Engine, it provides real-time sky visualization with sensor-based interaction, augmented reality overlays, and comprehensive offline astronomical data.

1.1 Purpose

Astara enables field personnel and aviation crews to:

- Identify celestial objects by pointing their device at the sky
- Navigate using astronomical references
- Access comprehensive star catalogs and deep sky object data offline
- Plan observations based on astronomical events and visibility

1.2 Key Features

Feature	Description
Gyroscope Mode	Point device to view corresp
AR Camera Overlay	Stars overlaid on live camer
Direction Tracking	Visual guide to locate speci
Offline Operation	60,000+ objects bundled loca
Astronomical Calendar	Moon phases, eclipses, conju
Multi-Constellation Support	Multiple sky cultures (IAU,

2. System Architecture

2.1 Architecture Overview

Astara follows a three-layer architecture that separates platform concerns from computational logic:

```
┌─────────────────────────────────────────────────────────────────┐
│                      NATIVE MOBILE RUNTIME                      │
│                  (Capacitor / Android Platform)                 │
│  ┌─────────────┐  ┌─────────────┐  ┌──────────────────────────┐ │
│  │   Sensors   │  │   Camera    │  │    Platform Services     │ │
│  │ (Gyroscope) │  │    (AR)     │  │ (Permissions, Storage)   │ │
│  └─────────────┘  └─────────────┘  └──────────────────────────┘ │
├─────────────────────────────────────────────────────────────────┤
│                      WEB APPLICATION LAYER                      │
│                        (Vue.js Frontend)                        │
│  ┌─────────────┐  ┌─────────────┐  ┌──────────────────────────┐ │
│  │ Components  │  │  Services   │  │       Vuex Store         │ │
│  │  (UI/UX)    │  │ (Platform)  │  │   (State Management)     │ │
│  └─────────────┘  └─────────────┘  └──────────────────────────┘ │
├─────────────────────────────────────────────────────────────────┤
│                    STELLARIUM WEB ENGINE CORE                   │
│               (C / WebAssembly Computational Core)              │
│  ┌─────────────┐  ┌─────────────┐  ┌──────────────────────────┐ │
│  │   Modules   │  │ Projections │  │    Algorithms (ERFA)     │ │
│  │  (Sky Obj)  │  │ (Rendering) │  │    (Ephemeris Calc)      │ │
│  └─────────────┘  └─────────────┘  └──────────────────────────┘ │
└─────────────────────────────────────────────────────────────────┘
```

2.2 Layer Descriptions

2.2.1 Native Mobile Runtime (Capacitor Layer)

The outermost layer provides native platform integration:

- Capacitor Shell: Wraps the web app as a native Android application
- Sensor Access: Gyroscope, accelerometer, compass via Capacitor plugins
- Camera Integration: Native camera feed for AR mode
- Permissions: Location, camera, sensor permissions
- Window Management: Fullscreen, orientation lock, immersive mode

2.2.2 Web Application Layer (Vue.js Frontend)

The presentation and interaction layer:

- Vue 2 Framework: Component-based UI architecture
- Vuex State Management: Centralized state for app and engine sync
- Services: Platform abstraction modules (gyroscope, camera, search)
- Components: Reusable UI elements (menus, panels, dialogs)

2.2.3 Stellarium Web Engine Core (WASM)

The computational core compiled from C to WebAssembly:

- Astronomical Algorithms: Position calculations via ERFA library
- Rendering Engine: OpenGL ES-based sky visualization
- Module System: Extensible sky object management (stars, planets, DSOs)
- Projection System: Multiple view projections (perspective, stereographic)

3. Technology Stack

3.1 Core Technologies

Layer	Technology	Purpose
Engine	C/C++	Astronomical calculations
Compilation	Emscripten 1.40.1	C to WebAssembly compilation
Runtime	WebAssembly	Cross-platform execution
Frontend	Vue.js 2.x	User interface framework
State	Vuex	Application state management
Styling	CSS/SCSS	User interface styling
Build	Webpack	Module bundling
Mobile	Capacitor 4.x	Native Android wrapper
Platform	Android 8.0+	Mobile operating system

3.2 Astronomical Libraries

Library	Source	Function
ERFA	IAU Reference	Earth rotation, star positio
SGP4	NORAD	Satellite orbit propagation
HiPS	CDS/IVOA	Hierarchical Progressive Sur
Hipparcos	ESA	Star catalog (~120,000 stars

3.3 Data Formats

Format	Usage
.eph	Binary ephemeris data (stars
.json	Configuration, name indices
.png	Textures (Milky Way, planets
.geojson	Constellation boundaries

---

4. Component Design

4.1 Frontend Components

```
┌──────────────────────────────────────────────────────────────────┐
│                         App.vue                                  │
│  ┌─────────────────────────────────────────────────────────────┐ │
│  │                    Canvas (Engine View)                     │ │
│  │          ┌─────────────────────────────────────────┐        │ │
│  │          │          Stellarium Rendering           │        │ │
│  │          └─────────────────────────────────────────┘        │ │
│  ├─────────────────────────────────────────────────────────────┤ │
│  │                       Overlay Layer                         │ │
│  │  ┌───────────────┐  ┌───────────────┐  ┌─────────────────┐  │ │
│  │  │ Gyro Direction│  │  AR Camera    │  │  DSO Overlays   │  │ │
│  │  │    Overlay    │  │   Preview     │  │                 │  │ │
│  │  └───────────────┘  └───────────────┘  └─────────────────┘  │ │
│  ├─────────────────────────────────────────────────────────────┤ │
│  │                        UI Layer                             │ │
│  │  ┌───────────────┐  ┌───────────────┐  ┌─────────────────┐  │ │
```

```
│  │  │  Bottom Bar    │  │ Search Panel  │  │ Settings Panel  │  ││
│  │  │  (Controls)    │  │               │  │                 │  ││
│  │  └────────────────┘  └───────────────┘  └─────────────────┘  ││
│  └────────────────────────────────────────────────────────────────┘│
└────────────────────────────────────────────────────────────────────┘
```

4.2 Service Architecture

Service	File	Responsibility
GyroscopeService	gyroscope-service.js	Device orientation to sky co
CameraService	camera-service.js	Camera feed management for A
FullscreenService	fullscreen-service.js	Immersive mode control
SearchEngine	search_engine.js	Sky object search and filter
AstronomyService	astronomy-service.js	Event calculations (eclipses
CalendarService	calendar-service.js	Astronomical calendar events

4.3 Engine Modules

The Stellarium engine is organized into modules, each handling a specific category of sky objects:

Module	File	Objects Handled
Stars	stars.c	Hipparcos catalog stars
Planets	planets.c	Solar system bodies
DSO	dso.c	Galaxies, nebulae, clusters
Constellations	constellations.c	Constellation lines and art
Satellites	satellites.c	Artificial satellites (TLE)
Comets	comets.c	Comets and asteroids
Milky Way	milkyway.c	Galactic panorama texture
Atmosphere	atmosphere.c	Sky gradient and refraction
Landscape	landscape.c	Horizon panoramas

5. Data Flow

5.1 Sensor to View Pipeline

```
┌─────────────────┐
│  Device Sensors │
│  (Gyroscope,    │
│   Accelerometer)│
└────────┬────────┘
         │
         ▼
┌─────────────────┐
│ GyroscopeService│
│ (Orientation    │
│  Quaternion)    │
└────────┬────────┘
         │
         ▼
┌─────────────────┐
│   Vue/Vuex      │
│ (State Update)  │
└────────┬────────┘
         │
         ▼
┌─────────────────┐
│ Engine Observer │
│ (set azimuth,   │
│  altitude)      │
└────────┬────────┘
         │
         ▼
┌─────────────────┐
│ Engine Render   │
│ (Sky View)      │
└─────────────────┘
```

5.2 Search Flow

```
┌─────────────────┐
│   User Input    │
│   "Orion"       │
└────────┬────────┘
         │
         ▼
┌─────────────────┐
│  SearchEngine   │
```

```
│  querySkySources│
└─────────────────┘
         │
         │
    ┌────┴──────────────────────────────┐
    │                                   │
    ▼                                   ▼
┌─────────────┐             ┌─────────────────┐
│ Engine      │             │  Name Index     │
│ (live query)│             │  (JSON offline) │
└─────┬───────┘             └────────┬────────┘
      │                              │
      └───────────┬──────────────────┘
                  │
                  ▼
         ┌─────────────────┐
         │  Merge & Rank   │
         │  Results        │
         └────────┬────────┘
                  │
                  ▼
         ┌─────────────────┐
         │  Display List   │
         │  to User        │
         └─────────────────┘
```

6. Deployment Architecture

6.1 Build Pipeline

```
┌──────────────────────────────────────────────────────────────────┐
│                        BUILD PIPELINE                            │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│  ┌─────────────┐       ┌─────────────┐        ┌────────────┐    │
│  │  C Source   │ make  │  WASM +     │  copy  │  Frontend  │    │
│  │  (Engine)   │ ────► │  JS Glue    │ ─────► │  Assets    │    │
│  └─────────────┘       └─────────────┘        └─────┬──────┘    │
│                                                      │           │
│  ┌─────────────┐                                     │           │
```

```
│  │ Vue Source │                                   │         │
│  │ (Frontend) │                                   │         │
│  └──────┬─────┘                                   │         │
│         │                                         │         │
│         └──────────────────────┬──────────────────┘         │
│                                │                            │
│                                ▼                            │
│                         ┌─────────────┐                     │
│                         │  npm build  │                     │
│                         │  (Webpack)  │                     │
│                         └──────┬──────┘                     │
│                                │                            │
│                                ▼                            │
│                         ┌─────────────┐                     │
│                         │  Capacitor  │                     │
│                         │    Sync     │                     │
│                         └──────┬──────┘                     │
│                                │                            │
│                                ▼                            │
│                         ┌─────────────┐                     │
│                         │   Android   │                     │
│                         │     APK     │                     │
│                         └─────────────┘                     │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

6.2 Output Artifacts

Artifact	Location	Size
stellarium-web-engine.wasm	apps/web-frontend/src/assets	~3 MB
stellarium-web-engine.js	apps/web-frontend/src/assets	~200 KB
dist/	apps/web-frontend/dist/	~50 MB
app-debug.apk	android/app/build/outputs/ap	~80 MB

---

7. Security Considerations

7.1 Data Security

Concern	Mitigation
Offline data integrity	Data bundled at build time,
Location privacy	Location stored locally only
No network dependency	All calculations performed l

7.2 Permission Model

Permission	Purpose	Required
CAMERA	AR overlay mode	Optional
ACCESSFINELOCATION	Observer position	Optional
INTERNET	Not required for operation	No

8. Performance Considerations

8.1 Rendering Optimization

- 60 FPS target on mid-range devices
- Level-of-detail for star rendering based on magnitude
- Tile-based loading for HiPS surveys
- Lazy loading for DSO images

8.2 Memory Management

Resource	Strategy
Star catalog	Binary format, memory-mapped
Textures	Loaded on demand, cached
Search index	Lazy-loaded on first search
DSO HiPS	Progressive loading

8.3 Battery Optimization

- Sensor polling rate adjustable
- Frame rate reduction when idle
- Camera disabled when not in AR mode

9. Scalability

9.1 Data Extensibility

Extension Point	Method
Additional stars	Add to .eph files
New DSO catalogs	Extend name index
Custom landscapes	Add panorama images
Sky cultures	Add constellation data

9.2 Platform Extensibility

- iOS support: Capacitor-ready architecture
- Web deployment: No platform-specific code in frontend
- Desktop: Electron wrapper possible

10. Glossary

Term	Definition
DSO	Deep Sky Object (galaxies, n
ERFA	Essential Routines for Funda
HiPS	Hierarchical Progressive Sur

TLE	Two-Line Element (satellite
WASM	WebAssembly (portable binary
Ephemeris	Table of celestial object po
Azimuth	Horizontal angle from north
Altitude	Vertical angle above horizon
Magnitude	Logarithmic brightness scale
Right Ascension	Celestial longitude (hours/m
Declination	Celestial latitude (degrees)

11. References

- Stellarium Web Engine - https://github.com/Stellarium/stellarium-web-engine
- ERFA Library - https://github.com/liberfa/erfa
- Capacitor Documentation - https://capacitorjs.com/docs
- HiPS Standard - https://www.ivoa.net/documents/HiPS/
- Hipparcos Catalog - https://www.cosmos.esa.int/web/hipparcos

Document prepared by: Astara Development Team Last updated: January 2026

Low-Level Design (LLD)

Version: 1.0 Date: January 2026 Classification: RESTRICTED Organization: Astara Development Team

Table of Contents

1. Project Structure

1.1 Root Directory Structure

```
astara/
├── apps/                           # Application packages
│   ├── web-frontend/               # Vue.js frontend application
│   ├── simple-html/                # Minimal test harness
│   └── test-skydata/               # Test data files
├── src/                            # Stellarium Web Engine source (C)
│   ├── algos/                      # Astronomical algorithms
```

```
│   ├── modules/                # Sky object modules
│   ├── projections/            # Map projections
│   ├── utils/                  # Utility functions
│   └── js/                     # JavaScript bindings
├── ext_src/                    # External dependencies
│   ├── erfa/                   # ERFA astronomical library
│   ├── stb/                    # Image loading
│   └── zlib/                   # Compression
├── data/                       # Bundled astronomical data
├── scripts/                    # Python utility scripts
├── tools/                      # Build and data tools
├── doc/                        # Engine documentation
├── docs/                       # Application documentation
├── android/                    # Android Capacitor project
├── Makefile                    # Engine build
├── SConstruct                  # SCons build config
└── README.md                   # Project overview
```

1.2 Frontend Structure

```
apps/web-frontend/
├── src/
│   ├── App.vue                 # Root Vue component
│   ├── main.js                 # Application entry point
│   ├── assets/                 # Static assets
│   │   ├── js/                 # WASM engine files
│   │   │   ├── stellarium-web-engine.js
│   │   │   └── stellarium-web-engine.wasm
│   │   ├── images/             # UI images
│   │   ├── gyroscope-service.js    # Sensor service
│   │   ├── camera-service.js       # AR camera service
│   │   ├── search_engine.js        # Search functionality
│   │   ├── astronomy-service.js    # Event calculations
│   │   └── fullscreen-service.js   # Fullscreen control
│   ├── components/             # Vue components
│   │   ├── bottom-bar.vue          # Main control bar
│   │   ├── search-panel.vue        # Search interface
│   │   ├── settings-panel.vue      # Settings drawer
│   │   ├── calendar-panel.vue      # Astronomical calendar
│   │   ├── GyroDirectionOverlay.vue
│   │   ├── AR-camera-preview.vue
│   │   └── ...
│   ├── store/                  # Vuex store
│   │   └── index.js
│   ├── plugins/                # Vue plugins
```

```
│   ├── locales/                # i18n translations
│   └── constants/              # Configuration constants
├── public/                     # Static public files
│   └── skydata/                # Astronomical data
├── android/                    # Capacitor Android project
├── package.json
├── vue.config.js
└── capacitor.config.json
```

2. Engine Architecture

2.1 Object System

The engine uses a C-based object-oriented system centered on obj_t:

```
// Base object structure
typedef struct obj_t {
    obj_klass_t *klass;        // Class definition
    char        *id;           // Unique identifier
    obj_t       *parent;       // Parent object
    obj_t       *children;     // Linked list of children
    obj_t       *next;         // Sibling link
    int         ref;           // Reference count

    // Position data for sky objects
    struct {
        double pvg[2][3];      // Position/velocity (geocentric)
        double unit;           // Distance unit
        double g_ra, g_dec;    // Geocentric RA/Dec
        double ra, dec;        // Apparent RA/Dec
        double az, alt;        // Azimuth/Altitude
    } pos;
} obj_t;

// Class definition
typedef struct obj_klass_t {
    const char *id;
    size_t size;
    uint32_t flags;
```

```
    // Virtual methods
    int (*init)(obj_t *obj, json_value *args);
    void (*del)(obj_t *obj);
    int (*update)(obj_t *obj, double dt);
    int (*render)(const obj_t *obj, const painter_t *painter);
    obj_t* (*get)(const obj_t *obj, const char *id, int flags);
    int (*list)(const obj_t *obj, observer_t *obs,
                double max_mag, uint64_t hint, void *user,
                int (*f)(void *user, obj_t *obj));
} obj_klass_t;
```

2.2 Module Registration

Modules are registered at compile time:

```
// Example: Stars module registration
static obj_klass_t stars_klass = {
    .id = "stars",
    .size = sizeof(stars_t),
    .flags = OBJ_IN_JSON_TREE | OBJ_MODULE,
    .init = stars_init,
    .update = stars_update,
    .render = stars_render,
    .get = stars_get,
    .list = stars_list,
    .attributes = (attribute_t[]) {
        ATTR("visible", "b", MEMBER(stars_t, visible)),
        ATTR("hints_mag_offset", "f", MEMBER(stars_t, hints_mag_offset)),
        {}
    },
};
MODULE_REGISTER(stars, &stars_klass, 0)
```

2.3 Core Modules

Module	File	Responsibility
core	core.c	Main engine, observer, time
stars	modules/stars.c	Hipparcos star catalog
planets	modules/planets.c	Solar system bodies
dso	modules/dso.c	Deep sky objects

constellations	modules/constellations.c	Constellation lines/art
satellites	modules/satellites.c	TLE-based satellites
lines	modules/lines.c	Equatorial/azimuthal grids
milkyway	modules/milkyway.c	Galactic texture
atmosphere	modules/atmosphere.c	Sky gradient
landscape	modules/landscape.c	Horizon panorama

2.4 Rendering Pipeline

```
// Main render loop
void core_render(core_t *core, double aspect, void *user) {
    painter_t painter;

    // Setup painter
    painter_init(&painter, core->proj, core->observer);

    // Render in order (back to front)
    module_render(core->milkyway, &painter);
    module_render(core->dso, &painter);
    module_render(core->stars, &painter);
    module_render(core->constellations, &painter);
    module_render(core->planets, &painter);
    module_render(core->satellites, &painter);
    module_render(core->lines, &painter);
    module_render(core->atmosphere, &painter);
    module_render(core->landscape, &painter);
    module_render(core->labels, &painter);
}
```

2.5 Projection System

Supported projections:

```
// Projection function signature
typedef bool (*proj_func_t)(
    const projection_t *proj,
    int flags,
    int out_dim,
    const double *v,
    double *out
);
```

```
// Available projections
typedef enum {
    PROJ_PERSPECTIVE,      // Normal camera view
    PROJ_STEREOGRAPHIC,    // Wide-angle, preserves circles
    PROJ_MERCATOR,         // Cylindrical
    PROJ_HEALPIX,          // HiPS survey projection
    PROJ_TOAST,            // TOAST survey projection
} projection_type_t;

// Projection flags
enum {
    PROJ_NO_CLIP            = 1 << 0,  // Skip clipping
    PROJ_BACKWARD           = 1 << 1,  // 2D → 3D
    PROJ_TO_NDC_SPACE       = 1 << 2,  // Output NDC
    PROJ_ALREADY_NORMALIZED = 1 << 3,  // Input normalized
};
```

3. Frontend Architecture

3.1 Vue Application Structure

```
// main.js - Application entry point
import Vue from 'vue'
import Vuex from 'vuex'
import App from './App.vue'
import store from './store'

Vue.use(Vuex)

// Initialize Stellarium engine
import StelWebEngine from '@/assets/js/stellarium-web-engine.js'

new Vue({
  store,
  render: h => h(App),
  beforeCreate() {
    // Engine initialization
    StelWebEngine({
      wasmFile: '/assets/js/stellarium-web-engine.wasm',
```

```
      onReady: (stel) => {
        this.$stel = stel
        Vue.prototype.$stel = stel
        store.commit('replaceStelWebEngine', stel)
      }
    })
  }
}).$mount('#app')
```

3.2 App.vue Structure

```
<template>
  <div id="app" :class="{ fullscreen: isFullscreen }">
    <!-- Engine canvas -->
    <canvas ref="stelCanvas" id="stel-canvas"></canvas>

    <!-- Overlay layers -->
    <GyroDirectionOverlay v-if="showGyroOverlay" />
    <AR-camera-preview v-if="arModeActive" />
    <DsoSkyOverlays v-if="showDsoOverlays" />

    <!-- UI layers -->
    <selected-object-info v-if="selectedObject" />
    <search-panel v-if="searchPanelOpen" />
    <settings-panel v-if="settingsPanelOpen" />
    <calendar-panel v-if="calendarPanelOpen" />

    <!-- Bottom bar (always visible) -->
    <bottom-bar />
  </div>
</template>

<script>
export default {
  name: 'App',
  computed: {
    ...mapState([
      'gyroModeActive',
      'arModeActive',
      'selectedObject',
      'searchPanelOpen',
      'settingsPanelOpen',
      'calendarPanelOpen'
    ])
  },
```

```
  mounted() {
    this.initializeEngine()
    this.initializeSensors()
  }
}
</script>
```

---

4. Service Layer Design

4.1 Gyroscope Service

```
// gyroscope-service.js

class GyroscopeService {
  constructor() {
    this.isActive = false
    this.calibrated = false
    this.quaternion = [0, 0, 0, 1]
    this.listeners = []

    // Sensor configuration
    this.config = {
      frequency: 60,           // Hz
      smoothingFactor: 0.2,    // Low-pass filter
      useCompass: true,        // Combine with magnetometer
    }
  }

  async start() {
    // Check for sensor support
    if (!window.DeviceOrientationEvent) {
      throw new Error('Device orientation not supported')
    }

    // Request permission (iOS 13+)
    if (typeof DeviceOrientationEvent.requestPermission === 'function') {
      const permission = await DeviceOrientationEvent.requestPermission()
      if (permission !== 'granted') {
        throw new Error('Sensor permission denied')
```

```
      }
    }

    // Start listening
    window.addEventListener('deviceorientation',
this.handleOrientation.bind(this))
    this.isActive = true
  }

  handleOrientation(event) {
    const { alpha, beta, gamma } = event

    // Convert Euler angles to quaternion
    const quat = this.eulerToQuaternion(alpha, beta, gamma)

    // Apply smoothing
    this.quaternion = this.slerp(this.quaternion, quat,
this.config.smoothingFactor)

    // Convert to azimuth/altitude
    const { azimuth, altitude } = this.quaternionToAltAz(this.quaternion)

    // Notify listeners
    this.notifyListeners({ azimuth, altitude, quaternion: this.quaternion })
  }

  eulerToQuaternion(alpha, beta, gamma) {
    // Convert degrees to radians
    const a = (alpha || 0) * Math.PI / 180
    const b = (beta || 0) * Math.PI / 180
    const g = (gamma || 0) * Math.PI / 180

    // Standard Euler to quaternion conversion
    const c1 = Math.cos(a / 2)
    const s1 = Math.sin(a / 2)
    const c2 = Math.cos(b / 2)
    const s2 = Math.sin(b / 2)
    const c3 = Math.cos(g / 2)
    const s3 = Math.sin(g / 2)

    return [
      s1 * c2 * c3 - c1 * s2 * s3,
      c1 * s2 * c3 + s1 * c2 * s3,
      c1 * c2 * s3 + s1 * s2 * c3,
      c1 * c2 * c3 - s1 * s2 * s3
    ]
  }
```

```
  quaternionToAltAz(q) {
    // Convert quaternion to viewing direction
    // Returns azimuth (0-360) and altitude (-90 to 90)

    // Extract rotation matrix from quaternion
    const [x, y, z, w] = q

    // Calculate forward vector
    const fx = 2 * (x * z + w * y)
    const fy = 2 * (y * z - w * x)
    const fz = 1 - 2 * (x * x + y * y)

    // Convert to alt/az
    const altitude = Math.asin(-fy) * 180 / Math.PI
    let azimuth = Math.atan2(fx, fz) * 180 / Math.PI
    if (azimuth < 0) azimuth += 360

    return { azimuth, altitude }
  }

  stop() {
    window.removeEventListener('deviceorientation', this.handleOrientation)
    this.isActive = false
  }

  subscribe(callback) {
    this.listeners.push(callback)
    return () => {
      this.listeners = this.listeners.filter(l => l !== callback)
    }
  }

  notifyListeners(data) {
    this.listeners.forEach(cb => cb(data))
  }
}

export default new GyroscopeService()
```

4.2 Search Engine

```
// search_engine.js

class SearchEngine {
```

```
  constructor() {
    this.nameIndex = null
    this.isLoaded = false
  }

  async loadIndex() {
    if (this.isLoaded) return

    const response = await fetch('/skydata/name_index_compact.json')
    this.nameIndex = await response.json()
    this.isLoaded = true
  }

  async search(query, options = {}) {
    const {
      maxResults = 50,
      categories = ['constellation', 'planet', 'star', 'dso'],
      minScore = 0.1
    } = options

    await this.loadIndex()

    const normalizedQuery = query.toLowerCase().trim()
    const results = []

    // Search priority order
    const priorities = {
      constellation: 100,
      planet: 90,
      star: 80,
      dso: 70,
      satellite: 60
    }

    // Search exact matches first
    for (const [name, data] of Object.entries(this.nameIndex)) {
      const normalizedName = name.toLowerCase()

      if (!categories.includes(data.type)) continue

      let score = 0

      if (normalizedName === normalizedQuery) {
        score = 1.0  // Exact match
      } else if (normalizedName.startsWith(normalizedQuery)) {
        score = 0.8  // Starts with
      } else if (normalizedName.includes(normalizedQuery)) {
```

```
        score = 0.5  // Contains
      }

      if (score >= minScore) {
        results.push({
          name,
          ...data,
          score: score + (priorities[data.type] || 0) / 1000
        })
      }
    }

    // Sort by score and limit
    return results
      .sort((a, b) => b.score - a.score)
      .slice(0, maxResults)
  }

  // Query engine directly for live objects
  queryEngine(stel, query) {
    const results = []

    // Search planets
    const planets = ['mercury', 'venus', 'mars', 'jupiter', 'saturn', 'uranus',
'neptune']
    planets.forEach(p => {
      if (p.includes(query.toLowerCase())) {
        const obj = stel.getObj(`planet/${p}`)
        if (obj) results.push({ name: p, type: 'planet', obj })
      }
    })

    // Search by designation
    const obj = stel.getObjByName(query)
    if (obj) {
      results.push({ name: query, obj })
    }

    return results
  }
}

export default new SearchEngine()
```

4.3 Astronomy Service

```
// astronomy-service.js

import * as Astronomy from 'astronomy-engine'

class AstronomyService {

  // Calculate moon phases for a month
  getMoonPhases(year, month) {
    const phases = []
    const date = new Date(year, month - 1, 1)
    const endDate = new Date(year, month, 0)

    while (date <= endDate) {
      const phase = Astronomy.MoonPhase(date)

      // Check for exact phase moments
      const newMoon = Astronomy.SearchMoonQuarter(date)
      if (newMoon && this.isSameMonth(newMoon.time.date, year, month)) {
        phases.push({
          type: this.getPhaseType(newMoon.quarter),
          date: newMoon.time.date,
          illumination: this.getMoonIllumination(newMoon.time.date)
        })
      }

      date.setDate(date.getDate() + 1)
    }

    return phases
  }

  getPhaseType(quarter) {
    const types = ['New Moon', 'First Quarter', 'Full Moon', 'Last Quarter']
    return types[quarter] || 'Unknown'
  }

  getMoonIllumination(date) {
    const phase = Astronomy.MoonPhase(date)
    return (1 - Math.cos(phase * Math.PI / 180)) / 2
  }

  // Calculate eclipses
  getEclipses(startYear, endYear) {
    const eclipses = []

    // Search for lunar eclipses
    let lunarSearch = Astronomy.SearchLunarEclipse(new Date(startYear, 0, 1))
```

```
    while (lunarSearch.peak.date.getFullYear() <= endYear) {
      eclipses.push({
        type: 'Lunar Eclipse',
        subtype: this.getLunarEclipseType(lunarSearch.kind),
        date: lunarSearch.peak.date,
        magnitude: lunarSearch.sd_total
      })
      lunarSearch = Astronomy.NextLunarEclipse(lunarSearch.peak)
    }

    return eclipses.sort((a, b) => a.date - b.date)
  }

  // Calculate planet conjunctions
  getPlanetConjunctions(year) {
    const conjunctions = []
    const planets = ['mercury', 'venus', 'mars', 'jupiter', 'saturn']

    for (let i = 0; i < planets.length - 1; i++) {
      for (let j = i + 1; j < planets.length; j++) {
        const conj = this.findConjunction(
          planets[i],
          planets[j],
          new Date(year, 0, 1),
          new Date(year, 11, 31)
        )
        if (conj) {
          conjunctions.push({
            type: 'Conjunction',
            bodies: [planets[i], planets[j]],
            date: conj.date,
            separation: conj.separation
          })
        }
      }
    }

    return conjunctions
  }

  findConjunction(body1, body2, startDate, endDate) {
    // Simplified conjunction search
    let minSep = Infinity
    let conjDate = null

    const date = new Date(startDate)
    while (date <= endDate) {
```

```
      const pos1 = Astronomy.GeoVector(body1, date, false)
      const pos2 = Astronomy.GeoVector(body2, date, false)

      const sep = this.angularSeparation(pos1, pos2)

      if (sep < minSep && sep < 5) { // Within 5 degrees
        minSep = sep
        conjDate = new Date(date)
      }

      date.setDate(date.getDate() + 1)
    }

    if (conjDate) {
      return { date: conjDate, separation: minSep }
    }
    return null
  }

  angularSeparation(pos1, pos2) {
    // Calculate angular separation in degrees
    const dot = pos1.x * pos2.x + pos1.y * pos2.y + pos1.z * pos2.z
    const r1 = Math.sqrt(pos1.x**2 + pos1.y**2 + pos1.z**2)
    const r2 = Math.sqrt(pos2.x**2 + pos2.y**2 + pos2.z**2)

    const cosAngle = dot / (r1 * r2)
    return Math.acos(Math.max(-1, Math.min(1, cosAngle))) * 180 / Math.PI
  }
}

export default new AstronomyService()
```

5. State Management

5.1 Vuex Store Structure

```
// store/index.js

import Vue from 'vue'
```

```
import Vuex from 'vuex'

Vue.use(Vuex)

export default new Vuex.Store({
  state: {
    // Engine reference
    stel: null,

    // UI state
    searchPanelOpen: false,
    settingsPanelOpen: false,
    calendarPanelOpen: false,

    // Feature state
    gyroModeActive: false,
    arModeActive: false,
    sensorsEnabled: true,

    // Selection
    selectedObject: null,

    // User preferences
    favorites: [],
    recentSearches: [],

    // Location
    location: {
      lat: 28.6139,  // Default: New Delhi
      lng: 77.2090,
      alt: 0,
      name: 'New Delhi'
    },

    // Engine state mirror
    stelState: {
      constellations: {
        lines_visible: true,
        labels_visible: true,
        art_visible: false
      },
      atmosphere: {
        visible: true
      },
      stars: {
        visible: true,
        mag_limit: 6.5
```

```
    },
    dso: {
      visible: true
    },
    planets: {
      visible: true
    },
    grids: {
      equatorial: false,
      azimuthal: false
    }
  }
},

mutations: {
  replaceStelWebEngine(state, stel) {
    state.stel = stel
  },

  setGyroModeActive(state, active) {
    state.gyroModeActive = active
  },

  setArModeActive(state, active) {
    state.arModeActive = active
  },

  setSelectedObject(state, obj) {
    state.selectedObject = obj
  },

  addFavorite(state, obj) {
    if (!state.favorites.find(f => f.id === obj.id)) {
      state.favorites.push(obj)
      localStorage.setItem('favorites', JSON.stringify(state.favorites))
    }
  },

  removeFavorite(state, id) {
    state.favorites = state.favorites.filter(f => f.id !== id)
    localStorage.setItem('favorites', JSON.stringify(state.favorites))
  },

  setLocation(state, location) {
    state.location = location
    if (state.stel) {
      state.stel.core.observer.latitude = location.lat * Math.PI / 180
```

```
      state.stel.core.observer.longitude = location.lng * Math.PI / 180
    }
  },

  updateStelState(state, { path, value }) {
    // Update mirrored engine state
    const keys = path.split('.')
    let obj = state.stelState
    for (let i = 0; i < keys.length - 1; i++) {
      obj = obj[keys[i]]
    }
    obj[keys[keys.length - 1]] = value
  }
},

actions: {
  async toggleGyroMode({ commit, state }) {
    if (state.gyroModeActive) {
      gyroscopeService.stop()
      commit('setGyroModeActive', false)
    } else {
      await gyroscopeService.start()
      commit('setGyroModeActive', true)
    }
  },

  async toggleArMode({ commit, state }) {
    if (state.arModeActive) {
      cameraService.stop()
      commit('setArModeActive', false)
    } else {
      await cameraService.start()
      commit('setArModeActive', true)
    }
  },

  selectObject({ commit, state }, obj) {
    commit('setSelectedObject', obj)
    state.recentSearches.unshift({
      id: obj.id,
      name: obj.name,
      type: obj.type
    })
    state.recentSearches = state.recentSearches.slice(0, 20)
  }
},
```

```
  getters: {
    isObjectSelected: state => !!state.selectedObject,

    favoriteIds: state => state.favorites.map(f => f.id),

    engineReady: state => !!state.stel
  }
})
```

6. Component Specifications

6.1 Bottom Bar Component

```
<!-- bottom-bar.vue -->
<template>
  <div class="bottom-bar">
    <bottom-button
      v-for="btn in buttons"
      :key="btn.id"
      :icon="btn.icon"
      :active="btn.isActive"
      @tap="btn.onTap"
      @longpress="btn.onLongPress"
    />
  </div>
</template>

<script>
export default {
  name: 'BottomBar',
  computed: {
    buttons() {
      return [
        {
          id: 'search',
          icon: 'search',
          isActive: this.$store.state.searchPanelOpen,
          onTap: () => this.togglePanel('search'),
          onLongPress: null
```

```
        },
        {
          id: 'constellations',
          icon: 'star',
          isActive: this.constellationLinesVisible,
          onTap: () => this.toggleConstellationLines(),
          onLongPress: () => this.openConstellationSettings()
        },
        {
          id: 'gyro',
          icon: 'compass',
          isActive: this.$store.state.gyroModeActive,
          onTap: () => this.$store.dispatch('toggleGyroMode'),
          onLongPress: () => this.openSensorSettings()
        },
        {
          id: 'camera',
          icon: 'camera',
          isActive: this.$store.state.arModeActive,
          onTap: () => this.$store.dispatch('toggleArMode'),
          onLongPress: () => this.openCameraSettings()
        },
        {
          id: 'calendar',
          icon: 'calendar',
          isActive: this.$store.state.calendarPanelOpen,
          onTap: () => this.togglePanel('calendar'),
          onLongPress: null
        },
        {
          id: 'settings',
          icon: 'settings',
          isActive: this.$store.state.settingsPanelOpen,
          onTap: () => this.togglePanel('settings'),
          onLongPress: null
        }
      ]
    }
  }
}
</script>

<style scoped>
.bottom-bar {
  position: fixed;
  bottom: 0;
  left: 0;
```

```
  right: 0;
  height: 60px;
  background: rgba(0, 0, 0, 0.8);
  display: flex;
  justify-content: space-around;
  align-items: center;
  z-index: 100;
}
</style>
```

6.2 Gyro Direction Overlay

```
<!-- GyroDirectionOverlay.vue -->
<template>
  <div class="gyro-overlay" :class="{ active: isActive }">
    <!-- Direction arrow for off-screen objects -->
    <div
      v-if="targetOffScreen"
      class="direction-arrow"
      :style="arrowStyle"
    >
      <svg viewBox="0 0 24 24" class="arrow-icon">
        <path d="M12 2L22 12L12 22L12 14L2 14L2 10L12 10Z"/>
      </svg>
      <span class="target-name">{{ targetName }}</span>
    </div>

    <!-- Calibration indicator -->
    <div class="calibration-status" :class="calibrationStatus">
      <span>{{ calibrationText }}</span>
    </div>
  </div>
</template>

<script>
export default {
  name: 'GyroDirectionOverlay',
  props: {
    targetObject: Object
  },
  data() {
    return {
      calibrationLevel: 0  // 0-1
    }
  },
```

```
  computed: {
    isActive() {
      return this.$store.state.gyroModeActive
    },
    targetOffScreen() {
      if (!this.targetObject) return false
      // Check if target is outside current view
      return !this.isInView(this.targetObject)
    },
    arrowStyle() {
      if (!this.targetObject) return {}
      const angle = this.calculateDirectionAngle()
      return {
        transform: `rotate(${angle}deg)`
      }
    },
    calibrationStatus() {
      if (this.calibrationLevel > 0.8) return 'good'
      if (this.calibrationLevel > 0.5) return 'fair'
      return 'poor'
    }
  },
  methods: {
    calculateDirectionAngle() {
      // Calculate angle from center to target
      const targetAz = this.targetObject.azimuth
      const currentAz = this.$store.state.currentAzimuth
      return (targetAz - currentAz + 360) % 360
    },
    isInView(obj) {
      // Check if object is within current view frustum
      const fov = this.$stel.core.fov
      const deltaAz = Math.abs(obj.azimuth - this.$store.state.currentAzimuth)
      const deltaAlt = Math.abs(obj.altitude -
this.$store.state.currentAltitude)
      return deltaAz < fov / 2 && deltaAlt < fov / 2
    }
  }
}
</script>
```

7. Algorithm Specifications

7.1 Sky Object Ranking

```
// Search result ranking algorithm

function rankSearchResults(results, query) {
  const weights = {
    exactMatch: 100,
    startsWithMatch: 50,
    containsMatch: 20,
    typeBonus: {
      constellation: 15,
      planet: 12,
      star: 8,
      dso: 5,
      satellite: 3
    },
    magnitudeBonus: (mag) => Math.max(0, 10 - mag),  // Brighter = higher
    visibilityBonus: (alt) => alt > 0 ? 10 : 0       // Above horizon
  }

  return results.map(result => {
    let score = 0
    const name = result.name.toLowerCase()
    const q = query.toLowerCase()

    // Match type
    if (name === q) {
      score += weights.exactMatch
    } else if (name.startsWith(q)) {
      score += weights.startsWithMatch
    } else if (name.includes(q)) {
      score += weights.containsMatch
    }

    // Type bonus
    score += weights.typeBonus[result.type] || 0

    // Magnitude bonus (for stars/DSOs)
    if (result.magnitude !== undefined) {
      score += weights.magnitudeBonus(result.magnitude)
    }
```

```
    // Visibility bonus
    if (result.altitude !== undefined) {
      score += weights.visibilityBonus(result.altitude)
    }

    return { ...result, score }
  }).sort((a, b) => b.score - a.score)
}
```

7.2 Gyroscope Smoothing

```
// Low-pass filter for sensor data

class SensorSmoother {
  constructor(alpha = 0.2) {
    this.alpha = alpha  // Smoothing factor (0 = no smoothing, 1 = no filtering)
    this.lastValue = null
  }

  smooth(newValue) {
    if (this.lastValue === null) {
      this.lastValue = newValue
      return newValue
    }

    // Exponential moving average
    this.lastValue = this.alpha * newValue + (1 - this.alpha) * this.lastValue
    return this.lastValue
  }

  // Spherical interpolation for quaternions
  slerp(q1, q2, t) {
    let dot = q1[0]*q2[0] + q1[1]*q2[1] + q1[2]*q2[2] + q1[3]*q2[3]

    // Ensure shortest path
    if (dot < 0) {
      q2 = q2.map(x => -x)
      dot = -dot
    }

    // If very close, use linear interpolation
    if (dot > 0.9995) {
      const result = q1.map((v, i) => v + t * (q2[i] - v))
      return this.normalize(result)
    }
```

```
    // Spherical interpolation
    const theta0 = Math.acos(dot)
    const theta = theta0 * t
    const sinTheta = Math.sin(theta)
    const sinTheta0 = Math.sin(theta0)

    const s0 = Math.cos(theta) - dot * sinTheta / sinTheta0
    const s1 = sinTheta / sinTheta0

    return q1.map((v, i) => s0 * v + s1 * q2[i])
  }

  normalize(q) {
    const len = Math.sqrt(q[0]**2 + q[1]**2 + q[2]**2 + q[3]**2)
    return q.map(v => v / len)
  }
}
```

8. Data Formats

8.1 Name Index Format

```
{
  "Sirius": {
    "type": "star",
    "id": "HIP 32349",
    "mag": -1.46,
    "ra": 101.2875,
    "dec": -16.7161,
    "alt_names": ["Alpha Canis Majoris", "Dog Star"]
  },
  "M31": {
    "type": "dso",
    "id": "NGC 224",
    "mag": 3.4,
    "ra": 10.6847,
    "dec": 41.2689,
    "alt_names": ["Andromeda Galaxy", "NGC 224"]
```

```
  },
  "Orion": {
    "type": "constellation",
    "id": "Ori",
    "stars": ["HIP 26727", "HIP 27989", "..."]
  }
}
```

8.2 Event Data Format

```
{
  "events": [
    {
      "type": "moon_phase",
      "subtype": "full_moon",
      "date": "2026-01-19T12:30:00Z",
      "data": {
        "illumination": 1.0
      }
    },
    {
      "type": "eclipse",
      "subtype": "lunar_total",
      "date": "2026-03-14T00:00:00Z",
      "data": {
        "magnitude": 1.2,
        "duration_minutes": 180
      }
    },
    {
      "type": "conjunction",
      "bodies": ["venus", "jupiter"],
      "date": "2026-02-20T06:00:00Z",
      "data": {
        "separation_degrees": 0.5
      }
    }
  ]
}
```

9. Build System

9.1 Makefile Targets

```
# Main targets
make              # Build WASM engine
make debug        # Build with debug symbols
make clean        # Clean build artifacts
make sync-android # Sync web build to Capacitor
make build-apk    # Build Android APK

# Engine build (SConstruct)
scons -j4 mode=release target=wasm
scons -j4 mode=debug target=wasm
```

9.2 npm Scripts

```
{
  "scripts": {
    "dev": "vue-cli-service serve",
    "build": "vue-cli-service build",
    "lint": "vue-cli-service lint",
    "android:sync": "npx cap sync android",
    "android:open": "npx cap open android"
  }
}
```

10. Testing Specifications

10.1 Unit Tests

```
// Example: Search engine tests
describe('SearchEngine', () => {
  beforeAll(async () => {
```

```
    await searchEngine.loadIndex()
  })

  test('exact match returns highest score', async () => {
    const results = await searchEngine.search('Sirius')
    expect(results[0].name).toBe('Sirius')
    expect(results[0].score).toBeGreaterThan(0.9)
  })

  test('partial match returns results', async () => {
    const results = await searchEngine.search('sir')
    expect(results.length).toBeGreaterThan(0)
    expect(results.some(r => r.name === 'Sirius')).toBe(true)
  })

  test('category filter works', async () => {
    const results = await searchEngine.search('M', { categories: ['dso'] })
    results.forEach(r => {
      expect(r.type).toBe('dso')
    })
  })
})
```

10.2 Integration Tests

```
// Example: Gyroscope service integration
describe('GyroscopeService Integration', () => {
  test('sensor data updates engine view', async () => {
    const mockStel = {
      core: {
        observer: {
          azimuth: 0,
          altitude: 0
        }
      }
    }

    gyroscopeService.subscribe(({ azimuth, altitude }) => {
      mockStel.core.observer.azimuth = azimuth
      mockStel.core.observer.altitude = altitude
    })

    // Simulate sensor event
    const event = new DeviceOrientationEvent('deviceorientation', {
      alpha: 180,
```

```
      beta: 45,
      gamma: 0
    })
    window.dispatchEvent(event)

    expect(mockStel.core.observer.azimuth).toBeCloseTo(180, 1)
    expect(mockStel.core.observer.altitude).toBeCloseTo(45, 1)
  })
})
```

Document prepared by: Astara Development Team Last updated: January 2026

Part III: User Guides

User Manual

Version: 1.0 Date: January 2026 Classification: RESTRICTED Organization: Astara Development Team

Table of Contents

1. Introduction

1.1 About Astara

Astara is a comprehensive mobile planetarium application. It transforms your Android device into a powerful astronomical tool that can identify stars, planets, constellations, and deep sky objects simply by pointing at the sky.

1.2 Key Capabilities

- Point and Identify: Use your device's sensors to identify what's in the sky
- Augmented Reality: Overlay star data on your camera feed
- Offline Operation: Full functionality without internet connection
- 60,000+ Objects: Stars, planets, galaxies, nebulae, satellites
- Multiple Sky Cultures: IAU, Indian, Chinese, Arabic constellations
- Astronomical Events: Moon phases, eclipses, planet conjunctions

1.3 System Requirements

Requirement	Specification
OS	Android 8.0 (Oreo) or higher
RAM	2 GB minimum, 4 GB recommend
Storage	200 MB free space
Sensors	Gyroscope, accelerometer (re
Camera	Rear camera (required for AR
GPS	For automatic location (opti

2. Installation

2.1 APK Installation

- Enable Unknown Sources
  - Go to Settings > Security > Install unknown apps
  - Enable installation for your file manager
- Install the APK
  - Locate the astara.apk file
  - Tap to install
  - Grant requested permissions
- Grant Permissions
  - Location (for accurate sky position)
  - Camera (for AR mode)

  - Sensors (automatically granted)

2.2 First Launch

On first launch, Astara will:

- Load the astronomical database (~5 seconds)
- Request necessary permissions
- Detect your location (if permitted)
- Display the current sky view

3. Quick Start Guide

3.1 Basic Operation

- Launch Astara - Tap the app icon
- Enable Gyroscope Mode - Tap the gyroscope icon in the bottom bar
- Point at the Sky - Hold your device towards the sky
- View Objects - Stars, constellations, and planets appear in real-time
- Tap an Object - Get detailed information

3.2 Control Gestures

Gesture	Action
Drag	Pan sky view (manual mode)
Pinch	Zoom in/out
Single Tap	Select celestial object
Double Tap	Center on tapped location
Long Press	Open context menu

3.3 Quick Access Icons

The bottom bar provides quick access to main features:

```
┌──────────────────────────────────────────────────────────────────┐
│  []    []    []    []    []    []                    │
│ Search  Stars  Gyro   Camera Calendar Settings           │
└──────────────────────────────────────────────────────────────────┘
```

4. User Interface

4.1 Main Screen Layout

```
┌──────────────────────────────────────────────────────────────────┐
│ [Time Display]                            [Location Display]  │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│                                                                  │
│                    SKY CANVAS                                    │
│                 (Main View Area)                                 │
│                                                                  │
│                                                                │
│                                                              │
│                                                              │
│                                                              │
│                                                                │
│                                                                  │
├──────────────────────────────────────────────────────────────────┤
│                     BOTTOM BAR                                   │
│  []    []    []    []    []    []                    │
└──────────────────────────────────────────────────────────────────┘
```

4.2 Bottom Bar Functions

Icon	Function	Tap	Long Press
	Search	Open search panel	-
	Constellations	Toggle lines	Constellation settings
	Gyroscope	Toggle sensor mode	Sensor settings
	AR Camera	Toggle AR overlay	Camera settings

	Calendar	Open calendar	-
	Settings	Open settings panel	-

4.3 Information Panel

When you select a celestial object, an information panel appears:

```
┌─────────────────────────────────────────────────────────────────┐
│                         SIRIUS                                  │
│                     Alpha Canis Majoris                         │
├─────────────────────────────────────────────────────────────────┤
│  Type:         Star                                             │
│  Magnitude:    -1.46 (Brightest star)                          │
│  Distance:     8.6 light years                                  │
│  RA/Dec:       06h 45m / -16° 43'                              │
│  Altitude:     45° 23'                                          │
│  Azimuth:      180° 15' (South)                                 │
├─────────────────────────────────────────────────────────────────┤
│  [Center View]    [Track Object]    [Add Favorite]              │
└─────────────────────────────────────────────────────────────────┘
```

5. Core Features

5.1 Gyroscope Mode

Gyroscope mode uses your device's sensors to display the sky region you're pointing at.

Enable Gyroscope Mode:

- Tap the compass icon () in the bottom bar
- Hold your device flat briefly for calibration
- Point towards the sky
- The view updates in real-time
- Calibrate by moving device in figure-8 pattern
- Avoid strong magnetic interference
- Hold device steady for stable view

- Works better outdoors away from buildings
- Green: Sensors active and calibrated
- Yellow: Sensors active, may need calibration
- Red: Sensor error

5.2 Augmented Reality (AR) Mode

AR mode overlays constellation lines and labels on your camera feed.

Enable AR Mode:

- Tap the camera icon () in the bottom bar
- Grant camera permission if prompted
- Point camera at the sky
- Stars and labels appear overlaid on the live feed
- Transparency: Adjust overlay opacity in settings
- Labels: Toggle star names on/off
- Lines: Toggle constellation lines

5.3 Constellation Display

Toggle constellation visualizations:

Mode	Description
Lines	Connect stars with lines
Art	Display mythology artwork
Boundaries	Show constellation boundarie
Labels	Show constellation names

Change Sky Culture:

- Open Settings ()
- Tap "Sky Culture"
- Select from:
  - IAU (International Astronomical Union)
  - Indian Traditional
  - Chinese
  - Arabic
  - And more...

5.4 Time Control

Astara can simulate the sky at any date and time.

Access Time Controls:

- Tap the time display at top of screen
- Use date/time picker to select
- Or tap Quick buttons:
  - Now: Return to current time
  - Sunrise/Sunset: Jump to these events
  - +1 Hour/-1 Hour: Step through time
- Tap play button to animate sky motion
- Adjust speed: 1x, 10x, 100x, 1000x

5.5 Location Settings

Automatic Location:

- Ensure GPS is enabled
- Open Settings > Location
- Tap "Use GPS Location"
- Open Settings > Location
- Enter coordinates manually:
  - Latitude: e.g., 28.6139
  - Longitude: e.g., 77.2090
- Or search by city name
- Save frequently used locations
- Quick switch between observing sites

---

6. Search and Navigation

6.1 Search Panel

Tap the search icon () to open the search panel.

Search by Name:

- Type object name (e.g., "Orion", "Mars", "M31")
- Results appear as you type
- Tap result to center view on object

Category	Examples
Constellations	Orion, Ursa Major, Scorpius
Stars	Sirius, Vega, Polaris, HIP 1
Planets	Mars, Jupiter, Saturn
DSOs	M31, NGC 224, Andromeda Gala
Satellites	ISS, Hubble

6.2 Direction Indicator

When tracking an object not in view:

- Select any object
- If object is below horizon or out of view
- An arrow indicator shows direction to object
- Follow arrow to locate object in sky

6.3 Favorites and Recent

Add to Favorites:

- Select an object
- Tap "Add Favorite" ()
- Object saved for quick access
- Open Search panel
- Tap "Favorites" or "Recent" tab
- Tap any item to navigate

7. Settings and Configuration

7.1 Display Settings

Access via Settings () > Display:

Setting	Options
Stars	Show/Hide, Limit magnitude
Star Names	Show/Hide, Limit by magnitud
Planets	Show/Hide, Show labels
DSOs	Show/Hide, Show markers
Atmosphere	Enable/Disable sky color sim
Milky Way	Show/Hide galactic band
Grid Lines	Equatorial, Azimuthal, Off

7.2 Star Rendering

Setting	Description
Limiting Magnitude	Faintest stars shown (defaul
Twinkling	Star scintillation effect
Colored Stars	Show star colors based on te
Size Scaling	Relative star sizes

7.3 Light Pollution

Simulate different observing conditions:

Bortle Scale	Description
1	Excellent dark site
4	Rural/suburban transition
6	Bright suburban
8-9	City center

7.4 Sensor Settings

Setting	Description
Sensitivity	Gyroscope response speed

Smoothing	Reduce jitter
Auto-Calibrate	Periodic recalibration
Compass Mode	Use compass with gyro

8. Astronomical Calendar

8.1 Calendar Panel

Tap the calendar icon () to view astronomical events.

Event Types:

Event	Icon	Description
Moon Phases		New, First Quarter, Full, La
Eclipses		Solar and Lunar eclipses
Conjunctions		Planet meetups
Meteor Showers		Peak dates
Planet Events		Opposition, Greatest Elongat

8.2 Filtering Events

- Open Calendar panel
- Tap filter icon
- Select event types to show
- Apply filter

8.3 Event Details

Tap any event to see:

- Date and time
- Visibility from your location
- Description

- Observing tips

9. Advanced Features

9.1 Deep Sky Object (DSO) Overlays

View high-resolution images of DSOs:

- Search for a DSO (e.g., "M42")
- Zoom in closely
- HiPS image overlay appears automatically
- Pinch to zoom into detailed structure

9.2 Satellite Tracking

View Satellites:

- Settings > Objects > Satellites
- Enable satellite display
- Satellites appear as moving dots
- Search "ISS"
- Tap to select
- View next visible pass times
- Track in real-time during pass

9.3 Screenshot Capture

- Compose desired view
- Settings > Screenshot
- Image saved to gallery

9.4 Observation Planning

- Set future date/time
- Check object visibility
- Note altitudes and directions

- Plan observation session

10. Troubleshooting

10.1 Gyroscope Issues

Problem: View drifts or jumps erratically

Solution:

- Calibrate by moving device in figure-8 pattern
- Move away from magnetic interference
- Restart the app
- In extreme cases, restart device
- Ensure device has gyroscope sensor
- Check app permissions
- Try enabling/disabling sensor mode

10.2 AR Camera Issues

Problem: Camera feed is black

Solution:

- Check camera permission
- Close other camera apps
- Restart Astara
- Ensure gyroscope is calibrated
- Hold device steady
- Check that location is set correctly

10.3 Performance Issues

Problem: App runs slowly

Solution:

- Close background apps

- Reduce star limiting magnitude
- Disable atmosphere effect
- Disable DSO overlays
- Clear app cache
- Reinstall app
- Check for updates

10.4 Location Issues

Problem: Wrong sky displayed (stars don't match)

Solution:

- Verify location is correctly set
- Check date and time settings
- Ensure time zone is correct

11. Appendix

11.1 Keyboard Shortcuts (When connected)

Key	Action
Space	Pause/Resume time
+/-	Zoom in/out
Arrow Keys	Pan view
F	Toggle fullscreen
G	Toggle gyroscope

11.2 Object Types Reference

Symbol	Type
	Star
●	Planet

○	Moon
	Moon Phase
	Galaxy
	Nebula
	Cluster
	Satellite

11.3 Magnitude Scale

Magnitude	Example	Visibility
-27	Sun	Daylight
-13	Full Moon	Bright night
-4	Venus (max)	Before sunrise/after sunset
-1.5	Sirius	Easy naked eye
0	Vega	Bright star
2	Polaris	Visible in city
4	Faint stars	Suburban sky
6	Limit naked eye	Dark sky only
10+	Telescope only	Requires equipment

11.4 Constellation Guide (Selected)

Constellation	Season	Notable Objects
Orion	Winter	Betelgeuse, Rigel, M42
Scorpius	Summer	Antares
Ursa Major	Year-round	Big Dipper, M81/M82
Leo	Spring	Regulus
Sagittarius	Summer	Galactic center, many DSOs
Cassiopeia	Year-round	W-shape, M52

11.5 Planet Visibility Quick Reference

Planet	Best Time	Brightness

Mercury	Twilight only	-2 to 0
Venus	Morning/Evening Star	-4.5 (very bright)
Mars	Near opposition	-2.9 (at best)
Jupiter	When visible	-2.9 to -2
Saturn	When visible	0 to +1
Uranus	Needs binoculars	+5.5
Neptune	Needs telescope	+7.8

Support

For technical support, contact the Astara development team through official IAF channels.

Document prepared by: Astara Development Team Last updated: January 2026

Setup & Installation

> *Windows users: Use WSL (Windows Subsystem for Linux) for the build process.*

> *Native Windows builds are not supported.*

Table of Contents

1. Setting Up Emscripten

Clone the Emscripten SDK:

```
git clone https://github.com/emscripten-core/emsdk.git
cd emsdk
```

Install and activate the SDK:

```
./emsdk install latest
./emsdk activate latest     # writes .emscripten file
source ./emsdk_env.sh       # Active PATH and other environment variables in
current terminal
```

Install additional versions required by Stellarium:

```
# Get a tip-of-tree
./emsdk install tot

# Install required emscripten version
./emsdk install 1.40.1
./emsdk activate 1.40.1
```

```
source ./emsdk_env.sh
```

> *Note: Stellarium Web Engine is sensitive to Emscripten versions.*

> *Version 1.40.1 is known to work reliably.*

Install scons (choose one):

```
sudo apt-get install scons        # Debian / Ubuntu
sudo dnf install scons            # Fedora
pip install scons                 # Python
```

---

2. Build Stellarium Web Engine (WASM + JS)

Ensure Docker is installed and running:

```
sudo systemctl status docker
sudo systemctl start docker
```

Clone the VayuView repository:

```
git clone https://github.com/DesolateSea/VayuView vayuview
cd VayuView
```

Activate Emscripten:

```
source /path/to/emsdk/emsdk_env.sh
```

Build the engine:

```
make
```

Copy the generated artifacts into the frontend:

```
cp build/stellarium-web-engine.* apps/web-frontend/src/assets/js/
```

This produces:

- stellarium-web-engine.js
- stellarium-web-engine.wasm

---

3. Build Web Frontend

Node.js 22 is recommended.

```
cd apps/web-frontend
npm install --legacy-peer-deps
```

Workaround for OpenSSL compatibility:

```
export NODE_OPTIONS=--openssl-legacy-provider
```

Run the dev server:

```
npm run dev
```

---

4. Build Android APK

Set required environment variables:

```
export CAPACITOR_ANDROID_STUDIO_PATH=/path/to/android-studio/bin/studio.sh
```

```
export JAVA_HOME=/path/to/java-21-openjdk
export PATH=$JAVA_HOME/bin:$PATH
```

Build the production frontend:

```
npm run build
```

Sync with Capacitor:

```
make sync-android
```

Build the debug APK:

```
make build-apk
```

The APK will be located at:

```
android/app/build/outputs/apk/debug/app-debug.apk
```

Part IV: Technical Reference

Architecture Overview

Architecture Overview

Astara is a mobile planetarium app built on the Stellarium Web Engine. We wrap the engine with a Vue frontend and package it as an Android app using Capacitor.

```
┌────────────────────────────────────────────────────────────┐
│                    Native Mobile Runtime                   │
│               (Capacitor / Android Platform)               │
├────────────────────────────────────────────────────────────┤
│                    Web Application Layer                   │
│                       (Vue Frontend)                       │
├────────────────────────────────────────────────────────────┤
│                 Stellarium Web Engine Core                 │
│             (C / WebAssembly Computational Core)           │
└────────────────────────────────────────────────────────────┘
```

Engine Layer

This is the Stellarium Web Engine core, compiled from C to WebAssembly. It does all the heavy work: astronomical calculations, sky state, projections, and rendering logic.

Nothing in this layer is mobile-specific. It behaves exactly like upstream.

If you want to understand how the engine is structured internally, read doc/internals.md.

Web Application Layer

This is the JavaScript / Vue layer that sits on top of the engine.

It handles things like:

- Taking user and sensor input.

- Converting that into engine state updates.
- Keeping UI state separate from simulation state.

Native Mobile Runtime

This is the Capacitor wrapper around the web app.

It mainly exists to provide the mobile shell: app manifest, permissions, window flags, orientation handling, and platform integration.

Most native functionality is accessed through @capacitor/* plugins from the JavaScript layer rather than custom Java / Kotlin code.

Data and Control Flow

The system follows a strict unidirectional flow model:

```
Native Sensors / OS Events
          ↓
  Web Application Layer
          ↓
  Engine State Updates
          ↓
Engine Render / Simulation
          ↓
    Visual Output
```

User interaction follows the same pipeline, entering through the web layer and propagating downward into the engine core.

This model ensures:

- Isolation of deterministic simulation logic.
- Platform independence of the engine.
- Replaceability of the mobile runtime without modifying the computational core.

Frontend Guide

The frontend lives in apps/web-frontend/src/. It's a Vue 2 app that talks to the Stellarium Web Engine and wraps it for mobile via Capacitor.

Engine Boundary

The engine is a WebAssembly module exposed as $stel on the Vue instance.

To read engine state:

```
this.$store.state.stel.constellations.lines_visible
```

To write engine state:

```
this.$stel.core.constellations.lines_visible = true
```

The Vuex store mirrors the engine state via replaceStelWebEngine. This keeps Vue's reactivity in sync with the engine. You read from the store, write to the engine directly.

Services

These are standalone modules in assets/ that handle platform features. Some examples are:

Service	File	What it does
Gyroscope	gyroscope-service.js	Device orientation → sky vie
Camera	camera-service.js	Camera feed for AR mode
Fullscreen	fullscreen-service.js	Android immersive mode
Search	search_engine.js	Sky object search with filte

Services are typically initialized in App.vue and accessed globally or through the Vuex store.

State Management

The Vuex store (store/index.js) holds two kinds of state:

Engine state — mirrored from the WASM engine:

```
state.stel = {
  constellations: { lines_visible: true, ... },
  atmosphere: { visible: true },
  ...
}
```

App state — things the engine doesn't know about:

```
state.gyroModeActive = false
state.arModeActive = false
state.sensorsEnabled = true
```

Mutations like setGyroModeActive update app state. The engine state is updated by writing to $stel.core.* and gets synced back via the watcher.

Component Patterns

Bottom Menu

The bottom menu (components/bottom-menu/) uses a pattern where:

- Tap toggles a feature on/off.
- Long-press opens a submenu with detailed settings.

Settings Panel

settings-panel.vue is a slide-out drawer with display options. Most settings are two-way bindings to engine properties:

```
constellationLines: {
  get() { return this.$stel.core.constellations.lines_visible },
  set(val) { this.$stel.core.constellations.lines_visible = val }
}
```

Includes: atmosphere, grids, constellation art, star rendering, light pollution (Bortle scale), and sky culture selection.

Search

The search panel (components/search-panel.vue) uses search_engine.js for querying sky objects.

How it works:

- User types a query.
- querySkySources() searches in priority order:
  - Constellations (highest)
  - Planets
  - Stars
  - DSOs (lowest)
- Results are deduplicated and ranked (exact → starts-with → contains).

Overlays

Components like GyroDirectionOverlay.vue and AR-camera-preview.vue are conditionally rendered based on store state. They sit on top of the canvas and don't interfere with engine rendering.

Upstream Relationship

Astara is a fork of Stellarium Web Engine.

The upstream project is a browser-based planetarium. We took it and turned it into a native Android app with sensor integration.

Why Fork?

The upstream engine is designed mainly as a desktop browser demo. It assumes constant network access, mouse-driven interaction, and a short-lived session model.

We wanted something different:

- A real native mobile app, not just a PWA.
- Gyroscope-based “point your phone at the sky” interaction.
- AR camera overlay.
- Touch-first UI behavior.
- Full offline usage with bundled data.
- A build pipeline that produces a production-grade Android application.

What Stays Upstream

The astronomical engine itself is treated as upstream-owned code. By default we do not change:

- Astronomical simulation logic.
- Rendering and projection system.
- Object model and module structure.
- Asset loading model and ephemeris algorithms.

What We Changed

Most of the work in Astara happens outside the core engine, but some practical changes exist across all layers in order to make the engine usable as a real mobile application.

This includes:

- A mobile-oriented application shell built with Capacitor.
- A frontend that treats touch input and device motion as first-class inputs.
- Sensor and camera integration paths that do not exist in the upstream web app.
- Full offline operation with sky data bundled locally.
- Support for arbitrary observer locations without relying on external APIs.
- Much larger bundled datasets, including additional deep sky objects and landscapes.
- Build tooling that produces a standalone Android app rather than a demo site.