ASTRA Backend · API Contract

Advanced System for Testbed Recording and Analysis · NASA JPL Capstone

v0.2 | Feb 2025

1. Backend Scope

Responsibility	Details
Receive & store notes from AI/Data module	POST /api/sessions/{sid}/notes → called by AI team after Whisper + LLM
Receive & store telemetry	POST /telemetry or /telemetry/batch → called by telemetry source
Provide telemetry query API for AI module	GET /telemetry/latest?channel=X → AI looks up live value to attach to note
Provide notes read / edit API for frontend	GET, PUT, DELETE /notes → operator reads & corrects AI-generated notes
WebSocket real-time push	WS /ws/sessions/{sid} → broadcasts note.created, note.updated, stt events
Session management	Create, list, get, update/end sessions
STT task lifecycle (v0.2)	POST/PUT /stt/tasks → AI team registers audio chunk, reports transcript back

Key Principle: *Backend = storage + query + real-time delivery. AI/Data module is responsible for audio processing and note generation.*

Caller Legend

Frontend	AI / Data Module	Telemetry Source	Frontend / AI

2. Sessions /api/sessions

Method	Endpoint	Description	Called By
POST	`/api/sessions`	Create new test session	Frontend / System
GET	`/api/sessions`	List all sessions (newest first)	Frontend
GET	`/api/sessions/{sid}`	Get specific session	Frontend
PATCH	`/api/sessions/{sid}`	Update metadata / set status=ended	Frontend / System

3. Notes /api/sessions/{sid}/notes

Method	Endpoint	Description	Called By
POST	`/{sid}/notes`	Create note (AI-generated)	AI/Data Module

Method	Endpoint	Description	Called By
GET	`/{sid}/notes`	List notes — filters: speaker, type, from/to	Frontend
GET	`/{sid}/notes/export`	Export as Markdown or JSON	Frontend
GET	`/{sid}/notes/{id}`	Get specific note	Frontend
PUT	`/{sid}/notes/{id}`	Edit note (operator correction)	Frontend
DELETE	`/{sid}/notes/{id}`	Delete note	Frontend

4. Telemetry /api/sessions/{sid}/telemetry

Method	Endpoint	Description	Called By
POST	`/{sid}/telemetry`	Ingest single telemetry point	Telemetry Source
POST	`/{sid}/telemetry/batch`	Batch ingest multiple points	Telemetry Source
GET	`/{sid}/telemetry`	Query (filters: channel, from/to, limit)	Frontend / AI
GET	`/{sid}/telemetry/latest`	Get latest value — ?channel=X (used by AI to attach telemetry to notes)	AI/Data Module
GET	`/{sid}/telemetry/channels`	List all channel names in session	Frontend

5. STT Tasks /api/sessions/{sid}/stt/tasks (New in v0.2)

Method	Endpoint	Description	Called By
POST	`/{sid}/stt/tasks`	Register new audio chunk task (status: pending)	AI/Data Module
GET	`/{sid}/stt/tasks`	List all tasks — newest first	Frontend
GET	`/{sid}/stt/tasks/{tid}`	Get task status / transcript	Frontend
PUT	`/{sid}/stt/tasks/{tid}`	Update result: status=done + transcript, or status=failed + error	AI/Data Module

6. WebSocket /ws/sessions/{sid}

Type	Endpoint	Description	Called By
WS	`/ws/sessions/{sid}`	Subscribe to real-time session events	Frontend

WebSocket Events

Event	Trigger	data contains
connected	Client connects	{ "message": "Connected to session ..." }

Event	Trigger	data contains
note.created	AI posts a new note	Full note object
note.updated	Operator edits a note	Updated note object
note.deleted	Note deleted	{ "id": "note_xxx" }
stt.task.created	AI registers audio chunk	STT task object (status: pending)
stt.task.done	Whisper transcript ready	STT task object (status: done, transcript filled)
error.occurred	STT failed / system error	{ "message": "...", "source": "stt" }

7. Data Formats

7.1 Note Object (POST /notes by AI/Data Module)

```
{ "timestamp": "2025-01-26T10:32:15Z", // ISO 8601 UTC, required "speaker": "Engineer A", // optional — from diarization "content": "Motor current rising", // required "type": "observation", // observation | command | system "tags": ["motor", "current"], // optional, default [] "telemetry_snapshot": { // optional "battery_voltage": 32.5, "motor_current": 2.3 } }
```

7.2 Telemetry Object (POST /telemetry)

```
{ "timestamp": "2025-01-26T10:32:15Z", "channel": "battery_voltage", "value": 32.5, "unit": "V" // optional }
```

7.3 STT Task Object (PUT /stt/tasks/{tid} by AI/Data Module)

```
{ "status": "done", // done | failed "transcript": "Motor current rising to 2.3 amps, temperature stable.", "error": null // error message if status=failed }
```

7.4 WebSocket Message Format (Server → Client)

```
{ "event": "note.created", "session_id": "sess_a1b2c3d4", "data": { ... note / task / error object ... } }
```

8. STT Workflow (pause-based segmentation)

#	Actor	Action
1	Frontend	Detects pause in speech → marks end of audio chunk
2	AI/Data Module	Uploads chunk to local storage
3	AI/Data Module	POST /stt/tasks → register task (status: pending)
4	Backend	Stores task, broadcasts stt.task.created via WebSocket
5	AI/Data Module	Sends chunk to Whisper for transcription
6	AI/Data Module	PUT /stt/tasks/{tid} → status=done, transcript filled
7	Backend	Broadcasts stt.task.done via WebSocket
8	AI/Data Module	POST /notes → creates structured note with content from transcript
9	Backend	Stores note, broadcasts note.created via WebSocket
10	Frontend	Receives note.created event, renders note in UI