=== INSERTED SECTION: Full Backend Component & API Inventory ===

## 🧠 EXPANDED CONSOLIDATED VIEW: FULL BACKEND COMPONENTS

**🔹 \*\*File Handler\*\***

• Description: Manages file uploads and temp storage for embeddings and goal extraction

• Usage: Used by upload endpoints and embedding engine

**🔹 \*\*Embedder\*\***

• Description: Embeds document content into vector database (e.g., FAISS or Pinecone)

• Usage: Called during file upload or goal reprocessing

**🔹 \*\*Chat Engine\*\***

• Description: Routes chat queries, invokes LLM, formats messages

• Usage: Used throughout all chat sessions

**🔹 \*\*Logger\*\***

• Description: Logs structured backend events and issues

• Usage: Used internally across modules

**🔹 \*\*Prompt Compiler\*\***

• Builds dynamic tutor prompt using templates, goals, snapshots, and placeholders.

• Usage: Called during chat session start or on-demand regen

**🔹 \*\*Session Starter Engine\*\***

• Description: Accepts goal ID, starts new tutoring session with state

• Usage: Used via /session/start

**🔹 \*\*Goal Plan Tracker\*\***

• Description: Stores canonical structure of milestones per goal

• Usage: Used during plan parsing and session prep

**🔹 \*\*Progress Summary Engine\*\***

• Description: Extracts user learning progress and updates trackers

• Usage: Called periodically or after session

**🔹 \*\*Tutoring Cycle Tracker\*\***

• Central orchestrator — routes all tutoring traffic across module

**🔹 \*\*Dynamic Placeholder Engine\*\***

• Description: Fills slots like {{user}}, {{topic}} into compiled prompts

• Usage: Called inside Prompt Compiler

**🔹 \*\*Report Generator\*\***

• Description: Creates end-of-session report from tracker snapshots

• Usage: Called at session close

**🔹 models.py**

Defines schemas for all backend database tables (sync manually with DB schema)

**🔹 Handler (Optional)**

Executes isolated logic units like goal fetching or chat prep. May replace/augment `chat\_engine.py`.

**🔹Sessine.py**

Starts and manages chat\_sessions

## 📡 EXPANDED API INTERFACE OVERVIEW

🔸 `/session/start` – Triggers new session, compiles prompt, initializes state

🔸 `/prompt/compiled/{session\_id}` – Returns prompt for given session

🔸 `/goal-plan/{goal\_id}` – Returns parsed goal milestones for tutoring

🔸 `/progress/summarize` – Analyzes chat logs and updates trackers

🔸 `/tracker/{goal\_id}` – Returns tutoring life-cycle tracker state

🔸 `/report/generate` – Builds PDF or JSON report from session logs

🔸 `/upload` – Receives uploaded file, triggers embedding & planning

🔸 `/status/{job\_id}` – Returns job lifecycle status for frontend tracking

🔸 `/embedding/reset/{kb\_id}` – Forces re-embedding and plan re-generation

**📘 Metatron Tutor Backend System Design – v1.2**

This document defines the backend architecture, data flow, and orchestration logic for the **Metatron Tutor Engine**, with specific focus on modularity, progress tracking, and prompt generation.

**🧠 Consolidated View: Core Backend Components**

| **🔧 Component** | **🧠 Function** | **🔌 API Role** | **🗃️ Database Tables** | **💬 Used During** |
| --- | --- | --- | --- | --- |
| **Prompt Compiler** | Builds prompt from all sources (master, goal, insights, trackers) | Generates final session prompt | None (reads from all trackers) | Session Start |
| **Session Starter Engine** | Accepts goal ID, calls compiler, starts chat session | /session/start | chat sessions, users, goals | Session Start |
| **Goal Plan Tracker** | Static structure from uploaded content or manual config | /goal-plan/{goal\_id} | goal\_plans, milestones, topic\_tree | Prompt build, tutoring |
| **Progress Summary Engine** | Tracks actual learner performance from chat | /progress/summarize | chat\_logs, progress\_snapshots, insight\_tags | Mid-session & End |
| **Tutoring Cycle Tracker** | Tracks pedagogical stages (Engage → Repeat) | /tracker/{goal\_id} | cycle\_stages, stage\_repeats, topic\_flags | During session |
| **Goal Content & Plan Progress Tracker** | Tracks learner progress against content & milestones | (Updated via Progress Engine) | plan\_progress\_snapshots | Mid-session |
| **Dynamic Placeholder Engine** | Injects real-time context into prompts ({{user}}, {{stage}}) | (Internal only) | None | Prompt Build |
| **Report Generator** | Builds per-session reports (PDF, JSON) | /report/generate | reports, summaries, flags | Session End |
| **Tutor Brain Backend Engine** | Orchestrates all engines above, routes calls, logs activity | (Internal service layer) | None (routes and orchestrates) | Always |

**🔄 High-Level Backend Flow (with Updated Snapshots)**

[❶ Upload File / Define Goal]

→ Goal Plan created from file (topics, milestones, headings, summary)

→ Goal Plan DB snapshot stored in `goal\_plans`

→ Initializes empty trackers:

- Progress Snapshot Tracker → `plan\_progress\_snapshots`

- Tutoring Cycle Tracker → `cycle\_stages`

- Goal Plan Tracker → `goal\_plans`

→ Vector store and knowledge base created if embeddings used

[❷ Tutor Session Starts]

→ `/session/start` called with goal ID

→ SessionMeta entry created

→ Prompt Compiler pulls:

- Master prompt

- Goal-specific prompt

- Dynamic placeholders (`{{user\_name}}`, `{{topic}}`)

- Goal plan

- Most recent progress + cycle snapshot

[❸ Chat Begins]

→ Messages stored in `chat\_logs`

→ Every N messages or idle timeout:

→ Progress Summary Engine:

- Analyzes content understanding

- Evaluates lifecycle position

- Updates progress snapshots

- Triggers insight generation

[❹ Tutor Adapts Based on Tracker Data]

→ Struggles → Repeat prior stage

→ Mastered → Skip ahead, quiz

→ Uncertain → Injects reflection questions

[❺ Session Ends]

→ Final progress + cycle + plan snapshots saved

→ Session summary report generated (`reports`)

→ Tracked for future recall

[❻ Future Chat Sessions]

→ Past snapshot + plan data injected into prompt

→ Tutor resumes seamlessly

**📡 API Interface Plan**

| **API Route** | **Purpose** |
| --- | --- |
| POST /session/start | Initiate session, compile prompt, create session record |
| GET /prompt/compiled/{session\_id} | Return generated prompt |
| GET /goal-plan/{goal\_id} | Return structured milestones, plan |
| PATCH /goal-plan/update | Update or override plan structure |
| POST /progress/summarize | Analyze last N messages, return tags, update tracker |
| GET /progress/{session\_id} | Fetch latest progress snapshot |
| GET /tracker/{goal\_id} | Fetch current cycle or lifecycle stage |
| POST /report/generate | Create downloadable report from session state |

**🗂️ CENTRAL FILES TO BE DESIGNED**

| **File** | **Purpose** |
| --- | --- |
| main.py | Entry point for FastAPI app and middleware |
| chat\_engine.py | Orchestrates tutoring chat flow, legacy file |
| prompt\_engine/compiler.py | Compiles prompt using master + goal + snapshot |
| models.py | Holds schema definitions, must sync with DB |
| api/session.py | Handles chat session start logic + chat\_session\_meta |
| api/goal\_plan.py | Serves/updates tutoring plan structure |
| api/progress.py | Summarizes learning progress + snapshot |
| file\_handler.py | Manages file upload, format detection, text extraction |
| embedder.py | Embeds parsed content into vector DB |
| logger.py | Central logging utility (non-schema) |
| utils/placeholder.py | Fills {{user}}, {{topic}}, etc., in prompt templates |
| handler.py | **[NEW]** Optional orchestrator wrapper (for fallback or inline logic) |

**🧠 PHASE 1: SYSTEM ANALYSIS & MAPPING**

Before writing code, we define all "Process → Data → Table → API" relationships:

| **Spec Piece** | **Description** |
| --- | --- |
| **Purpose** | What does this module do? Who needs it? |
| **Process Flow** | What steps does it perform and when? |
| **Input Requirements** | What data does it need and from where? |
| **Output/Actions** | What does it return, store, or call next? |
| **DB Fields Required** | Table name and key fields |
| **APIs Needed** | GET, POST, PATCH routes with expected params |

**✅ Next Design Steps (Before Coding)**

**🔲 Step 1: Finalize Field Schema Maps**

Each tracker must define:

1. Field names
2. Data types
3. Unique keys
4. Foreign keys (e.g., session\_id, goal\_id)
5. Validation rules (nullable, enum, etc.)

✅ **Ready to begin now**

**🔲 Step 2: Build System Flow Diagrams**

Recommended diagrams:

1. One master “Tutor Session Lifecycle”
2. One per engine:
   1. Prompt Compiler
   2. Tracker Coordination
   3. Progress Analyzer

**💡 Design Guidelines**

1. **Orchestration via Tutor Brain Backend Engine**: All modules interact via this layer
2. **Snapshot-Based Tracking**: All changes are saved as time-based snapshots for auditing and re-entry
3. **Tracker Extensibility**: Trackers support dynamic lifecycle stages to allow future modifications
4. **Placeholder Personalization**: Prompts adapt to context and learner dynamically
5. **Decoupled APIs**: APIs expose only what's needed to the frontend — never raw engine access
6. **Embeddings Optional**: Knowledge Base is modular and can swap between OpenAI and OSS models

Let me know if you’d like this content converted back into a Word or PDF file now that the full body has been reconstructed.

Would you like to proceed with **Step 1: Tracker Schema Field Maps** now?

# Metatron Backend Design – Current State & Alignment (v1.0)

## 9. Backend Source File Architecture

We use a modular FastAPI-based Python backend. Each engine and route is split for clarity, testing, and growth.

|  |  |
| --- | --- |
| File Path | Purpose |
| main.py | Initializes FastAPI app, sets up CORS and base routes |
| chat\_engine.py | Orchestrates tutoring chat sessions using the compiled prompt |
| embedder.py | Converts uploaded files into vector DB entries |
| file\_handler.py | Manages uploads, extraction, PDF/text/DocX parsing |
| logger.py | Unified logging utility across modules |
| models.py (to create) | Central schema definitions (ORM or manual Pydantic) |
| api/session.py (to create) | Starts chat sessions, stores session metadata |
| api/goal\_plan.py (to create) | Serves and updates goal plan structure |
| api/progress.py (to create) | Summarizes chat logs, triggers insights |
| prompt\_engine/compiler.py | Compiles master + session prompt using metadata |
| utils/placeholder.py | Injects dynamic context into prompt templates (e.g., {{user\_name}}) |

## 10. Process → Data → Table → API Mapping

Let’s map one full flow as a template. Each module below will follow this exact format.

### ✅ Session Starter

|  |  |
| --- | --- |
| Spec Element | Detail |
| Purpose | Launch tutoring session from defined goal |
| Input Requirements | `user\_id`, `goal\_id`, context state |
| Process Flow | 1. Call compiler → 2. Store session\_meta → 3. Return compiled prompt |
| Output / Actions | New session row, compiled prompt returned |
| DB Tables Required | `chat sessions`, `session\_meta`, `goal\_plans`, `goal\_progress\_snapshots` |
| API Endpoints | `POST /session/start` → Payload: `{ user\_id, goal\_id }` |

## 11. Backend Design Sprint Tracker

|  |  |  |
| --- | --- | --- |
| Sprint Phase | Components Defined | Status |
| 🟦 Sprint 1: Session Engine | Prompt Compiler, Session Starter | ✅ Finalized |
| 🟦 Sprint 2: Goal Plan Tracker | DB schema, logic, and update API | ✅ Finalized |
| 🟨 Sprint 3: Progress Summary Engine | Snapshot schema, analyzer, insight tags | ⚙️ In Progress |
| ⬜ Sprint 4: Tutor Lifecycle Tracker | Life stage state machine, per-concept | 🔜 Pending |
| ⬜ Sprint 5: Reporting Engine | PDF + JSON Report Builder | 🔜 Planned |
| ⬜ Sprint 6: Central Logging Engine | SystemPromptLog, TutorBehaviorLog | 🔜 Planned |

## 12. Summary: Design Alignment Status

|  |  |  |
| --- | --- | --- |
| Topic | Alignment Across Versions? | Resolution Plan |
| KB Meta + Goals Table | ✅ Already implemented | Kept as-is, used as `goal\_id` FK in new tables |
| Tutor Lifecycle State Tracking | ⚠️ In v1.2, not yet schema’d | Separate table designed to allow dynamic stage storage |
| Embedded Plan From Uploads | ✅ Present in existing flow | Now stored in `goal\_plans` |
| Placeholder System | ⚠️ Not fully defined | Defined as `utils/placeholder.py`, injects during compile |
| Unified Snapshots | ⚠️ Needed standardization | Now done via `\*\_snapshots` across all trackers |

**📦 Prompt Compiler Module – Full Backend Design Spec (v1.2 with Compiled Prompts Integration)**

**✅ PURPOSE**

**What does this module do? Who needs it?**

The **Prompt Compiler** is responsible for building the dynamic tutoring prompt used by the language model during each chat session. It consolidates all prompt components — the master template, goal-specific logic, learner state, and dynamic variables — into one compiled string.

Used by:

* 🧠 **Chat Session Starter Engine** — triggers initial prompt build
* 🎯 **Progress Summary Engine** — regenerates prompt on milestone/cycle updates
* 🧩 **Tutor Brain Engine** — central tutoring orchestrator
* 🛠️ **Admins/QA/Developers** — for reviewing prompt versioning, inputs used, debug tracing
* 📑 **Audit Systems** — to compare historical prompt construction across sessions

This module **writes outputs to two places**:

1. chat\_session\_meta — for operational tutoring use
2. compiled\_prompts — for **auditing, debugging, admin panel tools**, and regen workflows

**🔁 PROCESS FLOW**

plaintext

CopyEdit

[User selects goal and starts chat session]

↓

[Chat Session Starter triggers Prompt Compiler]

↓

Prompt Compiler Execution Steps:

1. Load base Master Prompt Template

- From `master\_prompts` using `master\_prompt\_id`

2. Check for Goal-specific Prompt Snippet

- From `goal\_prompts` using `goal\_id`

- If found, append or replace based on `override\_base`

3. Retrieve Parsed Goal Plan

- From `goal\_plans` (topics, milestones, tier\_structure)

4. Fetch Last Progress Snapshot

- From `progress\_snapshots` (user+goal state)

5. Inject Dynamic Context Placeholders

- Using values from `user\_meta`, `goal\_meta`, and live state

- Replace: {{user}}, {{goal}}, {{topic}}, {{cycle\_stage}}

6. Validate and Compile Final Prompt

- If missing data or invalid structure → return error trace

7. Store Result:

✅ `chat\_session\_meta` → compiled\_prompt, prompt\_version

✅ `compiled\_prompts` → full audit trace (admin/debug)

8. Return Compiled Prompt to Session Starter

**📥 INPUT REQUIREMENTS**

| **Source Table** | **Fields / Format** | **Notes** |
| --- | --- | --- |
| master\_prompts | id, template, version | Base system prompt, versioned, static content |
| goal\_prompts | id, goal\_id, prompt\_snippet, override\_base, version | Goal override/appended logic |
| goal\_plans | goal\_plan\_id, goal\_id, plan\_version, topics, milestones, summary, tier\_structure | Multi-level structure for session flow |
| progress\_snapshots | snapshot\_id, goal\_id, user\_id, timestamp, content\_score, cycle\_stage, notes | Snapshot of progress and state |
| user\_meta | user\_id, first\_name, language\_pref, last\_topic\_viewed | Personal context for dynamic prompt slots |
| chat\_sessions | chat\_session\_id, goal\_id, user\_id | Current tutoring session metadata |

**📤 OUTPUT / ACTIONS**

| **Target Table** | **Fields Written** | **Purpose** |
| --- | --- | --- |
| chat\_session\_meta | compiled\_prompt, compiled\_at, prompt\_version | Final compiled string used by the LLM |
| compiled\_prompts | chat\_session\_id, goal\_id, prompt\_text, inputs\_used, engine\_name, template\_id, debug\_trace | ✨ Full **audit trace**, versioning, admin review, dev inspection |

compiled\_prompts gives complete transparency of:

* Which data sources were used
* What content went into the final output
* How prompts evolved between tutoring cycles

**🗃️ DATABASE FIELD SCHEMA (ALL TABLES)**

**✅ master\_prompts**

| **Field** | **Type** | **Notes** |
| --- | --- | --- |
| master\_prompts**\_**id | UUID | PK |
| template | TEXT | Static base prompt |
| version | INT | Used in compiled\_prompts log |
| description | TEXT | Optional tooltip/admin view |

**✅ goal\_prompts**

| **Field** | **Type** | **Notes** |
| --- | --- | --- |
| Goal\_prompt\_ id | UUID | PK |
| goal\_id | UUID | FK |
| prompt\_snippet | TEXT | Appended or replaces master prompt template |
| override\_base | BOOL | Controls whether this replaces the base |
| version | INT | Used in compiler trace |

**✅ goal\_plans**

| **Field** | **Type** | **Notes** |
| --- | --- | --- |
| goal\_plan\_id | UUID | FK in compiled\_prompts inputs\_used |
| plan\_version | INT | Trace changes in plan updates |
| topics | JSON | Tier-1 headings |
| milestones | JSON | Ordered goals mapped to topics |
| tier\_structure | JSON | Deep hierarchy: H1 → H2 → H3 |
| summary | TEXT | High-level tutor-facing summary |

**✅ progress\_snapshots**

| **Field** | **Type** | **Notes** |
| --- | --- | --- |
| Snapshot\_id | UUID | FK in compiled\_prompts.inputs\_used |
| content\_score | INT/JSON | Summary score of coverage |
| cycle\_stage | TEXT | “Present”, “Repeat”, etc. |
| notes | TEXT | Human or AI-generated commentary |

**✅ user\_meta**

| **Field** | **Type** | **Notes** |
| --- | --- | --- |
| user\_id | UUID | Primary key |
| first\_name | TEXT | Injected into prompt |
| language\_pref | TEXT | Optional, for prompt tone/styling |
| last\_topic\_viewed | TEXT | Injected as starting point context |

**✅ chat\_session\_meta**

| **Field** | **Type** | **Notes** |
| --- | --- | --- |
| chat\_session\_id | UUID | FK to chat\_sessions |
| compiled\_prompt | TEXT | Final prompt used by tutor |
| compiled\_at | DATETIME | Generation timestamp |
| prompt\_version | INT | Matches master/goal prompt version |

**✅ compiled\_prompts (✨ new)**

| **Field** | **Type** | **Notes** |
| --- | --- | --- |
| compiled\_prompt\_id | UUID | Primary key |
| chat\_session\_id | UUID | FK — trace which session this belongs to |
| goal\_id | UUID | FK — for group queries |
| prompt\_text | TEXT | Full compiled output |
| inputs\_used | JSON | { master\_prompt\_id, goal\_prompt\_id, goal\_plan\_id, snapshot\_id } |
| engine\_name | TEXT | Compiler version or model (e.g., prompt\_compiler:v1) |
| template\_id | UUID | Links to master prompt |
| debug\_trace | JSON | Optional info: timestamps, merge steps, warnings |

**🌐 REQUIRED API ROUTES**

| **Route** | **Method** | **Purpose** | **Payload / Params** |
| --- | --- | --- | --- |
| /prompt/compile | POST | Trigger compilation | { user\_id, goal\_id, chat\_session\_id } |
| /prompt/compiled/{session\_id} | GET | Get prompt for chat session | chat\_session\_id as URL param |
| /prompt/debug | POST | Get compiled prompt + trace | { session\_id, trace: true } |
| /prompt/logs/{goal\_id} *(opt)* | GET | Audit history of all compiled prompts | goal\_id, pagination params, filter by user |

**🔍 CLARIFICATION HIGHLIGHTS**

1. ✅ **Prompt stored in two places**
   * **chat\_session\_meta** → used for tutoring
   * **compiled\_prompts** → used for debugging, audits, dev tooling
2. ✅ **Version Control**
   * Master + Goal Prompts versioned explicitly
   * Prompt Compiler logs what version was used
   * Enables future regen + report consistency
3. ✅ **Compiler accesses DB directly**  
   Internal module → uses ORM or direct SQL  
   External clients (frontend/tools) → use REST APIs
4. ✅ **"Optional" APIs**  
   Used only by admin tools, not the tutoring workflow  
   E.g., debug or audit log routes
5. ✅ **All “session” terms = renamed to chat\_session**  
   Field: chat\_session\_id  
   Table: chat\_sessions  
   Applies everywhere from schema to APIs

**🧠 SYSTEM FLOW DIAGRAM (Simplified Text)**

plaintext

CopyEdit

Frontend → POST /chat-session/start

↓

Chat Session Starter → calls PromptCompiler.compile()

↓

Reads:

- master\_prompts

- goal\_prompts

- goal\_plans

- progress\_snapshots

- user\_meta

↓

Writes:

- chat\_session\_meta.compiled\_prompt

- compiled\_prompts (admin/audit)

↓

Returns:

→ Compiled prompt to Tutor Engine