=== 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

**🔹 \*\*Metatron\*\***

• The Metatron.py module is the authoritative orchestrator that initiates chat sessions, routes calls to sub-engines (compiler, tracker, progress), and integrates all tutoring state logic. • 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\*\* Session.py**

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

• Usage: Used via /session/start

**🔹 \*\*Goal Plan Tracker\*\* ??.py**

• Description: Stores canonical structure of milestones per goal

• Usage: Used during plan parsing and session prep

**🔹 \*\*Progress Summary Engine\*\* ??.py**

• Description: Extracts user learning progress and updates trackers

• Usage: Called periodically or after session

**🔹 \*\*Tutoring Cycle Tracker\*\* ??.py**

• Central orchestrator — routes all tutoring traffic across module

**🔹 \*\*Dynamic Placeholder Engine\*\* ??.py**

• 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 (DB authoritive data**

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

**🔹Session.py**

Starts and manages chat\_sessions

🔹 **Tracker Manager**  
• Aggregates goal plan, content, and lifecycle tracker states into unified snapshots  
• Coordinates snapshot logic and serves high-level progress endpoints

## 📡 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**

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 |
| **Content Progress 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 | Mid-session & End |
| **Tracker Manager** | Tracks learner progress against content, milestones | Not sure?? | plan\_progress\_snapshots | Mid-session & End |
| **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 (Metatron)** | 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 |
| GET /tracker/snapshot/{goal\_id} | Get full snapshot from all 3 trackers |
| GET /tracker/state/{goal\_id} | Get current status of lifecycle + content tracker |

**🗂️ CENTRAL FILES TO BE DESIGNED**

| **File** | **Purpose** |
| --- | --- |
| main.py | Entry point for FastAPI app and middleware |
| Metatron.py | High-level orchestrator and tutoring brain (replaces chat\_engine.py) |
| prompt\_engine/compiler.py | Compiles prompt using master + goal + snapshot |
| models.py | Holds schema definitions, must sync with DB |
| session.py | Handles chat session start logic + chat\_session\_meta |
| goal\_plan.py | Serves/updates tutoring plan structure |
| 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) |
| placeholder.py | Fills {{user}}, {{topic}}, etc., in prompt templates |
|  |  |

**🧠 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.

**🧠 Tracker Framework Design – Detailed Spec (v1.5+)**

This section defines the modular, extensible tracking system powering tutoring intelligence in the Metatron Tutor Engine. Each tracker monitors a specific aspect of learner progress. Together, they provide real-time insight and enable personalized learning at scale.

**🎯 Tracker Roles and Responsibilities**

| **Tracker** | **What It Tracks** | **Persistence** | **API Exposure** | **Description** |
| --- | --- | --- | --- | --- |
| Goal Plan Tracker | Canonical plan of topics, milestones, tier structures | ✅ Yes | ✅ Yes | Static learning map derived from uploaded goals or manual plan editor |
| Content Progress Tracker | Topic-by-topic learner interaction + comprehension | ✅ Yes | ✅ Yes | Tracks user understanding of each topic, flags for mastery/difficulty |
| Lifecycle Tracker | Tutoring lifecycle stage (intro, reinforce, assess) | ✅ Yes | ✅ Yes | Drives tutor behavior and interaction type during session phases |
| **Tracker Manager** | Aggregates current state across trackers (not tracking itself) | ✅ Yes (via progress\_snapshots) | ✅ Yes (aggregated endpoints) | Combines tracker outputs into unified snapshot per chat session |

**🧩 Tracker Manager Responsibilities**

TrackerManager is a logic-only coordination module (no tracking logic of its own). It provides:

* Aggregated state for each user+goal session
* Snapshot generation logic (e.g., when to take a snapshot)
* Unified API layer for accessing tracker states

It **does persist data** in the progress\_snapshots table, acting as the system-of-record for:

* Last known learner understanding
* Most recent lifecycle stage
* Notes, metadata, tutor insights
* Future support for quiz scores and engagement metrics

**📊 Unified Progress Snapshot Schema**

Stored in the progress\_snapshots table and modeled in models.py as:

python

CopyEdit

class ProgressSnapshot(BaseModel):

snapshot\_id: UUID

goal\_id: UUID

user\_id: UUID

timestamp: datetime

content\_score: Optional[float] # % or score of content mastery (aggregated)

cycle\_stage: Optional[str] # Tutoring lifecycle phase

topic\_status: Optional[dict] # { topic\_id: "mastered"/"stuck" } (flattened)

notes: Optional[str] # Tutor or system-generated commentary

quiz\_summary: Optional[str] # Future: Summary of quiz results

engagement\_score: Optional[float] # Future: Derived from retries, time-on-page, etc.

✅ This table is **persisted**, and it serves as the **official time-series log** of a learner’s evolving journey.

**🔄 Data Relationships: Tracker → Snapshot**

| **Field** | **Source Tracker** | **Purpose** |
| --- | --- | --- |
| content\_score | Content Progress Tracker | Aggregated understanding score (0–100%) |
| cycle\_stage | Lifecycle Tracker | e.g., INTRO → PRACTICE → ASSESS |
| topic\_status[] | Goal Plan + Content Tracker | Flattened flags from per-topic progress |
| quiz\_summary | Content Tracker | Summary of scores, pass/fail, retry counts |
| engagement\_score | Content Tracker or Metatron | Time on topic, retries, session behavior |
| notes | Any module (via system) | Optional observations or insights |

**🧪 Future-Ready: Assessments, Engagement, and Intelligence**

From Day 1, we treat the progress\_snapshots as a platform for intelligent future features:

| **Future Field** | **Tracker To Extend** | **Description** |
| --- | --- | --- |
| quiz\_attempts[] | Content Progress Tracker | JSON history of quiz scores, timestamps |
| last\_assessment\_stage | Lifecycle Tracker | Track most recent recap/review cycle |
| engagement\_score | TrackerManager or Tutor | Auto-calculated from chat pacing, topic revisit |

All of these can be added to progress\_snapshots without needing new tables, maintaining backward compatibility.

**📡 Tracker Manager API – (Future-Facing Spec)**

| **Route** | **Method** | **Purpose** |
| --- | --- | --- |
| /tracker/snapshot/{goal\_id} | GET | Returns latest aggregated snapshot for user+goal |
| /tracker/state/{goal\_id} | GET | Returns real-time view across all 3 trackers |
| /progress/{goal\_id} | GET | Returns only progress\_snapshots for reporting/analysis |

**🛠 Developer Implementation Notes**

* Each tracker is a dedicated Python module (goal\_tracker.py, content\_tracker.py, etc.)
* TrackerManager will live in tracker\_manager.py, used only by internal modules
* progress\_snapshots table becomes the **single source of truth** for tutor regen, reporting, audits
* Snapshots should be taken:
  + At session start
  + Every N messages
  + At session end
  + On demand by admin/system

**📦 Component Design: Goal Plan Tracker - Phase One**

**(for initial implementation)**

**🧭 Purpose**

The **Goal Plan Tracker** is responsible for storing and representing the **static educational roadmap** associated with a specific user’s coaching goal. It defines:

* what the user should learn,
* in what sequence,
* organized by difficulty (tier structure),
* optionally aligned to milestones and themes.

It is the **foundational source of truth** for the tutoring system. It does not track live behavior but represents the **“what should be learned”** blueprint.

**Primary responsibilities:**

* Structuring content in tutoring sessions
* Generating milestone-based insights
* Driving scaffolding logic in tutor behavior (e.g., tiered support)
* Rendering the learning UI (tier visualizations, milestones, topic trees)
* Injecting summarized structure into prompt compiler
* Feeding comparison logic for snapshots and progress tracking

Once initialized, this tracker becomes immutable per version and referenced **across all system components**.

**🔄 Process Flow**

| **Step** | **Description** |
| --- | --- |
| 1. User Defines Goal | A coaching goal is created via frontend (e.g. CoachingGoalSetup.jsx) |
| 2. System Suggests Tiered Plan | Pre-built templates or AI generation logic populates tier/topic layout |
| 3. User Reviews/Edits | User refines topics, tiers, milestones; adds summaries |
| 4. Plan is Finalized and Saved | Stored in DB with a version number; previous versions retained |
| 5. Plan Is Referenced | All tutoring components reference it during learning, prompting, and reporting |

**📥 Input Requirements**

| **Source** | **Data** |
| --- | --- |
| User Input | goal\_id, learning intent, topic hints, tier depth (optional) |
| System Defaults | Templates, AI-generated tiered curriculum |
| Upload Data | PDF / content processed via embedder.py may influence structure |
| Coaching Goal Form | Pulls preferences like age group, timeframe, pacing target |

**📤 Output / Dependencies**

| **Consumer** | **Use** |
| --- | --- |
| Metatron.py | Drives sequencing, navigation, and flow decisions |
| Prompt Compiler | Injects roadmap context into the system prompt |
| Snapshot Generator | Used as the “ideal” comparison for what has been completed |
| Tracker Manager | Aggregates goal plan with lifecycle and progress tracker for insight |
| Frontend Dashboards | Renders tier structures and milestone progress views |
| CoachingGoalSetup.jsx | The plan builder is invoked directly after coaching goal definition |

**🧱 Data Schema (models.py)**

python

CopyEdit

class GoalPlan(BaseModel):

goal\_plan\_id: UUID # Unique version ID

goal\_id: UUID # FK to user-defined goal

plan\_version: int # Each saved state increments this

topics: Dict # Hierarchical topic tree

milestones: List[str] # Optional milestone headings

tier\_structure: Dict # { "Beginner": [...], "Advanced": [...] }

summary: Optional[str] = None # Tutor/system description

**Database Table: goal\_plans**

| **Field** | **Type** | **Notes** |
| --- | --- | --- |
| goal\_plan\_id | UUID | PK, immutable version ID |
| goal\_id | UUID | FK to goals table |
| plan\_version | INT | Starts at 1 |
| topics | JSON | Topic hierarchy with metadata |
| milestones | JSON | Optional |
| tier\_structure | JSON | Tiers mapped to topic IDs |
| summary | TEXT | Editable tutor/system comment |

**🔌 API Interfaces**

| **Route** | **Method** | **Purpose** |
| --- | --- | --- |
| /goal/plan/{goal\_id} | GET | Retrieve full plan |
| /goal/plan/{goal\_id} | POST | Save initial version |
| /goal/plan/{goal\_id} | PATCH | Edit & create new version |
| /goal/plan/version/{version\_id} | GET | Fetch specific version |
| /goal/plan/validate | POST | Validate structure (optional) |

**🧠 Internal Logic & Constraints**

* **Immutability**: Once saved, plans are versioned. Old plans are never overwritten.
* **Tier-Topic Normalization**: All topic IDs used in tiers must exist in topics. Prevent orphaned topics or ghost tiers.
* **Topic ID Convention**: topic\_id must be unique per goal and must be referenced by the Progress Tracker and Snapshot Generator.
* **Deep/Narrow Tolerance**: Plan builder is flexible for both low-content (e.g., 1–5 topics) and high-volume content (hundreds of topics).

**🧮 Interactions with Other Components**

| **Component** | **Interaction** |
| --- | --- |
| Content Progress Tracker | Uses topic\_id to track progress vs structure |
| Lifecycle Tracker | Syncs with milestone themes for tutor behavior alignment |
| Tracker Manager | Aggregates plan with actual progress and lifecycle |
| CompiledPrompt | Summarizes current tier, milestone, and next goals |
| Metatron.py | Orchestrates session based on plan roadmap |
| CoachingGoalSetup.jsx | Captures frontend input for topics and structures |
| Dashboard.jsx | Renders plan preview and snapshot dashboards |

**🧰 Extensibility & Future-Proofing**

| **Feature** | **Notes** |
| --- | --- |
| ✅ Topic weights | e.g., Critical vs Optional — influences tutor emphasis |
| ✅ Estimated time per topic | Used to align pacing and set expectations |
| ✅ Prerequisites | Maps required completion dependencies (e.g. “Algebra I before Algebra II”) |
| ✅ Editable topic summaries | Per-topic AI summaries injected into prompts |
| ✅ Metadata fields | For curriculum tags, test-prep alignment, national standards |
| ✅ AI-aided plan generation | Based on uploaded curriculum content |
| 🔒 Lock plans | Optional for managed institution curriculums |
| ⏱️ Dynamic Milestone Rendering | Support both goal-based and time-based milestones (free-form goals supported) |

**✅ Design Review Checklist**

| **Element** | **Complete?** |
| --- | --- |
| Full versioning logic | ✅ |
| API interfaces mapped to planner UI | ✅ |
| Tier structure cross-checked with topic tree | ✅ |
| Topic ID format and uniqueness defined | ✅ |
| Tracker & Snapshot compatibility | ✅ |
| Deep/narrow plan support | ✅ |
| Prompt & UI integration | ✅ |
| Frontend hook from CoachingGoalSetup.jsx | ✅ |
| Dashboard rendering logic planned | ✅ |

**🔍 Deep Analysis: Goal Plan Tracker – Prompt-Integrated, AI-Generated – Phase Two (not for initial implementation, also includes other future feature options to be considered)**

**🧭 Refined Purpose & Role**

The **Goal Plan Tracker** is more than just a static representation of the user’s curriculum. It now serves as a dynamic interface between:

* **User-driven goal setup (frontend)**
* **AI-generated planning and content scaffolding (backend)**
* **Session prompt logic and downstream tutoring behavior**

Whereas previous versions focused on static plans, this version enables **real-time plan generation**, **tier synthesis**, and **progress-aware iteration**.

**📤 Entry Points & Source Data**

Based on CoachingGoalSetup.jsx, we now know exactly what user data we collect during coaching goal setup:

| **Captured Field** | **Purpose** |
| --- | --- |
| Goal title | Short, user-defined theme |
| Description (goal intent) | Used as input to plan-generation prompt |
| Age Group | Influences plan difficulty, structure |
| Timeframe type + value | Guides milestone spacing or urgency (e.g., fixed, flexible) |
| File Uploads | Provide raw source material for plan generation |

This means the system can infer a reasonable tiered learning structure **immediately after coaching goal setup**, using either:

* A **default templating engine** (fixed plan)
* A **prompted AI model** (dynamic plan generation)

**🧠 Prompting Strategy for Plan Generation**

**🧩 Key Question:**  
Should the backend call an AI model to generate a personalized Goal Plan using the user inputs (goal title, age group, timeframe, files)?

✅ Yes — and here's how.

**Option A: Predefined Template**

Use internal logic (based on age, file type, timeframe) to build a fixed tier structure. Great for MVP and consistent UX.

**Option B: Prompted Generation**

Inject this into a short structured system prompt:

text

CopyEdit

Create a tiered learning plan for the following user:

- Age Group: [value]

- Learning Goal: [intent/description]

- Timeframe: [X weeks/months]

- Topics Covered: [parsed file metadata or GPT summary of uploaded content]

Return a JSON object with:

- Tiers (Beginner, Intermediate, Advanced)

- Topics per tier

- Optional Milestones

- Optional estimated time per topic

This plan is parsed and saved to the goal\_plans table.

**🧱 Schema Alignment – Full**

**GoalPlan model**

python

CopyEdit

class GoalPlan(BaseModel):

goal\_plan\_id: UUID

goal\_id: UUID

plan\_version: int

topics: Dict # {"topic\_id": {"name": "Algebra Basics", "subtopics": [...]}}

milestones: List[str] # ["Week 1", "Week 2: Review", "Final Assessment"]

tier\_structure: Dict # {"Beginner": ["topic1", "topic2"], ...}

summary: Optional[str]

🎯 Consider: Allow model to inject a short summary at the end of plan generation — this helps with prompts later.

**🧠 Prompt Integration Points**

**Session Prompt Compilation includes:**

* Milestone and tier list from Goal Plan
* Topic IDs for guiding tutoring logic
* Summary of plan as part of “instructional scope”

🧠 As seen in your Prompt Architecture doc — this happens in Step 2 of prompt lifecycle:  
*“Milestone plan from Goal Plan Tracker”* → included in every session init

**🛠️ Future Enhancements to Planning Logic**

| **Feature** | **Benefit** |
| --- | --- |
| Prerequisite tagging | Allow tutor to dynamically skip ahead or require completion |
| Estimated time per topic | Helps time-bound users visualize effort |
| AI-summary per tier/topic | Adds descriptions to UI and tutor prompts |
| Editable JSON editor | Let advanced users tweak the full plan |
| Dynamic milestone injection | “You’re 60% through Week 2’s targets” — powered by snapshot logic |

# 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 |
| metatron.py | High level Orchestrates tutoring chat sessions, using all the resources |
| 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 | Central schema definitions (ORM or manual Pydantic) |
| session.py (to create) | Starts chat sessions, stores session metadata |
| goal\_plan.py (to create) | Serves and updates goal plan structure |
| progress.py (to create) | Summarizes chat logs, triggers insights |
| compiler.py | Compiles master + session prompt using metadata |
| placeholder.py | Injects dynamic context into prompt templates (e.g., {{user\_name}}) |

## 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 |

**🗂️ Change Log (Draft v1.5)**

**✅ Unified all orchestration under `Metatron.py`, removing `chat\_engine.py` and `handler.py`**

**✅ All session-related schema/fields renamed to `chat\_session`, e.g., `chat\_session\_id`, `chat\_sessions` table**

**✅ Made `compiled\_prompts` table standard (not optional), required for admin/audit/debug**

**✅ Removed unused modules: `TutorChatBrain`, `session\_controller.py`**

**✅ Confirmed `models.py` is source of truth for DB schema definitions**

**📦 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 (Metatron.py)** — central tutoring orchestrator
* 🛠️ **Admins/QA/Developers** — for reviewing prompt versioning, inputs used, debug tracing
* 📑 **Audit Systems** — to compare historical prompt construction across sessions (future release)

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