# **Comprehensive Technical Analysis of Setarcos Project Components for AI/LLM Engineering**

This Master Technical Blueprint provides a comprehensive, unified, and detailed specification for the Setarcos philosophy application. It consolidates all previous planning documents, technical specifications, feature definitions, and user feedback into a single, cohesive business document designed to guide the engineering team through the entire app design and implementation process.1 The Setarcos app aims to be a modern, engaging platform for philosophical exploration, featuring AI-driven dialogues, guided quests, personal journaling, interactive concept visualization, community interaction, and a sophisticated gamification system (Wisdom XP).1

This report provides a comprehensive, categorized inventory of all APIs, frontend components, backend components, and services within the Setarcos project. It is specifically tailored for an AI LLM Engineer, aiming to provide a precise, actionable reference for understanding system interactions, data flows, and critical integration points for AI/LLM applications. The structure prioritizes clarity and technical accuracy to facilitate efficient analysis and development.

### **1. API Endpoints Overview**

The Setarcos API adheres to RESTful design principles, emphasizing resource-orientation, versioning (/api/v1/), JSON format, consistent patterns, pagination, filtering, and rate limiting. All requests and responses utilize JSON format, and collection endpoints support pagination via limit and offset parameters.1 JWTs (JSON Web Tokens) are employed for authentication, with Supabase Auth managing user sessions and token issuance. Authorization is role-based, supporting Anonymous, Free, Premium, Pro, and Admin tiers.1

The application's monetization strategy is directly reflected in the API design through tiered access and rate limiting. Various endpoints, such as those for asking questions, creating journal entries, or expanding insights, have specific daily usage limits that vary based on the user's subscription tier (Free, Premium, Pro).1 Section 8.5 of the blueprint further details these tiered rate limits, including per-minute limits and endpoint-specific daily quotas for features like AI interactions, quest starts, and forum posts.1 This architectural choice means the backend must actively enforce these restrictions, responding with appropriate HTTP status codes (e.g., 429 Too Many Requests) when limits are exceeded.1 For an engineer working with large language models, this is a crucial consideration: any interaction with the API on behalf of a user must account for their subscription level and associated constraints. This influences how the LLM might suggest actions to the user, such as recommending an upgrade if a daily limit is reached. Furthermore, the AI Router's cost optimization strategies, as described in Section 9.4, are intrinsically linked to these tiers, prioritizing more cost-effective models for free users to manage operational expenses.1 This creates a direct feedback loop where the application's business model dictates aspects of AI model selection.

The enforcement of daily question limits at the API level is managed by a rate\_limit\_service which checks and increments counters for user actions.1 The frontend is expected to communicate these limits to the user, potentially through UI elements that indicate remaining questions or suggest upgrades. The exact mechanism for communicating these limits to the frontend for UI display would typically involve the API response including the current usage and remaining allowance, or a dedicated endpoint for querying user-specific limits.

#### **1.1. Authentication & User Management Endpoints**

These endpoints handle user registration, login, session management, and password recovery. The authentication flow involves the user registering or logging in via Supabase Auth, which issues a JWT. This JWT is then included in the Authorization header for subsequent API requests, with the backend validating the token and identifying the user.1

* POST /api/v1/auth/register: Registers a new user.1
* POST /api/v1/auth/login: Logs in an existing user.1
* POST /api/v1/auth/refresh: Refreshes an expired JWT.1
* POST /api/v1/auth/logout: Logs out a user by invalidating their refresh token.1
* POST /api/v1/auth/password/reset: Requests a password reset.1
* POST /api/v1/auth/password/change: Changes a user's password.1

#### **1.2. Ask Feature Endpoints**

These endpoints facilitate user interaction with the AI for philosophical questions, tone selection, and interaction management. The core interaction is the POST /api/v1/ai/ask endpoint, which is not merely a chat interface but a sophisticated pipeline.1 The response includes the AI's textual output, along with extracted

concepts and xp\_earned.1 The data flow diagram illustrates that this process involves the AI Router, external AI models, database storage, Celery tasks for XP and concept extraction, and PostHog analytics.1 This design decouples the core AI response generation from post-processing, which enhances perceived performance.

For an LLM engineer, understanding that the response field is only one component of the interaction is vital. The concepts field is critical for linking AI dialogues to the broader Explore knowledge base, enabling deeper learning pathways. The xp\_earned indicates the integration with the gamification system, motivating user engagement. Any work on improving AI responses must consider not only textual quality but also the accuracy of concept extraction and the potential impact on XP calculations. The asynchronous nature of concept extraction and XP calculation means that while the primary AI response is delivered quickly, the complete processing and data updates may have a slight delay. The typical latency for these background tasks is not explicitly stated in the document, but they are designed to run asynchronously to avoid blocking the main thread.1 The frontend would typically handle this eventual consistency by displaying the primary response immediately and then updating with concepts or XP notifications once the background tasks complete.

* POST /api/v1/ai/ask: Submits a question and retrieves an AI-generated response. The request includes the question, tone\_id, include\_concepts flag, and user\_context. The response contains interaction\_id, question, response text, tone details, extracted concepts, xp\_earned, and processing\_time\_ms.1
* GET /api/v1/ai/interactions: Retrieves a user's interaction history, supporting limit, offset, and saved\_only query parameters.1
* GET /api/v1/ask/tones: Retrieves a list of available philosophical tones.1
* POST /api/v1/ask/tone-preview: Obtains a sample response for a selected tone and question, allowing users to preview tone characteristics.1
* POST /api/v1/ask/interactions/{interaction\_id}/save: Saves an AI interaction to the user's journal.1
* POST /api/v1/ask/interactions/{interaction\_id}/expand: Expands an insight with deeper philosophical analysis, a feature available to Premium/Pro users.1
* GET /api/v1/ask/suggestions: Retrieves personalized question suggestions for the user.1

#### **1.3. Seek Clarity Feature Endpoints**

These endpoints support the five pathways for deeper philosophical engagement stemming from AI responses. Many of these endpoints, such as practical-examples and practical-rephrasing, return a task\_id and a status: "processing".1 This design pattern for AI-driven content generation explicitly indicates asynchronous processing, requiring a subsequent

GET call to retrieve the final result. This approach ensures that complex or potentially long-running AI tasks do not block the main API thread, thereby maintaining responsiveness for the user. The offloading of heavy computation to Celery workers is confirmed by the presence of seek\_clarity\_tasks.py.1

For an LLM engineer, this asynchronous pattern is a critical architectural consideration. The quality and latency of the underlying AI models directly influence the user's waiting time for these "clarity" features. Optimizing the AI models for speed and accuracy in these specific content generation tasks (e.g., generating real-world examples, rephrasing content for practical application) is paramount. The estimated\_time field in the asynchronous response implies a user expectation that needs to be managed, which can be achieved through careful model selection and prompt optimization. Specific performance SLAs for the completion of seek\_clarity\_tasks are not detailed in the provided documentation, but the design implies that these tasks are critical for user engagement and should be optimized for timely completion.

* POST /api/v1/seek-clarity/practical-examples: Requests AI-generated real-world examples based on an AI response and associated concepts.1
* GET /api/v1/seek-clarity/practical-examples/{task\_id}: Retrieves the generated practical examples once the asynchronous processing is complete.1
* POST /api/v1/seek-clarity/practical-rephrasing: Requests the AI to rephrase philosophical content for a specified real-world context.1
* POST /api/v1/seek-clarity/journal-challenge: Creates a structured challenge linked to the user's Journal based on an AI insight.1
* POST /api/v1/seek-clarity/wisdom-sharing: Generates curated sharing options with philosophical framing for insights.1
* POST /api/v1/seek-clarity/learning-path: Generates a concept-based learning path derived from an AI interaction.1

#### **1.4. Quest Feature Endpoints**

These endpoints manage philosophical quests, user progress, and skill tree visualization.

* GET /api/v1/quests: Retrieves a list of available quests, supporting filtering by page, limit, difficulty, concept\_id, status, and skill\_tree view.1
* GET /api/v1/quests/{id}: Retrieves detailed information about a specific quest, including its steps, concepts, and user's progress.1
* GET /api/v1/quests/skill-tree: Retrieves the full skill tree visualization data, showing relationships and unlock status of quests.1
* POST /api/v1/quests/{id}/start: Initiates a quest or resumes existing progress.1
* POST /api/v1/quests/{id}/steps/{step\_id}/complete: Marks a quest step as complete and submits the user's response for validation.1
* GET /api/v1/quests/recommended: Provides personalized quest recommendations based on user activity and interests.1
* POST /api/v1/quests/{id}/concepts/{concept\_id}/explore: Tracks when a user explores a concept linked from a quest and retrieves details about that concept.1
* GET /api/v1/quests/daily: Retrieves the user's daily quest.1

#### **1.5. Explore Feature Endpoints**

These endpoints facilitate interactive exploration of philosophical concepts and learning pathways. The Explore feature's API includes endpoints for create-path, connect-concepts, and create-nudge.1 These are explicitly for user creation and interaction with the knowledge graph, indicating a strategic move beyond passive content consumption to active user participation and knowledge co-creation. Users are empowered to define their own wisdom paths, connect ideas in their personal constellations, and share "nudges" (brief insights or questions).1 This also implies a need for robust moderation and quality control for user-generated content (UGC).

For an LLM engineer, this opens up significant opportunities and challenges. The LLM could assist users in creating paths (e.g., suggesting next concepts in a sequence), generating nudges (e.g., rephrasing a user's thought into a concise fragment), or even assisting in moderating user-contributed content. The LLM needs to be aware of this "user-generated" layer of the knowledge graph, not just the system-defined one, as it can influence content retrieval and recommendations. The "Organic Discovery System" 1 mentions thought fragments, revelation echoes, and question seeds, which are all forms of UGC, further emphasizing the importance of this interactive layer. The moderation workflow for user-generated wisdom paths and nudges is not explicitly detailed in the provided document, but it is a critical consideration for maintaining content quality and adherence to community guidelines.

* GET /api/v1/concepts: Retrieves a paginated list of concepts, supporting filtering by category, tradition, era, difficulty, search term, sort field, and order.1
* GET /api/v1/concepts/{concept\_id}: Retrieves detailed information about a specific philosophical concept.1
* GET /api/v1/concepts/{concept\_id}/relationships: Retrieves relationships for a specific concept, with options to filter by relationship\_type, min\_strength, and direction.1
* POST /api/v1/explore/history/start: Initiates a new concept exploration session.1
* PUT /api/v1/explore/history/{session\_id}/update: Updates an ongoing exploration session with a newly viewed concept.1
* POST /api/v1/explore/history/{session\_id}/end: Concludes an exploration session.1
* GET /api/v1/explore/visualize/map: Generates data for a concept map visualization, including nodes and links, with options for central\_concept\_id, depth, min\_strength, layout, and max\_nodes.1
* GET /api/v1/explore/visualize/constellation: Generates data for a constellation view visualization, clustering concepts thematically.1
* GET /api/v1/explore/recommendations: Retrieves personalized concept recommendations for the user.1
* GET /api/v1/explore/pathways: Retrieves available learning pathways, with filtering options for difficulty and tradition.1
* GET /api/v1/explore/pathways/{pathway\_id}: Retrieves detailed information about a specific learning pathway, including its concept sequence.1
* POST /api/v1/explore/pathways/{pathway\_id}/start: Initiates a learning pathway for the user.1
* POST /api/v1/explore/pathways/{pathway\_id}/concepts/{concept\_id}/complete: Marks a concept as completed within a learning pathway.1
* POST /api/v1/explore/create-path: Allows users to create and share their own wisdom paths.1
* POST /api/v1/explore/connect-concepts: Enables users to manually connect concepts within their personal constellation.1
* POST /api/v1/explore/create-nudge: Allows users to create brief user-generated content nudges (thought fragments, revelation echoes, question seeds) associated with concepts.1
* GET /api/v1/explore/nudges: Retrieves user-generated content nudges, filterable by concept\_id and nudge\_type.1
* GET /api/v1/concepts/schools: Retrieves a list of philosophical schools or categories for concepts.1

#### **1.6. Journal Feature Endpoints**

These endpoints manage personal philosophical reflections and activity history.

* GET /api/v1/journal: Retrieves a user's journal entries, supporting limit, offset, and concept\_id filtering.1
* POST /api/v1/journal: Creates a new journal entry.1
* GET /api/v1/journal/{id}: Retrieves details of a specific journal entry.1
* PUT /api/v1/journal/{id}: Updates an existing journal entry.1
* DELETE /api/v1/journal/{id}: Deletes a specific journal entry.1
* GET /api/v1/history: Retrieves a user's aggregated activity history across the entire application.1

#### **1.7. Forum Feature Endpoints**

These endpoints facilitate community discussions and content moderation.

* GET /api/v1/forum/threads: Retrieves forum threads, supporting limit, offset, concept\_id filtering, and sort options.1
* POST /api/v1/forum/threads: Creates a new discussion thread.1
* GET /api/v1/forum/threads/{id}: Retrieves details of a specific discussion thread, including its comments.1
* PUT /api/v1/forum/threads/{id}: Updates an existing discussion thread.1
* DELETE /api/v1/forum/threads/{id}: Deletes a specific discussion thread.1
* POST /api/v1/forum/threads/{id}/comments: Adds a comment to a discussion thread.1
* PUT /api/v1/forum/comments/{id}: Updates an existing comment.1
* DELETE /api/v1/forum/comments/{id}: Deletes a specific comment.1
* POST /api/v1/forum/vote: Allows users to upvote or downvote a thread or comment.1
* POST /api/v1/forum/report: Allows users to report inappropriate content.1

#### **1.8. Profile & Subscription Endpoints**

These endpoints manage user profiles, settings, and subscription details.

* GET /api/v1/users/me: Retrieves the current authenticated user's profile information.1
* PUT /api/v1/users/me: Updates the current user's profile information.1
* GET /api/v1/users/me/settings: Retrieves the current user's application settings and preferences.1
* PUT /api/v1/users/me/settings: Updates the current user's application settings and preferences.1
* GET /api/v1/users/me/badges: Retrieves a list of badges and achievements earned by the current user.1
* GET /api/v1/subscriptions: Retrieves the current user's subscription information.1
* POST /api/v1/subscriptions/create: Initiates the creation of a new subscription.1
* PUT /api/v1/subscriptions/update: Updates an existing subscription.1
* DELETE /api/v1/subscriptions/cancel: Cancels an active subscription.1

#### **1.9. Notification & Rating System Endpoints**

These endpoints manage user notifications and the contemplative orb rating system.

* GET /api/v1/notifications: Retrieves a user's notification history, with options to filter by limit, offset, status, and type.1
* PUT /api/v1/notifications/preferences: Updates a user's notification preferences, including push and email settings, and quiet hours.1
* POST /api/v1/notifications/read/{id}: Marks a specific notification as read.1
* POST /api/v1/notifications/read-all: Marks all notifications as read for the user.1
* POST /api/v1/ratings/submit: Submits a rating for an AI response, including interaction metrics such as hover duration and time to rate.1
* GET /api/v1/ratings/analytics: Retrieves aggregated rating insights, filterable by philosopher\_tone, model, and time\_range.1

#### **1.10. System Health & Metrics Endpoints**

These endpoints provide insights into the system's operational status and performance. The presence of /health, /health/db, /health/redis, /health/detailed, and /metrics endpoints 1 indicates a strong commitment to observability and proactive issue detection within the Setarcos system. These endpoints are not for direct user interaction but are crucial for system monitoring and operational visibility. The

detailed health check, in particular, offers a composite view of critical dependencies, including the database, Redis, and Celery workers.1 The

/metrics endpoint, which exposes Prometheus metrics, points to a robust monitoring stack capable of providing granular performance data.1

For an LLM engineer, understanding these endpoints is fundamental for diagnosing issues that might impact AI service performance or availability. If AI responses are slow or failing, checking these health endpoints can quickly identify underlying infrastructure problems such as database latency, Redis connectivity, or Celery worker issues. The metrics endpoint provides invaluable data on AI request counts and latencies, which is essential for performance tuning and A/B testing of different LLM models. This level of operational visibility enables data-driven optimization of LLM integration. Specific dashboards or alerts configured for AI-related metrics (e.g., AI model latency, error rates, token usage) would typically be accessible through the PostHog platform, as indicated by the DashboardService.1

* GET /api/v1/xp: Retrieves the current user's XP information, including total XP, current level, and progress to the next level.1
* GET /api/v1/xp/history: Retrieves a user's XP transaction history.1
* POST /api/v1/xp/check-in: Allows a user to perform a daily check-in, contributing to their streak and earning XP.1
* GET /health: Provides a basic health check, returning an "ok" status and timestamp.1
* GET /health/db: Performs a health check on the database connection.1
* GET /health/redis: Performs a health check on the Redis connection.1
* GET /health/detailed: Provides a detailed health check, reporting the status of multiple components including the database, Redis, and Celery.1
* GET /metrics: Exposes Prometheus metrics for application performance monitoring.1

**Table 1: Consolidated API Endpoints**

| HTTP Method | Endpoint Path | Feature | Description | Authentication Required | Tier Restrictions |
| --- | --- | --- | --- | --- | --- |
| POST | /api/v1/auth/register | Auth | Register a new user | No | None |
| POST | /api/v1/auth/login | Auth | Log in an existing user | No | None |
| POST | /api/v1/auth/refresh | Auth | Refresh an expired JWT | Yes (Refresh Token) | None |
| POST | /api/v1/auth/logout | Auth | Invalidate refresh token | Yes | None |
| POST | /api/v1/auth/password/reset | Auth | Request password reset | No | None |
| POST | /api/v1/auth/password/change | Auth | Change password | Yes | None |
| POST | /api/v1/ai/ask | Ask | Submit question, get AI response | Yes | Free (10/day), Premium (50/day), Pro (Unlimited) |
| GET | /api/v1/ai/interactions | Ask | Get user's interaction history | Yes | None |
| GET | /api/v1/ask/tones | Ask | Get available philosophical tones | Yes | None |
| POST | /api/v1/ask/tone-preview | Ask | Get sample response for tone | Yes | None |
| POST | /api/v1/ask/interactions/{interaction\_id}/save | Ask | Save AI interaction to journal | Yes | None |
| POST | /api/v1/ask/interactions/{interaction\_id}/expand | Ask | Expand insight | Yes | Premium, Pro |
| GET | /api/v1/ask/suggestions | Ask | Get personalized question suggestions | Yes | None |
| POST | /api/v1/seek-clarity/practical-examples | Seek Clarity | Request AI-generated examples | Yes | None |
| GET | /api/v1/seek-clarity/practical-examples/{task\_id} | Seek Clarity | Retrieve generated examples | Yes | None |
| POST | /api/v1/seek-clarity/practical-rephrasing | Seek Clarity | Request AI rephrasing | Yes | None |
| POST | /api/v1/seek-clarity/journal-challenge | Seek Clarity | Create journal challenge | Yes | None |
| POST | /api/v1/seek-clarity/wisdom-sharing | Seek Clarity | Generate curated sharing options | Yes | None |
| POST | /api/v1/seek-clarity/learning-path | Seek Clarity | Generate concept learning path | Yes | None |
| GET | /api/v1/quests | Quest | Get available quests | Yes | None |
| GET | /api/v1/quests/{id} | Quest | Get quest details | Yes | None |
| GET | /api/v1/quests/skill-tree | Quest | Get full skill tree data | Yes | None |
| POST | /api/v1/quests/{id}/start | Quest | Start/resume a quest | Yes | None (some quests may be Premium) |
| POST | /api/v1/quests/{id}/steps/{step\_id}/complete | Quest | Complete quest step | Yes | None |
| GET | /api/v1/quests/recommended | Quest | Get personalized quest recommendations | Yes | None |
| POST | /api/v1/quests/{id}/concepts/{concept\_id}/explore | Quest | Track concept exploration from quest | Yes | None |
| GET | /api/v1/quests/daily | Quest | Get daily quest | Yes | None |
| GET | /api/v1/concepts | Explore | Get list of concepts | Yes | None |
| GET | /api/v1/concepts/{concept\_id} | Explore | Get concept details | Yes | None |
| GET | /api/v1/concepts/{concept\_id}/relationships | Explore | Get concept relationships | Yes | None |
| POST | /api/v1/explore/history/start | Explore | Start exploration session | Yes | None |
| PUT | /api/v1/explore/history/{session\_id}/update | Explore | Update exploration session | Yes | None |
| POST | /api/v1/explore/history/{session\_id}/end | Explore | End exploration session | Yes | None |
| GET | /api/v1/explore/visualize/map | Explore | Generate concept map visualization | Yes | None |
| GET | /api/v1/explore/visualize/constellation | Explore | Generate constellation view | Yes | None |
| GET | /api/v1/explore/recommendations | Explore | Get personalized concept recommendations | Yes | None |
| GET | /api/v1/explore/pathways | Explore | Get available learning pathways | Yes | None |
| GET | /api/v1/explore/pathways/{pathway\_id} | Explore | Get learning pathway details | Yes | None |
| POST | /api/v1/explore/pathways/{pathway\_id}/start | Explore | Start learning pathway | Yes | None |
| POST | /api/v1/explore/pathways/{pathway\_id}/concepts/{concept\_id}/complete | Explore | Complete concept in pathway | Yes | None |
| POST | /api/v1/explore/create-path | Explore | Create user-defined wisdom path | Yes | Pro |
| POST | /api/v1/explore/connect-concepts | Explore | Connect concepts in constellation | Yes | None |
| POST | /api/v1/explore/create-nudge | Explore | Create user-generated content nudge | Yes | None |
| GET | /api/v1/explore/nudges | Explore | Retrieve user-generated content nudges | Yes | None |
| GET | /api/v1/concepts/schools | Explore | Get concept schools/categories | Yes | None |
| GET | /api/v1/journal | Journal | Get user's journal entries | Yes | None |
| POST | /api/v1/journal | Journal | Create journal entry | Yes | Free (5/day), Premium (Unlimited), Pro (Unlimited) |
| GET | /api/v1/journal/{id} | Journal | Get journal entry details | Yes | None |
| PUT | /api/v1/journal/{id} | Journal | Update journal entry | Yes | None |
| DELETE | /api/v1/journal/{id} | Journal | Delete journal entry | Yes | None |
| GET | /api/v1/history | Journal | Get user's activity history | Yes | None |
| GET | /api/v1/forum/threads | Forum | Get forum threads | Yes | None |
| POST | /api/v1/forum/threads | Forum | Create a thread | Yes | Free (3/day), Premium (10/day), Pro (Unlimited) |
| GET | /api/v1/forum/threads/{id} | Forum | Get thread details | Yes | None |
| PUT | /api/v1/forum/threads/{id} | Forum | Update a thread | Yes | None |
| DELETE | /api/v1/forum/threads/{id} | Forum | Delete a thread | Yes | None |
| POST | /api/v1/forum/threads/{id}/comments | Forum | Add a comment | Yes | Free (10/day), Premium (30/day), Pro (Unlimited) |
| PUT | /api/v1/forum/comments/{id} | Forum | Update a comment | Yes | None |
| DELETE | /api/v1/forum/comments/{id} | Forum | Delete a comment | Yes | None |
| POST | /api/v1/forum/vote | Forum | Vote on thread or comment | Yes | None |
| POST | /api/v1/forum/report | Forum | Report inappropriate content | Yes | None |
| GET | /api/v1/users/me | Profile | Get current user profile | Yes | None |
| PUT | /api/v1/users/me | Profile | Update user profile | Yes | None |
| GET | /api/v1/users/me/settings | Profile | Get user settings | Yes | None |
| PUT | /api/v1/users/me/settings | Profile | Update user settings | Yes | None |
| GET | /api/v1/users/me/badges | Profile | Get user badges | Yes | None |
| GET | /api/v1/subscriptions | Subscription | Get subscription information | Yes | None |
| POST | /api/v1/subscriptions/create | Subscription | Create a subscription | Yes | None |
| PUT | /api/v1/subscriptions/update | Subscription | Update a subscription | Yes | None |
| DELETE | /api/v1/subscriptions/cancel | Subscription | Cancel a subscription | Yes | None |
| GET | /api/v1/notifications | Notification | Get user's notification history | Yes | None |
| PUT | /api/v1/notifications/preferences | Notification | Update notification preferences | Yes | None |
| POST | /api/v1/notifications/read/{id} | Notification | Mark notification as read | Yes | None |
| POST | /api/v1/notifications/read-all | Notification | Mark all notifications as read | Yes | None |
| POST | /api/v1/ratings/submit | Rating System | Submit AI response rating | Yes | None |
| GET | /api/v1/ratings/analytics | Rating System | Retrieve aggregated rating insights | Yes | None |
| GET | /api/v1/xp | XP System | Get user's XP information | Yes | None |
| GET | /api/v1/xp/history | XP System | Get user's XP transaction history | Yes | None |
| POST | /api/v1/xp/check-in | XP System | Perform daily check-in | Yes | None |
| GET | /health | System | Basic health check | No | None |
| GET | /health/db | System | Database health check | No | None |
| GET | /health/redis | System | Redis health check | No | None |
| GET | /health/detailed | System | Detailed health check (DB, Redis, Celery) | No | None |
| GET | /metrics | System | Prometheus metrics endpoint | No | None |

### **2. Frontend Components Inventory**

The Setarcos mobile application is built with React Native, leveraging Zustand for state management, React Navigation for routing, and NativeWind for styling.1 This cross-platform approach, enabling deployment on both iOS and Android from a single codebase, streamlines development while aiming for a native user experience.1

The frontend architecture is modular and scalable, organized into common/, core/, and feature/ components.1 This structure promotes reusability, maintainability, and scalability.

common components provide a consistent UI, core components handle app-wide concerns like headers and navigation, and feature components encapsulate specific functionalities.1 Zustand's lightweight global state management allows for efficient data sharing across the application without excessive boilerplate.1 This well-structured frontend facilitates understanding how AI-generated content or recommendations are displayed and integrated into the user interface. For an LLM engineer, knowing the component hierarchy and state management approach is crucial for dynamically adjusting UI elements based on AI output or user interaction. The clear separation of concerns means that efforts to enhance LLM features can focus on the data interaction and presentation logic within specific feature components without requiring a deep understanding of the entire UI framework. This modularity also simplifies the implementation of A/B testing for LLM features, as components can be easily swapped or their data inputs modified.

Global state within the application, such as user profile, XP progress, and current AI interaction context, is managed using Zustand stores located in the src/store/ directory.1 Specifically,

auth.ts manages authentication state, user.ts handles user profile state, and xp.ts manages the XP system state.1 These stores coordinate with API services for data operations, ensuring that relevant AI interaction data is accessible across features.

#### **2.1. Core Frontend Components**

These components form the foundational structure and common UI elements across the application.

* **React Native**: The core framework for building the cross-platform mobile application.1
* **Zustand**: Utilized for local state management, providing a lightweight and flexible state solution.1
* **React Navigation**: Manages navigation flows and routing within the application, ensuring a consistent user journey.1
* **NativeWind (styling)**: Used for styling components, leveraging Tailwind CSS principles for rapid UI development.1
* **Axios/Fetch (API calls)**: Handles communication with the backend API, enabling data exchange.1
* **Common Components (**src/components/common/**)**: A collection of general-purpose UI elements, including Button/, Card/, Input/, Modal/, and Typography/, designed for reusability across the app.1
* **Core Components (**src/components/core/**)**: Essential app structure components such as AppHeader/, BottomNav/, ErrorBoundary/, and LoadingState/, which provide consistent UI elements and handle global application concerns.1
* **App.tsx**: The main application component, serving as the entry point for the React Native application.1
* **index.js**: The primary entry point for the mobile application.1
* **AppNavigator.tsx**: Defines the main navigation structure of the application, orchestrating different screen flows.1
* **AuthNavigator.tsx**: Manages authentication-related navigation flows, such as login and registration.1
* **linking.ts**: Configures deep linking, allowing external URLs to navigate directly to specific app content.1

#### **2.2. Ask Feature Frontend Components**

These components are specifically designed for the AI-driven philosophical dialogue feature. The Ask feature's extensive list of specialized frontend components, such as QuestionInput, ToneSelector, ResponseCard, ConceptBadge, ContemplativeOrb, and ActionButtonGroup 1, indicates a highly interactive and user-centric design for AI interaction. The UI is not merely displaying text; it actively guides user input (e.g.,

QuestionAssistancePanel), manages AI persona selection (e.g., ToneSelector, TonePreviewModal), structures AI output (e.g., ResponseCard, ConceptBadgeStrip), and captures explicit user feedback (e.g., ContemplativeOrb).

For an LLM engineer, this implies that the frontend provides rich context and feedback loops crucial for model fine-tuning and quality evaluation. The ToneSelector components are particularly important for understanding how philosophical tones are presented and chosen by the user, directly impacting the AI Router's prompt workflow.1 The

ContemplativeOrb represents the direct channel for explicit user feedback, which is vital for quality evaluation and continuous model improvement.1 The presence of

QuestionAssistancePanel and ReflectionPrompt suggests opportunities for the LLM to proactively guide user inquiry, thereby improving the quality of input provided to the AI. The ToneSelector component communicates the chosen tone\_id to the backend via the POST /api/v1/ai/ask endpoint.1 The frontend can dynamically update available tones by querying the

GET /api/v1/ask/tones endpoint, which returns a list of tones, potentially indicating user-specific availability based on subscription tier.1

* **AskScreen.tsx**: The main screen for the Ask feature, serving as the primary interface for AI dialogues.1
* **QuestionInput.tsx**: A text input component for users to type their philosophical questions.1
* **QuestionAssistancePanel.tsx**: Provides smart suggestions and reformulation options to help users craft their questions.1
* **ReflectionPrompt.tsx**: A pre-question prompt designed to encourage thoughtful inquiry and deeper engagement before asking.1
* **ToneSelector.tsx**: A component for selecting philosophical tones, displaying options and loading definitions from a configuration file.1
* **TonePreviewModal.tsx**: Displays sample responses for selected tones, allowing users to preview characteristics before committing.1
* **ToneSelectionGrid.tsx**: An interactive grid presenting available philosophical tones with descriptions.1
* **ToneExplorationGallery.tsx**: An interactive gallery for discovering and comparing different philosophical tones.1
* **ProgressIndicator.tsx**: An animated, tone-appropriate loading indicator displayed during AI response generation.1
* **ResponseCard.tsx**: The primary display component for AI-generated responses.1
* **ConceptBadgeStrip.tsx**: A horizontal, scrollable strip displaying concept badges extracted from AI responses.1
* **ConceptBadge.tsx**: Clickable tags representing philosophical concepts, enabling navigation to the Explore feature.1
* **ActionButtonGroup.tsx**: A container for interaction buttons associated with AI responses, such as Save, Expand, Clarity, and Share.1
* **SaveButton.tsx**: Allows users to save AI responses to their personal journal.1
* **ShareButton.tsx**: Enables sharing of philosophical insights generated by the AI.1
* **Expand.tsx**: A component for expanding insights, providing deeper philosophical analysis.1
* **SeekClarity.tsx**: A component for accessing the "Seek Clarity" pathways for further exploration of an AI response.1
* **ContemplativeOrb.tsx**: A minimalist rating component for users to provide feedback on AI responses.1

#### **2.3. Seek Clarity Feature Frontend Components**

These components enable deeper engagement with philosophical insights from AI responses.

* **SeekClarityPanel.tsx**: The main component that expands from a button to an action panel, presenting various clarity pathways.1
* **PathwayCard.tsx**: A reusable component used to represent each distinct pathway option within the Seek Clarity feature.1
* **PracticalExampleModal.tsx**: Displays real-world case studies and examples related to philosophical concepts.1
* **PracticalRephrasingModal.tsx**: Shows context-specific applications and rephrasing of philosophical ideas.1
* **WisdomJournalIntegration.tsx**: Facilitates the creation of structured challenges and prompts within the user's Journal.1
* **WisdomSharingModal.tsx**: Provides curated options for sharing philosophical insights with others.1
* **LearningPathModal.tsx**: Presents concept-based learning paths for structured exploration of related ideas.1

#### **2.4. Cross-Feature Integration Frontend Components**

These components facilitate seamless navigation and data flow between different core features. The extensive list of "Cross-Feature Integration Frontend Components," such as ConceptBadge, SaveButton, ShareButton, and their associated preview/confirmation modals 1, highlights that Setarcos is designed as a cohesive learning ecosystem rather than a collection of siloed features. The UI/UX actively encourages users to transition between different modes of engagement (e.g., Ask to Explore, Ask to Journal, Ask to Forum) by making these transitions explicit and seamless. This design reinforces the overarching vision of the application by connecting dialogue, reflection, and community interaction.

For an LLM engineer, this means that the AI's output in one feature often serves as an input or trigger for engagement in another. For example, the LLM should be aware that the concepts it extracts from a dialogue will become clickable badges leading to the Explore feature. This creates opportunities for optimizing LLM-generated content to naturally facilitate these cross-feature journeys, perhaps by suggesting follow-up actions like "Consider journaling this insight" or "Explore the concept of 'virtue' further." This design also allows for LLM-driven "nudges" to encourage deeper application usage. Analytics events are indeed tracked for these cross-feature navigation and saving/sharing actions, including ask\_question, save\_insight, and share\_insight events, which capture properties like interaction\_id and question\_topic.1 These events provide valuable data that can inform LLM-driven recommendations for user engagement.

* **ConceptBadge.tsx**: An interactive component displaying philosophical concepts extracted from Ask responses, enabling direct navigation to the Explore feature for deeper understanding.1
* **ConceptNavigationService.tsx**: A service handling navigation between features, ensuring state preservation during transitions, and managing deep linking.1
* **ExploreContextProvider.tsx**: A context provider for the Explore feature's entry points, preserving the originating context (e.g., Ask interaction ID, question) to allow for a seamless return.1
* **SaveButton.tsx**: A button component for saving Ask interactions directly to the user's Journal.1
* **JournalEntryPreview.tsx**: A preview component that shows how a saved Ask interaction will appear as a journal entry, allowing for pre-save customization.1
* **SaveConfirmationModal.tsx**: A modal that confirms the successful saving of an Ask interaction to the Journal, often displaying XP earned.1
* **ShareButton.tsx**: A button component for sharing Ask interactions to the community Forum or external platforms.1
* **ForumPostPreview.tsx**: A preview component that shows how a shared Ask interaction will appear as a forum post, allowing for pre-share customization.1
* **ShareConfirmationModal.tsx**: A modal that confirms the successful sharing of an Ask interaction to the Forum, often displaying XP earned.1
* **ConceptBadgeStrip.tsx**: A container component that arranges and displays concept badges in a horizontal, scrollable strip.1
* **ConceptTooltip.tsx**: A tooltip component that appears on hover or focus of a concept badge, providing a brief definition and mastery level.1

#### **2.5. Quest Feature Frontend Components**

These components provide the user interface for structured philosophical explorations.

* **QuestScreen.tsx**: The main screen for browsing and selecting quests, featuring filtering and recommendation integration.1
* **QuestDetailScreen.tsx**: Displays detailed information about an individual quest, including its skill tree visualization and prerequisites.1
* **SkillTreeView.tsx**: A zoomable visualization component for the quest progression system, rendering nodes and connections with WebGL for smooth performance.1
* **QuestStep.tsx**: An individual step within a quest, dynamically rendering different content types (e.g., reading, multiple-choice, reflection) and handling user input.1
* **QuestProgress.tsx**: A progress indicator that renders a progress bar, step indicators, and displays XP earned for a quest.1
* **QuestNodeComponent.tsx**: Represents an individual node in the skill tree, indicating completion status and handling interactions.1
* **ConceptMiniTree.tsx**: A mini visualization displaying hierarchical relationships of concepts related to a quest, with interactive navigation.1

#### **2.6. Explore Feature Frontend Components**

These components enable the interactive cosmic visualization and concept discovery.

* **ExploreScreen.tsx**: The main screen for the Explore feature, implementing filtering and rendering concept cards within a visualization.1
* **ConceptMap.tsx**: An interactive visualization of concept relationships, utilizing WebGL for smooth performance with zoom and pan capabilities.1
* **ConceptDetail.tsx**: Provides a detailed view of individual philosophical concepts, including descriptions, examples, and related concepts.1
* **ConceptSearch.tsx**: A search interface for finding specific concepts, featuring real-time suggestions and filtering.1
* **ConceptFilter.tsx**: Offers filtering options for concept exploration, allowing users to narrow down concepts by category, tradition, era, and difficulty.1
* **PathwaySelector.tsx**: An interface for selecting curated learning pathways, showing progression and prerequisites.1
* **ConstellationView.tsx**: An alternative visualization mode that organizes concepts into thematic clusters, resembling star constellations.1
* **RelatedConceptsList.tsx**: Displays a list of concepts related to the current selection, indicating relationship types and strengths.1
* **ConceptHistoryTracker.tsx**: A component that tracks the user's exploration history, providing breadcrumb navigation and insights into learning patterns.1
* **NotificationBanner.tsx**: A component for displaying exploration-specific notifications, such as milestone achievements and recommendations.1
* **CuriosityTrigger.tsx**: A component designed to display organic user-generated content (UGC) related social proof nudges, often animated as "shooting stars" with intriguing questions.1
* **PathCreator.tsx**: A tool allowing highest-tier paid users to create and share their own wisdom paths, linking concepts and other app features.1
* **ConceptColorSystem.tsx**: Manages the color-coding system for concept types and relationships, ensuring proper categorization and symbology.1
* **OrganicNudgeSystem.tsx**: Handles the display and interaction with user-generated content nudges, including thought fragments, revelation echoes, and question seeds.1

#### **2.7. Journal Feature Frontend Components**

These components provide the user interface for personal reflection and activity tracking.

* **JournalScreen.tsx**: The main view for the user's personal journal.1
* **JournalEntryEditor.tsx**: A component for creating and editing journal entries, allowing users to record their reflections.1
* **JournalEntryList.tsx**: Displays a list of the user's journal entries, often with filtering and sorting options.1
* **UserHistoryTab.tsx**: A tab within the journal or profile that displays the user's aggregated activity history across the entire application.1

#### **2.8. Forum Feature Frontend Components**

These components enable community discussions and interactions.

* **ForumScreen.tsx**: The main view for the community forum.1
* **ThreadList.tsx**: Displays a list of discussion threads within the forum.1
* **ThreadDetail.tsx**: Provides a detailed view of an individual discussion thread, including its content and comments.1
* **CommentEditor.tsx**: A component for creating and editing comments within forum threads.1

#### **2.9. Profile Feature Frontend Components**

These components manage user profiles, settings, and achievements.

* **ProfileScreen.tsx**: The main view for the user's personal profile, displaying XP progress, stats, and account management options.1
* **SettingsScreen.tsx**: Allows users to configure various application settings and preferences.1
* **SubscriptionScreen.tsx**: Manages the user's subscription details and options.1
* **AchievementsScreen.tsx**: Displays the badges and achievements earned by the user.1
* **AccessibilityControls.tsx**: Provides controls for various accessibility options, such as high-contrast mode, text size adjustment, screen reader optimization, and audio narration preferences.1

#### **2.10. Notification Feature Frontend Components**

These components handle in-app and push notifications.

* **NotificationScreen.tsx**: A dedicated inbox for viewing and managing user notifications.1
* **NotificationSettings.tsx**: A component within the Profile section for users to manage their notification preferences.1
* **NotificationBadge.tsx**: An unread count indicator typically displayed on tab navigation icons to alert users to new notifications.1
* **NotificationToast.tsx**: Transient, non-intrusive in-app alerts that appear briefly to convey information.1

#### **2.11. Contemplative Orb Rating System Frontend Components**

These components provide the minimalist user feedback mechanism for AI responses.

* **ContemplativeOrb.tsx**: The main rating component that appears alongside AI responses, allowing users to provide feedback on quality.1
* **OrbAnimation.tsx**: Handles subtle animation effects for interactions with the Contemplative Orb, providing visual feedback.1
* **RatingConfirmation.tsx**: Provides a minimal visual confirmation to the user after a rating has been submitted.1

**Table 2: Frontend Components by Feature**

| Component Name | Primary Feature | Key Responsibility/Function |
| --- | --- | --- |
| App.tsx | Core | Main application component, entry point |
| index.js | Core | Application entry point |
| AppNavigator.tsx | Core | Main navigation structure |
| AuthNavigator.tsx | Core | Authentication flows navigation |
| linking.ts | Core | Deep linking configuration |
| Button/, Card/, Input/, Modal/, Typography/ | Core (Common) | Reusable general-purpose UI elements |
| AppHeader/, BottomNav/, ErrorBoundary/, LoadingState/ | Core (Core Structure) | Core app structure components |
| AskScreen.tsx | Ask | Main screen for AI philosophical dialogues |
| QuestionInput.tsx | Ask | Text input for user questions |
| QuestionAssistancePanel.tsx | Ask | Provides smart suggestions for questions |
| ReflectionPrompt.tsx | Ask | Pre-question prompt for thoughtful inquiry |
| ToneSelector.tsx | Ask | Component for selecting philosophical tones |
| TonePreviewModal.tsx | Ask | Displays sample responses for selected tones |
| ToneSelectionGrid.tsx | Ask | Interactive grid for tone selection |
| ToneExplorationGallery.tsx | Ask | Interactive gallery for tone discovery |
| ProgressIndicator.tsx | Ask | Animated loading indicator for AI responses |
| ResponseCard.tsx | Ask | Display for AI-generated responses |
| ConceptBadgeStrip.tsx | Ask | Horizontal strip of concept badges from responses |
| ConceptBadge.tsx | Ask, Cross-Feature | Clickable concept tags, navigates to Explore |
| ActionButtonGroup.tsx | Ask | Container for response interaction buttons |
| SaveButton.tsx | Ask, Cross-Feature | Allows saving responses to Journal |
| ShareButton.tsx | Ask, Cross-Feature | Enables sharing insights to Forum/external |
| Expand.tsx | Ask | Component for expanding insights |
| SeekClarity.tsx | Ask | Access point for Seek Clarity pathways |
| ContemplativeOrb.tsx | Ask, Rating System | Rating component for AI responses |
| SeekClarityPanel.tsx | Seek Clarity | Main action panel for clarity pathways |
| PathwayCard.tsx | Seek Clarity | Reusable component for pathway options |
| PracticalExampleModal.tsx | Seek Clarity | Displays real-world case studies |
| PracticalRephrasingModal.tsx | Seek Clarity | Shows context-specific applications |
| WisdomJournalIntegration.tsx | Seek Clarity | Creates structured journal challenges |
| WisdomSharingModal.tsx | Seek Clarity | Provides curated sharing options |
| LearningPathModal.tsx | Seek Clarity | Presents concept-based learning paths |
| ConceptNavigationService.tsx | Cross-Feature | Handles navigation between features |
| ExploreContextProvider.tsx | Cross-Feature | Context provider for Explore entry points |
| JournalEntryPreview.tsx | Cross-Feature | Preview of saved entry in Journal |
| SaveConfirmationModal.tsx | Cross-Feature | Confirmation modal for saving to Journal |
| ForumPostPreview.tsx | Cross-Feature | Preview of shared post in Forum |
| ShareConfirmationModal.tsx | Cross-Feature | Confirmation modal for sharing to Forum |
| ConceptTooltip.tsx | Cross-Feature | Tooltip for concept badges |
| QuestScreen.tsx | Quest | Quest browsing and selection |
| QuestDetailScreen.tsx | Quest | Individual quest view with skill tree |
| SkillTreeView.tsx | Quest | Zoomable visualization of quest progression |
| QuestStep.tsx | Quest | Individual step in a quest |
| QuestProgress.tsx | Quest | Progress indicator for quests |
| QuestNodeComponent.tsx | Quest | Individual node in skill tree |
| ConceptMiniTree.tsx | Quest | Mini visualization of related concepts |
| ExploreScreen.tsx | Explore | Main screen for concept exploration |
| ConceptMap.tsx | Explore | Interactive visualization of concept relationships |
| ConceptDetail.tsx | Explore | Detailed view of individual concepts |
| ConceptSearch.tsx | Explore | Search interface for concepts |
| ConceptFilter.tsx | Explore | Filtering options for concept exploration |
| PathwaySelector.tsx | Explore | Interface for selecting learning pathways |
| ConstellationView.tsx | Explore | Alternative visualization of concept clusters |
| RelatedConceptsList.tsx | Explore | List of related concepts |
| ConceptHistoryTracker.tsx | Explore | Tracks user's exploration history |
| NotificationBanner.tsx | Explore | Displays exploration notifications |
| CuriosityTrigger.tsx | Explore | Displays organic UGC social proof nudges |
| PathCreator.tsx | Explore | Tool for users to create wisdom paths |
| ConceptColorSystem.tsx | Explore | Manages color-coding for concepts |
| OrganicNudgeSystem.tsx | Explore | Handles thought fragments, revelation echoes, question seeds |
| JournalScreen.tsx | Journal | Main journal view |
| JournalEntryEditor.tsx | Journal | Create/edit journal entries |
| JournalEntryList.tsx | Journal | List of journal entries |
| UserHistoryTab.tsx | Journal | User activity history |
| ForumScreen.tsx | Forum | Main forum view |
| ThreadList.tsx | Forum | List of discussion threads |
| ThreadDetail.tsx | Forum | Individual thread view |
| CommentEditor.tsx | Forum | Create/edit comments |
| ProfileScreen.tsx | Profile | Main profile view |
| SettingsScreen.tsx | Profile | User settings |
| SubscriptionScreen.tsx | Profile | Subscription management |
| AchievementsScreen.tsx | Profile | Badges and achievements display |
| AccessibilityControls.tsx | Profile | Controls for accessibility options |
| NotificationScreen.tsx | Notification | Dedicated notification inbox |
| NotificationSettings.tsx | Notification | User preference management for notifications |
| NotificationBadge.tsx | Notification | Unread count indicator for navigation |
| NotificationToast.tsx | Notification | Transient in-app alerts |
| OrbAnimation.tsx | Rating System | Subtle animation effects for orb interactions |
| RatingConfirmation.tsx | Rating System | Minimal confirmation of rating submission |

### **3. Backend Components Inventory**

The Setarcos backend is built with FastAPI, utilizing Uvicorn as the ASGI server, Pydantic for data validation, and SQLAlchemy for ORM-based database interaction.1 It is designed for high performance and asynchronous API development.1

Asynchronous processing is a core architectural principle of the Setarcos backend. Celery and Redis are explicitly listed as key libraries and core components, with a dedicated tasks/ directory for Celery task definitions.1 The detailed Celery configuration in Section 10.1 and task definitions in Section 10.3 confirm a heavy reliance on asynchronous processing.1 This design ensures that potentially long-running or non-critical operations, such as complex XP calculations, concept extraction, generating practical examples, and quality correlation analysis, are offloaded to Celery workers.1 This approach is crucial for maintaining a responsive user interface and efficient resource utilization, especially for AI-driven tasks that can introduce variable latency. For an LLM engineer, this means that while the initial AI response might be fast, subsequent processing and data updates might have a slight delay. When designing LLM features, it is important to consider which parts can be synchronous and which should be asynchronous to avoid blocking the user experience. This also implies that LLM-generated content might be post-processed or analyzed by these background tasks, potentially impacting how it is stored or used by other features. Celery queues are configured with priorities (e.g.,

notifications\_critical at priority 10, ratings\_realtime at 8, analytics\_realtime at 6, and default at 0) to ensure high-priority tasks are processed first.1 Task failures are managed through retry mechanisms with exponential backoff, and tasks are re-queued if a worker dies (

task\_reject\_on\_worker\_lost = True).1

The AI Router's development follows a phased approach, progressing from a Foundational Router to Advanced Integration, and finally to ML Enhancement.1 This phased development is a strategic decision to manage the complexity and risk associated with integrating multiple large language models and advanced machine learning capabilities. It allows for incremental development, testing, and optimization, ensuring stability at each stage before introducing further sophistication. This design also implies a commitment to continuous improvement in AI model selection and performance. For an LLM engineer, understanding the current operational phase of the AI Router is crucial for interpreting its behavior and planning future work. For example, in Phase 1, model selection is rule-based and predictable, whereas in Phase 3, it becomes dynamic and data-driven.1 This impacts how LLM models are integrated, how their performance is evaluated, and how new models might be introduced. The "Training Data Collection" and "Continuous Learning" components of Phase 3 highlight that user interactions directly feed into the LLM optimization loop, making user feedback (e.g., Contemplative Orb ratings) critically important for the LLM engineer's work.1 The specific data sources for

Training Data Collection include user satisfaction ratings, response quality metrics, engagement measurements, and cost efficiency tracking.1

#### **3.1. Core Backend Components**

These components form the foundational structure and common functionalities of the backend.

* **FastAPI**: The primary Python web framework, chosen for its high performance and asynchronous API development capabilities.1
* **Uvicorn**: The ASGI (Asynchronous Server Gateway Interface) server responsible for running the FastAPI application.1
* **Pydantic**: Used for data validation, ensuring that incoming request data conforms to defined schemas, and for managing application settings.1
* **SQLAlchemy**: The Object-Relational Mapper (ORM) used for interacting with the PostgreSQL database, abstracting SQL queries into Python objects.1
* **Celery**: A distributed task queue system that handles asynchronous processing of long-running or non-critical tasks, improving API responsiveness.1
* **Redis**: An in-memory data structure store, serving as both a caching layer to improve API performance and a message broker for Celery tasks.1
* **python-jose (JWT)**: A library used for handling JSON Web Token (JWT) authentication, specifically for validating tokens issued by Supabase Auth.1
* **httpx (async HTTP requests)**: An asynchronous HTTP client used for making non-blocking HTTP requests to external services, such as the various AI models.1
* **API Versioning (**src/api/v1/**)**: A structural approach to organizing API endpoints by version, allowing for future changes without breaking existing clients.1
* **Routes (**src/api/v1/routes/**)**: Contains the definitions of all API endpoints, mapping URL paths to specific backend logic.1
* **Schemas (**src/api/v1/schemas/**)**: Houses Pydantic models that define the structure and validation rules for request and response data.1
* **Core Logic (**src/core/**)**: Encapsulates fundamental functionalities, including authentication logic, the AI Router implementation, and dependency injection definitions.1
* **Database Models/Session Management (**src/database/**)**: Manages the setup for database connections, SQLAlchemy models, and session management for interacting with PostgreSQL.1
* **Services (**src/services/**)**: The business logic layer, where core application functionalities are implemented and orchestrated.1
* **Background Tasks (**src/tasks/**)**: Contains the definitions and logic for Celery tasks that run asynchronously in the background.1
* **Utilities (**src/utils/**)**: Provides a collection of utility functions, such as formatters, validators, and helpers, used across the backend.1
* **main.py**: The primary entry point for the FastAPI application.1
* **celery\_app.py**: Configures and initializes the Celery application, defining its settings and task loading.1

#### **3.2. Ask Feature Backend Components**

These components manage the AI interaction logic, including model selection, prompt handling, and data storage for the Ask feature.

* ai\_router.py: The intelligent model selection system responsible for choosing the optimal AI model for each user query.1
* ai\_service.py: Handles the integration with external AI model providers (Grok, Claude, Gemini, ChatGPT) through their respective APIs.1
* concept\_service.py: Manages the extraction and linking of philosophical concepts from AI responses, connecting them to the broader concept knowledge base.1
* tone\_service.py: Responsible for managing philosophical tone configurations and providing tone-specific prompt templates to the AI models.1
* ask\_service.py: The core service that orchestrates the entire question processing and response generation flow for the Ask feature.1
* question\_suggestion\_service.py: Generates contextual question suggestions for users, enhancing engagement and guiding further philosophical inquiry.1
* user\_preference\_service.py: Tracks and applies user tone preferences and other interaction patterns to personalize the AI experience.1
* response\_formatter.py: Formats AI responses for optimal display on mobile devices, ensuring readability and aesthetic presentation.1
* xp\_service.py: Calculates and awards experience points (XP) for various interactions within the Ask feature, integrating with the gamification system.1

#### **3.3. Seek Clarity Feature Backend Components**

These components manage the logic for the "Seek Clarity" pathways.

* seek\_clarity\_service.py: The core service responsible for managing all interactions and content generation within the Seek Clarity feature.1
* seek\_clarity\_tasks.py: Contains Celery tasks for asynchronous processing related to Seek Clarity, such as generating practical examples or rephrasing content.1
* API endpoints in seek\_clarity\_routes.py: Defines the specific API routes for interacting with the Seek Clarity service.1

#### **3.4. Cross-Feature Integration Backend Components**

These components handle the backend logic for seamless transitions and data flow between features.

* concept\_navigation\_service.py: Manages the tracking of concept navigation events between different features, preserving context for seamless transitions.1
* concept\_relevance\_service.py: Calculates and scores the relevance of philosophical concepts, used for display order and contextual linking across features.1
* journal\_integration\_service.py: Handles the conversion and saving of Ask interactions into the Journal feature.1
* reflection\_prompt\_service.py: Generates contextual reflection prompts for journal entries, often based on AI interactions.1
* forum\_integration\_service.py: Manages the conversion and sharing of Ask interactions as new threads or posts in the Forum feature.1
* discussion\_prompt\_service.py: Generates engaging discussion prompts to accompany shared content in the Forum, encouraging community interaction.1
* concept\_extraction\_service.py: A specialized service for identifying and extracting philosophical concepts from textual content, particularly AI responses.1

#### **3.5. Quest Feature Backend Components**

These components manage quest content, user progress, and rewards.

* quest\_service.py: The central service for quest management, handling quest retrieval, user progress tracking, step validation, and coordination with the XP system.1
* concept\_service.py: Integrates philosophical concepts into quests, mapping them to content and steps, and tracking concept mastery through quest completion.1
* xp\_service.py: Manages the awarding of XP for quest-related activities, tracks user XP totals, and determines badge eligibility.1
* QuestRepository: The data access layer for quest-related operations, providing optimized queries for quest entities.1
* UserQuestRepository: Manages data access for user-quest relationships, tracking individual user progress across quests and steps.1

#### **3.6. Explore Feature Backend Components**

These components manage philosophical concepts, relationships, and user exploration patterns.

* concept\_service.py: The core service for managing concept data, including retrieval, filtering, search, and providing concept recommendation algorithms.1
* concept\_relationship\_service.py: Manages the definition and rules for relationships between concepts, calculating their strength and providing graph traversal capabilities.1
* exploration\_history\_service.py: Tracks user exploration patterns, recording concept viewing sequences and durations to generate insights for personalization.1
* recommendation\_service.py: Generates personalized concept suggestions and identifies knowledge gaps based on user behavior and content.1
* visualization\_service.py: Responsible for generating optimized layouts for concept maps and constellation views, ensuring visual balance and performance.1
* notification\_service.py: Manages the generation and delivery of notifications related to concept exploration, such as milestone achievements.1
* explore\_service.py: The overarching service managing concept relationships and the core logic of the Explore feature.1
* constellation\_manager.py: Tracks and updates a user's personal concept constellation, reflecting their unique philosophical journey.1
* path\_service.py: Handles both system-generated and user-created wisdom paths, defining sequences of concepts for structured learning.1
* concept\_recommendation.py: A specialized component focused on generating personalized concept recommendations.1
* cross\_feature\_integration.py: Manages the connections and data flow between the Explore feature and other core features like Quest, Ask, Journal, and Forum.1
* organic\_nudge\_service.py: Processes and delivers user-generated content nudges (e.g., thought fragments, question seeds) to encourage organic discovery.1
* exploration\_analytics.py: Tracks user exploration patterns and engagement metrics specifically within the Explore feature.1
* ConceptRepository: The data access layer for concept-related operations, including CRUD operations and optimized queries.1
* UserConceptRepository: Manages data access for user-concept relationships, tracking individual user mastery and interaction history.1

#### **3.7. Journal Feature Backend Components**

These components manage user journal entries and activity history.

* journal\_service.py: The core service responsible for managing the creation, retrieval, updating, and deletion of user journal entries.1
* history\_service.py: Aggregates and manages the user's activity history from across different features of the application.1

#### **3.8. Forum Feature Backend Components**

These components manage forum threads, comments, and moderation.

* forum\_service.py: Manages the creation, retrieval, and interaction with discussion threads and comments within the community forum.1
* moderation\_service.py: Handles content moderation for forum posts and comments, including automated filtering and reporting mechanisms.1

#### **3.9. Profile Feature Backend Components**

These components manage user profiles, subscriptions, and achievements.

* user\_service.py: Manages user profile information, including display name, bio, avatar, and other user-specific settings.1
* subscription\_service.py: Handles all aspects of user subscriptions, including tier management, payment processing integration, and subscription status.1
* achievement\_service.py: Tracks and manages user badges and achievements earned through the Wisdom XP system.1

#### **3.10. Notification Feature Backend Components**

These components manage notification delivery and user preferences.

* notification\_service.py: The central service for managing notification logic, including user preference management and channel selection for delivery.1
* notification\_tasks.py: Contains Celery tasks for asynchronous notification processing, such as sending push notifications via FCM or emails via SendGrid.1

#### **3.11. Contemplative Orb Rating System Backend Components**

These components handle the processing and analysis of user feedback for AI responses.

* user\_rating\_service.py: Manages the capture and validation of user ratings provided via the Contemplative Orb.1
* quality\_correlation\_service.py: Analyzes the correlation between user ratings and predicted AI response quality, feeding into model optimization.1
* rating\_analytics\_service.py: Processes rating data for analytics, providing insights into user satisfaction and model performance.1

#### **3.12. AI Implementation Components**

These components are central to the AI Router's functionality, model selection, and quality evaluation.

* **Model Registry**: A component that defines and stores metadata for all available AI models, including their name, provider, cost, maximum tokens, strengths, philosophical tones they excel at, and availability score.1
* **Basic Selection Logic**: In Phase 1, this component implements rule-based selection of AI models primarily based on the chosen philosophical tone.1
* **Fallback Mechanism**: Ensures system robustness by defining primary, secondary, and tertiary fallback models in case the initial model fails or is unavailable, typically defaulting to a highly reliable model like ChatGPT.1
* **Context Analyzer**: Analyzes incoming user questions to determine characteristics such as complexity, philosophical depth, relevant concept categories, emotional tone, and length.1
* **User Context Integration**: Retrieves user-specific context, including subscription tier, past usage history, preferred AI models, concept mastery levels, and interaction patterns, to inform model selection.1
* **Dynamic Selection Algorithm**: In Phase 2, this algorithm selects the optimal AI model by scoring each available model based on the question context, selected tone, and user context.1
* **Model Selection ML Pipeline**: In Phase 3, this machine learning pipeline predicts the optimal model by extracting features from the question and user context, and then using an ML model selector and quality predictor to make a decision, considering quality-cost trade-offs.1
* **Training Data Collection**: Gathers data from various sources, including user satisfaction ratings (from the Contemplative Orb), AI response quality metrics, user engagement measurements, and cost efficiency tracking, to train and refine AI models.1
* **Continuous Learning**: Implements online learning algorithms, integrates feedback loops, detects performance drift in models, and manages automatic retraining pipelines to ensure ongoing AI improvement.1

#### **3.13. Database Models**

The Setarcos application leverages PostgreSQL, managed via Supabase, for persistent data storage.1 The schema is designed with clear relationships, UUID primary keys, and comprehensive indexing strategies to ensure data integrity and optimize query performance.1

The extensive number of tables dedicated to tracking user activity, progress, and feedback, such as user\_xp, xp\_transactions, user\_streaks, user\_badges, user\_concept\_progress, user\_rating\_interactions, user\_exploration\_history, user\_pathway\_progress, ask\_save\_events, ask\_share\_events, and concept\_badge\_interactions 1, underscores the application's core value proposition: gamification and personalized learning. This detailed data collection enables the system to track granular user behavior, award XP, manage streaks, and assess concept mastery. Crucially, user feedback from the Contemplative Orb is explicitly stored in

user\_rating\_interactions and linked to ai\_responses and response\_quality\_metrics 1, indicating a direct data pipeline for AI model improvement.

For an LLM engineer, this rich dataset is invaluable for training, fine-tuning, and evaluating models. The user\_rating\_interactions table, with its connection to AI responses and quality metrics, provides explicit feedback on AI quality. The detailed user progress data allows for deep personalization of LLM interactions, enabling the tailoring of responses based on a user's concept\_mastery\_level or user\_tone\_preferences.1 This robust data infrastructure is a powerful asset for continuous LLM improvement and for developing new LLM-driven features that leverage comprehensive user history and preferences. The data retention policy for granular user interaction data is not explicitly specified in the provided document, but it is a critical consideration for balancing user privacy with the needs for long-term LLM training and personalization. This would typically be defined in a separate data governance policy document.

* **User-Related Tables**:
  + users: Stores core user account information, including id (PK), auth\_id, username, email, display\_name, bio, avatar\_url, and timestamps.1
  + user\_settings: Holds user preferences such as notification\_preferences, theme, language, and privacy\_settings.1
  + user\_subscriptions: Details user subscription information, including subscription\_tier, stripe\_customer\_id, stripe\_subscription\_id, current\_period\_start/end, and status.1
  + user\_badges: Tracks badges earned by users, with badge\_id and earned\_at.1
  + user\_xp: Records overall user experience points (total\_xp) and level.1
  + xp\_transactions: Logs individual XP awards, detailing amount, source, source\_id, and reason.1
  + user\_streaks: Tracks user streak information, including current\_streak, longest\_streak, last\_activity\_date, and freeze\_count.1
  + user\_notification\_preferences: Stores user-specific notification settings, such as push\_enabled, email\_enabled, and quiet\_hours.1
  + user\_rating\_interactions: Records user ratings from the Contemplative Orb, linking to response\_id, rating\_value, interaction\_metrics, and quality correlation data like predicted\_quality\_score.1
  + user\_tone\_preferences: Stores user tone preferences and usage statistics, including usage\_count and is\_favorite.1
  + user\_question\_suggestions: Stores user-specific question suggestions, often linked to interaction\_id or concept\_id, with an expires\_at timestamp.1
  + user\_concept\_progress: Tracks a user's mastery\_level for individual concepts, along with interaction\_count and last\_interaction.1
  + user\_exploration\_history: Records user exploration sessions, including session\_id, the exploration\_path (sequence of concepts), and start/end times.1
  + user\_pathway\_progress: Tracks a user's progress on learning pathways, including current\_position and completed\_concepts.1
  + user\_constellations: Tracks the container for a user's personal concept constellation.1
  + user\_concept\_connections: Records connections users have made between concepts in their personal constellation.1
* **Content Tables**:
  + concepts: Defines core philosophical concepts, including name, description, category, tradition, era, difficulty\_level, image\_url, and color\_code.1
  + concept\_relationships: Defines connections between concepts, specifying source\_id, target\_id, relationship\_type, and strength.1
  + concept\_schools / concept\_domains: Categorizes concepts into philosophical schools or domains, with name, description, and color.1
  + philosophical\_tones: Stores attributes of philosophical tones, such as name, description, tier, and sample\_response.1
  + wisdom\_paths: Stores system-generated and user-created wisdom paths, including name, description, creator\_id, total\_xp, and estimated\_completion\_time.1
  + wisdom\_path\_steps: Defines the sequence of concepts within a wisdom\_path, specifying concept\_id and step\_order.1
  + curiosity\_triggers: Stores intriguing questions that serve as entry points to concepts, linked to concept\_id and domain\_id.1
  + organic\_nudges: Stores user-generated content nudges for organic discovery, including nudge\_type, content, and creator\_id.1
  + concept\_learning\_pathways: Stores curated learning paths, including name, description, difficulty\_level, and concepts\_sequence.1
* **Feature-Specific Tables**:
  + ai\_interactions: Records interactions within the Ask feature, including question, response, tone, model\_used, concepts (JSONB), is\_saved, and processing\_time\_ms.1
  + ask\_interaction\_concepts: Links ai\_interactions to concepts, with relevance\_score and is\_primary indicators.1
  + quests: Defines philosophical quests, including title, description, difficulty, xp\_reward, concepts (JSONB), and prerequisites.1
  + quest\_steps: Details individual steps within quests, specifying step\_number, title, content, step\_type, interaction\_data (JSONB), and xp\_reward.1
  + user\_quests: Tracks a user's progress on quests, including status, current\_step, started\_at, and completed\_at.1
  + user\_quest\_steps: Records a user's progress on individual quest steps, including status, user\_response (JSONB), and xp\_earned.1
  + quest\_badges: Defines badges awarded for quest achievements, with name, description, xp\_threshold, and icon\_url.1
  + journal\_entries: Stores user journal entries, including title, content, and is\_private status.1
  + journal\_tags: Links journal\_entries to concepts as tags.1
  + forum\_threads: Stores discussion threads, including title, content, view\_count, is\_pinned, and is\_locked status.1
  + forum\_thread\_tags: Links forum\_threads to concepts as tags.1
  + forum\_comments: Stores user comments on forum threads, including parent\_id for nesting and is\_solution status.1
  + forum\_votes: Records user upvotes/downvotes on threads or comments, specifying target\_type, target\_id, and vote\_type.1
  + forum\_reports: Stores reports of inappropriate content, detailing reason, description, and status.1
  + user\_history: Aggregates user activity across the app, including activity\_type, activity\_id, and metadata (JSONB).1
  + notifications: Stores notification history, including type, channel, content, status, and sent\_at/read\_at timestamps.1
  + seek\_clarity\_interactions: Tracks all interactions within the Seek Clarity pathways, including pathway\_type and interaction\_data (JSONB).1
  + practical\_examples: Stores AI-generated practical examples for Seek Clarity, with title, content, and source\_url.1
  + journal\_challenges: Links challenges generated by Seek Clarity to journal entries, including challenge\_type and due\_date.1
  + feature\_navigation\_events: Tracks navigation events between features, specifying source\_feature, destination\_feature, and session\_id.1
  + concept\_navigation\_context: Stores contextual information for cross-feature concept navigation, such as return\_path.1
  + ask\_journal\_links: Links Ask interactions to Journal entries, including an initial\_reflection.1
  + ask\_save\_events: Tracks save actions from Ask to Journal, including xp\_awarded and save\_context (JSONB).1
  + ask\_forum\_links: Links Ask interactions to Forum posts, including user\_commentary.1
  + ask\_share\_events: Tracks share actions from Ask to Forum or external platforms, including destination\_type and xp\_awarded.1
  + concept\_badge\_interactions: Tracks user interactions with concept badges, specifying interaction\_type (e.g., 'view', 'click').1
  + exploration\_notifications: Stores notifications specifically related to concept exploration, with notification\_type and related\_concept\_id.1
  + user\_concept\_interactions: Records granular user interactions with concepts, including interaction\_type, duration\_seconds, and source\_feature.1
  + ai\_responses (updates): Existing table updated with user\_rating, rating\_timestamp, and user\_engagement\_duration\_ms.1
  + response\_quality\_metrics (updates): Existing table updated with user\_rating\_correlation and rating\_prediction\_accuracy.1

**Table 3: Database Schema Overview**

| Table Name | Primary Purpose | Key Columns (PK, FKs) | Important Indexes |
| --- | --- | --- | --- |
| users | Core user accounts | id (PK) | username, email |
| user\_settings | User preferences | id (PK), user\_id (FK) | user\_id (Unique) |
| user\_subscriptions | Subscription details | id (PK), user\_id (FK) | user\_id (Unique) |
| user\_badges | Earned badges | id (PK), user\_id (FK), badge\_id | user\_id, badge\_id (Unique Composite) |
| user\_xp | XP and level tracking | id (PK), user\_id (FK) | user\_id (Unique) |
| xp\_transactions | XP award history | id (PK), user\_id (FK) | user\_id, created\_at |
| user\_streaks | Streak tracking | id (PK), user\_id (FK) | user\_id (Unique), current\_streak |
| user\_notification\_preferences | User notification settings | id (PK), user\_id (FK) | user\_id (Unique) |
| user\_rating\_interactions | User ratings from Contemplative Orb | id (PK), response\_id (FK), user\_id (FK) | response\_id, user\_id, philosopher\_tone, model\_used, created\_at |
| user\_tone\_preferences | User tone preferences | user\_id (FK), tone\_id (FK) | user\_id, tone\_id (PK Composite) |
| user\_question\_suggestions | User-specific question suggestions | id (PK), user\_id (FK) | user\_id |
| user\_concept\_progress | User's concept mastery | mastery\_id (PK), user\_id (FK), concept\_id (FK) | user\_id, concept\_id (Unique Composite), mastery\_level, exposure\_count, last\_interaction |
| user\_exploration\_history | User exploration sessions | history\_id (PK), user\_id (FK) | user\_id, session\_id, start\_time, entry\_point, exit\_point |
| user\_pathway\_progress | User progress on learning pathways | progress\_id (PK), user\_id (FK), pathway\_id (FK) | user\_id, pathway\_id (Unique Composite) |
| user\_constellations | User's personal constellation container | id (PK), user\_id (FK) | user\_id |
| user\_concept\_connections | Connections between user-discovered concepts | id (PK), user\_id (FK), source\_discovery\_id (FK), target\_discovery\_id (FK) | user\_id, source\_discovery\_id, target\_discovery\_id (Unique Composite) |
| concepts | Core concept definitions | concept\_id (PK) | name (Unique), Full-text search on name, description |
| concept\_relationships | Relationships between concepts | relationship\_id (PK), source\_concept\_id (FK), target\_concept\_id (FK) | source\_concept\_id, target\_concept\_id, relationship\_type, relationship\_strength |
| concept\_schools/concept\_domains | Philosophical schools/categories | id (PK) | name (Unique) |
| philosophical\_tones | Philosophical tones attributes | id (PK) | id (Unique) |
| wisdom\_paths | System/user-created wisdom paths | id (PK), creator\_id (FK) | None |
| wisdom\_path\_steps | Sequence of concepts in a wisdom path | id (PK), path\_id (FK), concept\_id (FK) | path\_id, step\_order (Unique Composite) |
| curiosity\_triggers | Intriguing questions for concepts | id (PK), concept\_id (FK), domain\_id (FK) | None |
| organic\_nudges | User-generated content nudges | id (PK), concept\_id (FK), creator\_id (FK) | None |
| concept\_learning\_pathways | Curated learning paths | pathway\_id (PK) | difficulty\_level, is\_premium, estimated\_duration\_minutes |
| ai\_interactions | Ask feature interactions | id (PK), user\_id (FK), tone\_id (FK) | created\_at |
| ask\_interaction\_concepts | Concepts extracted from ask interactions | interaction\_id (FK), concept\_id (FK) | interaction\_id, concept\_id (PK Composite) |
| quests | Quest definitions | id (PK) | difficulty, is\_premium, xp\_threshold, concepts (GIN) |
| quest\_steps | Individual quest steps | id (PK), quest\_id (FK) | quest\_id, step\_number (Unique Composite), step\_type, is\_challenge\_node, concept\_links (GIN) |
| user\_quests | User quest progress | id (PK), user\_id (FK), quest\_id (FK) | user\_id, quest\_id (Unique Composite), status, completed\_at |
| user\_quest\_steps | User progress on individual steps | id (PK), user\_quest\_id (FK), step\_id (FK) | user\_quest\_id, step\_id (Unique Composite), status, completed\_at |
| quest\_badges | Badges for quest achievements | id (PK) | quest\_category, xp\_threshold |
| journal\_entries | User journal entries | id (PK), user\_id (FK) | Full-text search on title, content |
| journal\_tags | Tags for entries | id (PK), journal\_entry\_id (FK), concept\_id (FK) | journal\_entry\_id, concept\_id (Unique Composite) |
| forum\_threads | Discussion threads | id (PK), user\_id (FK) | user\_id, created\_at, Full-text search on title, content |
| forum\_thread\_tags | Tags for threads | id (PK), thread\_id (FK), concept\_id (FK) | thread\_id, concept\_id (Unique Composite) |
| forum\_comments | User comments | id (PK), thread\_id (FK), user\_id (FK), parent\_id (FK) | thread\_id, created\_at |
| forum\_votes | Upvotes/downvotes | id (PK), user\_id (FK), target\_type, target\_id | user\_id, target\_type, target\_id (Unique Composite) |
| forum\_reports | Content reports | id (PK), user\_id (FK) | None |
| user\_history | Aggregated user activity | id (PK), user\_id (FK) | user\_id, created\_at, activity\_type |
| notifications | Notification history | id (PK), user\_id (FK) | user\_id, type, is\_read, created\_at, related\_concept\_id |
| seek\_clarity\_interactions | Seek Clarity pathway interactions | id (PK), user\_id (FK), response\_id (FK) | None |
| practical\_examples | AI-generated examples for Seek Clarity | id (PK), interaction\_id (FK) | None |
| journal\_challenges | Challenges linked to journal entries | id (PK), interaction\_id (FK), journal\_entry\_id (FK) | None |
| feature\_navigation\_events | Navigation between features via concepts | id (PK), user\_id (FK), concept\_id (FK) | None |
| concept\_navigation\_context | Context for cross-feature navigation | navigation\_event\_id (PK, FK) | None |
| ask\_journal\_links | Links Ask interactions to Journal entries | ask\_interaction\_id (FK), journal\_entry\_id (FK) | ask\_interaction\_id, journal\_entry\_id (PK Composite) |
| ask\_save\_events | Tracks save actions for analytics/XP | id (PK), user\_id (FK), ask\_interaction\_id (FK), journal\_entry\_id (FK) | None |
| ask\_forum\_links | Links Ask interactions to Forum posts | ask\_interaction\_id (FK), forum\_post\_id (FK) | ask\_interaction\_id, forum\_post\_id (PK Composite) |
| ask\_share\_events | Tracks share actions for analytics/XP | id (PK), user\_id (FK), ask\_interaction\_id (FK) | None |
| concept\_badge\_interactions | Tracks concept badge interactions | id (PK), user\_id (FK), interaction\_id (FK), concept\_id (FK) | None |
| exploration\_notifications | Notifications related to exploration | notification\_id (PK), user\_id (FK), related\_concept\_id (FK) | user\_id, notification\_type, is\_read, created\_at, related\_concept\_id |
| user\_concept\_interactions | User interactions with concepts | interaction\_id (PK), user\_id (FK), concept\_id (FK) | user\_id, concept\_id, interaction\_type, timestamp, source\_feature |

### **4. Services Inventory**

The Setarcos backend services are designed with a modular approach, encapsulating specific business logic and responsibilities. This promotes separation of concerns, reusability, and maintainability across the application.1

The AI implementation services, including CostOptimizer, ResponseCache, TokenOptimizer, PerformanceMonitor, QualityEvaluator, UserFeedbackCollector, and QualityImprovementEngine 1, represent a holistic approach to AI quality management. This comprehensive suite of services ensures that AI responses are not only philosophically insightful but also cost-efficient, fast, and continuously improving based on user feedback. The

QualityEvaluator assesses various aspects of AI responses, such as relevance, depth, and tone consistency, while the UserFeedbackCollector gathers both implicit (e.g., reading time, saves) and explicit (e.g., ratings) user feedback.1 This data then feeds into the

QualityImprovementEngine, which analyzes performance, updates model selection weights, and generates recommendations for ongoing enhancements.1 This integrated approach means that the system is designed to learn and adapt, ensuring that the AI models are constantly optimized for user satisfaction and operational efficiency. For an LLM engineer, this comprehensive quality management framework provides the necessary tools and data streams to monitor, evaluate, and iteratively improve the performance of the integrated LLMs.

#### **4.1. Core System Services**

These services provide fundamental functionalities utilized across multiple features.

* ToneService: Manages philosophical tone configurations, including loading them from file or cache, providing lists of available tones, retrieving specific tone details, and generating model-specific prompt templates.1
* AIService: The primary service responsible for generating AI responses by interfacing with various external AI models (Grok, Claude, Gemini, ChatGPT).1
* AnalyticsService: Tracks single and batch events across the application, identifies users with specific traits, and fetches feature flags for personalized experiences.1
* DashboardService: Facilitates the creation of new dashboards within PostHog and the addition of specific insights to these dashboards for monitoring key metrics.1
* PrivacyService: Manages data protection and privacy, including retrieving and exporting user data for GDPR compliance, deleting user data (right to be forgotten), and anonymizing analytics data.1
* ModerationService: Implements content moderation functionalities, checking for inappropriate material using pattern matching and AI moderation APIs, particularly for forum posts.1

#### **4.2. Ask Feature Services**

These services manage the core AI interaction logic for the Ask feature.

* ai\_router.py: The intelligent system that selects the optimal AI model for each user query based on various criteria.1
* ai\_service.py: Handles the direct integration with external AI model providers, managing API calls and responses.1
* concept\_service.py: Responsible for extracting and linking philosophical concepts from AI responses, connecting them to the broader concept knowledge base.1
* tone\_service.py: Manages the philosophical tone configurations and ensures consistent tone application across AI models.1
* ask\_service.py: The core service that orchestrates the entire question processing and response generation workflow for the Ask feature.1
* question\_suggestion\_service.py: Generates contextual and personalized question suggestions to guide user inquiry.1
* user\_preference\_service.py: Tracks and applies user tone preferences and other interaction patterns to personalize the AI experience.1
* response\_formatter.py: Formats AI responses for optimal display on mobile devices, ensuring readability and aesthetic presentation.1
* xp\_service.py: Calculates and awards experience points (XP) for interactions within the Ask feature, integrating with the gamification system.1

#### **4.3. Seek Clarity Feature Services**

These services handle the logic for the "Seek Clarity" pathways.

* seek\_clarity\_service.py: The central service for managing all interactions and content generation within the Seek Clarity feature.1
* seek\_clarity\_tasks.py: Contains Celery tasks for asynchronous processing related to Seek Clarity, such as generating practical examples or rephrasing content.1

#### **4.4. Cross-Feature Integration Services**

These services manage the backend logic for seamless transitions and data flow between features.

* concept\_navigation\_service.py: Handles the tracking of concept navigation events between different features, preserving context for seamless transitions.1
* concept\_relevance\_service.py: Calculates and scores the relevance of philosophical concepts, used for display order and contextual linking across features.1
* journal\_integration\_service.py: Manages the conversion and saving of Ask interactions into the Journal feature.1
* reflection\_prompt\_service.py: Generates contextual reflection prompts for journal entries, often based on AI interactions.1
* forum\_integration\_service.py: Handles the conversion and sharing of Ask interactions as new threads or posts in the Forum feature.1
* discussion\_prompt\_service.py: Generates engaging discussion prompts to accompany shared content in the Forum, encouraging community interaction.1
* concept\_extraction\_service.py: A specialized service for identifying and extracting philosophical concepts from textual content, particularly AI responses.1

#### **4.5. Quest Feature Services**

These services manage quest content, user progress, and rewards.

* quest\_service.py: The central service for quest management, handling quest retrieval, user progress tracking, step validation, and coordination with the XP system.1
* concept\_service.py: Integrates philosophical concepts into quests, mapping them to content and steps, and tracking concept mastery through quest completion.1
* xp\_service.py: Manages the awarding of XP for quest-related activities, tracks user XP totals, and determines badge eligibility.1

#### **4.6. Explore Feature Services**

These services manage philosophical concepts, relationships, and user exploration patterns.

* concept\_service.py: The core service for managing concept data, including retrieval, filtering, search, and providing concept recommendation algorithms.1
* concept\_relationship\_service.py: Manages the definition and rules for relationships between concepts, calculating their strength and providing graph traversal capabilities.1
* exploration\_history\_service.py: Tracks user exploration patterns, recording concept viewing sequences and durations to generate insights for personalization.1
* recommendation\_service.py: Generates personalized concept suggestions and identifies knowledge gaps based on user behavior and content.1
* visualization\_service.py: Responsible for generating optimized layouts for concept maps and constellation views, ensuring visual balance and performance.1
* notification\_service.py: Manages the generation and delivery of notifications related to concept exploration, such as milestone achievements.1
* explore\_service.py: The overarching service managing concept relationships and the core logic of the Explore feature.1
* constellation\_manager.py: Tracks and updates a user's personal concept constellation, reflecting their unique philosophical journey.1
* path\_service.py: Handles both system-generated and user-created wisdom paths, defining sequences of concepts for structured learning.1
* concept\_recommendation.py: A specialized component focused on generating personalized concept recommendations.1
* cross\_feature\_integration.py: Manages the connections and data flow between the Explore feature and other core features like Quest, Ask, Journal, and Forum.1
* organic\_nudge\_service.py: Processes and delivers user-generated content nudges (e.g., thought fragments, question seeds) to encourage organic discovery.1
* exploration\_analytics.py: Tracks user exploration patterns and engagement metrics specifically within the Explore feature.1

#### **4.7. Journal Feature Services**

These services manage user journal entries and activity history.

* journal\_service.py: The core service responsible for managing the creation, retrieval, updating, and deletion of user journal entries.1
* history\_service.py: Aggregates and manages the user's activity history from across different features of the application.1

#### **4.8. Forum Feature Services**

These services manage forum threads, comments, and moderation.

* forum\_service.py: Manages the creation, retrieval, and interaction with discussion threads and comments within the community forum.1
* moderation\_service.py: Handles content moderation for forum posts and comments, including automated filtering and reporting mechanisms.1

#### **4.9. Profile Feature Services**

These services manage user profiles, subscriptions, and achievements.

* user\_service.py: Manages user profile information, including display name, bio, avatar, and other user-specific settings.1
* subscription\_service.py: Handles all aspects of user subscriptions, including tier management, payment processing integration, and subscription status.1
* achievement\_service.py: Tracks and manages user badges and achievements earned through the Wisdom XP system.1

#### **4.10. Notification Feature Services**

These services manage notification delivery and user preferences.

* notification\_service.py: The central service for managing notification logic, including user preference management and channel selection for delivery.1

#### **4.11. Contemplative Orb Rating System Services**

These services handle the processing and analysis of user feedback for AI responses.

* user\_rating\_service.py: Manages the capture and validation of user ratings provided via the Contemplative Orb.1
* quality\_correlation\_service.py: Analyzes the correlation between user ratings and predicted AI response quality, feeding into model optimization.1
* rating\_analytics\_service.py: Processes rating data for analytics, providing insights into user satisfaction and model performance.1

#### **4.12. AI Implementation Services**

These services are central to the AI Router's functionality, model selection, and quality evaluation.

* CostOptimizer: Selects cost-optimal AI models based on user tier, current usage, and quality requirements, ensuring efficient resource allocation.1
* ResponseCache: Manages the caching of AI responses to improve performance and reduce costs associated with repeated queries.1
* TokenOptimizer: Optimizes prompt and response token usage to minimize costs and improve efficiency of AI model interactions.1
* PerformanceMonitor: Monitors AI model response times and error rates, providing real-time data for performance tuning and model selection.1
* QualityEvaluator: Systematically evaluates AI response quality based on various metrics such as relevance, philosophical depth, tone consistency, and concept accuracy.1
* UserFeedbackCollector: Gathers both implicit (e.g., reading time, saves) and explicit (e.g., ratings from Contemplative Orb) user feedback on AI responses.1
* QualityImprovementEngine: Analyzes performance data and user feedback to update model selection weights and generate recommendations for continuous improvement of AI responses.1

### **Conclusions**

The Setarcos application demonstrates a robust and thoughtfully designed technical architecture, particularly in its approach to integrating and managing AI functionalities. The detailed blueprint reveals a system built on a modern, decoupled foundation, leveraging FastAPI for a high-performance backend and React Native for a cross-platform mobile client.1

A key architectural strength lies in the application's comprehensive data strategy, which supports both gamification and continuous AI improvement. The extensive collection of user activity, progress, and feedback data, stored across numerous specialized database tables, provides a rich dataset for understanding user behavior and refining AI models.1 The explicit linkage between user ratings (via the Contemplative Orb) and AI response quality metrics establishes a direct feedback loop, which is fundamental for iterative LLM optimization.1 This data-driven approach positions the system for advanced personalization and adaptive learning experiences.

The phased development of the AI Router is a prudent strategy for managing complexity and risk associated with integrating multiple large language models.1 Progressing from rule-based selection to dynamic algorithms and ultimately to an ML-enhanced pipeline allows for controlled evolution and ensures stability at each stage. This incremental sophistication, coupled with continuous learning mechanisms that incorporate user feedback, means the AI capabilities are designed for ongoing refinement and performance enhancement.1

Furthermore, the application's commitment to asynchronous processing, evident in its heavy reliance on Celery and Redis, is critical for maintaining a responsive user experience, especially given the variable latency of AI-driven tasks.1 This architectural choice ensures that complex AI-related operations do not impede the primary user interaction flow. The emphasis on observability, demonstrated by the comprehensive health and metrics endpoints, provides essential tools for diagnosing operational issues and monitoring AI service performance in real-time.1

The inclusion of user-generated content features within the Explore section, such as user-defined wisdom paths and "nudges," represents a strategic move towards fostering a collaborative learning ecosystem.1 This not only enriches the philosophical content but also creates new avenues for LLM integration, such as assisting in content creation or moderation.

In conclusion, the Setarcos project exhibits a robust, scalable, and intelligently designed technical foundation. Its emphasis on data-driven AI improvement, phased development, and a highly interactive user experience positions it well for delivering on its vision of an engaging platform for philosophical exploration. For an AI LLM engineer, this blueprint provides a clear roadmap for integrating, optimizing, and extending the application's core AI functionalities, with ample opportunities for leveraging user data to drive continuous innovation.