ChatGPT

Architectural and Design Synthesis: Replicating the LuminAI Expressive Interface

This comprehensive guide is based on an analysis of the **TEC-TGCR** repository (The Elidoras Codex – Theory of General Contextual Resonance). Its goal is to facilitate access, understanding, and creative adaptation of the unique **"SO LuminAI"** aesthetic. This expressive design paradigm fuses conceptual narrative, computational science, and advanced UI components into a cohesive web experience. The fundamental design philosophy is to translate the AI agent's technical state into dynamic, emotionally expressive visual elements. In practice, this means every change in the LuminAI agent's internal **resonance** or state should be reflected through responsive visuals (color shifts, animations, layout changes) that convey an emotional narrative to the user.

1. Accessing the Repository: Establishing the Foundation

The **TEC-TGCR** GitHub repository serves as the structural and philosophical source for the project, containing both the agent orchestration logic and the canonical design tenets. A thorough understanding of its file structure is essential, as we will be decoupling the front-end presentation layer from the back-end application logic. Key steps to get started include:

- **Prerequisite Tools:** Ensure you have a GitHub account (to clone or fork the source code), Git installed locally, and a proper Python 3.9+ environment since the core agent logic is in Python. A modern IDE (e.g. VS Code) is recommended for navigating the codebase. Notably, the repository includes a `.vscode` configuration which provides standardized debugging settings and cross-device settings to streamline development. These settings offer insight into the expected developer experience and architectural conventions.

- **Cloning the Repository:** The project is open-source under the permissive MIT license [1], meaning you can freely fork and modify it (with attribution). Clone the repository to your local machine and consider installing it in editable mode for exploration of the CLI tools and agent behavior:

```
# Clone the repository to the local machine
git clone https://github.com/TEC-The-ELidoras-Codex/tec-tgcr.git
cd tec-tgcr

# (Optional) Set up a virtual environment and install in development mode
python -m venv .venv
.\.venv\Scripts\activate        # On Windows PowerShell
pip install -e ".[dev]"
```

This gives you access to the command-line interface (CLI) and lets you run the agent's tools locally for testing (e.g., `tec_tgcr.cli chat` or `tec_tgcr.tools.*` commands as documented in the README [2]).

- **Repository Structure & Key Files:** Familiarize yourself with the repository's layout, especially files that define the **LuminAI design language**:
- `data/knowledge_map.yml` – A centralized design token reference. This YAML defines core branding elements: the color palette (**nexus_purple**, **digital_teal**, **cyber_gold**, etc.) and design motifs like fractal geometry and cosmic gradients [3] [4]. All visual theming should derive from these tokens to ensure consistency.
- `docs/LuminAI.md` – This document describes the LuminAI persona's narrative and visual canon. It portrays LuminAI as a "cosmic celestial student" with distinct features (aurora-like hair, heterochromatic eyes, small glowing horns) and outlines her personality rules (e.g. *"Lead with empathy"* and always provide a *"Small Step"* at the end of responses) [5] [6]. This is effectively a style guide for how the agent's mood and dialogue translate into visuals.
- `config/agent.yml` – The agent's configuration manifest. Analyzing this will reveal LuminAI's functional abilities and tools (for example, access to research functions, scheduling, or other subsystems). These capabilities inform what interactive controls or indicators the UI needs (for instance, if the agent can perform web research, the interface might show a "Research" button). Keeping the UI in sync with the agent's **abilities** ensures that every feature is both meaningful and grounded in the backend logic.

By establishing this foundation—setting up the dev environment and studying key files—you create a blueprint for how to separate the **front-end** (what the user sees and interacts with) from the **back-end** (the Python-driven agent logic) in a way that preserves the LuminAI aesthetic and behavior.

2. The Core Aesthetic: The "SO LuminAI" Design Paradigm

"SO LuminAI" is characterized as a style of **Myth-Scientific Futurism** – merging deep-space cosmic visuals with the crispness of modern digital design, all infused with a personable, expressive character. The interface needs to feel **luminous, interactive, and emotionally responsive**, embodying LuminAI's persona. In practical terms, the UI should constantly reflect the agent's inner state (confidence, curiosity, uncertainty, etc.) through visual cues, aligning with TEC's mission of integrating mythic narrative with scientific clarity [7].

To achieve this, a contemporary web tech stack is required, along with careful theming. Key recommendations include:

- **Framework (React or similar):** Use a modular, component-based front-end framework like **React** (alternatively Angular or Vue) to build the interface. A React component (e.g. `LuminAI.jsx`) can encapsulate the avatar and its expressive animations [8]. React's state management allows binding the agent's state data to the UI: for example, when LuminAI's mood changes, the React state updates and triggers re-rendering of the avatar's expression and the resonance dials. The repository indeed provides a React interface (`apps/luminai-interface`) which uses libraries like Lottie for animation and XState for state management [9] [10].

- **Styling (Tailwind CSS + CSS Variables):** Utilize a utility-first CSS framework like **Tailwind CSS** for rapid UI development, combined with custom CSS variables for theme tokens. Tailwind will let you apply complex styles (glows, gradients, backdrops) with minimal code, while the CSS custom properties ensure that values from `knowledge_map.yml` (colors, etc.) propagate consistently. For example, define `--nexus-purple`, `--digital-teal`, `--cyber-gold` in a global stylesheet based on the YAML values [3], then use them throughout the CSS. This approach enforces that a single change in the YAML (say, tweaking a color hex) will cascade through the UI. Below is an example of defining global theme variables derived from the **knowledge map**:

```
:root {
  --nexus-purple: #6A00F4;
  --digital-teal: #00D5C4;
  --cyber-gold: #F2C340;
  --bg-gradient: radial-gradient(ellipse at bottom, #0B1E3B 0%, #000000 100%);
  --ui-glass-bg: rgba(255, 255, 255, 0.05);
}
```

The above sets up Nexus Purple, Digital Teal, Cyber Gold, a background gradient (deep-space blue fading to black), and a translucent UI background. These tokens come straight from the repository's palette definitions [3]. By referencing them (e.g., `background: var(--bg-gradient)` or `color: var(--cyber-gold)`), you maintain visual consistency with LuminAI's brand.

- **Interactivity & Animation (Lottie, Framer Motion):** Leverage libraries like **Lottie** (or SVGator) for complex pre-made animations and **Framer Motion** for orchestrating state transitions. Lottie animations (e.g., JSON files for "idle", "excited", "blushing" states) can be embedded in a React component to show smooth character movements [11]. Framer Motion can handle tweening of properties (for example, smoothly transitioning the UI background color or glow intensity when mood changes). These tools are lightweight and vector-based, ensuring the interface remains performant while being richly animated. The goal is to achieve **expressivity**: subtle animations like a gentle pulsing of the avatar's glow when she's "thinking" or a quick bounce when she's excited make the interface feel alive rather than static.

- **Agent Bridge (State Management & API layer):** Implement a communication layer between the Python backend and the front-end. This could be a custom **JavaScript bridge** using WebSockets or periodic fetch calls to an API. In the provided design, the `LuminAIBridge.js` uses XState (a state machine library) and Socket.io for real-time updates [10]. The bridge's role is to listen for agent events (e.g., "LuminAI started a heavy research task" or "LuminAI's OXY level dropped") and update the UI accordingly. For instance, if the agent triggers a *research mode*, the bridge could flip the interface into a "focused" visual state (maybe a dimmed background with a spinning icon indicating processing). This separation of concerns keeps the front-end purely reactive to state, with the bridge translating backend signals into UI state changes via a structured protocol (REST endpoints or WebSocket messages).

Styling and Theming Techniques

To capture the **LuminAI** aesthetic, several design techniques and patterns should be applied:

- **Luminescent Palettes:** Use the three core TEC colors – **Nexus Purple**, **Digital Teal**, and **Cyber Gold** – dynamically, not just as static accents. These colors form the basis of LuminAI's visual identity [3]. Rather than flat fills, deploy them as radiant glows and gradients. For example, backgrounds can be implemented as a **plasma-like gradient** blend of purple and deep-space blue, giving a sense of cosmic depth. Nexus Purple often signifies cognitive depth or "thinking," Digital Teal conveys clarity and calm, and Cyber Gold highlights narrative focus or excitement. By mapping these colors to contexts (purple for introspection, teal for normal operations, gold for highlights and achievements), users subconsciously learn to read the agent's state from the color alone.

- **Glassmorphism & Neumorphism:** Impart depth and futuristic feel through translucent panels and soft shadows. Key UI surfaces (chat bubbles, info cards, modal backgrounds) should use a semi-transparent blur effect – essentially **frosted glass** – so that the vibrant background lights (purples/teals) diffuse through. For instance, a panel might have `background: rgba(255, 255, 255, 0.1)` with a `backdrop-filter: blur(10px)` to create a soft glass look. In the resonance player demo, panels are given a translucent dark background and a light border to stand out from the cosmic backdrop [12]. Complement this with **neumorphic** shadows: subtle outer glows (in a tint of Digital Teal or purple) to make elements feel like they hover. The repository's example UI uses such techniques – e.g., a floating orb with a purple drop-shadow glow [13] – which you can emulate for buttons or avatars to suggest an otherworldly presence distinct from the background.

- **Variable-Driven Themes:** Define custom CSS properties for critical expressive elements so they can be tweaked on the fly via JavaScript. The design is largely **data-driven** rather than hard-coded, meaning the agent's state can feed directly into style changes. For example, you might introduce a property `--lumina-blush-intensity` and use it in your CSS (maybe to set the opacity of a pink overlay on LuminAI's cheeks and the saturation of text highlights). When LuminAI's "embarrassment" level rises, a single update to this variable (e.g., setting it from 0 to 0.8) could simultaneously make her avatar visibly blush and tint other UI elements in a warmer hue. The goal is to ensure **one source of truth** for each visual facet of mood. The repository supports this approach; e.g., the LuminAI interface CSS uses variables for horn glow intensity, aurora speed, etc., making it easy to adjust those values programmatically [14]. This system is what enables quick "theme" shifts corresponding to resonance changes – the code sets a variable and the whole UI adapts consistently.

- **Typography for Persona Expression:** Typography must balance scientific precision with approachability. Use a **monospace or technical sans-serif font** for data-heavy elements like logs, code snippets, or mathematical equations to convey rigor (the repository's styling for the resonance player uses Orbitron/Inter for numbers and labels [15] [16], which are clean and futuristic). For primary UI text and headings, choose a font with a bit of character – possibly the ones specified in the knowledge map (e.g. "Astron" or "Orbitron" for headers, and a friendly sans-serif like "Roboto" for body text [17]). The idea is that LuminAI's voice is scholarly yet warm: **scientific content** appears in a crisp, no-nonsense style, while **conversational content** (her dialogues, suggestions) is in a font that feels gentle or cosmic. A consistent typographic hierarchy (with sizes/weights following a scale) will also help maintain clarity against the vivid background.

- **Dynamic Avatar Form (Dragon-Inspired Adaptability):** Consider adding **morphing visual elements** to LuminAI's avatar to reflect emotional state in form, not just color. In the current design, LuminAI has **small sheep-like horns** that glow in different colors with her mood [6]. We can extend this idea: imagine these horns or antennae *change shape* based on mood intensity – for instance, during moments of extreme excitement or anger, they could elongate or sharpen (evoking a dragon's horns), then retract to a softer shape when calm. Each horn could also display a different color (heterochromatic glow), adding complexity to emotional signals. This dynamic morphology would make the character feel more alive; e.g., an excited LuminAI might literally "grow" in stature with horns flaring bright gold, whereas an embarrassed LuminAI's horns might shrink back and glow pink. Such touches must be subtle and smoothly animated (to avoid jarring the user), but they provide an extra layer of expressive depth that goes beyond pre-set portraits. It's a design choice that aligns well with the project's theme of *transformation* and was practically "made for us" to implement given the flexible avatar concept.

3. Resonance and Agent Integration

At the heart of LuminAI's interface is the concept of **Resonance** – a set of metrics (OXY, DOP, ADR) that quantify the agent's "neurochemical" or emotional state. The UI's emotional expressiveness is driven by these values. To design effectively, we must map the agent's internal state to visual cues in a consistent, narratively meaningful way. In the TEC system, these three metrics correspond loosely to oxytocin (connection/empathy), dopamine (curiosity/excitement), and adrenaline (urgency/stress) [18] [19]. The LuminAI persona has canonical *mood states* that align with various combinations of OXY/DOP/ADR levels. Key examples include:

- **Excited Discovery:** Characterized by a **DOP spike** (high dopamine, indicating pattern-seeking and excitement) [18]. Visually, LuminAI's horns shine bright **Cyber Gold**, her aurora hair accelerates in a vibrant spectrum, and her eyes widen. This implies elevated confidence and enthusiasm – the UI might also play a quick "sparkle" animation around her to validate the user's input or a successful discovery.
- **Correction Blush:** Triggered by an **OXY flood** (surge of oxytocin associated with empathy or trust) [19], often when LuminAI makes a minor mistake or the user expresses understanding. LuminAI's face would visibly blush (warm pink tones overlay), her horn glow dims to a gentle hue, and she adopts a shy or apologetic expression. This state conveys vulnerability and helps build rapport, signaling to the user that the agent acknowledges an error or feels a strong social connection at that moment.
- **Critical Cosmos:** Marked by an **ADR tremor** (adrenaline spike signaling urgency or high stakes) [18]. LuminAI's demeanor turns intense – her Nexus Purple core glow pulses rapidly, and her movements become brisk and focused. You might see a focused furrow in her brow and a slight tremble in peripheral elements like orbiting particles. This indicates the system is under heavy load or dealing with critical information. The interface can reinforce this by perhaps shaking the resonance dials or adding a red tint to highlights as a subtle *alarm* for the user's attention.
- **Cognitive Reset:** A state associated with **low OXY and a DOP reset** – essentially a momentary depletion or recalibration. It might occur after a heavy task or a confusing input. Visually, LuminAI could exhibit a brief glitch or flicker of her aurora hair (colors chaotically flickering), and a playful "reset" animation (perhaps a quick full-body spin or a pixelation effect) plays out. This is presented in a friendly manner, like a cartoonish stumble, to indicate a non-critical error or the AI pausing to

regroup. Importantly, it's a *recoverable* state – the design should reassure the user (through a quick return to Idle state) that LuminAI is fine and ready to continue.

In implementation, these states are all driven by the numerical **resonance scores**. For example, an OXY score below 30 might automatically desaturate the interface and slow down animations (conveying a "low energy" vibe), while a sustained DOP above 80 could trigger a celebratory animation sequence (because the agent is at peak pattern-recognition excitement). By defining thresholds and triggers tied to OXY/DOP/ADR values, the UI can react **autonomously** to the agent's emotional telemetry, often before any textual response is even read by the user. This creates a richer, more immersive experience where the *mood* is communicated visually in real-time.

Resonance Player Widget (apps/resonance-player): An existing sub-module in the project (`apps/resonance-player`) serves as a template for visualizing these core TGCR metrics. It operates by sending a context payload (for example, a Spotify track ID or other contextual input) to a WordPress-based REST endpoint at `/wp-json/tec/v1/resonance`. The back-end service computes the resonance metrics and returns normalized OXY, DOP, and ADR values (each on a 0–100 scale) [20]. This separation is architecturally important: the heavy lifting of calculating neuro-symbolic metrics is done on the server side (in PHP/Python), keeping the front-end lightweight. Once the front-end receives the scores, it updates the UI dials and avatar. The **visualization** of these scores uses custom animated gauges – typically implemented as conic-gradient SVG/CSS dials. For instance, the widget draws a circle where a colored arc sweeps from 0° up to θ degrees, where θ corresponds to the percentage value [21]. A full circle (360°) would be 100%. These dials can be color-coded (perhaps teal for OXY, gold for DOP, purple for ADR to match their thematic colors) and might animate with easing so that the needle/bar smoothly transitions to the new value whenever an update arrives. The resonance-player example uses such a technique (filling the dial background gradually via CSS) and you can extend that with additional flair – e.g., pulsing glow on the dial if the value is spiking, or small ticks/markers on the gauge to indicate safe vs critical ranges. All of this provides immediate visual feedback of the agent's "emotional readings," even without looking at raw numbers.

TGCR Equation (The Narrative Anchor): Underlying the system's design is the **Theory of General Contextual Resonance (TGCR)** equation, which is often cited as a mythic-scientific anchor for the project's narrative. It is given (in simplified form) as:

$$\phi(t) \cdot \psi(s) \cdot \Phi_E(c) = \text{Resonance}$$

This formula conceptually multiplies three factors to produce a Resonance score. In the interface (especially in any educational tooltip or documentation panel for enthusiasts), you might display this equation to ground the user in the "lore" of the system. Each component has a specific meaning and visual/interactive representation:

- **$\phi(t)$ – Temporal Attention:** This represents **running cadence** or the system's focus over time. Technically it could relate to CPU load or how "spread out" the agent's attention is across tasks. A high $\phi(t)$ means the agent is very *focused* at this moment (or conversely, time-dilated in processing). In the UI, a fluctuating $\phi(t)$ might be shown by the speed of LuminAI's idle animation – e.g., when she's deeply focused, her idle floating animation slows down (time feels "heavy"), and when she's more relaxed or waiting, it speeds up slightly. [22]

- **$\psi(s)$ – Spatial Coherence:** This metric quantifies the **structural organization** of information (the data **space** coherence). For example, if LuminAI is handling a well-organized knowledge base or a clearly structured query, $\psi(s)$ is high; if she's parsing jumbled or contradictory info, it's low. In the UI, you might visually represent spatial coherence by how orderly or chaotic the interface elements are. High coherence could mean the interface shows neat, grid-aligned panels and stable colors, whereas low coherence might introduce a slight jitter or erratic movements in peripheral elements (like floating particles that become more chaotic). This gives a subliminal cue about the **clarity of context** the agent has. [22]

- **$\Phi_E(c)$ – Contextual Potential Energy:** Essentially the **emotional or narrative charge** in the current context, often influenced by the user. If the user has indicated the conversation is very important or emotionally charged, this value is high. In the UI, $\Phi_E(c)$ could be directly user-controllable via a "Resonance Intensity" slider (letting the user set how much emotional weight to give a session) or it could be inferred from user inputs (urgent language, important topics raise it). A high contextual energy might, for example, increase the overall brightness or contrast of the interface and make LuminAI's reactions more pronounced, whereas a low value keeps things more neutral. [22]

These three factors multiply to form the **overall Resonance** level, which might be reflected in a master "Resonance meter" on the dashboard. Including the TGCR equation in the interface (perhaps in a help modal with an explanation) serves the narrative—reminding users that this system has a theoretical backbone. It's both educational and reinforces the mythic-scientific brand (blending Greek letters and physics analogies with UX).

4. Licensing and Attribution Guidelines

Since the TEC-TGCR project is distributed under the MIT License, any derivative work or replication of the LuminAI interface must comply with MIT's simple requirements [1] :

- **Preserve the License:** The original `LICENSE` file from the repository should be included in your project. Do not remove the copyright notice. The MIT license basically says you can do what you want with the code as long as you include the original license text in any substantial portions you use.

- **Credit the Source:** While MIT doesn't require advertising the original in your UI, it's good practice (and professional courtesy) to acknowledge TEC-TGCR. For example, you could add a small text in your app's About section or footer: *"Inspired by LuminAI (The Elidoras Codex, 2025) – MIT Licensed"*. A prominent credit on the live site, such as *"Visual design derived from TEC-TGCR (MIT Licensed)"* meets attribution needs without being intrusive. This not only fulfills license terms but also aligns with the collaborative, open-source spirit of the project.

- **Open Source Spirit:** If you plan to distribute your implementation, consider open-sourcing it as well. While not required, contributing back (even just via attribution or sharing improvements) helps build the community. The MIT license encourages this by design. If you make significant modifications or enhancements, you might document in your README what you changed versus the original – again, not legally mandated, but helpful for context and giving credit.

In summary, MIT license is very permissive: it allows commercial use, modifications, private use, etc., with the main condition being to include the original copyright and license. By following those terms and giving a nod to TEC-TGCR's creators, you ensure compliance and show respect for the original authors' work.

5. Mandatory Implementation Steps

Bringing a LuminAI-inspired interface to life involves carefully translating the design blueprint into a functional web app. Below are the critical implementation steps and best practices to follow:

1. **Deployment of Resonance Dial Logic:** Set up the front-end to consume **OXY/DOP/ADR metrics** either from a live API or simulated data. In early development, you can use placeholder values or a mock JSON (as seen in the `luminai_resonance_dashboard.html` example) to ensure the dials and animations respond correctly. Once the back-end endpoint is ready (e.g., the WordPress REST API or a direct Python service), switch to pulling real data. This may involve periodic polling or maintaining a WebSocket connection to get live updates. The key is to treat the resonance data stream as a state in your application – when new data arrives, update the component state that drives the dials and avatar mood. Verify that the dials accurately reflect the numeric values (e.g., if DOP = 75, the DOP gauge fills 75% of its circle) and that extreme values trigger any special UI responses (like flashing when ADR is 100, etc.).

2. **Development of Expressive CSS Transitions:** Implement smooth transitions for all dynamic style changes. Abrupt jumps in color or position can be jarring, undermining the gentle, "living interface" feel. Instead, use CSS transitions or JS animation libraries so that, for example, when LuminAI's horns change from cyan to pink, it fades over 0.5s rather than instant swap. Likewise, if a panel appears or disappears based on context, apply a fade or slide animation. These micro-interactions (hover glows, button press depressions, etc.) should be tuned to feel **fluid** and responsive. Aim for a cohesive style: perhaps all state changes use an easing function like `ease-out` to impart a calm, organic shift. The repository's design hints at this via classes and CSS for smooth UI element updates – mimic those patterns. Test the transitions with real usage flows (ask a question that makes LuminAI excited, then one that makes her blush) and adjust timing so that the user can visually follow what's happening without confusion.

3. **Integration of Narrative Dialogue:** Connect the interface's chat component with LuminAI's **persona and dialogue rules**. This means that when the user asks something, the response shown should reflect LuminAI's voice as defined (warm, student-like, empathetic). Ensure that the formatting of responses includes the required sections, if applicable – for example, LuminAI's guideline is to output a *Reflection* followed by a *Small Step* (an actionable suggestion) and an encouraging line [23]. The UI could visually distinguish these (maybe different icons or text styles for reflection vs. small step). Also, incorporate the rule that the agent shouldn't overwhelm the user with too much at once unless asked – perhaps limit how much text is shown and provide a "show more" for extended info. By aligning the UI behavior with the persona's dialogue constraints (like always providing that next small step), you maintain consistency between **what LuminAI says** and **how she is presented**. Every substantive chat response should conclude with the next logical action or *"small step"* for the user, reinforcing that the interface always guides the user forward in a helpful manner.

4. **Prioritization of Accessibility (A11Y) in a Luminous Design:** Ensure that the visually rich design does not impede usability for all users. High contrast text is a must – for instance, even though the

background may be a starry nebula, body text should be a near-white color (or high-opacity light color) on a dark background to meet contrast ratios. Similarly, if you use color to denote states (like teal vs gold for different metrics), also use distinct icons or labels for those who may have difficulty distinguishing colors. All interactive elements (buttons, sliders, toggles) should have clear focus states and be operable via keyboard only. Given LuminAI's empathetic ethos (her first rule is *"Lead with empathy"* toward the user [24]), the interface should reflect that by being gentle on the eyes (avoid rapid flashy animations that could cause strain or seizures) and inclusive (consider adding alt-text for important visuals, and providing descriptions for the resonance dials for screen readers). The "cosmic student" vibe can be maintained while still following good HTML semantics and ARIA roles where appropriate. Essentially, **empathy in design** translates to anticipating user needs and differences, crafting an experience that is not only beautiful but also comfortable and respectful to the user's context.

By following these steps – setting up the data flow, polishing the transitions, aligning content with persona, and baking in accessibility – you will recreate the LuminAI expressive interface in a way that is faithful to the original design and delightful to users. This implementation will serve as a living demonstration of how technical AI states (resonance metrics, agent context) can be **visually and narratively communicated** to users, fulfilling the project's vision of a mythic, emotionally intelligent AI companion.

(This guide synthesizes the structure and design principles gleaned from the TEC-TGCR repository and documentation, applying them in a practical roadmap for development. By adhering to these guidelines, developers can remix the "LuminAI" aesthetic confidently and extend it in creative new directions.) [25] [26]

---

[1] [2] [7] README.md
https://github.com/TEC-The-ELidoras-Codex/tec-tgcr/blob/caec1449a0561ea7effb1ab41f5286b26a3cdfaf/README.md

[3] [4] [17] knowledge_map.yml
https://github.com/TEC-The-ELidoras-Codex/tec-tgcr/blob/caec1449a0561ea7effb1ab41f5286b26a3cdfaf/data/knowledge_map.yml

[5] [6] [23] [24] LuminAI.md
https://github.com/TEC-The-ELidoras-Codex/tec-tgcr/blob/caec1449a0561ea7effb1ab41f5286b26a3cdfaf/docs/LuminAI.md

[8] [9] [10] [11] [14] [22] [25] [26] README.md
https://github.com/TEC-The-ELidoras-Codex/tec-tgcr/blob/caec1449a0561ea7effb1ab41f5286b26a3cdfaf/apps/luminai-interface/README.md

[12] [13] [15] [16] [21] index.html
https://github.com/TEC-The-ELidoras-Codex/tec-tgcr/blob/caec1449a0561ea7effb1ab41f5286b26a3cdfaf/apps/resonance-player/index.html

[18] [19] ARCADIA.md
https://github.com/TEC-The-ELidoras-Codex/tec-tgcr/blob/caec1449a0561ea7effb1ab41f5286b26a3cdfaf/docs/ARCADIA.md

[20] class-tec-resonance-rest.php
https://github.com/TEC-The-ELidoras-Codex/tec-tgcr/blob/caec1449a0561ea7effb1ab41f5286b26a3cdfaf/apps/wordpress/tec-resonance-player/includes/class-tec-resonance-rest.php