# Aurora v1 Modular Pipeline & Architecture Redesign

## Executive Summary

The current project (Aurora v1) is organized as a tightly coupled collection of WebGPU/WebGL components driven by a single monolithic script (e.g. an all-in-one app.js)[[1]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L6-L9). This central script mixes disparate concerns – scene setup, physics simulation, audio routing, UI controls, and post-processing – within large, interdependent modules that rely on implicit globals and ad-hoc state synchronization[[2]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L6-L11). The result is a **messy structure** with redundant code and complex cross-dependencies. Configuration is managed through a mutable singleton object updated by timers, and modules use inconsistent import patterns, making the pipeline neither cohesive nor modular. Features like the particle physics simulator or post-effects cannot be easily reused outside the current runtime without pulling in large parts of the system[[3]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L9-L11).

**Goal:** This document proposes a comprehensive refactor to achieve a more **modular, organized, and flexible pipeline**. We will transition to a modern **ECMAScript Module (ESM)** and **TypeScript**-based architecture that splits functionality into focused, standalone components[[4]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L14-L19). Each major subsystem (stage rendering, physics simulation, audio, camera/lens effects, post-processing, etc.) will become an independent module with clear interfaces and lifecycle hooks, orchestrated by a small application shell. A unified configuration store and event messaging system will replace the current global state, allowing **hot-swappable** modules and real-time updates across components[[4]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L14-L19). In addition, we will introduce robust tooling – build scripts, linting, testing, and continuous integration – to improve code quality and maintainability[[5]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L4-L5)[[6]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L18-L19). The end result will be a cleaner, more performant codebase that is easier to extend, reuse, and **optimize** for future needs.

## Current State Assessment

The existing implementation suffers from several structural issues that motivate this redesign:

* **Monolithic orchestration** – A single bootstrap file (app.js) performs renderer creation, scene/stage setup, physics stepping, audio initialization, input handling, and applying post-effects, all in one place with minimal abstraction boundaries[[1]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L6-L9). This makes the code hard to navigate and modify in isolation.
* **Implicit configuration sync** – Global config (conf.js) is used as a mutable singleton to drive many subsystems, with values being polled or updated on intervals (e.g. a glass refraction parameter updating every 150 ms)[[7]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L7-L9). This implicit, time-based synchronization leads to brittle behavior and makes it unclear which module owns certain state.
* **Mixed module patterns** – The codebase inconsistently mixes CommonJS-style requires with newer ES module imports (referred to as "JSM" in notes), preventing effective tree-shaking and complicating the bundler configuration[[8]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L9-L10). The lack of a unified module format increases technical debt and hinders using modern build optimizations.
* **Sparse pipeline tooling** – There is **no type checking or linting** in place, and no automated formatting or tests[[9]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L10-L11). Maintaining consistency between scripts (and shaders) is manual and error-prone. The absence of a robust build/test pipeline means regressions or performance issues can slip in unnoticed.
* **Limited modularity & reuse** – Key components (e.g. the MLS-MPM physics simulator, post-processing effects, audio router) are tightly integrated and not exposed as independent modules[[9]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L10-L11). They cannot be extracted or reused in another project or even run in isolation for testing without pulling in large portions of the system. This lack of modularity makes the pipeline inflexible and harder to extend or replace parts (for example, swapping out the physics engine or adding a new visual effect).

These issues result in a codebase that is difficult to **maintain, scale, or upgrade**. Any change in one area (say, the camera focus or an audio effect) might unintentionally impact others due to implicit couplings. The goal of the refactor is to eliminate these pain points by redesigning the architecture from the ground up.

## Refactor Goals and Objectives

To address the above problems, the refactor will pursue the following key objectives[[4]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L14-L19):

1. **Explicit modular architecture** – Introduce an application shell that manages distinct feature modules (Stage, Physics, Audio, PostFX, Camera, UI, etc.) via clear **dependency injection** and lifecycle hooks. Each module will be loadable, removable, or runnable independently, which facilitates testing and “hot-swapping” components without affecting others[[4]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L14-L19).
2. **Adopt TypeScript throughout** – Migrate the codebase to TypeScript for **static type checking** and better developer tooling. A strict tsconfig will be used to enforce type safety. We will leverage a fast bundler (Vite with ESBuild) for development to support hot module reloading and efficient production builds[[10]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L14-L17).
3. **ESM-based project structure** – Restructure the code into a collection of **ES modules** (ECMAScript modules) organized by feature. This modular layout will allow building smaller standalone demos or tools (e.g. a physics-only sandbox or an audio-visualizer mode) by including only the necessary modules[[11]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L16-L19). It also enables better tree-shaking and optimized bundle sizes.
4. **Unified configuration system** – Replace the ad-hoc global config with a **central configuration store** that holds all tunable parameters. This store will support schema validation (ensuring values are in valid ranges/types), observable subscriptions for modules to react to changes, and easy serialization to save or load presets[[11]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L16-L19). This way, all components stay in sync via a single source of truth for config, without fragile timing logic.
5. **Automated pipeline & quality controls** – Integrate modern dev tooling: linting (ESLint + Prettier) to enforce code style, continuous type checking, and a testing framework for unit tests and regression tests (including GPU frame comparisons for visual features)[[6]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L18-L19). The build process will be managed via Node scripts and include tasks for analysis (bundle sizes, performance metrics) to keep the project healthy.
6. **Improved runtime flexibility** – Decouple subsystems by using an event-driven design and explicit **interfaces** between modules. Renderer and simulation loops will be isolated, and an event/message bus will allow modules to communicate (for example, audio beat events triggering a physics reaction, or simulation state influencing camera focus)[[6]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L18-L19). Optional features can be loaded dynamically, and the app could even support running with certain modules disabled (for performance or testing), demonstrating the flexibility of the new architecture.

By achieving these goals, we expect the project to be far more **maintainable** and extensible. New features (or third-party plugins) can be added as new modules without fear of breaking the entire system. Development will be safer and faster with types and tests catching issues early, and the end-user experience will benefit from an optimized, well-structured pipeline.

## Proposed Modular Architecture

Under the new design, the codebase will be reorganized into clear directories and modules, each with a well-defined responsibility. The top-level structure is outlined as follows[[12]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L23-L31)[[13]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L31-L39):

* **src/core/** – Core framework and infrastructure. This includes the application host and orchestrator (AppHost.ts) which bootstraps the renderer and manages the module lifecycle, the module registry (ModuleRegistry.ts) for registering/swapping modules, a new **configuration store** (ConfigStore.ts) for centralized config management, an event hub (EventHub.ts) for cross-module messaging, and other core utilities (asset loaders, diagnostics, etc.)[[14]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L23-L30). These are the building blocks that the feature modules will rely on.
* **src/modules/** – Self-contained *feature modules* for each major subsystem of the application. Each module encapsulates an entire domain:
* StageModule.ts – Manages the stage or scene setup, including environment lighting, background, ground plane, and camera setup (camera rigs and controls)[[15]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L32-L36).
* PhysicsModule.ts – Contains the **MLS-MPM physics simulation** (particle solver) along with any emitters or boundaries and the rendering of simulated particles/materials[[15]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L32-L36). This module updates the physics state each frame and provides data (particle positions, counts, etc.) to other parts like the renderer or UI.
* PostFxModule.ts – Implements the post-processing pipeline (post effects). This includes effects like camera **depth-of-field (DOF)** blur, bloom, color grading, vignetting, chromatic aberration, etc., all composed in a single frame-graph[[16]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L33-L38). The module applies these as a stack of passes to the rendered scene, and it listens for relevant config changes (e.g. aperture or focus distance for DOF).
* MaterialModule.ts – Consolidates all shader materials and related GPU resources. Rather than spreading shader code across multiple files, this module maintains a **material library** (e.g. definitions for particle material, floor material, any special surface shaders) in one place[[17]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L34-L37). It provides an interface for other modules to request materials or update material parameters.
* AudioModule.ts – Manages the audio engine and analysis. This module sets up the Web Audio API context, handles audio input or music playback, and runs analyzers (FFT, beat detection, etc.) to produce audio feature data[[18]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L35-L37). It routes these audio features to other modules – for instance, linking low-frequency energy to particle emission rates or linking beat events to visual bloom intensity, via the event hub or direct calls.
* UiModule.tsx – Encapsulates the user interface, particularly the control panels and heads-up display. The UI module will likely be built with a UI library (React or Lit) and contains the definitions for all controls, arranged per feature domain (e.g. a panel for all **PostFX and camera lens controls**, a panel for physics parameters, another for audio/reactivity settings)[[19]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L35-L38)[[20]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/chats.md#L7-L13). This module binds UI elements to the ConfigStore so that user input updates the central config and thereby the respective modules.

Each modules/\*.ts file is designed to be **standalone** and cohesive, encapsulating all logic for that subsystem. This means internal helper classes or functions (for example, functions to set up lighting rigs or to configure the physics solver) will reside inside the module file (possibly namespaced or as static helpers) instead of scattered across many files[[21]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L60-L64). By having one dense file per module, we reduce import overhead and make it easy to enable/disable a feature by including or omitting that file. This approach also aligns with the desire for **modularity** (each piece can be treated independently) and ensures faster compilation (fewer files to transpile/pack). It will be important to keep each module well-organized internally (using clear sections or inner namespaces for sub-tasks) to maintain readability within the larger file[[22]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L60-L63). - **src/presets/** – A collection of preset configurations and default parameter sets. Here we will define, for example, StagePresets.ts for environment/lighting presets, PostFxPresets.ts for predefined camera lens or color filter settings, and SimulationPresets.ts for different physics or audio-reactivity configurations[[23]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L38-L41). These preset modules simply export data (or functions to apply data) that can be loaded into the ConfigStore. They make it easy to switch between **scenes or effect profiles** (e.g. day vs night lighting, or a “max intensity” postFX setting) and also serve as documentation of recommended parameter values. - **src/bootstrap/** – Application entry points and bootstrap scripts. For example, main.ts will be the entry that initializes the AppHost, registers all necessary modules, and attaches the rendering canvas to the DOM[[24]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L42-L45). If the physics or audio processing can be offloaded to a Web Worker thread, a worker-entry.ts would set up a worker with a subset of modules (e.g. running the PhysicsModule in a worker) and communicate with the main thread. This separation allows the main thread to handle rendering and UI while heavy computation runs in parallel, improving frame rates. - **tests/** – Test suites for the project. We will include both unit tests (e.g. tests for math utilities or ConfigStore logic) and integration tests (e.g. ensuring that when the config changes, the correct module updates, or that enabling/disabling PostFx yields expected outcomes)[[25]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L46-L49). Going forward, having a dedicated tests directory ensures we can continually verify the refactoring doesn’t introduce regressions. We may also include **GPU snapshot tests** – for instance, render a frame of the simulation and compare it to a baseline image to detect unintended visual changes. - **scripts/** – Automation and build scripts. This will house Node scripts for tasks like building the project (build.mjs), running type checks (check-types.mjs), linting (lint.mjs), or generating bundle reports[[26]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L50-L54). Having these in a scripts folder keeps package.json simpler and allows writing more complex build logic in JavaScript/TypeScript if needed (for example, pre-processing assets or customizing the build for different targets). We will also include any asset pipeline tools here (discussed later).

**Inter-module Communication:** In this new architecture, modules interact through well-defined channels rather than direct, hard-coded references. A central **AppHost** (in core/AppHost.ts) will load each module and call its lifecycle methods (init(), start(), update(delta), dispose(), etc.) at the appropriate times[[27]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L59-L61). The AppHost and a **ModuleRegistry** will allow modules to be hot-swapped (for example, replacing the PostFxModule at runtime with a different implementation, if needed for debugging or upgrades) by standardizing how modules register and deregister themselves[[27]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L59-L61).

To avoid the previous global-state issues, we introduce two key infrastructure pieces:

* A **Config Store** (core/ConfigStore.ts) which holds all configuration/state that might be shared or tweaked at runtime[[28]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L66-L67). This store will be the single source of truth for parameters (replacing the old conf singleton). It will likely be implemented with a small state-management library or pattern (for example, using **Zustand** or a similar reactive store for efficient updates)[[29]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L2-L5). The ConfigStore will expose typed getters/setters and a subscription mechanism; modules can subscribe to specific config keys or namespaces. When the UI or any system changes a config value, the store will emit change events, and only the relevant modules will react (for instance, the PostFx module listens to lens aperture changes, the Physics module listens to particle spawn rate changes, etc.). We will also incorporate **schema validation** for config – using a library like **Zod** to define a schema for config objects – so that we can validate and even migrate config presets easily[[30]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L96-L101). Preset saving/loading functionality will allow exporting the current config to JSON and re-importing it, which is invaluable for testing and for users sharing their favorite visual setups.
* An **Event & Messaging Hub** (core/EventHub.ts) to publish and subscribe to cross-cutting events[[28]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L66-L67). Rather than modules directly calling each other (which creates tight coupling), modules will broadcast events for things like "physics simulation completed a step", "audio beat detected", or "camera moved to new focus". Other modules can listen for those events and respond accordingly. The EventHub will be a lightweight pub-sub or observer pattern implementation (potentially using a typed event emitter so that events are type-safe). This decoupling means, for example, the PhysicsModule doesn't need to know about the AudioModule – it can just emit a "particleCollision" event that AudioModule may listen to for triggering a sound, or vice-versa. This design greatly improves **flexibility**, as new interactions can be created just by wiring events, and modules can be added or removed without breaking others (if a module isn't present, its events simply go unhandled).

Together, the AppHost, ConfigStore, and EventHub provide the **infrastructure for a cohesive pipeline** despite the code being split into modules. Each module focuses on its domain logic, while the core takes care of coordination. This separation also enables running subsets of the app (for instance, launching just the StageModule and PhysicsModule for a headless simulation test) without the overhead of the full UI or audio stack – a big boost for reusability.

Finally, it’s worth noting that we plan to maintain **dense, feature-focused modules** (instead of fragmenting logic into too many tiny files) as a conscious choice to reduce complexity[[21]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L60-L64). Each module will expose a clear interface (possibly an abstract FeatureModule base or interface that defines methods like init() and optional hot-reload hooks)[[27]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L59-L61). This ensures consistency across modules and makes the AppHost logic generic. By keeping related code within one file per module, we minimize the number of imports and files, which can improve build times and reduce cognitive load when navigating the code (developers can find all stage-related code in StageModule.ts, for example). Of course, if a module grows too large to manage, we will internally structure it (using internal classes or splitting into sub-sections with comments) rather than breaking the modular design.

## Development Pipeline & Tooling Enhancements

Refactoring the structure goes hand-in-hand with improving the development pipeline. We will set up a robust Node/TypeScript build and workflow to support the new architecture:

* **TypeScript Adoption:** We will convert the codebase to TypeScript (.ts / .tsx files) to gain static typing benefits[[31]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L70-L73). A strict compiler configuration (targeting modern ES2022 output for ESM) will be used to catch errors early. Initially, we might use JSDoc comments for types on existing JS as a transitional step, but the end goal is full TypeScript across all modules. The build will continue to use **Vite** (which already supports ESM) and **ESBuild** for fast bundling and Hot Module Replacement during development[[32]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L70-L74). We will introduce proper tsconfig.json and configure path aliases (e.g. @core/\* pointing to src/core and @modules/\* to src/modules) for cleaner import paths and to ensure tree-shaking of unused modules[[33]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L71-L74).
* **Build Targets and Assets:** The new pipeline will support multiple build targets. By default, it will target **WebGPU** (for browsers that support it) to leverage modern graphics, but we will also provide a fallback **WebGL** build for broader compatibility[[34]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L72-L75). This might involve maintaining conditional code or separate shader code paths (the repository already uses Three.js’s WebGPURenderer, and we can allow a WebGL path if needed). We will use Vite plugins for handling GLSL/WGSL shader imports and possibly for loading other assets (like HDR environment maps) at build time[[33]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L71-L74). Additionally, an optional **headless** build could be configured for running the app without a GUI (useful for server-side rendering or automated testing of the simulation)[[34]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L72-L75).
* **Automated Scripts:** We will introduce a series of npm scripts to streamline common tasks[[35]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L78-L84):
* npm run dev – start a development server (Vite) with live reload and hot module replacement for rapid development.
* npm run build – produce a production build, including minification and generating a bundle analysis report to inspect module sizes[[36]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L78-L81). This ensures we keep the bundle efficient and can track the impact of changes.
* npm run check – run a full suite of checks: TypeScript type checking, ESLint, Prettier formatting validation, and unit tests[[37]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L78-L82). This will be useful in continuous integration to gate any broken code from being merged.
* npm run lint – run ESLint (with our config) to catch code style or potential error issues[[38]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L80-L82).
* npm run test – run the test suite (using **Vitest** for unit tests, which is a Vite-friendly test runner) and any integration tests. We plan to include **GPU tests** (for example using a headless browser with Puppeteer or Playwright) to render a frame and compare outputs[[38]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L80-L82)[[39]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L118-L121).
* (Additional scripts like npm run profile can be added to capture performance profiles or run custom analysis – e.g., using Chrome DevTools protocol to capture a frame timeline – to help optimize graphics performance[[39]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L118-L121).)
* **Linting and Formatting:** We will set up **ESLint** with the TypeScript parser and a custom rule set to enforce best practices[[40]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L86-L89). For example, we will forbid usage of any types, enforce boundaries so that modules don’t accidentally import private code from others (helping keep the architecture decoupled), and possibly create rules to catch common errors (like using the old conf singleton or improper WebGPU resource handling). **Prettier** will be used for consistent code formatting, and an .editorconfig will ensure consistent indentation and line endings across contributors[[40]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L86-L89). These tools will run on each commit via a Git hook (using Husky or a simple npm script) and also run in CI to ensure code quality is maintained automatically.
* **Testing & QA:** Quality assurance will be significantly enhanced. We will implement unit tests for critical logic (math routines, config merging, etc.) and **integration tests** for module contracts. For example, we might simulate a config change and assert that only the intended module responded. We also plan to use **Vitest** (a fast testing framework compatible with Vite) for writing tests, and possibly utilize **headless browser tests** for graphics. Using a tool like Puppeteer or Playwright, we can programmatically load the app in a headless browser, simulate user interactions or time steps, and capture rendered frames. These frames can be compared to baseline images (snapshot testing) to detect if a refactor unintentionally changed the visuals[[38]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L80-L82)[[39]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L118-L121). Such GPU regression tests are important for an audio-visual project where many changes could subtly affect rendering or performance. The testing infrastructure will run as part of CI and can also be run locally with npm run test.
* **Continuous Integration (CI):** We will set up a CI pipeline (for example, GitHub Actions) that runs on each push/PR to the repo[[40]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L86-L89). This CI will run the full npm run check suite: linting, type checks, tests, and attempt a build. By automating this, we catch errors early and ensure that the main branch of the project is always in a deployable state. Any issues introduced by a commit (like a type regression or a failed test) will be flagged immediately.
* **Asset Pipeline:** To optimize load times and manage assets, we will add an **asset processing pipeline**. For instance, a script scripts/asset-manifest.mjs can pre-compute metadata for heavy assets like HDR images or 3D meshes[[41]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L91-L93). This could involve generating lower-resolution versions, pre-calculating bounding boxes, or converting assets to more efficient formats (such as compressing textures, converting JSON meshes to binary). We can also include shader compilation tests in a tools/ folder – small programs that compile the shaders (WGSL/GLSL) offline to catch errors early. Automating asset handling ensures the application initializes faster and uses GPU/CPU resources more efficiently. For example, we might convert large JSON particle emitter definitions into binary data that can be loaded faster, or bake certain calculations into lookup tables stored in assets.
* **External Libraries & Enhancements:** As part of this refactor, we will assess opportunities to incorporate well-vetted libraries to enhance functionality without reinventing the wheel. For example, the **MLS-MPM** fluid/particle simulation is currently based on the WebGPU-Ocean reference[[42]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/CREDITS.md#L3-L6); we will continue to use and improve that core algorithm, but the new modular structure will make it easier to update or swap out if a better physics library emerges. For the configuration schema, using **Zod** (for schema definition and validation) is planned[[30]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L96-L101). For the state management in ConfigStore, as noted, a small library like **Zustand** or even the Reactivity API from a framework could be used to avoid writing a custom solution[[29]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L2-L5). The UI might leverage a component framework (React, as implied by the .tsx extension, or the lightweight Lit library) to build complex control panels more maintainably[[29]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L2-L5). We will also use Three.js and its WebGPU renderer as a foundation but within our own modules – this gives us access to a rich graphics API while still allowing custom control (e.g., custom shader materials, or using Three.js just for the rendering context and doing physics visuals manually). Overall, external tools and libraries will be used judiciously to accelerate development: if a library can provide a needed feature (like robust state management or schema validation) with low overhead, the modular design will let us plug it in without upsetting other parts of the system.

By establishing this improved development pipeline and tooling, we not only make the **refactoring process safer** (since mistakes will be caught by tests and type checks), but we also set the project up for long-term **sustainability**. New contributors or team members will have an easier time understanding the structure (thanks to consistent code style and documentation), and routine tasks (like running the app or deploying it) will be straightforward and automated.

## Refactor Implementation Plan

Refactoring such a large project will be done **incrementally** to manage risk. The following phased plan outlines how we can transition from the current state to the target architecture, ensuring that at each step we have a working application and can verify progress:

1. **Foundation Setup:** Begin by laying the groundwork for the new architecture. This includes initializing the TypeScript project configuration and adding basic tooling (ESLint, Prettier, testing framework)[[43]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L103-L106). At this stage, we will create the skeleton of the new structure: for example, introduce a minimal AppHost class and a placeholder ConfigStore. The existing app.js can be wrapped or lightly modified to use these new pieces without altering functionality yet (for instance, start by having AppHost simply call the old initialization, to ensure we haven't broken anything). This phase is about setting up the *scaffolding* for the refactor while still running the old code – **no major feature changes** here, just prep work.
2. **Introduce Unified Config System:** Implement the new **ConfigStore** and start moving configuration management to it[[44]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L106-L109). We will define a schema for config (using Zod or similar) covering all current adjustable parameters (camera settings, material properties, physics toggles, audio levels, etc.). The store will support change subscriptions. During this phase, we refactor parts of the code that read from the old conf object to instead use ConfigStore. We can do this gradually: for a while, both systems might coexist, but we keep them in sync. For example, when conf.x changes, we update ConfigStore and vice versa until the old usage is fully removed. By the end of this phase, the **global conf singleton should be eliminated or minimized**, and all modules should be able to get their settings via the new centralized store. This is a crucial step to untangle the implicit state and prepare for module isolation.
3. **Module Extraction (Incremental):** This is the core of the refactor – peeling apart the monolithic logic into the new modules. We will tackle this subsystem by subsystem, verifying functionality at each step:
4. **Stage Module:** Carve out all stage, lighting, and camera setup code into StageModule.ts[[45]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L108-L111). The existing app.js (or now AppHost) will instantiate StageModule instead of directly creating the scene/camera. We ensure that after this extraction, the scene still renders correctly (same lights, camera behavior, etc.), just that the code has moved into the module with a clean interface (e.g., StageModule.init(renderer) to set up the scene).
5. **Physics Module:** Migrate the MLS-MPM simulation code into PhysicsModule.ts[[45]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L108-L111). This includes particle state management, update loop, and hooking into the renderer for visualizing the particles. We encapsulate emitter definitions and physics tick timing inside this module. After this, physics updates should still occur as before, but triggered via AppHost calling physics.update(dt) each frame rather than from inside app.js. We also integrate the physics config (like material properties, spawn rates) with the ConfigStore so that changes in config (from UI) flow into the PhysicsModule cleanly.
6. **Material/Rendering Module:** (This might be done alongside Physics or Stage.) We create MaterialModule.ts to hold shader code and material creation logic[[46]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L63-L65). As we extract Stage and Physics, we relocate any shader definitions or material setup into this module. For example, if the project has custom shaders for the particles or floor, those go here. The Stage and Physics modules can then request materials from MaterialModule (or MaterialModule can register materials with Stage). The goal is to have a single place to tweak shaders and manage GPU buffers.
7. **Audio Module:** Extract all audio-related code (Web Audio context, analyzer setup, sound reactivity logic) into AudioModule.ts[[47]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L109-L112). After this, the main loop should call AudioModule.update() each frame to get audio analysis data. Audio controls move to ConfigStore as well. The AudioModule will also publish events (e.g. beat detected) to the EventHub, which the Physics or PostFX modules might subscribe to (linking music to visuals). We'll verify that audio-reactive features still function, but now decoupled (for instance, instead of physics directly reading from an audio global, it listens for an event).
8. **PostFX Module:** Refactor the post-processing pipeline into PostFxModule.ts[[48]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L110-L113). We will take all the code related to DOF, bloom, color correction, etc., and put it in this module, likely implementing a small **frame graph** or fixed sequence of passes. The StageModule will provide the main render target (scene rendered to a texture), and PostFxModule will produce the final composited frame. Controls for these effects (focus distance, bloom intensity) will be read from the ConfigStore inside PostFxModule. It's important to ensure that toggling post-processing on/off or changing an effect parameter reflects correctly (there was a noted bug where disabling postFX caused a white screen[[49]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/chats.md#L10-L13); with a clean module, these interactions should be more predictable).
9. **Camera (Lens) Module:** If not already fully covered by Stage or PostFX, we isolate any camera movement or lens-specific logic into a CameraModule (or incorporate it into StageModule with clear separation)[[50]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/ARCHITECTURE_EXECUTION_PLAN.md#L16-L19). This handles things like depth-of-field focus calculations, camera path animations, or user input for camera control. By doing this last, we ensure it integrates with both Stage (for camera rig) and PostFX (for lens effects). After extraction, camera updates (including focus changes for DOF) will be driven by either ConfigStore events or explicit calls from Physics (e.g., physics might provide a focal point that camera uses).

After each module extraction, we'll run the app and cross-check that behavior matches the pre-refactor state. The extraction can be done in the listed order to manage dependencies (Stage first because others rely on it, then Physics which might depend on Stage for coordinate space or materials, then Audio and PostFX which are more independent). 4. **UI Integration & Panel Refactor:** With the core modules in place, the UI (control panels) can be rebuilt to interface with them properly[[51]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L112-L115). In this phase, we refactor the tweaking panels to use the new ConfigStore and module APIs. Instead of directly manipulating globals or calling functions in app.js, the UI controls will dispatch actions to update ConfigStore values or invoke module methods through the AppHost. We'll create distinct panels (or sections) for each domain: e.g., **PostFX Panel** for camera lens, DOF, bloom parameters; **Physics Panel** for simulation parameters; **Audio Panel** for audio controls; **Rendering/Materials Panel** for things like particle color, material toggles, etc.[[20]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/chats.md#L7-L13). The UI module will likely use a modern framework (e.g. React components) but maintain the look of the current panel (possibly through styling or a gradual migration from the existing tweakpane). We ensure that all redundant or duplicate controls are cleaned up (for example, the earlier notes mentioned removing duplicate DOF controls from the wrong panel[[52]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/chats.md#L11-L14)). By the end of this step, the UI fully operates through our new pipeline – adjusting a slider updates the ConfigStore, which triggers the corresponding module to update, and the effect is seen in real-time. 5. **Cleanup & Optimization:** With all features moved into modules and the UI updated, we perform a thorough cleanup[[53]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L4-L5). This involves removing any leftover legacy code paths (for instance, delete the old conf.js and any now-unused functions in app.js). All hacky solutions (like polling loops or setInterval-based updates) are replaced with event-driven or timeline-driven updates[[53]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L4-L5). We ensure that each module properly **disposes** of resources when not needed (e.g., removing event listeners, releasing GPU buffers, stopping audio contexts on unload) to prevent memory leaks. We also audit the performance: run profile sessions to compare FPS before and after refactor, and optimize any identified bottlenecks. Since the new architecture separates concerns, we can easily toggle modules to isolate performance issues. For example, we might find that with the new modular rendering, the particle draw call could be optimized or the postFX pass could use a lower resolution target for performance – such tweaks can be done now systematically. Any redundant computations (like multiple modules computing the same value) can be unified. This phase is about solidifying the architecture so it’s not only clean but also efficient. 6. **Testing & Verification:** In parallel with the latter phases, but especially once the refactor is largely complete, we will rigorously test the system[[39]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L118-L121). Write unit tests for critical pieces (does the ConfigStore correctly propagate changes? Are events dispatched and received as expected? Does the physics solver output remain consistent?). Also, run integration tests and visual regression tests on various scenarios: with postFX on/off, under heavy physics load, different audio inputs, etc. The goal is to catch any regressions introduced by the refactor. Because we maintained the application in a working state through each phase, verification is easier – at each stage we can compare the module-refactored output to the known-good output from before. Now we formalize that with tests. We will also test edge cases (like toggling modules at runtime, or loading an old preset into the new config system to ensure backward compatibility or at least a smooth migration). 7. **Documentation & Handover:** Finally, update all documentation to reflect the new structure[[54]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L120-L124). The README will get an architecture overview section explaining the module system and how to run or build the project with the new scripts. We will also prepare developer docs for each module (e.g. how to add a new post-processing pass via PostFxModule, or how to create a new preset). If this project is to be continued or open-sourced, providing clear documentation is key to onboarding others to the now much cleaner codebase. We can include diagrams of the new architecture (showing how modules connect via the ConfigStore and EventHub) to give contributors a high-level understanding. This documentation phase ensures the longevity of our improvements by making the design clear and explicit (a form of "rubric" for how future additions should be structured).

Throughout these phases, we will maintain a branch or forks such that the application remains usable. The **phased approach** mitigates risk by not rewriting everything at once: after each phase, we'll have a working intermediate state that can be tested and even shipped if needed[[55]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/ARCHITECTURE_EXECUTION_PLAN.md#L70-L73). For instance, after Phase 3, we might choose to release an update that already has better performance or some new minor features enabled by the partial refactor, while Phase 4 and beyond are in progress.

By following this plan, we upgrade the project to the target modular architecture methodically. Each step brings us closer to the goal while allowing verification that nothing critical is broken.

## Conclusion and Future Outlook

By redesigning the project with a modular, ESM-based architecture and rigorous pipeline, we expect **significant benefits**: the code will be easier to maintain, new features can be developed in isolation, and components can be reused or swapped with minimal effort. The final refactored system will have **clear boundaries** between subsystems, a single source of truth for configuration, and well-defined communication channels, eliminating the confusion and fragility of the current setup.

In addition to solving current problems, this refactor opens the door to future enhancements. For example, with modules decoupled and registered via a central registry, we could allow **feature toggling or plugins**, where new modules (perhaps from third-party contributors or other libraries) can be added to provide additional effects or input methods[[56]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L125-L128). The application could be built as smaller “editions” – e.g., a build that includes only the Stage and PhysicsModule for a lightweight physics sandbox, or a build that focuses on audio visualization – thanks to the modular design and tree-shaking[[57]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L125-L130). We could even explore **server-side rendering** or offline rendering of frames (using the headless mode) to generate high-quality visuals for marketing or testing[[58]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L127-L130). Because we plan to use standardized interfaces and strong typing, integrating with external tools or engines (or upgrading to future WebGPU features) will be smoother.

Ultimately, this deep re-architecture, guided by “maximal thinking” and careful planning, will transform the project from a tightly bound prototype into a **robust, scalable platform**. The pipeline will be cohesive and modern: fully TypeScript, modular, with continuous integration ensuring quality. Developers and artists using the project will have confidence in its stability and flexibility, and we will have a strong foundation to build upon for subsequent versions. All scripts and components will be cleaner, redundant-free, and **polished**, reflecting the careful thought (“deep think”) and structured approach (“rubrik”) that went into this redesign.

**Sources:**

* Aurora v1 Refactor Proposal and Execution Plan[[2]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L6-L11)[[4]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L14-L19)[[59]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L59-L67)[[39]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L118-L121) (project internal documentation outlining current issues, goals, and planned architecture changes)
* Project Codebase (AURORA v1 repository) – file structure and module definitions[[12]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L23-L31)[[13]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L31-L39), config and event system design[[28]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L66-L67), tooling configuration[[60]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L70-L78)[[35]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L78-L84), and phase-wise migration strategy[[61]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L102-L111)[[62]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L112-L120).
* *Credits:* The MLS-MPM physics implementation is adapted from **WebGPU-Ocean** by matsuoka-601[[42]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/CREDITS.md#L3-L6), and visuals draw inspiration from Refik Anadol’s work[[63]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/CREDITS.md#L2-L5), informing our approach to simulation and rendering in the redesign.

[[1]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L6-L9) [[2]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L6-L11) [[3]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L9-L11) [[4]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L14-L19) [[5]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L4-L5) [[6]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L18-L19) [[7]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L7-L9) [[8]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L9-L10) [[9]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L10-L11) [[10]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L14-L17) [[11]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L16-L19) [[12]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L23-L31) [[13]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L31-L39) [[14]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L23-L30) [[15]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L32-L36) [[16]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L33-L38) [[17]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L34-L37) [[18]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L35-L37) [[19]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L35-L38) [[21]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L60-L64) [[22]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L60-L63) [[23]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L38-L41) [[24]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L42-L45) [[25]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L46-L49) [[26]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L50-L54) [[27]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L59-L61) [[28]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L66-L67) [[29]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L2-L5) [[30]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L96-L101) [[31]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L70-L73) [[32]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L70-L74) [[33]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L71-L74) [[34]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L72-L75) [[35]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L78-L84) [[36]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L78-L81) [[37]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L78-L82) [[38]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L80-L82) [[39]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L118-L121) [[40]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L86-L89) [[41]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L91-L93) [[43]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L103-L106) [[44]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L106-L109) [[45]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L108-L111) [[46]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L63-L65) [[47]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L109-L112) [[48]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L110-L113) [[51]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L112-L115) [[53]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L4-L5) [[54]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L120-L124) [[56]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L125-L128) [[57]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L125-L130) [[58]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L127-L130) [[59]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L59-L67) [[60]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L70-L78) [[61]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L102-L111) [[62]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md#L112-L120) REARCHITECTURE\_PROPOSAL.md

<https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/REARCHITECTURE_PROPOSAL.md>

[[20]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/chats.md#L7-L13) [[49]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/chats.md#L10-L13) [[52]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/chats.md#L11-L14) chats.md

<https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/chats.md>

[[42]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/CREDITS.md#L3-L6) [[63]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/CREDITS.md#L2-L5) CREDITS.md

<https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/CREDITS.md>

[[50]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/ARCHITECTURE_EXECUTION_PLAN.md#L16-L19) [[55]](https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/ARCHITECTURE_EXECUTION_PLAN.md#L70-L73) ARCHITECTURE\_EXECUTION\_PLAN.md

<https://github.com/artinkavousi/AURORAv1/blob/5fa44c2df8de98b669b119ae0f2a91d09f00e9cc/ARCHITECTURE_EXECUTION_PLAN.md>