Aerune System Architecture Overview (English)

1. Basic Application Structure

•Platform: Electron (desktop app for Windows and Mac)
•Frontend: Vanilla JavaScript (no frameworks such as React or Vue) + HTML/CSS
•API Client: @atproto/api (official Bluesky package)
•Architectural policy: Prioritizes lightweight performance and speed. The design reduces rendering overhead by directly manipulating the DOM and heavily using Event Delegation and DocumentFragment.
•Why not React/Vue? → To keep it lightweight
•Why Event Delegation? → To save memory

2. Role of Each JavaScript File (Module Structure)

Communication / Data Fetch Layer
•bsky-api.js (API wrapper)
•A class that wraps BskyAgent from @atproto/api for easier use.
•All communication with Bluesky servers goes through this module (login, fetching posts, sending likes/reposts, fetching notifications, etc.).
•view-loader.js (data building / view instructions)
•A class that organizes raw API responses into "how they should be displayed on screen".
•Responsible for threading the timeline (tracing parent posts and building a conversation tree), generating notification lists, and loading profile views.

Rendering / UI Construction Layer
•post-renderer.js (post creation / rendering)
•A "craftsman" module that assembles a single post as HTML (DOM elements).
•Handles image embedding, expanding quoted posts, NSFW blur detection, and calculating/caching relative time (e.g., "X minutes ago").
•utils.js (utility toolkit)
•A backstage utility file that contains generic helpers such as linkification (linkify), rich-text conversion (renderRichText), and local image compression (compressImage).

User Interaction / Action Layer
•actions.js (actions on posts)
•A class responsible for API-side operations after user actions such as Like, Repost, Bookmark, Follow, and Block.
•navigation.js (navigation / history management)
•A router for the Back button that stores the user's navigation history (stack) and the scroll position at each point.

Controller
•renderer.js (main entry point)
•Runs first on app launch and orchestrates all other modules.
•Manages global state (logged-in account, composing text, selected images, etc.).
•Handles centralized click monitoring (event delegation), view switching (switchView), and posting (sendPost).

Static Data

- constants.js (constants / translations / icons)
- A “warehouse” that caches i18n text and SVG icon HTML strings to prevent the codebase from bloating.

3. Core Mechanisms (Key Design Points)

1.Lightweight Design via Event Delegation

If you attach click events (addEventListener) to every Like button or every image, simply scrolling the timeline will consume a lot of memory.
To avoid this, Aerune watches clicks in a single place (document). When the clicked element has an attribute like data-act=“like”, Aerune triggers the corresponding action. This is a very lightweight approach (implemented in renderer.js: installDelegates).

2.Optimistic UI Updates

When the user presses Like, Repost, or the trash button, Aerune updates the UI immediately (e.g., changing button color or hiding a post) without waiting for the server to respond “success”.
This keeps the app responsive regardless of API latency and can feel snappier than the browser version (implemented in renderer.js inside switch(act)).

3.Threaded Timeline

An algorithm that takes scattered replies in the timeline, traces upward to the API fetch limit, rebuilds them into an ordered structure (“parent → child → grandchild”), and then renders the result.
In addition, a more advanced filter is applied in view-loader.js (fetchTimeline): “drop unrelated conversations from non-followed users” while “keeping replies directed to you”.

Module Relationship (Overview)

```
[renderer.js (Controller)]
↓
 ├── [bsky-api.js]    ← server communication
 ├── [view-loader.js]  ← data shaping
 ├── [post-renderer.js]← DOM building
 ├── [actions.js]     ← user actions
 └── [navigation.js]   ← navigation
```