Reflex LiveKit Audio Chat — Architecture & Design

1) Project Summary

Goal: a minimal real-time **audio room** experience in the browser.

Core idea:

- **Python (Reflex)** renders the UI and manages app state.
- A **browser-side LiveKit JS client** handles WebRTC and room events.
- A small “bridge” pushes LiveKit events (JSON) back into Reflex state via a **hidden input**.

—

2) Repo Map

```
reflex_livekit_audio_chat-main/
├── README.md
├── plan.md
├── pyproject.toml
├── rxconfig.py
├── .env.template
├── assets/
│   ├── favicon.ico
│   └── placeholder.svg
├── docs/images/...
└── reflex_livekit_audio_chat/
    ├── __init__.py
    ├── reflex_livekit_audio_chat.py      # Pages + UI composition
    ├── livekit_bridge.py                 # Reflex State <-> Browser LiveKit JS bridge
    └── states/
        ├── __init__.py
        └── settings_state.py             # Admin-gated settings page + .env persistence
```

—

3) Key Modules & Responsibilities

`reflex_livekit_audio_chat/reflex_livekit_audio_chat.py`

- Defines pages:

- `/` = lobby + room (conditional)
- `/settings` = configuration page (admin gated)
- Mounts LiveKit JS “head scripts” via `LIVEKIT_UI.head_components()`
- Adds the hidden bridge input via `LIVEKIT_UI.bridge_input()`

`reflex_livekit_audio_chat/livekit_bridge.py`

- `LiveKitBridgeState` (Reflex state)
- Fields: `room_name`, `username`, `token`, `participants`, `connection_status`, `is_muted`, `error_message`, `loading`, `is_connected`
- Events:
  - `join_room(form_data)` → mints token and calls JS `connect()`
  - `leave_room()` → calls JS `disconnect()` and clears state
  - `toggle_mute()` → calls JS `setMicrophone(enabled)`
  - `handle_js_message(json)` → updates state from browser events
  - `_LiveKitUI`
- Provides:
  - `head_components()` (loads LiveKit client + custom JS runtime)
  - `bridge_input()` (hidden input to deliver JSON into Reflex)
  - `volume_bar(identity, width)` (DOM element updated by JS volume visualizer)

`reflex_livekit_audio_chat/states/settings_state.py`

- `SettingsState`
- Admin gate using `ADMIN_PASSCODE`
- Loads/saves `LIVEKIT_URL`, `LIVEKIT_API_KEY`, `LIVEKIT_API_SECRET` into `.env`

—

4) Configuration & Secrets

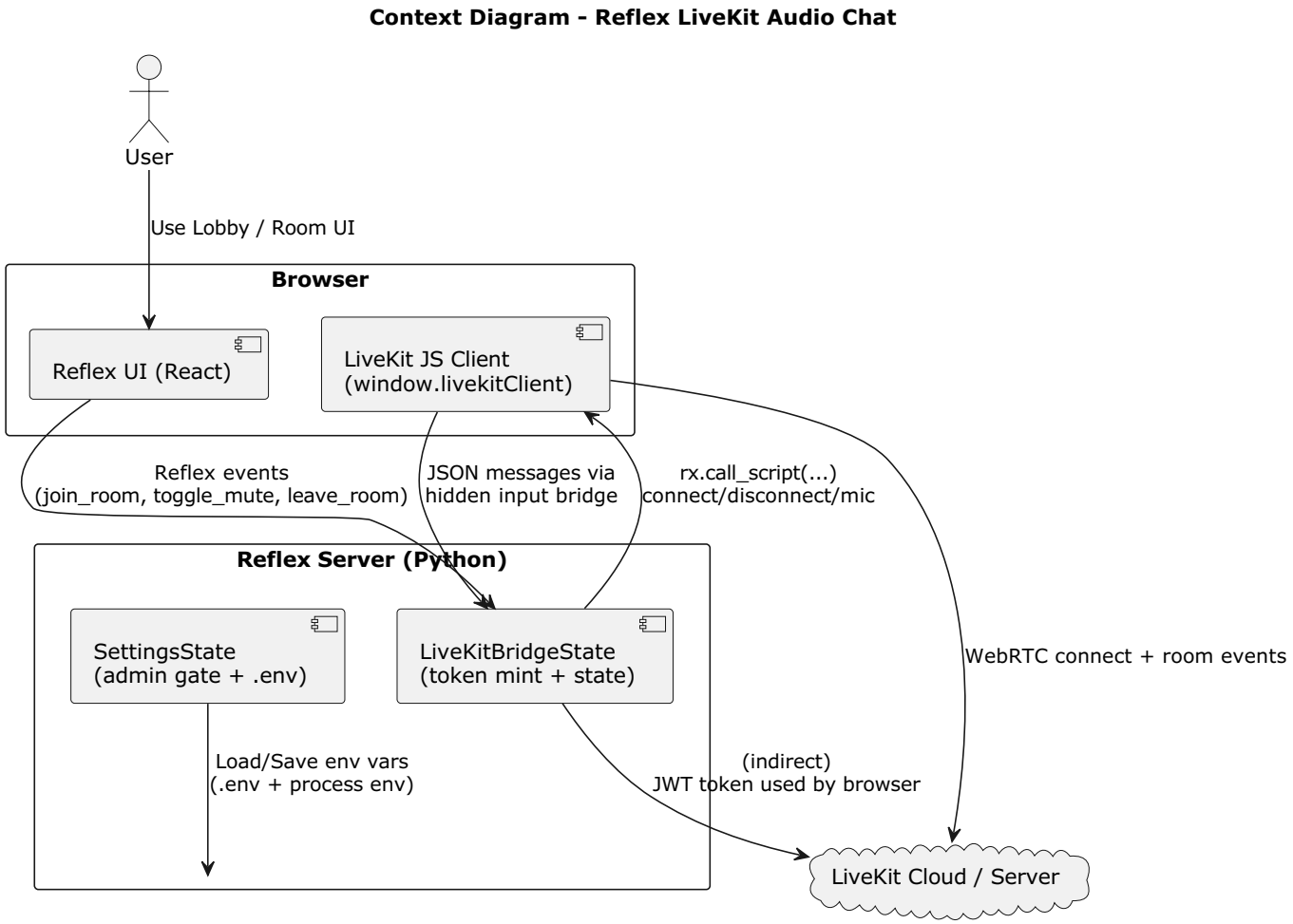
`.env.template` defines:

- `LIVEKIT_URL` (wss URL for browser connection)
- `LIVEKIT_API_KEY` + `LIVEKIT_API_SECRET` (server-side token minting)
- `ADMIN_PASSCODE` (unlock `/settings`)

Security note: never commit real `LIVEKIT_API_SECRET` to a public repo.

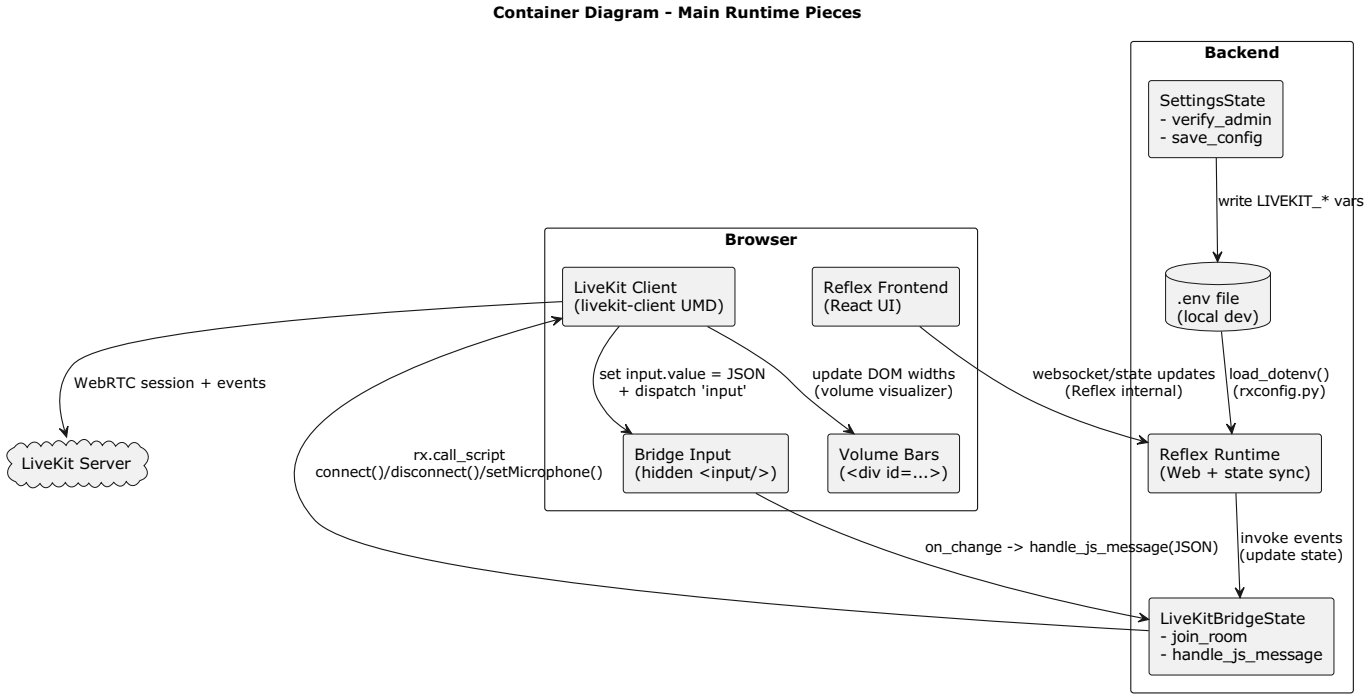
—

5) High-Level Architecture (Context)



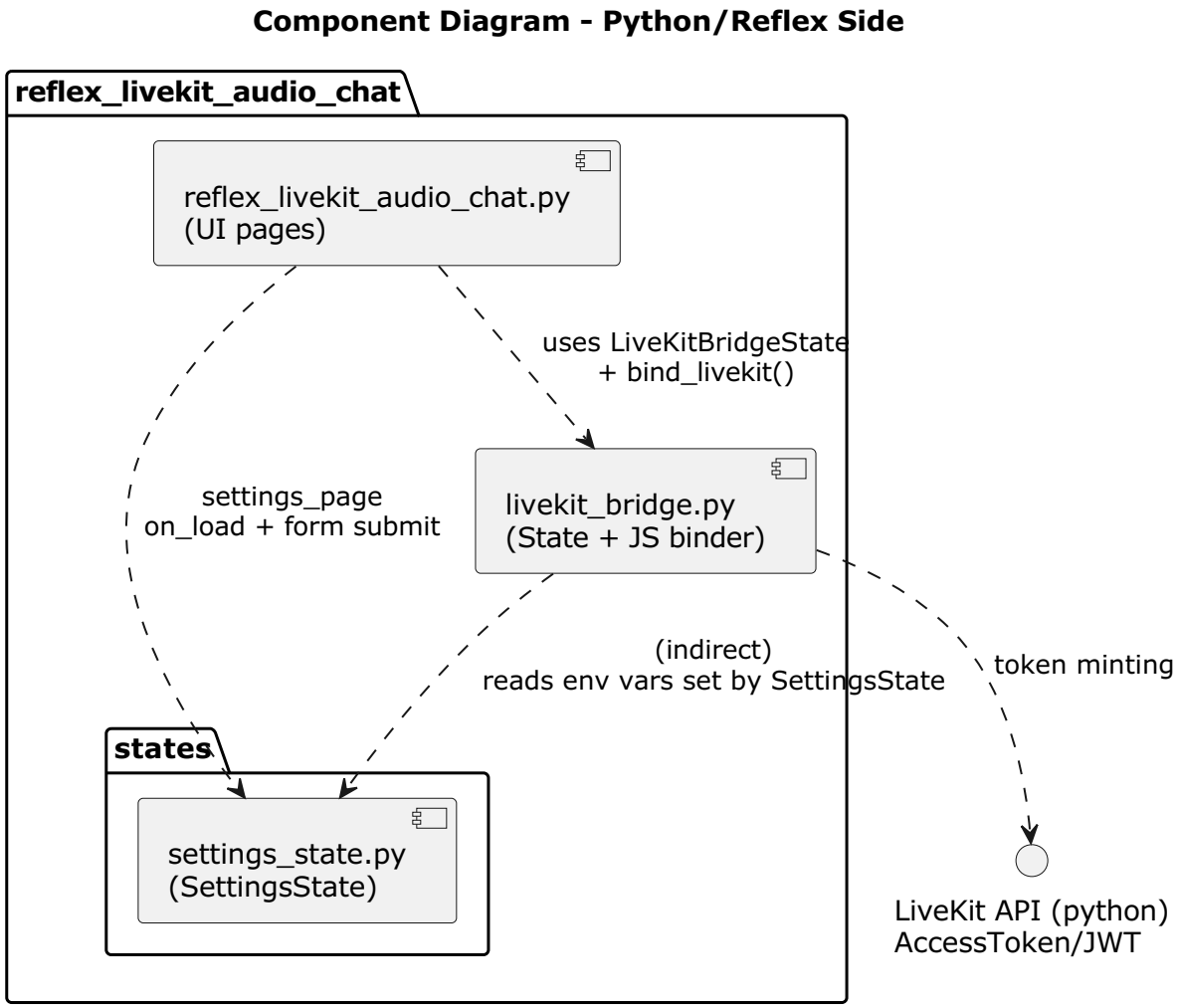
—

6) Container-Level View (C4-ish without external includes)



—

7) Component Diagram (Server-Side)



—

8) Data Model (What the UI Renders)

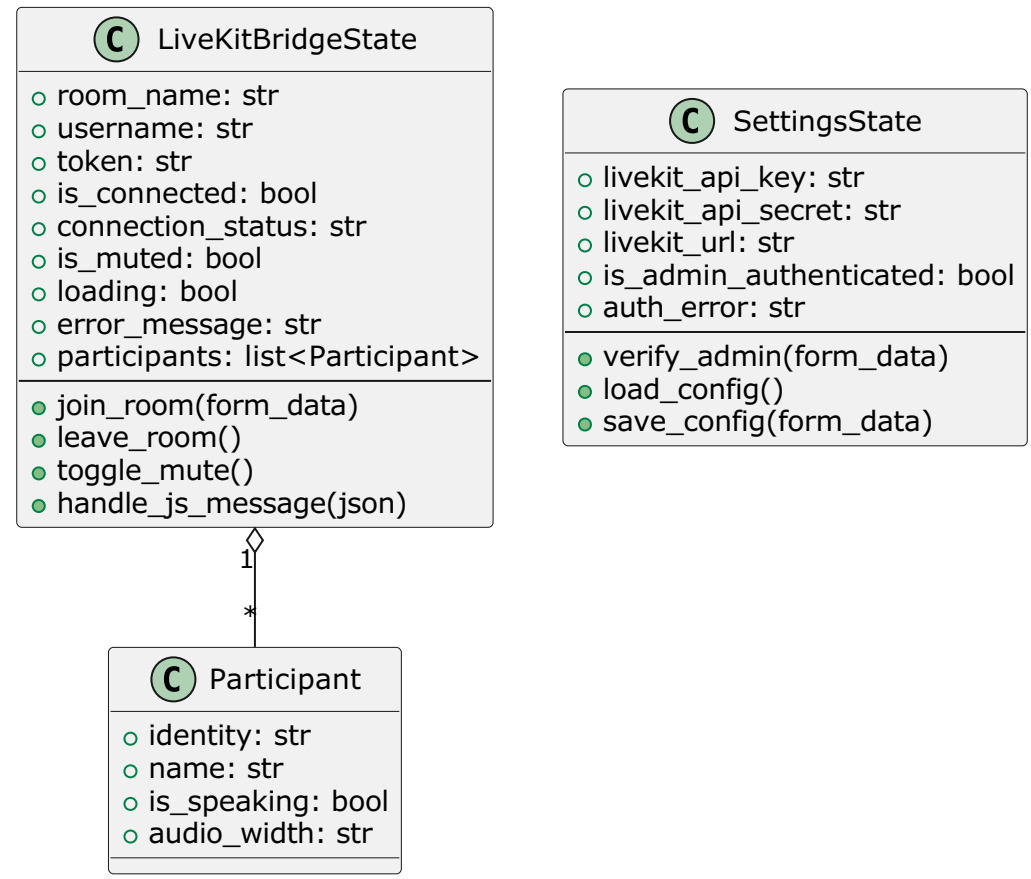
`LiveKitBridgeState.participants`

A list of dict-like objects used to render participant cards. Typical fields used by UI:

- `identity` (unique id / username)
- `name` (display name)
- `is_speaking` (boolean)
- `audio_width` (string like `"0%"` … `"100%"`, used to render volume bar)
- `is_local` (optional, if you mark local participant)
- `is_muted` (optional if you include it in JS payload)

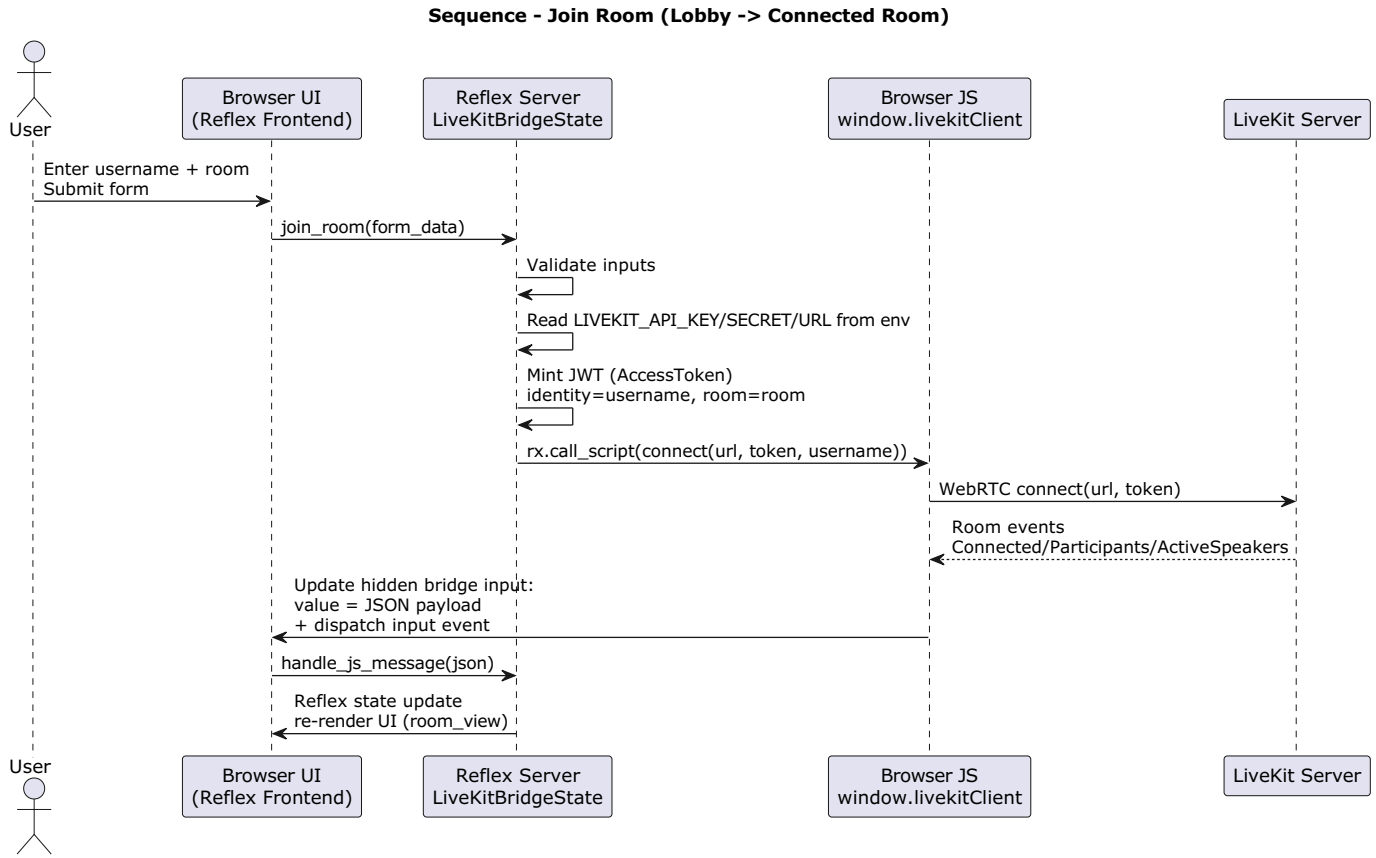
You can document it as:

Class Diagram - State Shapes (Simplified)



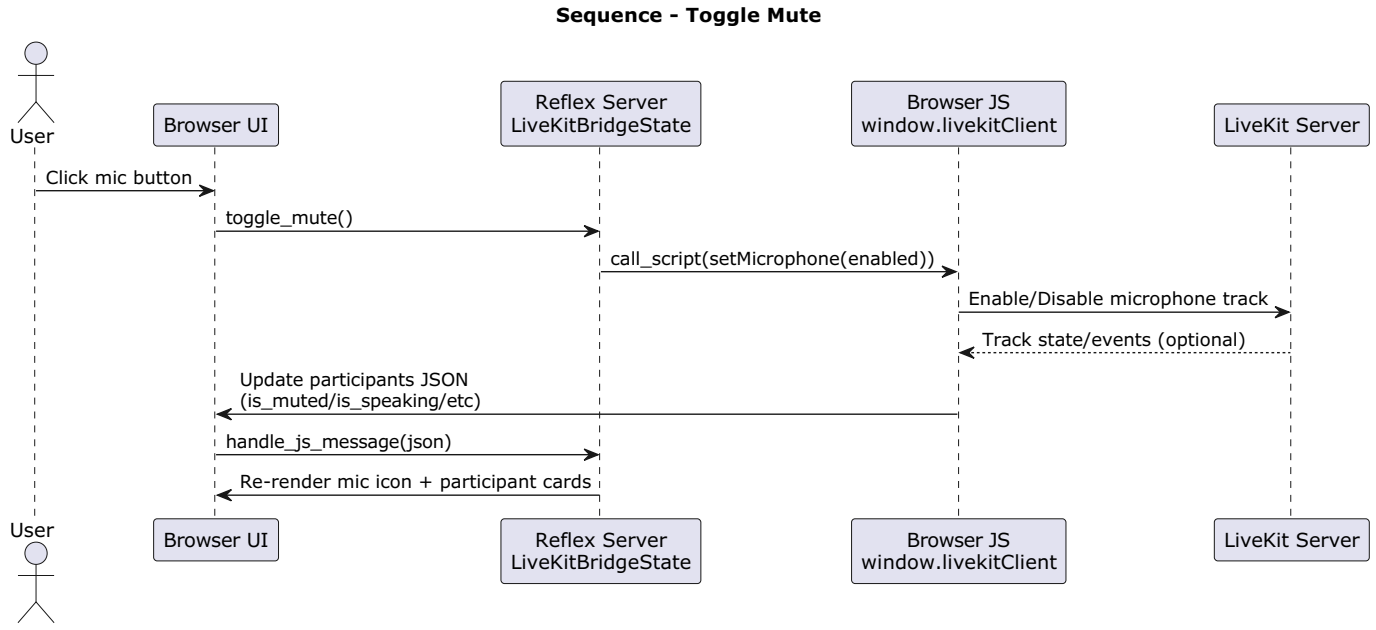
—

9) Main User Flow (Join Room) — Sequence Diagram



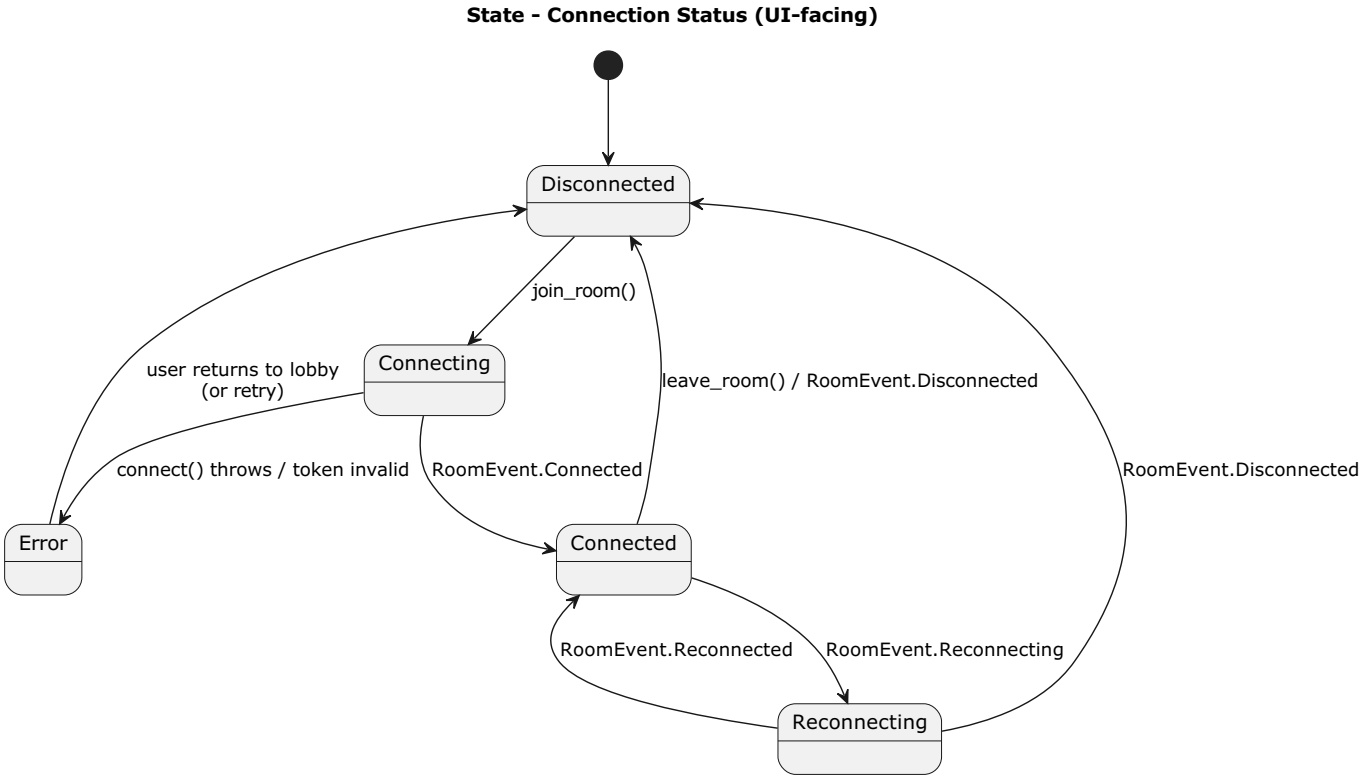
—

10) Mute / Unmute Flow — Sequence Diagram



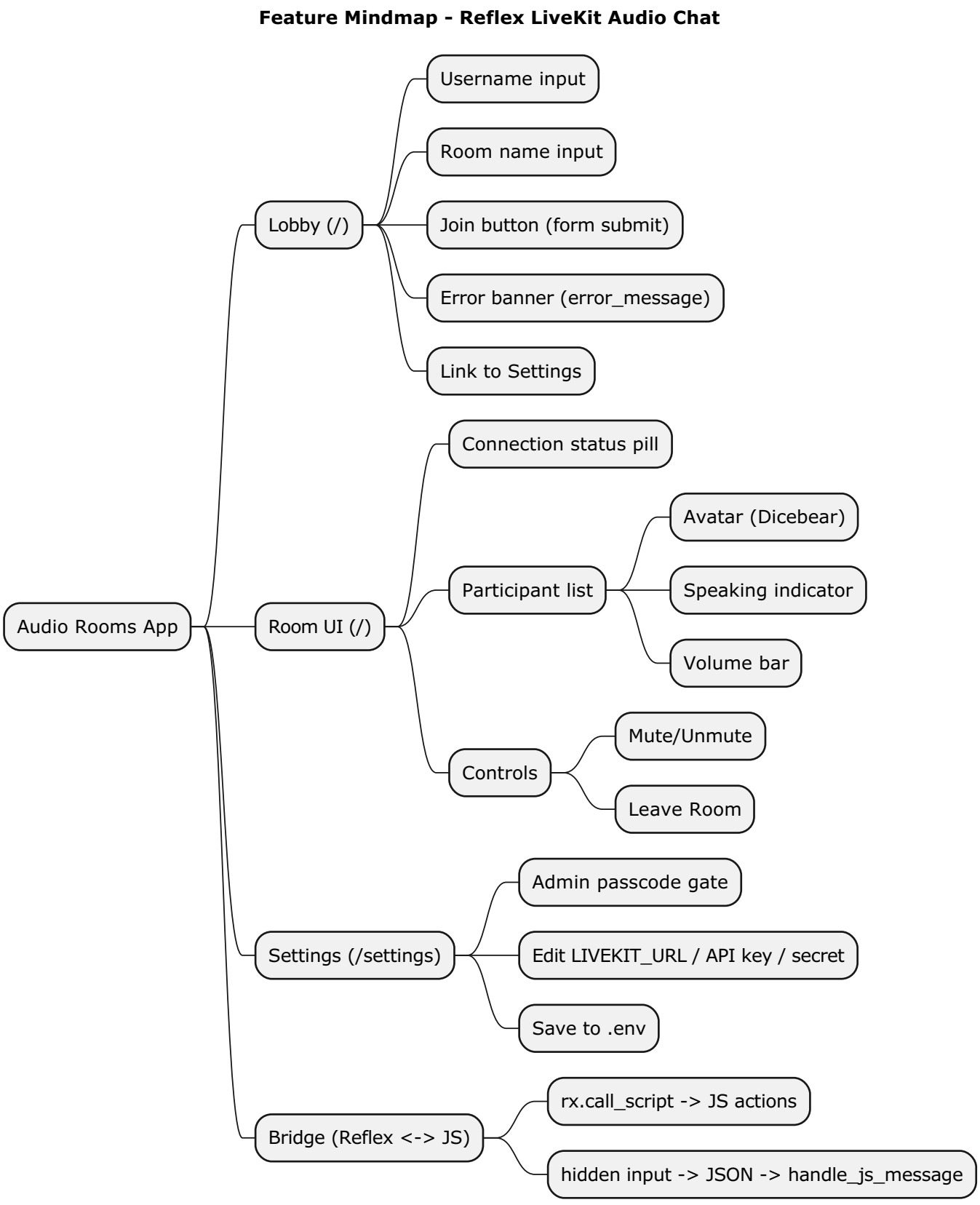
—

11) Connection Lifecycle — State Machine



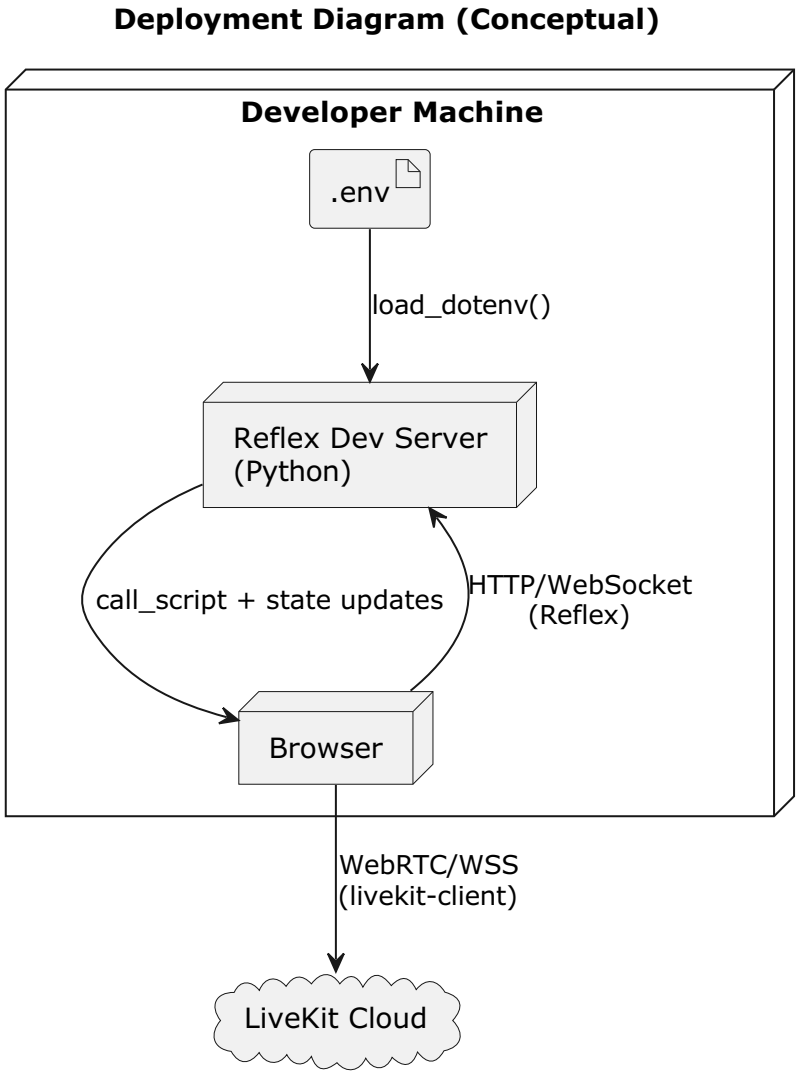
—

12) Feature Breakdown — Mindmap



—

13) Deployment View (Local Dev vs Production)



—

14) Notes / Practical Design Choices

- **Token minting happens on the server** (`join_room`), using `LIVEKIT_API_SECRET`.
- **WebRTC happens in the browser** (LiveKit JS client), because the mic/device permissions and media tracks are browser-native.
- The **bridge** uses a reliable "DOM → Reflex event" method:
- JS writes JSON into a hidden input's `.value`
- JS dispatches an `input` event
- Reflex `on_change` triggers `handle_js_message(json)` and updates state

—