Relack Project Design and Architecture Overview

> Relack is a self-hosted real-time chat system implemented with **Reflex (Python)**. It supports **Guest** and **Google OAuth** sign-in, and provides an **Admin Dashboard** for data maintenance, permission toggles, and export/import.

1) Project Structure (Repo Map)

```
relack-main/
├── relack/
│   ├── relack.py                # Reflex app entry: registers pages
│   ├── models.py                # Pydantic data models
│   ├── pages/
│   │   ├── index.py             # Home: Auth + Chat Dashboard
│   │   ├── profile.py           # /profile/[username]
│   │   └── admin.py             # /admin-dashboard
│   ├── components/
│   │   ├── auth_views.py        # Guest / Google login UI
│   │   ├── chat_views.py        # Sidebar / ChatArea / UsersPanel, etc.
│   │   ├── navbar.py            # Top navigation
│   │   └── profile_views.py     # Profile UI
│   └── states/
│       ├── shared_state.py      # GlobalLobbyState / RoomState / TabSessionState (core)
│       ├── auth_state.py        # AuthState (GoogleAuthState + Guest)
│       ├── admin_state.py       # AdminState (PASSCODE)
│       ├── permission_state.py  # PermissionState (permission toggle UI)
│       └── profile_state.py     # ProfileState (view/edit)
├── rxconfig.py                  # Reflex config + Tailwind plugin + sitemap
├── .env.template                # GOOGLE_CLIENT_ID / SECRET / ADMIN_PASSCODE
└── testcases/                   # Playwright e2e tests (guest/google/admin)
```

—

2) Technology and Execution Model (Key Concepts)

2.1 Tech Stack

- **Backend / Fullstack**: Reflex (Python) `reflex>=0.8.23`
- **UI styling**: Tailwind V3 plugin (`rx.plugins.TailwindV3Plugin()`)
- **Login**: `reflex-google-auth` + Google ID token verification
- **E2E tests**: Playwright

2.2 Where is the data stored?

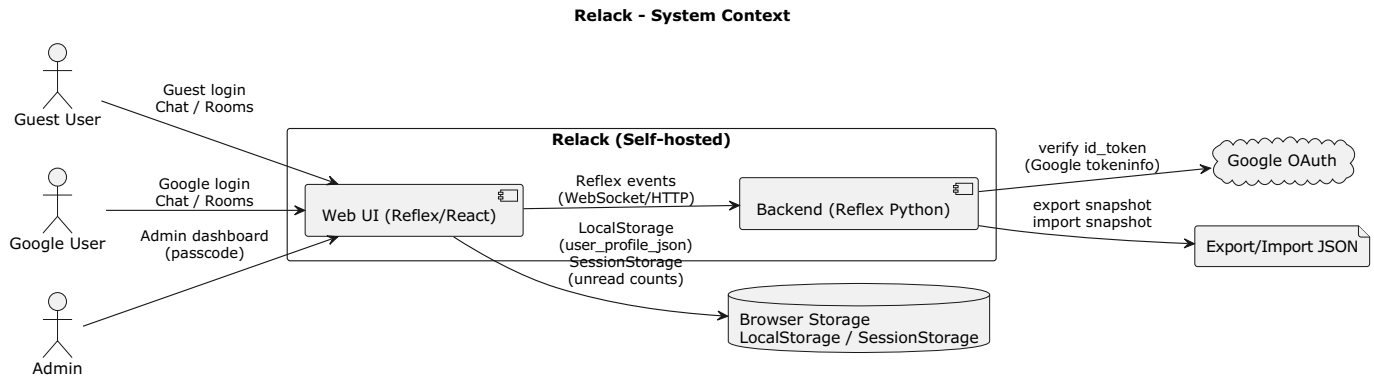
In the current version, the system is primarily **state-driven** and **in-memory oriented**:

- **GlobalLobbyState (rx.SharedState)**: globally shared site-wide (rooms, profiles, messages_by_room, permissions)
- **RoomState (rx.SharedState)**: one shared instance per room (token like `room-{slug}`), tracks presence and that room's messages
- **TabSessionState (rx.State)**: stored in **SessionStorage** (per tab) for unread counts and last room
- **AuthState (GoogleAuthState)**: uses **LocalStorage** to store `user_profile_json`

> Therefore:
> **if the server restarts, the in-memory content of GlobalLobbyState/RoomState will be lost**. However, you can back up and restore via the Admin **Export/Import JSON**.

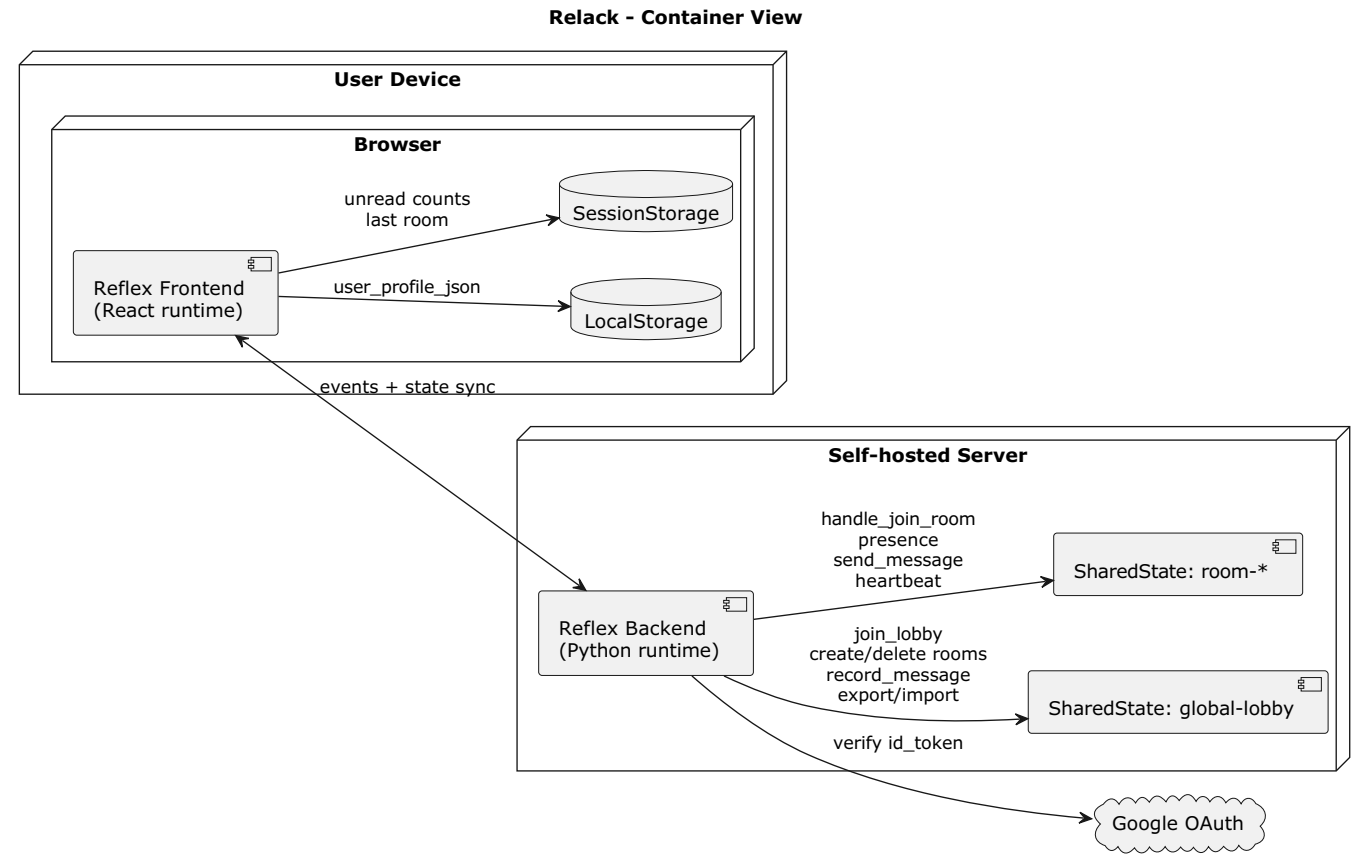
—

3) Context Diagram (System Context)



—

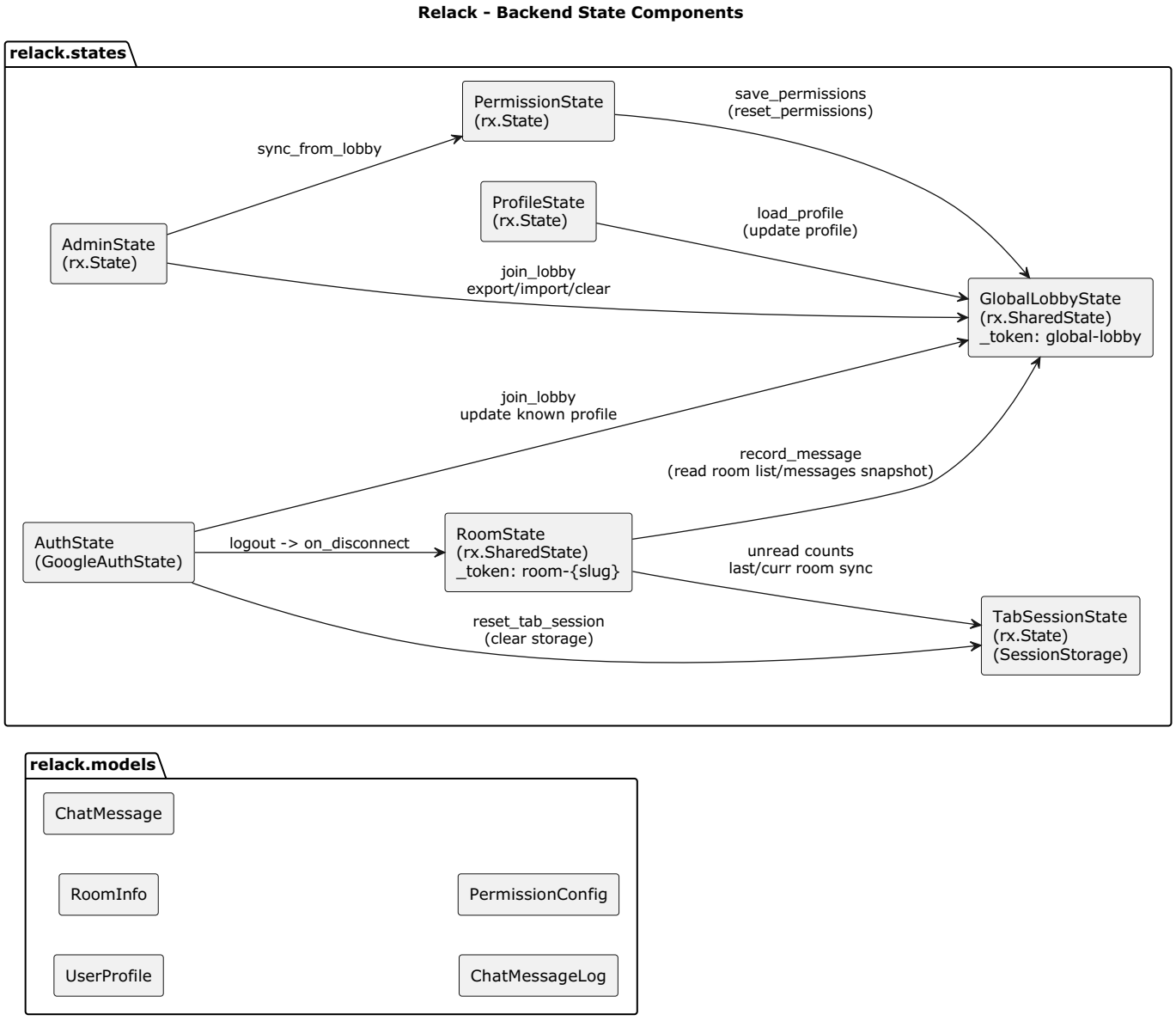
4) Container Diagram (Frontend/Backend Containers and Boundaries)



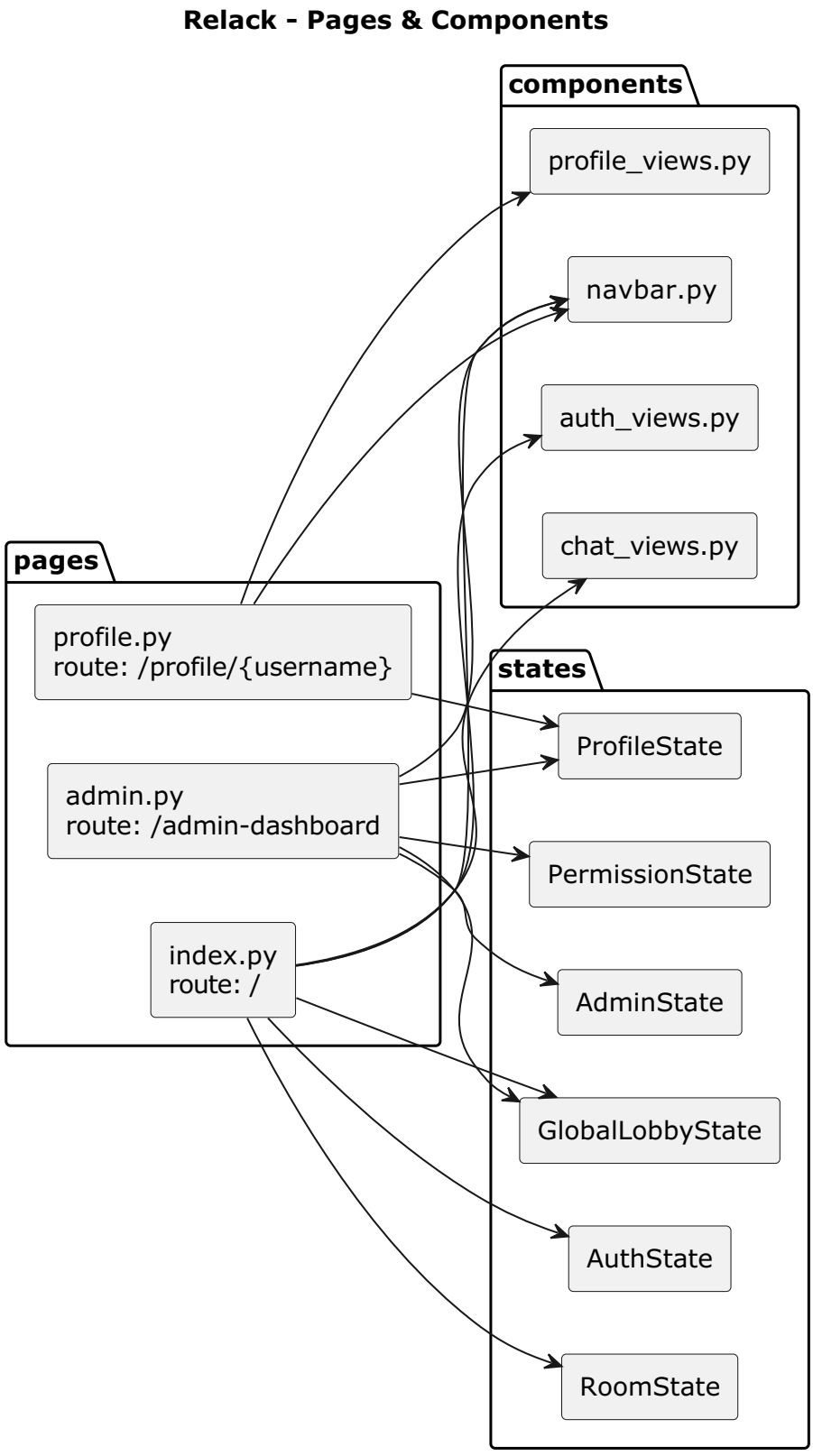
—

5) Component Diagram (Module / State Decomposition)

5.1 Backend / State Components

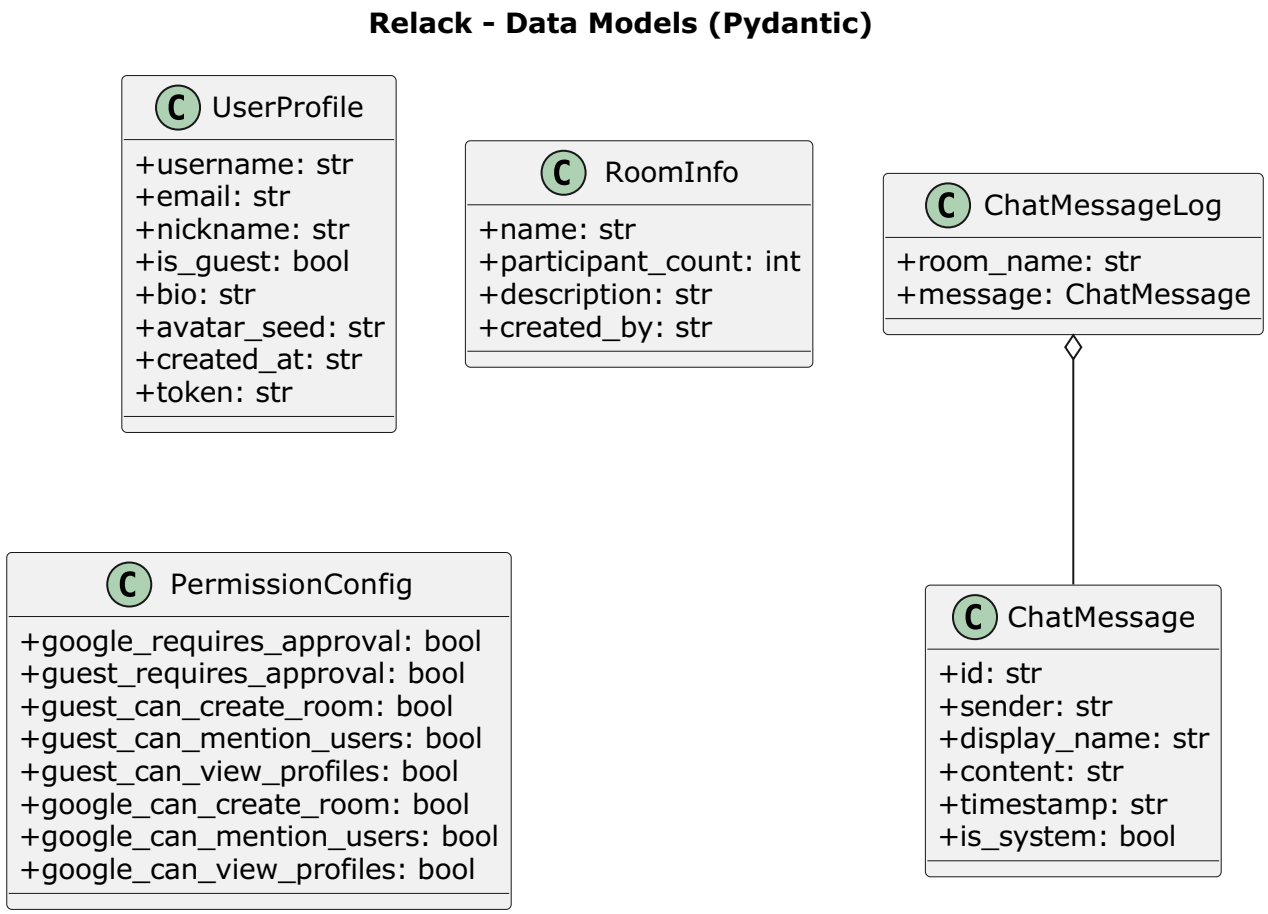


5.2 UI / Pages / Components (Who uses what)



—

6) Data Model (Pydantic models)



—

7) Core State Internal Data Structures (What you should master)

7.1 GlobalLobbyState (token: `"global-lobby"`)

Purpose: the global lobby (room directory, site-wide profiles, messages_by_room, permissions)

- `_rooms: dict[str, RoomInfo]`
- `_known_profiles: dict[str, UserProfile]`
- `_user_locations: dict[str, str]` (client_token -> room_name)
- `_messages_by_room: dict[str, list[ChatMessage]]`
- `_permissions: PermissionConfig`
- `export_payload / import_payload`

7.2 RoomState (token: `"room-{slug}"`)

Purpose: per-room shared state + presence (who's online) + message list + snapshots required for unread calculation

- `_active_users: dict[str, str]` (client_token -> username)
- `_active_user_profiles: dict[str, UserProfile]`
- `_active_user_last_seen: dict[str, float]`
- `_messages: list[ChatMessage]`
- `_current_room_by_client: dict[str, str]`
- `_message_counts_by_room: dict[str, int]` (used to compute unread)
- `STALE_WINDOW_SECONDS = 180` (heartbeat timeout treated as offline)
- `current_message`, `is_sidebar_open`, `is_user_list_open`

7.3 TabSessionState (SessionStorage)

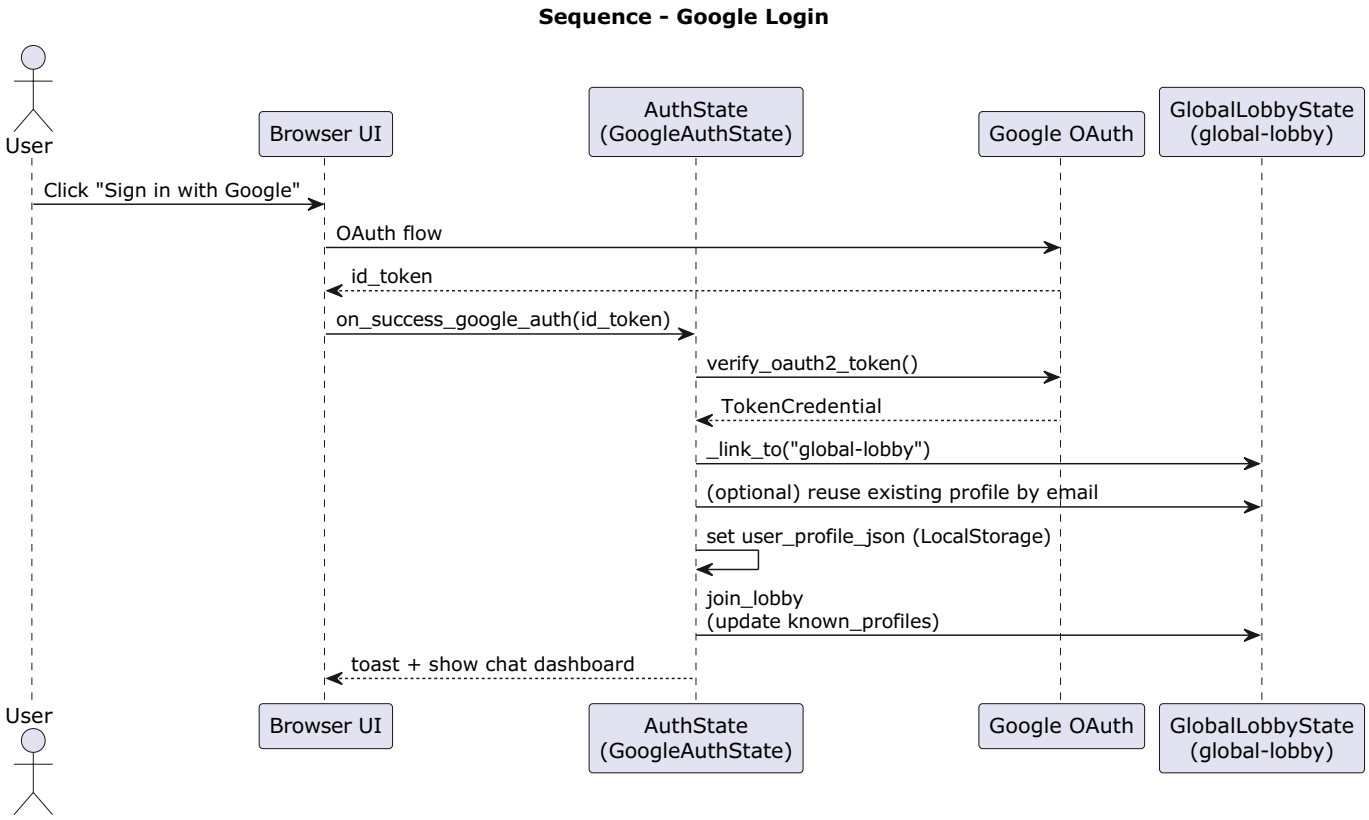
Purpose: each tab stores its own unread counts and last room, so badges don't disappear after reload

- `all_room_counts_json` (total message counts per room)
- `all_room_read_counts_json` (how many messages have been read per room)
- `last_room_name / curr_room_name`

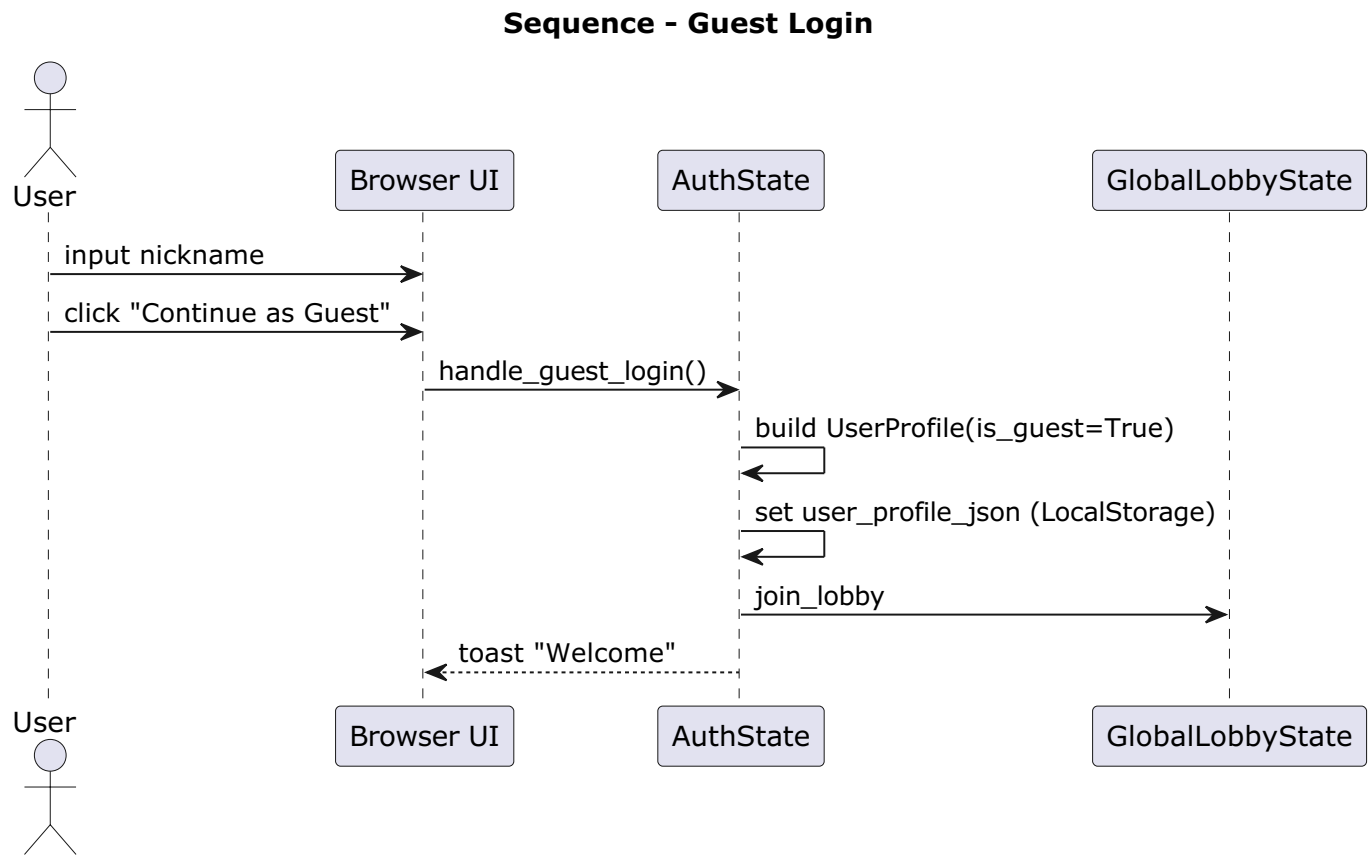
—

8) Sequence Diagrams (Most critical interaction loops)

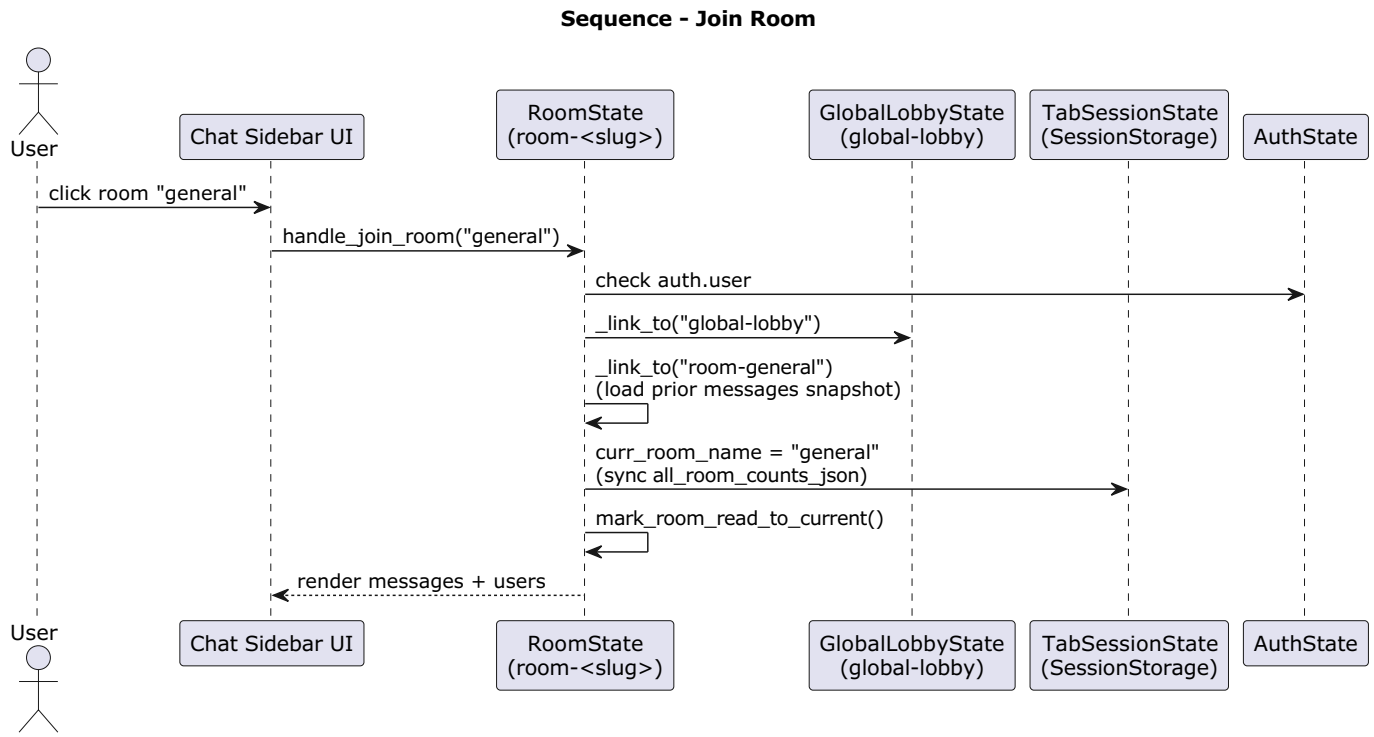
8.1 Google Login (on_success_google_auth)



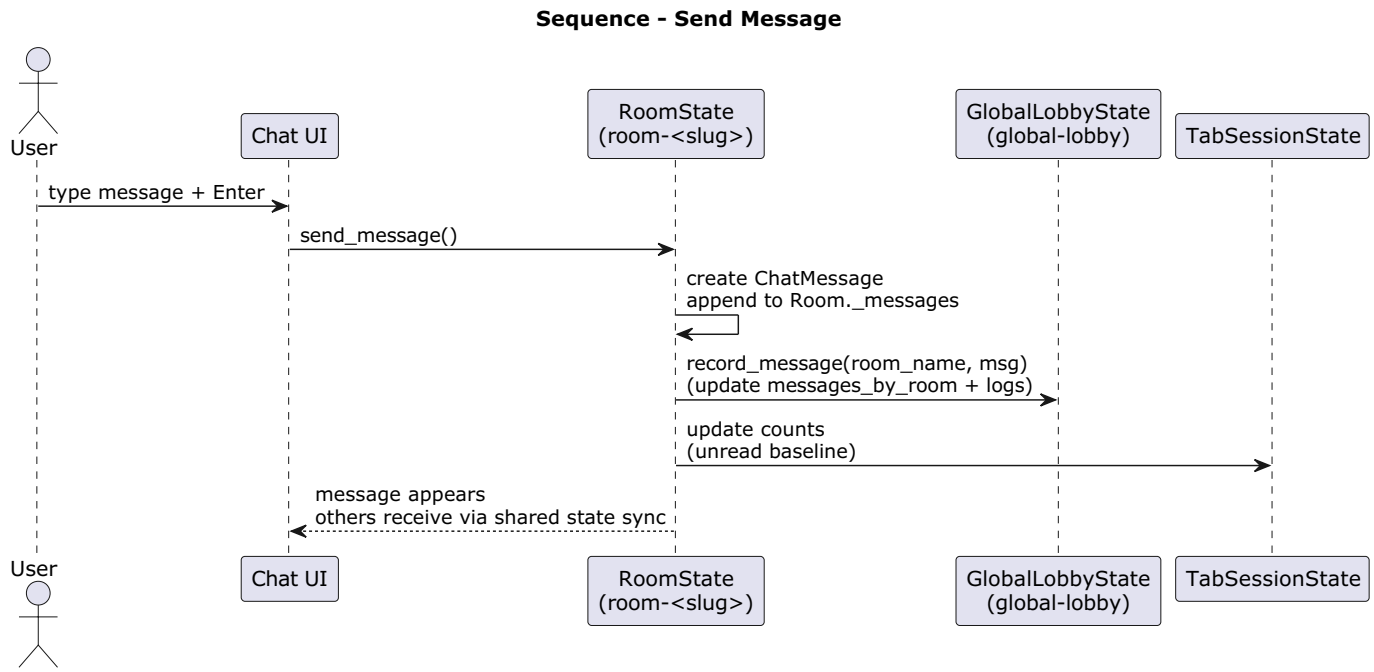
8.2 Guest Login (handle_guest_login)



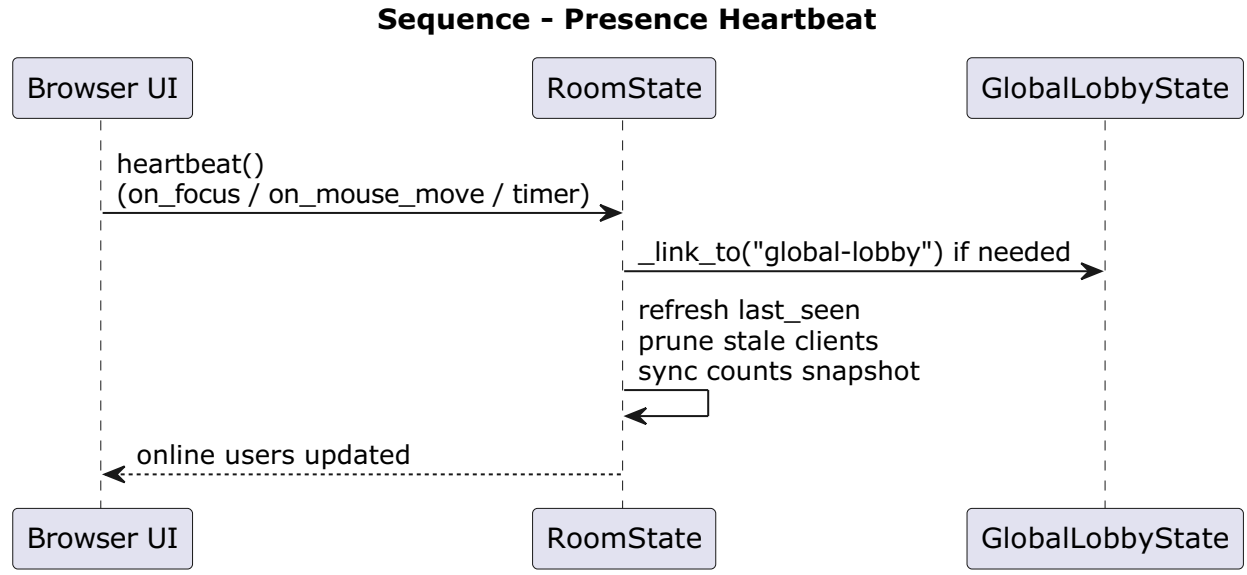
8.3 Join Room (RoomState.handle_join_room)



8.4 Send Message (RoomState.send_message -> GlobalLobbyState.record_message)

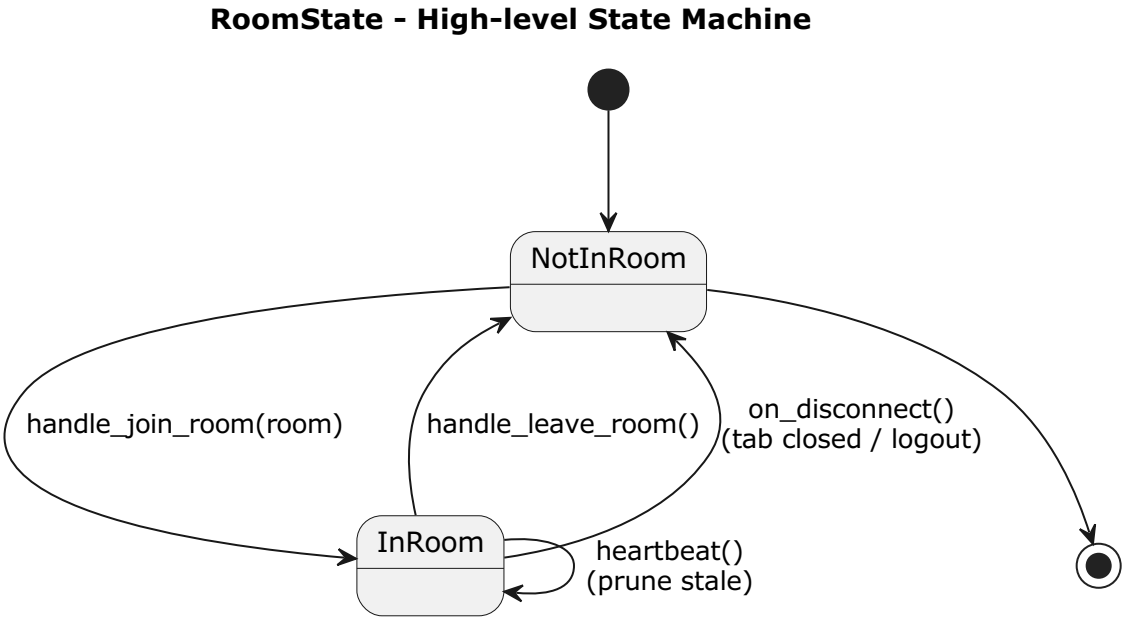


8.5 Heartbeat / Presence (RoomState.heartbeat)



—

9) State Diagram (Room state and disconnect cleanup)

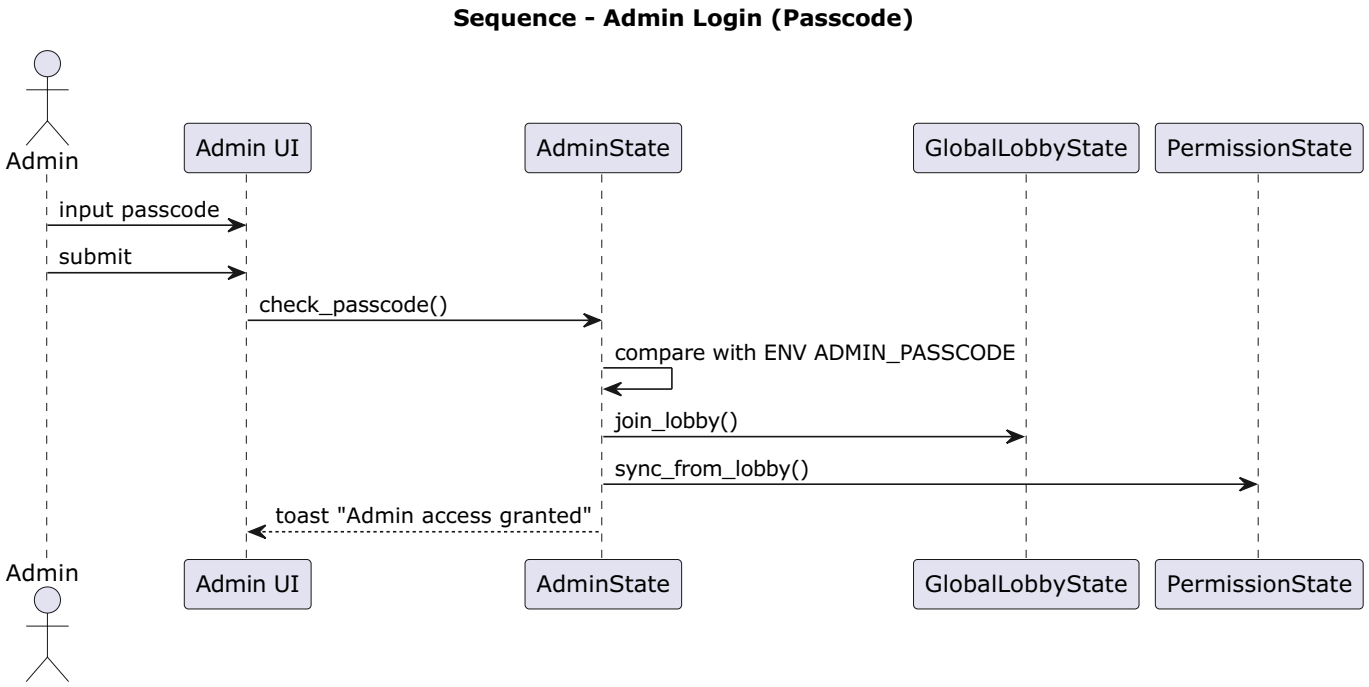


—

10) Admin / Permissions (Management plane)

10.1 Admin Login (PASSCODE)

- In `.env`: `ADMIN_PASSCODE=...`
- After `AdminState.check_passcode()` succeeds:
- `GlobalLobbyState.join_lobby()` ensures lobby data is available
- `PermissionState.sync_from_lobby()` syncs the UI toggles with the lobby snapshot



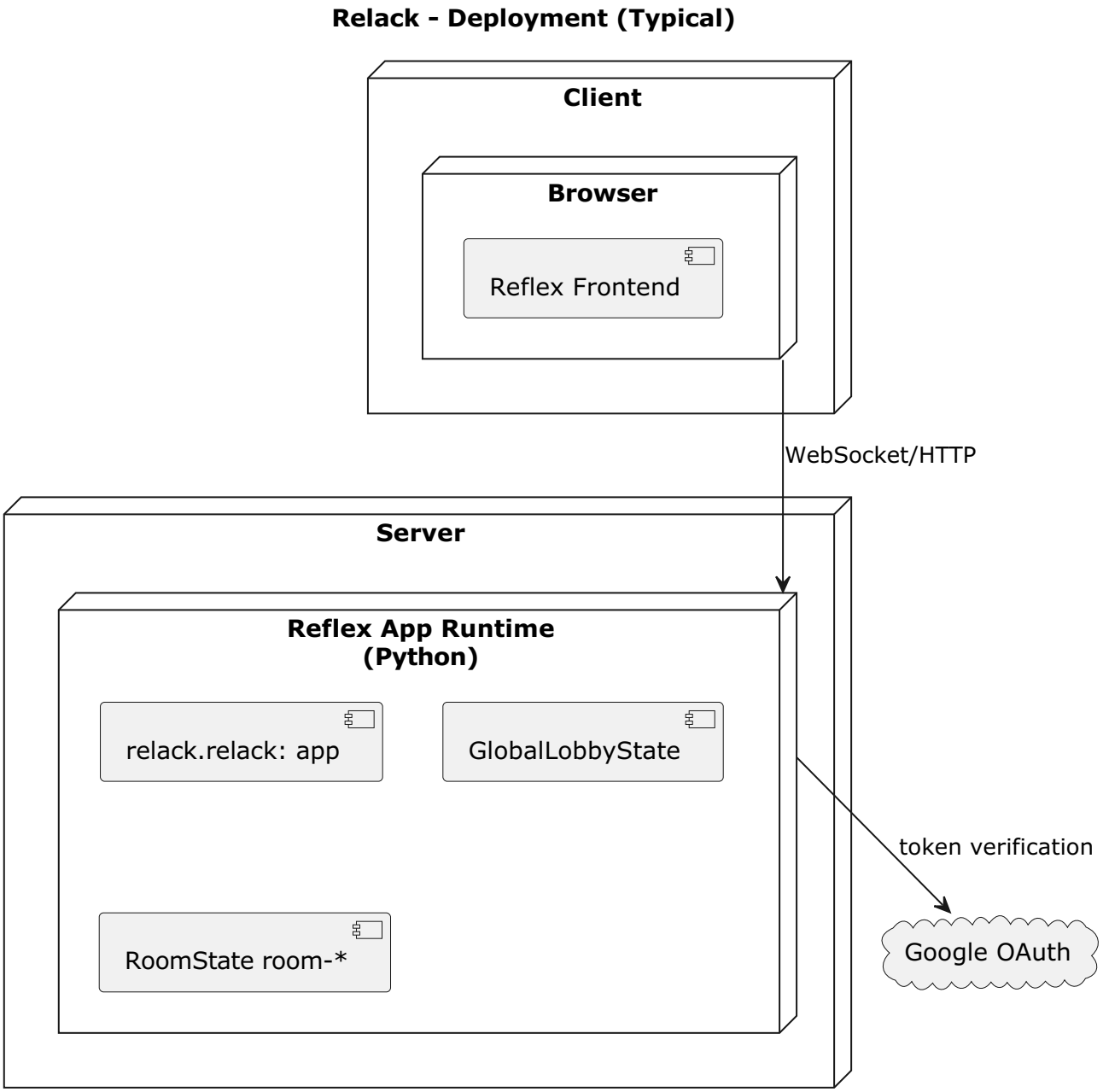
10.2 Data Maintenance (Export / Import / Clear)

GlobalLobbyState provides:

- `_snapshot()`: rooms / profiles / messages_by_room / permissions
- `export_data()` / `import_data(payload)` / `clear_all_data()`

—

11) Deployment Diagram (Typical self-hosted deployment)



> If you plan to support multi-worker / multi-server in the future: SharedState is currently in-memory, so you typically need external storage or pub/sub (e.g., Redis) to guarantee cross-worker synchronization (this would be a next-step extension).

—

12) Testing (Playwright e2e)

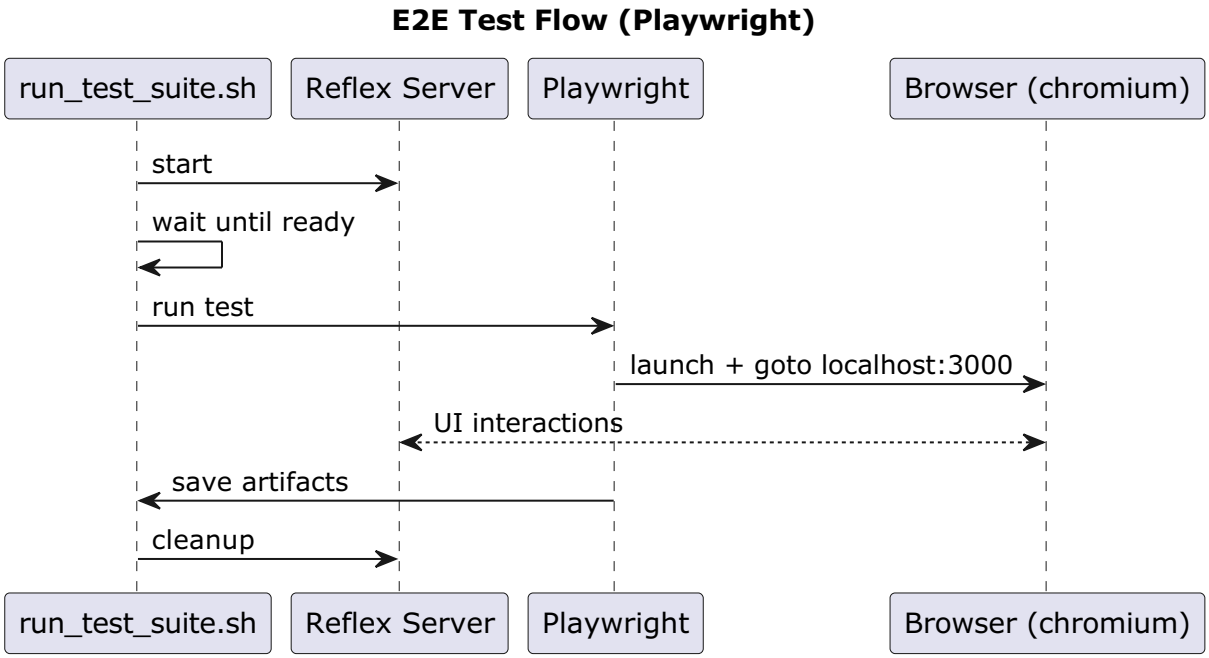
`testcases` provides:

- `guest_login`
- `google_login`

- `admin_login`

Overall strategy:

- `run_test_suite.sh`: start the reflex server → wait until ready → run the corresponding Playwright script → collect artifacts



—

13) WBS (System capability breakdown)