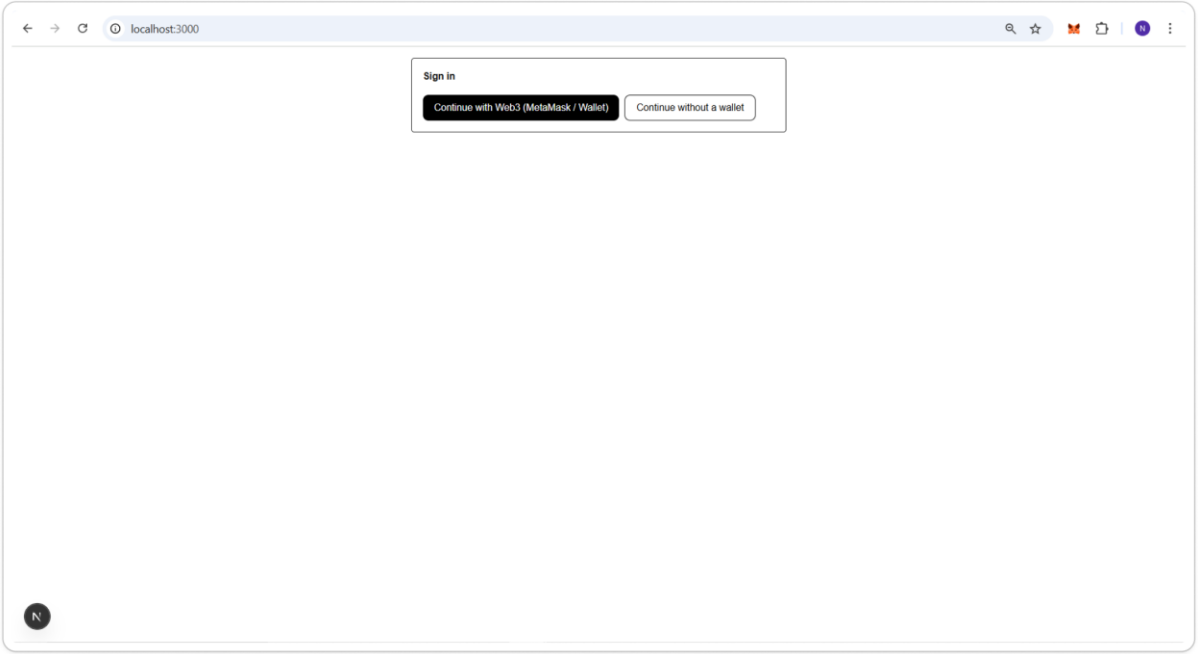
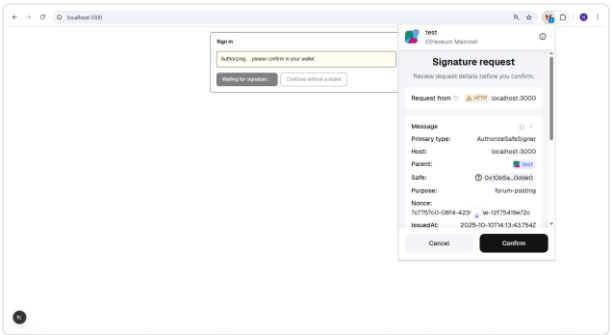
Login & Create Account



Create Account – User creates a new crypto account

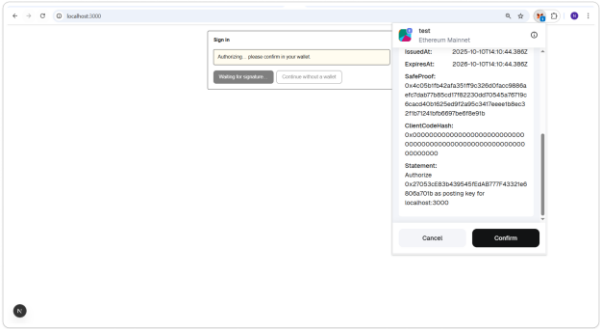
Web3 Login – User has an existing crypto account

- The user connects and signs an EIP 712 which creates and binds a new account to the parent signing.
- The new account is used so the user can post messages on the forum, signing posts for verification with EIP-191



Web3 Login

User has a Web3 wallet e.g. Metamask



EIP-712

User has a Web3 wallet e.g. Metamask

Scripts:

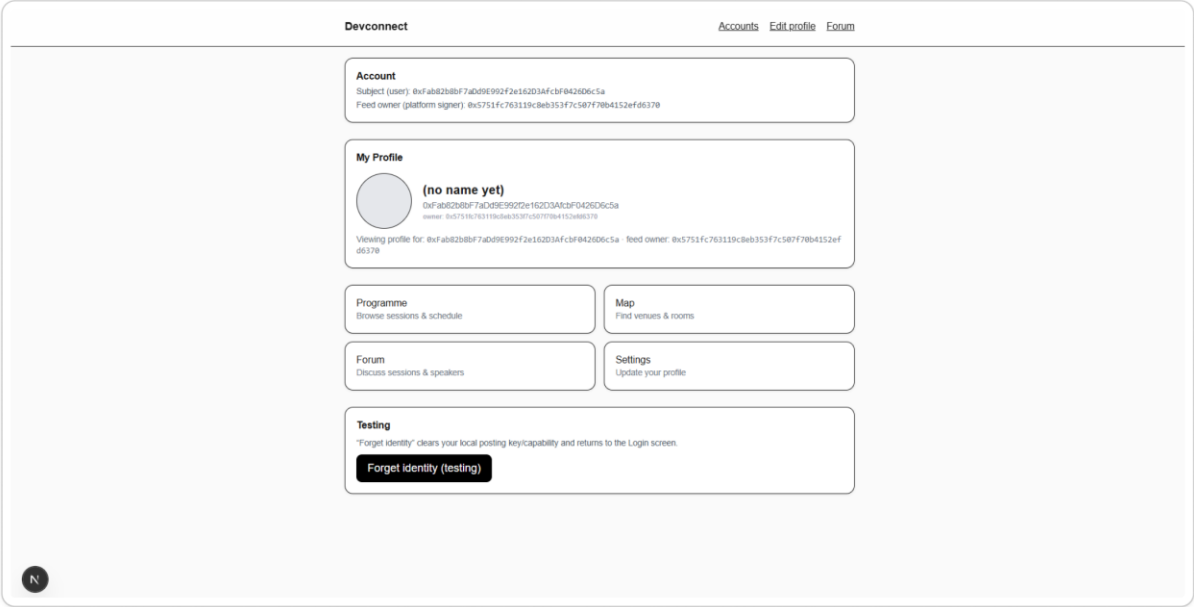
UI – https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/app/page.tsx

Lib (main script – EIP-712 creation, account creation, login) - https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/lib/auth/usePostingIdentity.tsx

Components –

- https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/components/auth/LoginScreen.tsx
- https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/components/auth/PostingAuthNudge.tsx

Home Screen/Dashboard

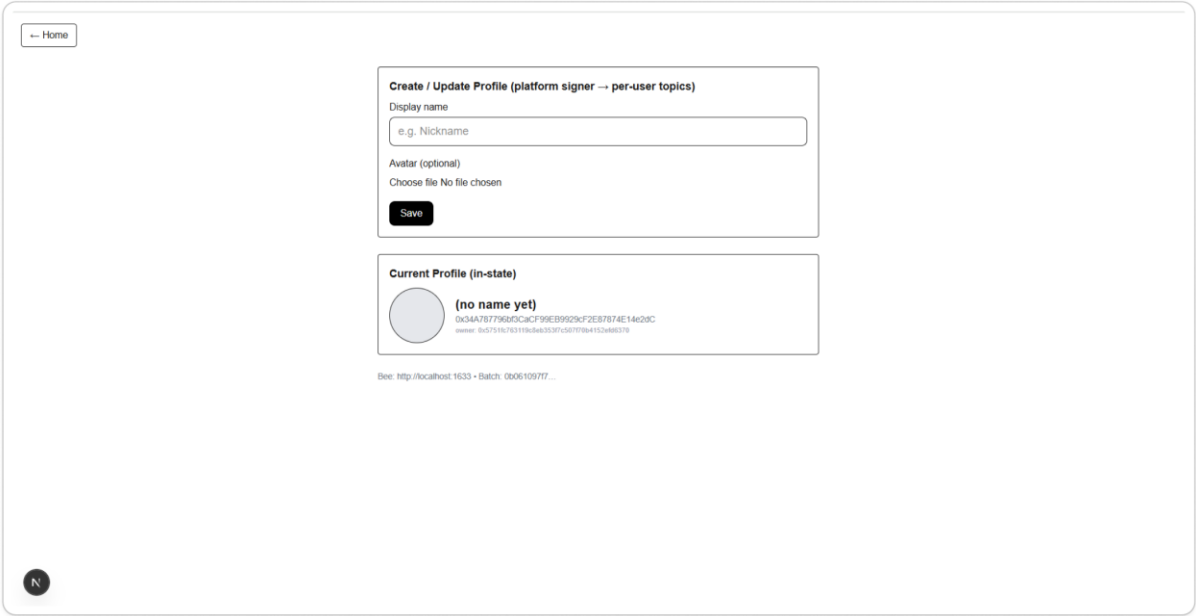


First time login – Blank profile information – Displays public address (parent account for web3 login)

Scripts

UI - https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/app/dashboard/page.tsx

Edit Profile



User Input – Profile Picture and Nickname

Feeds are created using the platform signer and the topic below

Topic for Profile Feeds = | Deterministic prefix | Account Public Address |

- devconnect/profile/name/public-address
- devconnect/profile/avatar/public-address

A) Save name (text/JSON)

1. Client → Server
    - POST /api/profile
    - Body: { kind: "name", payload: { subject, name } }
2. Server (platform signer) → Swarm (Bee)
    - Derive topic: devconnect/profile/name/{subjectNo0x}
    - Create/serialize the name doc (e.g., { "name": "Alice" }) or plain UTF-8.
    - Upload the doc → immutable ref (usually via /bytes or /bzz depending on your writer helper).

    - Publish feed: set the sequence feed at that topic to point to the new ref (signed by feedOwner).

3. Server → Client (response)
    - Returns { owner: <feedOwner> } (and optionally the ref).
4. Client (UI)
    - Persist owner (e.g., localStorage["woco.owner0x"] = owner).
    - applyLocalUpdate({ name }) for instant UI.
    - Schedule a gentle ensureFresh() so the provider re-reads the feed head from Swarm and confirms.

Swarm touchpoints

- Write content: POST /bytes (or POST /bzz) → ref
- Publish feed: POST /feeds/{owner}/{topicHex}?type=sequence (bee-js feed writer)

B) Save avatar (file/blob)

1. Client → Swarm (Bee)
    - Upload file with Bee-JS: bee.uploadFile(POSTAGE_BATCH_ID, file, file.name) → returns immutable imageRef (under the hood this is a POST /bzz with your batch)
2. Client → Server
    - POST /api/profile
    - Body: { kind: "avatar", payload: { subject, imageRef } }
3. Server (platform signer) → Swarm (Bee)
    - Derive topic: devconnect/profile/avatar/{subjectNo0x}
    - Write sequence feed at that topic to point to imageRef (signed by feedOwner).
4. Server → Client (response)
    - Returns { owner: <feedOwner> } (and optionally echo imageRef).
5. Client (UI)
    - Persist owner for future reads.

- applyLocalUpdate({ avatarRef: imageRef }) → avatar appears immediately.
- Kick a short-delay ensureFresh() to confirm feed head.
- The actual image renders at: src = ${BEE_URL}/bzz/${imageRef}?v=${avatarMarker} (?v= cache-busts when avatar changes)

Swarm touchpoints

- Upload avatar blob: POST /bzz → imageRef
- Publish feed: POST /feeds/{owner}/{topicHex}?type=sequence

Scripts:

UI - https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/app/profile/page.tsx

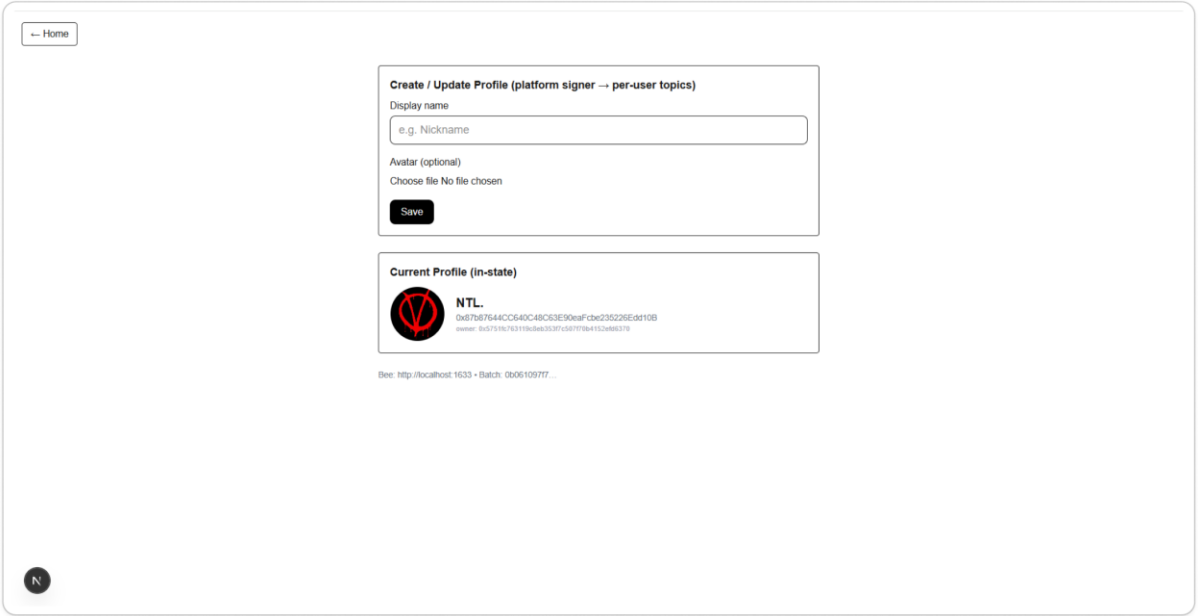
Profile save - https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/app/profile/ProfileTab.tsx

Lib –

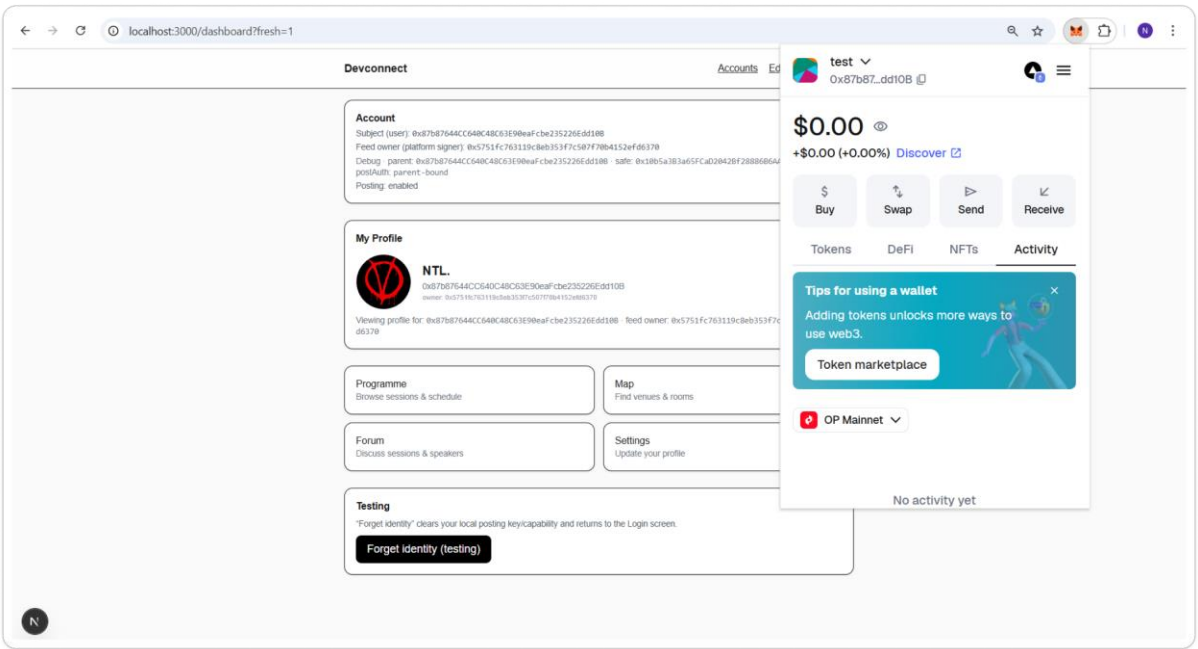
- https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/lib/profile/context.tsx
- https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/lib/profile/service.ts
- https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/lib/profile/storage.ts
- https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/lib/profile/swarm.ts
- https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/lib/profile/types.ts

API - https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/app/api/profile/route.ts

Viewing Profile Elements



- Once uploaded the profile content is also cached.
- Content can be recovered from Swarm using GET function with the:
    - Platform feed signer
    - Deterministic prefix
    - Users public address



Profile View (READ)

1. Client (UI) mounts reader
    - Mount <ProfileProvider beeUrl feedOwner subject>.
    - Hydrate cached profile for instant paint.
    - Immediately call ensureFresh().
2. Client (service/provider) builds topics
    - Name topic: devconnect/profile/name/{subjectNo0x}
    - Avatar topic: devconnect/profile/avatar/{subjectNo0x}
3. Client → Swarm (Bee): resolve feed heads → refs
    - Name feed head: GET {beeUrl}/feeds/{feedOwner}/{topicHex}?type=sequence
    - Avatar feed head: GET {beeUrl}/feeds/{feedOwner}/{topicHex}?type=sequence
    - Result: If 200 → extract immutable ref (e.g., nameRef, avatarRef). If 404 → treat as "no data yet".
4. Client → Swarm (Bee): dereference NAME
    - Prefer raw bytes, fallback to bzz:
        - GET {beeUrl}/bytes/{nameRef} (use cache: no-store)
        - If not found → GET {beeUrl}/bzz/{nameRef}
    - Parse JSON { "name": "Alice" } if possible; otherwise treat as UTF-8 text.
5. Browser (direct) → Swarm (Bee): render AVATAR
    - No JS fetch needed; render immutable ref:
        - src = {beeUrl}/bzz/{avatarRef}?v={avatarMarker}
    - ?v={avatarMarker} forces re-fetch when avatar changes (cache-buster).
6. Client (provider): update UI only if changed
    - Compute lightweight markers (e.g., hash of name text and avatar ref/payload).
    - If unchanged → keep previous object identity (avoid re-renders/flicker).
    - If avatar parsing fails → keep previous avatarRef.

Swarm touchpoints

- Resolve feeds (name & avatar): GET /feeds/{owner}/{topicHex}?type=sequence → current immutable ref
- Dereference name (text/JSON): GET /bytes/{ref} → fallback GET /bzz/{ref}
- Render avatar (image/file): Browser GET /bzz/{ref}?v={marker}

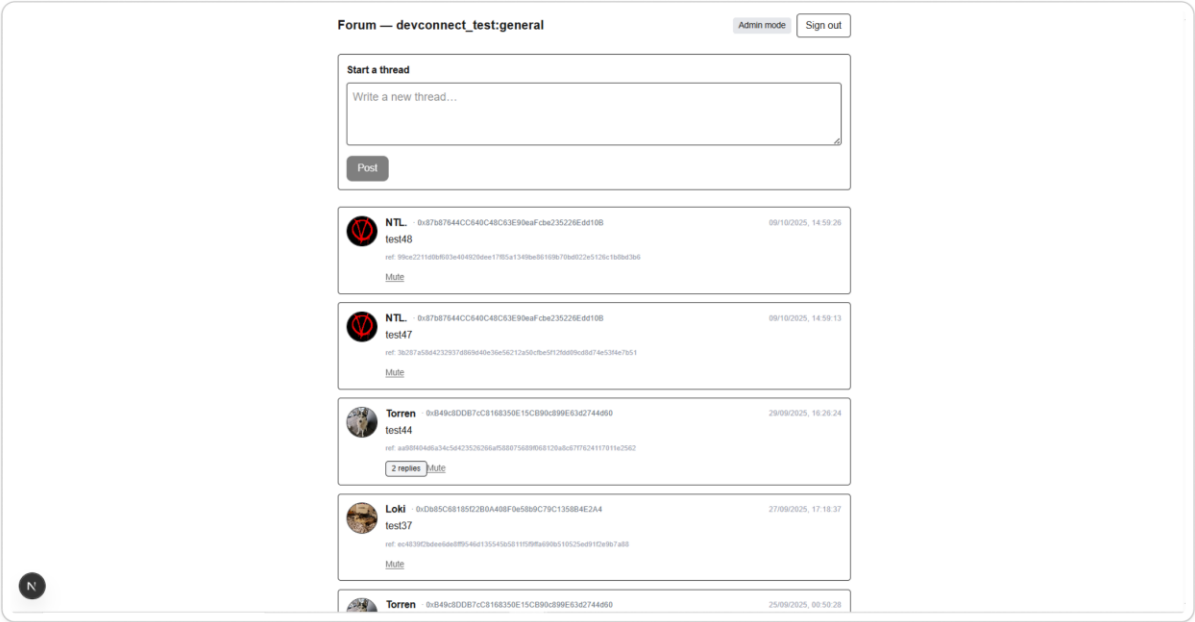
Scripts

Profile view - https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/app/profile/ProfileView.tsx

API - https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/app/api/profile/route.ts

Forum/Message boards

Main Forum UI's



- Message boards are saved as a feed, with the topic being a predetermined name (BOARD_ID) e.g. devconnect_fourm
- The platform feed signer is used to retrieve the individual posts for the forum UI

A) Board page (list of threads)

1. Client (UI) mounts board page
    - Call fetchBoard(BOARD_ID).
2. Client (or API) → Swarm (Bee): resolve Board feed
    - Topic: topicBoard(BOARD_ID)
    - GET {beeUrl}/feeds/{feedOwner}/{topicHex}?type=sequence
    - Collect thread root refs (newest-first; across feed pages if needed).
3. Client → Swarm (Bee): dereference each thread root
    - For each threadRef:
    -
        - GET {beeUrl}/bytes/{threadRef} → fallback GET {beeUrl}/bzz/{threadRef}
    - Parse canonical post JSON; render previews.

Swarm touchpoints (Board)

- Resolve board feed: GET /feeds/{owner}/{topicHex}?type=sequence
- Deref thread root: GET /bytes/{ref} → fallback GET /bzz/{ref}

Scripts:

UI - https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/app/forum/page.tsx

Components –

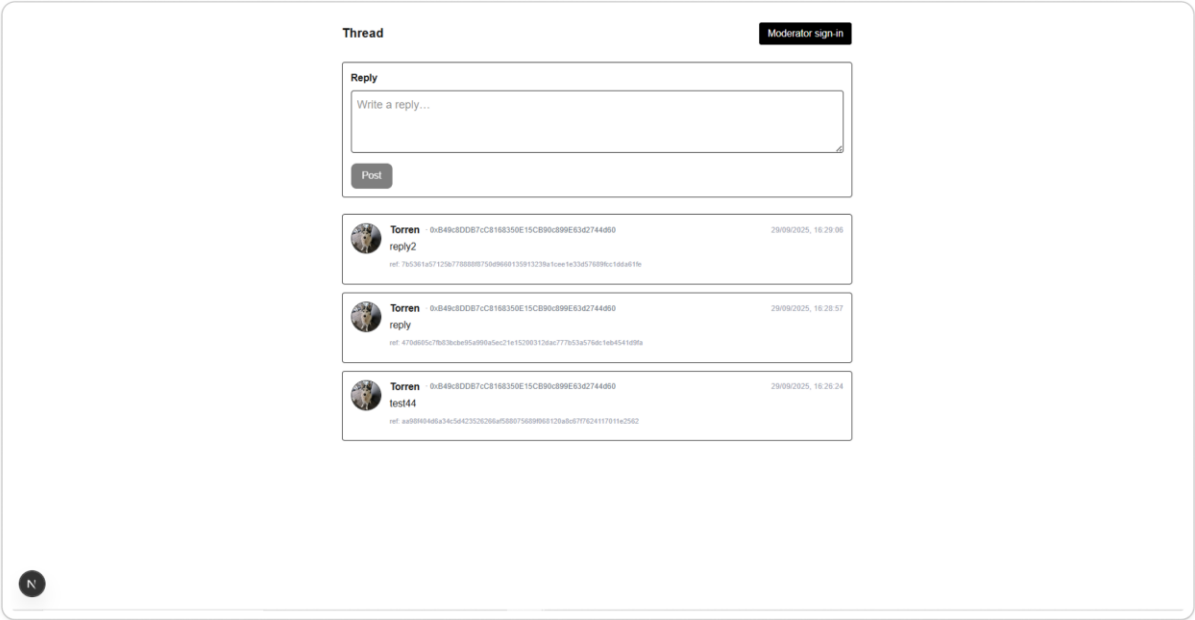
- Display replies: https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/components/forum/ReplyBadge.tsx

Lib –

- https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/lib/forum/boardID.ts
- https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/lib/forum/bytes.ts
- https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/lib/forum/client.ts
- https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/lib/forum/crypto.ts
- https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/lib/forum/memory.ts

API - https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/app/api/forum/board/route.ts

Post Reply UI's



- The reply UI's feed topics are determined by the BOARD_ID and the threadRef (the hash of the post being replied to)
- Again the platform signer is used to render the reply board

B) Thread page (root + replies)

1. Client (UI) mounts thread page
    - Read threadRef from route param.
    - Call fetchThread(BOARD_ID, threadRef).
2. Client → Swarm (Bee): dereference root post
    - GET {beeUrl}/bytes/{threadRef} → fallback GET {beeUrl}/bzz/{threadRef}
    - Parse canonical post JSON; render root.
3. Client (or API) → Swarm (Bee): resolve Thread feed (replies)
    - Topic: topicThread(BOARD_ID, threadRef)
    - GET {beeUrl}/feeds/{feedOwner}/{topicHex}?type=sequence
    - Collect reply refs (newest-first; across feed pages if needed).
4. Client → Swarm (Bee): dereference each reply
    - For each replyRef:

        - GET {beeUrl}/bytes/{replyRef} → fallback GET {beeUrl}/bzz/{replyRef}
    - Parse canonical post JSON; render reply items.

Swarm touchpoints (Thread)

- Resolve thread feed: GET /feeds/{owner}/{topicHex}?type=sequence
- Deref reply: GET /bytes/{ref} → fallback GET /bzz/{ref}
- Avatars in posts: Browser GET /bzz/{avatarRef}?v={marker}

Swarm touchpoints (Thread)

- Resolve thread feed: GET /feeds/{owner}/{topicHex}?type=sequence
- Deref reply: GET /bytes/{ref} → fallback GET /bzz/{ref}
- Avatars in posts: Browser GET /bzz/{avatarRef}?v={marker}

Scripts:

UI - https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/tree/main/src/app/forum/%5BthreadRef%5D

Lib – Above

API - https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/app/api/forum/thread/route.ts

Posting a Message/Thread

- The user inputs the message and the when the Post button is clicked they auto sign the JSON payload using EIP-191
- The avatarRef and displayName are both included I the JSON payload.
- When the post is sent the client checks that the signature and posting account match

- The platform feed signer receives the payload calls POST function to upload and create a hash (threadRef)
- The threadRef is saved (newest first) to the board feed: topicBoard(BOARD_ID)
    - Each BOARD_ID is 4096 bytes
    - threadRef = 32 bytes
    - Total threadRef's per BOARD_ID = 128
- Client receives the postRef, confirming upload and moving the optimistic to a confirmed post with the threadRef displayed.

Example threadRef:

{"kind":"post","payload":{"subject":"0x87b87644CC640C48C63E90eaFcbe235226Edd10B","boardId":"devconnect_test:general","content":"Hello World...Computer","contentSha256":"0xa8acb92936598b58e9be4b9a19c18863773ad0e4c4a16f6e57ee0533b72f2f80","displayName":"NTL.","avatarRef":"065bbc1c0b79b2b79fc4c4426f6288cdbbbe026437abfcbab635b8b9fe58a1c6","createdAt":1760470350156,"nonce":"2631084304-3566832716-2018442257-3490255555","version":1},"signature":"0xcd1e12afa78db649b31d181b44af4341230715cb3b673e7a1012e190fb6b9c4b050ce4d44aaccd6c9b49f10370898cf9feac557154ef945cadcf158045b641031c","signatureType":"eip191","server":{"receivedAt":1760470354896,"boardTopic":"0x938e48a0c8c1d1cf7dc153cb3d5669338763bee7dcdfb5fd2b2eaa6a7f8484c1","threadTopic":"0x0000000000000000000000000000000000000000000000000000000000000000"},"v":1}

Forum Post (WRITE)

A) Start a new thread (root post)

1. **Client (Composer) builds payload & optimistic row** - Composer
    - Snapshot identity + profile: subject (actor), displayName, avatarRef (from useProfile()), boardId.
    - Hash content: contentSha256 = sha256HexString(content).
    - Build SignedPostPayload with { subject, boardId, content, contentSha256, displayName?, avatarRef?, createdAt, nonce, version }.
    - Emit onOptimistic({ clientTag, postRef: "local:<uuid>", threadRef: "local:<uuid>", payload }) to insert immediately.
2. **Client (Composer) signs & submits** - Composer

- Sign JSON(payload) → signature (EIP-191).
- If Web3, include capability headers when calling submitPost() → POST /api/forum/post with:
  - Headers (web3):
    - x-posting-kind: web3
    - x-posting-parent: <parent addr>
    - x-posting-key: <safe posting key>
    - x-posting-auth: parent-bound
  - Body: { payload, signature, signatureType: "eip191" }.

3. **Server (publisher) → Swarm (Bee): write content & publish feeds**
   - Upload canonical post JSON (immutable) → **POST /bytes** → **postRef**.
   - Because this is a **new thread** (no replyTo):
     - **Set threadRef = postRef** (the root defines the thread).
     - Append **threadRef** to **Board feed** at topicBoard(BOARD_ID) (sequence feed → newest first).
4. **Server → Client (response)**
   - Return { postRef, threadRef }.
5. **Client (UI) finalizes** - Composer
   - Receive { postRef, threadRef } in onPosted({ ..., clientTag }).
   - Replace the optimistic row matching clientTag with the real refs.

Swarm touchpoints (New thread)

- **Write post JSON:** POST /bytes → postRef
- **Publish feed (board index):** POST /feeds/{owner}/{topicBoard(BOARD_ID)}?type=sequence (append threadRef)

Scripts:

Components –

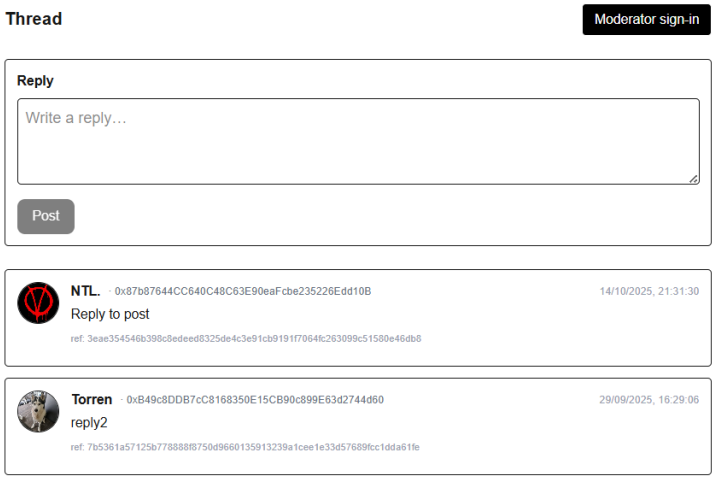
- Composer: https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/components/forum/Composer.tsx
- Post: https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/components/forum/PostItem.tsx

Lib –

- Packs 128 x 32B refs: https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/lib/forum/pack.ts
- Publisher: https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/lib/forum/publisher.ts
- Topics: https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/lib/forum/topics.ts
- Types: https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/lib/forum/types.ts

API - https://github.com/yea-80y/DevConnect-Profile-Forum-Sandbox/blob/main/src/app/api/forum/post/route.ts

Posting a Reply



- Replies follows a similar model to the original post, with threadRef: replyTo
- JSON payloads include the same data and are signed (EIP-191) and checked like original posts
- The platform feed signer receives the payload, calls POST, receiving the reply hash (threadRef: replyTo)
- The reply hash is saved to the post reply feed: topicThread(BOARD_ID, threadRef)
- As with the main boards, each reply hash is saved to the feeds chunk = 128 replies (newest first)
- Optimistic reply removed when postRef received.

Example threadRef: replyTo

{"kind":"post","payload":{"subject":"0xDb85C68185f22B0A408F0e58b9C79C1358B4E2A4","boardId":"devconnect_test:general","threadRef":"d008a57992a419eaa351f3b5cf9a7b37288f354a9c15e018e3e5d38096b6d3d5","content":"reply1","contentSha256":"0x1ea01959e3788decc7d12daabe520155e6bebeeace3c582013d989010e340e90","displayName":"Loki","avatarRef":"9bfc26526be45e7616bb87b10a0c2e6cdfa1ff6db92dbf16a7e07ba8d43888ca","createdAt":1758665258561,"nonce":"3498222231-738894226-1668960140-1384429286","version":1},"signature":"0x54d3f7465f1f7ac4b0c2b41d72d62e7cfd511a4483a02de5f05a3796a62e4d24604ea5b5a5d313c82a29c55da39a8748de8cd599a16abf7c6c2d7b4935e2db531b","signatureType":"eip191","server":{"receivedAt":1758665258816,"boardTopic":"0x938e48a0c8c1d1cf7dc153cb3d5669338763bee7dcdfb5fd2b2eaa6a7f8484c1","threadTopic":"0x0000000000000000000000000000000000000000000000000000000000000000"},"v":1}

B) Reply to a thread

1. **Client (Composer) builds payload & optimistic row** - Composer
    - Same as above, but include replyTo (a 64-hex **threadRef** from the URL/page).
    - Build SignedPostPayload with threadRef: replyTo.
    - Emit onOptimistic({ clientTag, postRef: "local:<uuid>", threadRef: replyTo, payload }).
2. **Client (Composer) signs & submits** - Composer
    - Sign JSON(payload) → signature (EIP-191).
    - If Web3, send capability headers (same as above).
    - POST /api/forum/post with { payload, signature, signatureType: "eip191" }.
3. **Server (publisher) → Swarm (Bee): write content & publish feeds**
    - Upload canonical **reply** JSON → **POST /bytes** → **postRef**.
    - Append **postRef** to the **Thread feed** at topicThread(BOARD_ID, threadRef) (sequence feed → newest first).
4. **Server → Client (response)**
    - Return { postRef, threadRef }.
5. **Client (UI) finalizes** - Composer

- onPosted({ postRef, threadRef, clientTag }) replaces the optimistic reply with the real one.

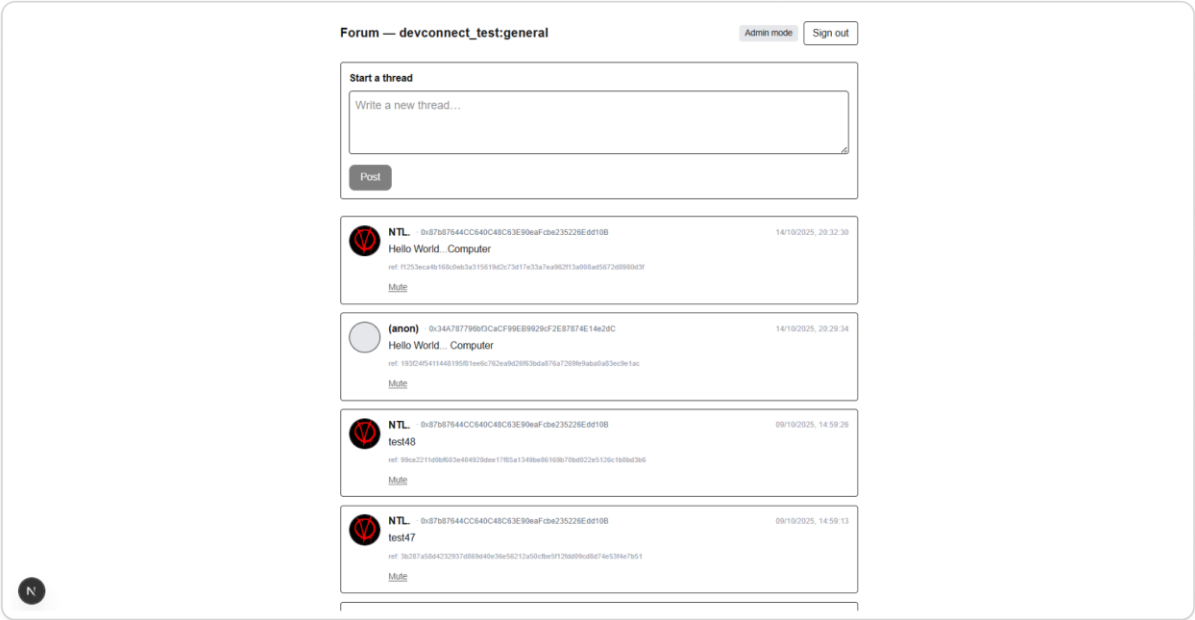
Swarm touchpoints (Reply)

- **Write reply JSON:** POST /bytes → postRef
- **Publish feed (thread timeline):** POST /feeds/{owner}/{topicThread(BOARD_ID, threadRef)}?type=sequence (append postRef)

Scripts:

Above in Post a Thread with append - topicThread(boardId: string, threadRef: string)

<u>Moderation – v1</u>



- Currently there is an administrator whitelist, permitting certain accounts to login as an admin and moderate the forums with a mute button.
- Muted posts are written by the platform signer to new feed with the topic: topicModThreads(BOARD_ID)
- Muted replies follow the same flow, but are written with the topic: topicModReplies(BOARD_ID, threadRef)
- The UI then resolves both boards, filtering only posts that are not muted.

A) Admin visibility & auth

1. **Client (UI)**
    - On mount, call /api/auth/me and cache isAdmin.

    - Show **Mute** (and **Unmute**) buttons in <PostItem> **only if** isAdmin === true.
2. **ClientProviders**
    - Exposes isAdmin to pages/components so buttons render consistently.

B) Mute a thread (board timeline)

1. **Client (admin clicks "Mute thread")**
    - Target: threadRef (the root post ref for that thread).
    - POST /api/forum/moderate
    - Body: { action: "mute", kind: "thread", boardId: BOARD_ID, ref: threadRef }
2. **Server (publisher) → Swarm (Bee)**
    - **Topic (deterministic):** topicModThreads(BOARD_ID)
    - **Write action unit (immutable JSON):** { op: "mute", kind: "thread", ref: threadRef, ts }
    - POST /bytes → modRef
    - **Append modRef** to moderation **threads** feed (sequence): POST /feeds/{owner}/{topicModThreads(BOARD_ID)}?type=sequence
3. **Server → Client**
    - Return { ok: true } (and optionally modRef).
4. **Client (UI)**
    - Optimistically mark the thread as muted in local state.

Swarm touchpoints (Mute thread)

- **Write moderation item:** POST /bytes → modRef
- **Publish feed (board-level mutes):** POST /feeds/{owner}/{topicModThreads(BOARD_ID)}?type=sequence

---

C) Mute a reply (inside a thread)

1. **Client (admin clicks "Mute reply")**
    - Target: postRef (the reply's immutable ref) and the thread's threadRef.
    - POST /api/forum/moderate

    - Body: { action: "mute", kind: "reply", boardId: BOARD_ID, threadRef, ref: postRef }

2. **Server (publisher) → Swarm (Bee)**
    - **Topic (deterministic):** topicModReplies(BOARD_ID, threadRef)
    - **Write action unit (immutable JSON):** { op: "mute", kind: "reply", ref: postRef, ts }
    - POST /bytes → modRef
    - **Append modRef** to moderation **replies** feed (sequence): POST /feeds/{owner}/{topicModReplies(BOARD_ID, threadRef)}?type=sequence
3. **Server → Client**
    - Return { ok: true }.
4. **Client (UI)**
    - Optimistically mark the reply as muted in local state.

Swarm touchpoints (Mute reply)

- **Write moderation item:** POST /bytes → modRef
- **Publish feed (thread-level mutes):** POST /feeds/{owner}/{topicModReplies(BOARD_ID, threadRef)}?type=sequence

D) How the UI applies moderation (READ)

1. **Board page (list of threads)**
    - Resolve **Board feed** → get threadRef[].
    - Resolve **Moderation-Threads feed** (topicModThreads(BOARD_ID)) → reduce ops to a **MutedThreadSet` .
    - **Filter**: drop any thread whose threadRef ∈ MutedThreadSet.
    - Render remaining threads.
2. **Thread page (root + replies)**
    - Resolve **Thread feed** → get postRef[] (replies).
    - Resolve **Moderation-Replies feed** (topicModReplies(BOARD_ID, threadRef)) → reduce ops to a **MutedReplySet**.

    - **Filter**: drop any reply whose postRef ∈ MutedReplySet.
    - Render remaining replies.

Swarm touchpoints (READ for moderation)

- **Resolve moderation feeds:**
  GET /feeds/{owner}/{topic}?type=sequence → collect modRef[]
- **Dereference moderation items:**
  GET /bytes/{modRef} → JSON { op, kind, ref, ts }
- **Apply reducer:** last-writer-wins per ref (i.e., the most recent op for that ref in the feed determines visibility).