EVzone SDK — Cookie-Based Account Detection & Selection (Single Doc)

This guide tells you **exactly** what to implement so the SDK:

- reads **user numbers** from cookies,
- resolves them to **full user profiles** `{ userNo, walletId, owner, email, photo }`,
- and shows the correct modal:
  - **0 users** → *No account / Sign in*
  - **1 user** → *Summary*
  - **2+ users** → *Account Picker*

It also shows where to **console the full selected profile object** and how to **wire sign-in** so cookies are detected after login.

---

1) Cookies → Profiles (`src/utils/cookie.js`)

You already have helpers like `getUserNoFromCookie`, `getUserNosFromCookie`, etc. Add **one** function that turns userNos → full profiles.

```
// src/utils/cookie.js

/**
 * Return an array of profiles for the given userNos.
 * Each profile MUST have: { userNo, walletId, owner, email, photo }.
 *
 * Replace the example map with your own cookie/storage-backed values.
 */
export function resolveUserProfilesFromCookies(userNos = []) {
  const arr = Array.isArray(userNos) ? userNos.filter(Boolean) : [];

  // ✅ Example stub — replace with your cookie-backed data
  const known = {
    'U-000123': {
      walletId: 'W-256-48392018',
      owner: 'John Doe',
      email: 'john@x.com',
      photo: 'https://api.dicebear.com/7.x/initials/svg?seed=John%20Doe',
    },
    'U-000789': {
      walletId: 'W-256-74731323',
      owner: 'Jane Smith',
      email: 'jane@x.com',
      photo: 'https://api.dicebear.com/7.x/initials/svg?seed=Jane%20Smith',
    },
  };

  return arr.map((u) => {
    const k = String(u);
    const row = known[k];
    // IMPORTANT: always return the full shape
    return row
      ? { userNo: k, ...row }
      : {
          userNo: k,
          walletId: null,
          owner: `User ${k.slice(-3)}`,
          email: `${k.replace(/[^a-z0-
9]/gi,'').toLowerCase()}@example.com`,
          photo: `https://i.pravatar.cc/80?u=${encodeURIComponent(k)}`,
        };
  });
}
```

> When you wire in your real cookie logic later: **keep the same function name and return shape**—just populate fields from your data source.

---

2) Account lookup uses your cookie resolver (`src/sdk/paykitClient.js`)

The SDK calls `lookupUsersByNo(userNos)` to populate the Account Picker. Make it delegate to the resolver above (no external calls).

```
// src/sdk/paykitClient.js
import { resolveUserProfilesFromCookies } from '../utils/cookie.js';

async function lookupUsersByNo(userNos) {
  const arr = Array.isArray(userNos) ? userNos.filter(Boolean) : [];
  if (arr.length === 0) return { users: [] };

  // Resolve from cookies/storage only
  const users = resolveUserProfilesFromCookies(arr);
  return { users };
}
```

> Nothing else in this file needs changing for this flow.

---

3) Log the **full profile** when an account is chosen (`src/WalletPaymentForm.js`)

During the Account Picker step, add one line to log the chosen object.

```
// src/WalletPaymentForm.js
// ...
else if (view === 'accountPicker') {
  content = (
    <AccountPickerModal
      open
      zIndex={zIndex}
      accounts={accounts || []}
      onSelect={(userNo) => {
        // 🔎 Log the full selected profile object:
        const selected = (accounts || []).find(a => a.userNo === userNo);
        try { console.log('[EVZ SDK] selected account:', selected); } catch
{}

        setPasscode('');
        selectAccount?.(userNo); // continue the flow
      }}
      onClose={closeAndReset}
    />
  );
}
```

If the user goes through **Sign In** instead of the picker, you can also log immediately after the login succeeds (optional):

```
// src/WalletPaymentForm.js
else if (view === 'signin') content = (
  <HasAccountSummary
    open
    onLoginSuccess={(userNo) => {
      // Optional: if you want to log here as well:
      // import { resolveUserProfilesFromCookies } from
'./utils/cookie.js';
      // const [profile] = resolveUserProfilesFromCookies([userNo]);
      // console.log('[EVZ SDK] selected account (signin):', profile);

      setPasscode('');
      selectAccount?.(userNo);
    }}
    onClose={closeAndReset}
    zIndex={zIndex}
  />
);
```

> Result: when someone picks or signs in, the console shows: `{ userNo, walletId, owner, email, photo }`.

---

4) Make the Sign-In modal trigger your auth & cookie set (`src/HasAccountSummary.js`)

Use the **adapter prop** `startAuth` so your sign-in flow runs, sets cookies, and returns the active `userNo`. The SDK will then re-check cookies and continue automatically.

Where you render `<WalletPaymentForm />`, pass:

```
<WalletPaymentForm
  /* your existing props */
  startAuth={async () => {
    // 1) Run your sign-in UI here (popup/redirect/etc.)
    // 2) On success, set your cookies so getUserNo(s)FromCookie see them.
    // 3) Return the active user number:
    const userNo = window.myAuth?.getActiveUserNo?.(); // your own helper
    if (!userNo) throw new Error('cancelled'); // closes gracefully
    return { userNo };
  }}
/>
```

In `HasAccountSummary.js`, ensure it calls `startAuth` if provided (fallbacks remain intact):

```
// Inside HasAccountSummary component, when "Sign in" is clicked:
if (typeof startAuth === 'function') {
  try {
    const { userNo } = await startAuth();
    if (userNo) onLoginSuccess?.(userNo);
  } catch {
    onClose?.();
  }
  return;
}
```

> After `onLoginSuccess(userNo)`, the SDK sets the cookie (via its helpers) and restarts the flow. **0 users** → Sign in, **1 user** → Summary, **2+ users** → Account Picker.

---

5) How the SDK decides which modal to show (already implemented)

The hook (`useWalletPaymentFlow`) checks cookies and behaves as follows:

- **No user numbers found** → shows **"No account / Sign in"** modal.
- **Exactly 1 user number** → auto-selects and shows **Summary**.
- **2 or more user numbers** → calls `lookupUsersByNo` (your resolver) and shows **Account Picker**.

You do **not** need to change the hook logic—just make sure:

- `getUserNosFromCookie()` returns all userNos.

- `resolveUserProfilesFromCookies(userNos)` returns full profiles.

---

6) Contract you must keep

- `resolveUserProfilesFromCookies(userNos)` **returns**:

```
[
    {
    userNo: string,
    walletId: string | null,
    owner: string,
    email: string,
    photo: string
  },{
    userNo: string,
    walletId: string | null,
    owner: string,
    email: string,
    photo: string
  },{
    userNo: string,
    walletId: string | null,
    owner: string,
    email: string,
    photo: string
  }
]
```

- `getUserNosFromCookie()` **returns** all available userNos (active first is fine).

- **Do not** rename these functions or change their return shapes.

---

7) Quick checklist

- [ ] `getUserNosFromCookie()` returns the list you support.
- [ ] `getUserNoFromCookie()` returns the active user after sign-in.
- [ ] `resolveUserProfilesFromCookies(userNos)` yields `{ userNo, walletId, owner, email, photo }`.
- [ ] `lookupUsersByNo()` delegates to the resolver (no external calls).
- [ ] `WalletPaymentForm` logs the selected profile on pick (and optionally on sign-in).
- [ ] `startAuth` is passed to `WalletPaymentForm` and returns `{ userNo }` after your sign-in completes.