School Event Check-In — Full SUI-targeted Code (Move module + tests + React frontend)

Această pagină conține codul complet (ready-to-copy) pentru MVP-ul proiectului **School Event Check-In** construit pe Sui.

Ce primești aici - Un pachet Move (Sui) cu modul `attendance` care implementează: Registry (shared object), AttendanceBadge (resource), flux de check-in și logică de upgrade (stretch). - Teste Move (unit tests) care acoperă scenarii importante. - Un frontend React + TypeScript folosind `@mysten/dapp-kit` (dapp-kit) pentru: conectare wallet, scanare QR (client-side), fetch badges, submit tranzacții.

> **Notă:** Codul e scris pentru Sui; poate necesita mici adaptări dacă versiunea CLI/SDK din mediul tău diferă. Am folosit convenții și module din Sui standard library: `sui::table`, `sui::transfer::share_object`, `sui::object`.

---

1) Move package: `packages/attendance`

Move.toml (esentials)

```
[package]
name = "attendance"
version = "0.1.0"
authors = ["Team Hackathon"]

[dependencies]
sui-framework = { git = "https://github.com/MystenLabs/sui", subdir =
"crates/sui-framework" }
```

---

sources/attendance.move

```
module 0x1::attendance {
    use sui::tx_context::{Self, TxContext};
    use sui::object::{Self, UID};
    use sui::transfer;
    use sui::table::{Self, Table};
    use std::vector;
    use std::signer;
    use std::string;
    use std::option::{Self, Option};

    /// Errors
    const E_EVENT_NOT_FOUND: u64 = 1;
    const E_NOT_EVENT_ADMIN: u64 = 2;
```

```
    const E_ALREADY_CHECKED_IN: u64 = 3;

    /// AttendanceBadge: soulbound NFT-like resource (non-transferable by
default)
    struct AttendanceBadge has key, store {
        id: u64,
        event_id: u64,
        owner: address,
        ts: u64,
        count_for_upgrade: u64
    }

    /// Reward resource minted after reaching milestone
    struct Reward has key, store {
        id: u64,
        owner: address,
        note: vector<u8>
    }

    /// Event metadata
    struct Event has key, store {
        id: u64,
        name: vector<u8>,
        organizer: address,
        timestamp: u64,
        // Table of attendance: student address -> badge id
        attendance: Table<address, u64>,
        // simple list of badge ids for quick queries (optional)
        badges: Table<u64, address>
    }

    /// Global registry stored as a shared object by the deployer
    struct Registry has key {
        next_event_id: u64,
        next_badge_id: u64,
        events: Table<u64, Event>
    }

    /// Initialize registry under deployer (call once)
    public fun init_registry(deployer: &signer, ctx: &mut TxContext) {
        let deployer_addr = signer::address_of(deployer);
        // Create empty tables
        let events_tbl = Table::new<u64, Event>(ctx);
        let reg = Registry { next_event_id: 1u64, next_badge_id: 1u64,
events: events_tbl };
        let reg_obj = object::new(reg, ctx);
        // Make it shared so multiple transactions can reference it
concurrently
        transfer::share_object(reg_obj);
        // transfer admin ownership handle to deployer (we'll keep address
check for admin ops)
```

```
        // Note: ownership/permission model here: deployer address is
considered admin for simplicity
    }

    /// Admin creates an event
    public fun create_event(admin: &signer, reg_id: object::ID, name:
vector<u8>, ts: u64, ctx: &mut TxContext) {
        let admin_addr = signer::address_of(admin);
        let mut reg = Table::borrow_mut(&mut reg_id, &ctx);
        // create attendance table + badges table for this event
        let attendance_tbl = Table::new<address, u64>(ctx);
        let badges_tbl = Table::new<u64, address>(ctx);
        let event_id = reg.next_event_id;
        reg.next_event_id = event_id + 1;
        let event = Event { id: event_id, name, organizer: admin_addr,
timestamp: ts, attendance: attendance_tbl, badges: badges_tbl };
        Table::insert(&mut reg.events, event_id, event);
    }

    /// Student check-in: mint AttendanceBadge and attach to student's
address
    /// - `reg_obj_id` is the shared registry object id
    /// - `event_id` is the event to check-in
    public fun check_in(student: &signer, reg_obj_id: object::ID, event_id:
u64, ts: u64, ctx: &mut TxContext) {
        let student_addr = signer::address_of(student);
        // borrow registry
        let mut reg = Table::borrow_mut(&mut reg_obj_id, &ctx);
        // find event
        if (!Table::contains_key(&reg.events, event_id)) {
            abort E_EVENT_NOT_FOUND;
        }
        let mut ev = Table::borrow_mut(&mut reg.events, event_id);
        // check already checked-in
        if (Table::contains_key(&ev.attendance, student_addr)) {
            abort E_ALREADY_CHECKED_IN;
        }
        let badge_id = reg.next_badge_id;
        reg.next_badge_id = badge_id + 1;
        let badge = AttendanceBadge { id: badge_id, event_id, owner:
student_addr, ts, count_for_upgrade: 1 };
        // move badge to student's object ownership
        let badge_obj = object::new(badge, ctx);
        transfer::transfer(badge_obj, signer::address_of(student));
        // record in event attendance
        Table::insert(&mut ev.attendance, student_addr, badge_id);
        Table::insert(&mut ev.badges, badge_id, student_addr);
    }

    /// Query helpers (entry reads are done via RPC; these functions are
illustrative)
```

```
    /// Stretch: called after check-in to attempt upgrade
    public fun try_upgrade(student_addr: address, reg_obj_id: object::ID,
threshold: u64, ctx: &mut TxContext) {
        let mut reg = Table::borrow_mut(&mut reg_obj_id, &ctx);
        // count badges owned by student across all events
        let mut count = 0u64;
        let mut iter = Table::keys(&reg.events);
        while (vector::length(&iter) > 0) {
            let k = vector::pop_back(&mut iter);
            let ev = Table::borrow(&reg.events, k);
            if (Table::contains_key(&ev.attendance, student_addr)) {
                count = count + 1;
            }
        }
        if (count >= threshold) {
            // mint reward to student
            let reward = Reward { id: reg.next_badge_id, owner: student_addr,
note: vector::empty<u8>() };
            reg.next_badge_id = reg.next_badge_id + 1;
            let r_obj = object::new(reward, ctx);
            transfer::transfer(r_obj, student_addr);
        }
    }
}
```

⚠️**Important implementation notes:** - The code above is conceptual and mixes some higher-level helpers. In Sui Move, operations such as borrowing a shared object and using `Table::borrow_mut` require the exact `object::ID` and correct TxContext usage. Use Sui's `move` compiler and test harness to ensure all calls match the exact stdlib signatures.

---

2) Move Tests (sources/attendance_tests.move)

```
module 0x1::attendance_tests {
    use 0x1::attendance;
    use std::vector;
    use sui::tx_context::TxContext;
    use std::signer;

    #[test]
    fun test_create_registry_and_event() {
        let deployer = test::create_signer(0x1);
        let ctx = test::tx_context();
        // init registry
        attendance::init_registry(&deployer, &mut ctx);
        // get registry id from context (test harness specific)
        // create event
```

```
        let name = string::utf8("Book Club");
        attendance::create_event(&deployer, /*reg id*/
test::get_registry_object_id(), name, 123456, &mut ctx);
        // assert event exists
        assert!(/*some existence check*/ true);
    }

    #[test]
    fun test_check_in_and_no_double() {
        let deployer = test::create_signer(0x1);
        let student = test::create_signer(0x2);
        let mut ctx = test::tx_context();
        attendance::init_registry(&deployer, &mut ctx);
        let name = string::utf8("Seminar");
        attendance::create_event(&deployer, test::get_registry_object_id(),
name, 111, &mut ctx);
        // student checks in
        attendance::check_in(&student, test::get_registry_object_id(), 1,
200, &mut ctx);
        // second check-in should abort
        let res = test::should_abort(E_ALREADY_CHECKED_IN, fun() {
            attendance::check_in(&student, test::get_registry_object_id(), 1,
201, &mut ctx);
        });
        assert!(res);
    }
}
```

Tests above use pseudocode test harness functions (`test::create_signer`, `test::tx_context`, `test::get_registry_object_id`, `test::should_abort`) — replace them with the exact Sui test helpers (e.g., `sui::test::run_test`, `TxContext::new()`, etc.) from your environment. Use `sui move test` to run tests.

---

3) React + TypeScript Frontend (client)

Assumptions - You use `create-react-app` or Vite. Below uses Vite with React + TypeScript. - You have `@mysten/dapp-kit` (dapp-kit) and `@mysten/sui.js` installed.

package.json (relevant deps)

```
{
  "dependencies": {
    "react": "^18.x",
    "react-dom": "^18.x",
    "@mysten/dapp-kit": "^1.x",
    "@mysten/sui.js": "^1.x",
    "qrcode.react": "^1.x",
    "react-qr-reader": "^3.x"
```

```
  }
}
```

src/main.tsx

```
import React from 'react'
import { createRoot } from 'react-dom/client'
import App from './App'
import { DAppKitProvider } from '@mysten/dapp-kit'

createRoot(document.getElementById('root')!).render(
  <DAppKitProvider>
    <App />
  </DAppKitProvider>
)
```

src/App.tsx

```
import React, { useEffect, useState } from 'react'
import { ConnectButton, useWallet } from '@mysten/dapp-kit'
import { JsonRpcProvider, TransactionBlock } from '@mysten/sui.js'
import QRReader from 'react-qr-reader'

const provider = new JsonRpcProvider({ fullnode: 'https://
fullnode.devnet.sui.io:443' })
const REGISTRY_OBJECT_ID = '0x...'; // set after deploying Move package

export default function App() {
  const { account, signAndExecuteTransactionBlock } = useWallet();
  const [scanResult, setScanResult] = useState<string | null>(null)
  const [badges, setBadges] = useState<any[]>([])

  useEffect(() => {
    if (account?.address) fetchBadges();
  }, [account])

  async function fetchBadges() {
    if (!account) return;
    // call provider to get objects owned by account and filter by
AttendanceBadge type
    const objs = await provider.getOwnedObjects({ owner: account.address })
    const badgesObjs = objs.data.filter(o => o.type &&
o.type.includes('attendance::AttendanceBadge'))
    setBadges(badgesObjs)
  }

  async function onScan(data: string | null) {
    if (data) {
      setScanResult(data);
```

```
      // data expected: JSON with event_id and reg_obj_id
      try {
        const info = JSON.parse(data);
        await submitCheckIn(info.event_id)
      } catch (e) {
        console.error('invalid QR payload', e)
      }
    }
  }

  async function submitCheckIn(eventId: number) {
    if (!account) return alert('connect wallet');
    // Build tx block calling Move function: attendance::check_in
    const tx = new TransactionBlock();
    // The exact call shape depends on package/object ids; using generic
'moveCall'
    tx.moveCall({
      target: '0x1::attendance::check_in',
      arguments: [tx.pure(REGISTRY_OBJECT_ID), tx.pure(eventId),
tx.pure(Math.floor(Date.now()/1000))],
    })
    const result = await signAndExecuteTransactionBlock({ transactionBlock:
tx })
    console.log('tx result', result)
    await fetchBadges()
  }

  return (
    <div className="p-4">
      <h1>School Event Check-In</h1>
      <ConnectButton />
      <section>
        <h2>Scan QR to Check-In</h2>
        <QRReader
          onResult={(result, error) => { if (result) onScan(result?.text); }}
          constraints={{ facingMode: 'environment' }}
        />
        {scanResult && <div>Scanned: {scanResult}</div>}
      </section>

      <section>
        <h2>Your Badges</h2>
        {badges.length === 0 ? <div>No badges found</div> : (
          <ul>
            {badges.map(b => <li key={b.objectId}>{b.objectId} — {b.type}</
li>)}
          </ul>
        )}
      </section>
    </div>
```

```
  )
}
```

Notes - Replace `REGISTRY_OBJECT_ID` with the actual shared registry id after deploying the Move package. You can display it in the CLI output or explorer. - `tx.moveCall` arguments ordering and types must match the compiled Move function ABI. Use `sui client` or `sui.js` helpers to inspect the package and method signature.

---

4) Deployment & Local testing notes

1. Install Sui CLI & Tooling: `npm i -g @mysten/sui` (or follow Sui docs). See Sui docs for Move package creation. More on `sui move` and `sui move test` in Sui docs. cite turn0search3 turn0search13
2. Create Move package: `sui move new attendance` then add module, adjust `Move.toml`.
3. Compile: `sui move build`.
4. Publish to local devnet or testnet: `sui client publish --gas-budget 10000` (check exact CLI flags) — note: publishing returns package id and object ids (registry id if created). Use that in frontend. See Sui docs for shared objects and Table. cite turn0search11 turn0search3
5. Run Move tests: `sui move test` (adapt harness). See Sui Move testing guides. cite turn0search20
6. Frontend: `npm install && npm run dev` (Vite) and configure `REGISTRY_OBJECT_ID` and RPC endpoint.

---

5) What's next / How I can help now

Pick one: - I can **produce a fully compiling Sui Move package** with exact signatures tailored to Sui CLI version you have (tell me Sui CLI version / target: testnet/devnet). - Or I can **generate the full React repo** in this document as downloadable files.

Spune-mi ce opțiune preferi și îți trimit imediat fișierele gata de copiat.

References / docs used: - Sui Table module docs. cite turn0search3 - Sui shared objects guide. cite turn0search11 - dApp Kit (Sui) docs. cite turn0search1 turn0search4