```
#!/bin/bash
# One-click setup script for JChain/JBits repo

BASE="jchain_stack"

echo "Creating repo structure..."
mkdir -p "$BASE"/{core,instruments,settlement,interface,auth,rate_limit,storage}

echo "Creating core files..."
cat > "$BASE/core/ledger.h" << 'EOF'
#pragma once
#include <string>

enum class Status { PENDING, CLEARED, SETTLED };
enum class Dir { DEBIT, CREDIT };

struct Entry {
    std::string tx_id;
    int ref_id;  // ⁝13⁝
    double amount;
    Dir dir;
    Status status;
    std::string ts;
};
EOF

cat > "$BASE/core/reference.h" << 'EOF'
#pragma once
struct Reference {
    int ref_id;  // ⁝13⁝
    bool active;
};
EOF

cat > "$BASE/core/storage_sqlite.h" << 'EOF'
#pragma once
#include <sqlite3.h>
#include <string>
#include <stdexcept>
#include "ledger.h"

struct StorageSQLite {
    sqlite3* db;
    StorageSQLite(const std::string& path) {
        if (sqlite3_open(path.c_str(), &db)) {
            throw std::runtime_error("Cannot open SQLite database");
        }

        const char* ledger_table =
            "CREATE TABLE IF NOT EXISTS ledger("
            "tx_id TEXT PRIMARY KEY,"
            "ref_id INTEGER,"
            "amount REAL,"
            "dir TEXT,"
            "status TEXT,"
            "ts TEXT);";

        const char* balances_table =
            "CREATE TABLE IF NOT EXISTS balances("
            "ref_id INTEGER PRIMARY KEY,"
            "available REAL,"
            "pending REAL,"
            "last_sync TEXT);";

        sqlite3_exec(db, ledger_table, nullptr, nullptr, nullptr);
        sqlite3_exec(db, balances_table, nullptr, nullptr, nullptr);
    }

    void appendLedger(const Entry& e) {
        std::string sql = "INSERT INTO ledger(tx_id, ref_id, amount, dir, status, ts) VALUES('" +
            e.tx_id + "'," +
            std::to_string(e.ref_id) + "," +
            std::to_string(e.amount) + ",'" +
            (e.dir == Dir::CREDIT ? "CREDIT" : "DEBIT") + "','" +
            (e.status == Status::PENDING ? "PENDING" :
             e.status == Status::CLEARED ? "CLEARED" : "SETTLED") + "','" +
            e.ts + "');";
        sqlite3_exec(db, sql.c_str(), nullptr, nullptr, nullptr);
    }

    ~StorageSQLite() {
        sqlite3_close(db);
    }
};
EOF

echo "Creating instrument files..."
```

```
cat > "$BASE/instruments/ach_event.h" << 'EOF'
#pragma once
#include <string>

struct AchEvent {
    std::string event_id;
    int ref_id;
    double amount;
    std::string status; // RECEIVED, CLEARED
};
EOF

cat > "$BASE/instruments/card_event.h" << 'EOF'
#pragma once
#include <string>

struct CardEvent {
    std::string event_id;
    int ref_id;
    double amount;
    std::string token;
    std::string status; // PENDING, CLEARED
};
EOF

cat > "$BASE/instruments/check_event.h" << 'EOF'
#pragma once
#include <string>

struct CheckEvent {
    std::string check_id;
    int ref_id;
    double amount;
    std::string image_hash;
    std::string status; // PENDING, CLEARED
};
EOF

echo "Creating settlement file..."
cat > "$BASE/settlement/settle.h" << 'EOF'
#pragma once
#include "../core/ledger.h"
#include "../core/storage_sqlite.h"
#include <vector>
```

```
struct Settlement {
    StorageSQLite& storage;

    void settleBatch(int ref_id, std::vector<Entry>& entries) {
        for (auto& e : entries) {
            if(e.status == Status::CLEARED){
                e.status = Status::SETTLED;
                storage.appendLedger(e);
            }
        }
    }
};
EOF

echo "Creating auth file..."
cat > "$BASE/auth/api_key.h" << 'EOF'
#pragma once
#include <string>
#include <unordered_set>

struct ApiAuth {
    std::unordered_set<std::string> valid_keys
{"sk-proj-rx10IyNzVrd5ljczsPCz-SUpJDjJHJgEaspna4hs1bqxaFO8okzGNpd1BtUuvLUbSHEjz2
po5ZT3BlbkFJmBm4P1opX8vvynRISEem_tfFwJjyohxZUwSkdtGNuhUs0SChbeqCal937RMtvG
OTekC0nbnZgA
"};

    bool validate(const std::string& key){
        return valid_keys.find(key) != valid_keys.end();
    }
};
EOF

echo "Creating rate limiter..."
cat > "$BASE/rate_limit/limiter.h" << 'EOF'
#pragma once
#include <unordered_map>
#include <chrono>

struct RateLimiter {
    std::unordered_map<std::string,int> counter;
    std::chrono::steady_clock::time_point last_reset = std::chrono::steady_clock::now();
    int limit = 10;
    int window_sec = 60;

    bool allow(const std::string& key){
        auto now = std::chrono::steady_clock::now();
        if(std::chrono::duration_cast<std::chrono::seconds>(now - last_reset).count() >
window_sec){
            counter.clear();
            last_reset = now;
        }
        counter[key]++;
        return counter[key] <= limit;
    }
};
EOF

echo "Creating interface files..."
cat > "$BASE/interface/cli.cpp" << 'EOF'
#include <iostream>
#include "../core/storage_sqlite.h"
#include "../core/ledger.h"
#include <ctime>

int runCLI() {
    StorageSQLite storage("storage/jchain.db");
    std::string cmd;

    while(true){
        std::cout << "Command (add/show/exit): ";
        std::cin >> cmd;

        if(cmd == "add"){
            std::string tx_id;
            double amount;
            std::cout << "TX ID: "; std::cin >> tx_id;
            std::cout << "Amount: "; std::cin >> amount;

            Entry e;
            e.tx_id = tx_id;
            e.ref_id = 13;
            e.amount = amount;
            e.dir = Dir::CREDIT;
            e.status = Status::PENDING;
            e.ts = std::to_string(time(0));
            storage.appendLedger(e);
            std::cout << "Entry added to SQLite.\n";
        }
        else if(cmd == "show"){
            sqlite3_stmt* stmt;
            const char* query = "SELECT tx_id, ref_id, amount, dir, status, ts FROM ledger;";
            sqlite3_prepare_v2(storage.db, query, -1, &stmt, nullptr);

            while(sqlite3_step(stmt) == SQLITE_ROW){
                std::cout << sqlite3_column_text(stmt,0) << ","
                          << sqlite3_column_int(stmt,1) << ","
                          << sqlite3_column_double(stmt,2) << ","
                          << sqlite3_column_text(stmt,3) << ","
                          << sqlite3_column_text(stmt,4) << ","
                          << sqlite3_column_text(stmt,5) << "\n";
            }
            sqlite3_finalize(stmt);
        }
        else if(cmd == "exit") break;
    }

    return 0;
}
EOF

cat > "$BASE/interface/http_server.cpp" << 'EOF'
#include "httplib.h"
#include "../core/storage_sqlite.h"
#include "../core/ledger.h"
#include "../auth/api_key.h"
#include "../rate_limit/limiter.h"
#include <iostream>

void runHttp(StorageSQLite& storage){
    httplib::Server svr;
    ApiAuth auth;
    RateLimiter limiter;

    svr.Post("/event/card", [&](const httplib::Request &req, httplib::Response &res){
        auto key =
req.get_header_value("sk-proj-rx10lyNzVrd5IjczsPCz-SUpJDjJHJgEaspna4hs1bqxaFO8okzGN
pd1BtUuvLUbSHEjz2po5ZT3BlbkFJmBm4P1opX8vvynRISEem_tfFwJjyohxZUwSkdtGNuhUs0
SChbeqCal937RMtvGOTekC0nbnZgA
");
```

```
        if(!auth.validate(key)){
            res.status = 401; return;
        }
        if(!limiter.allow(key)){
            res.status = 429; return;
        }

        std::string tx_id = "tx_" + std::to_string(time(0));
        double amount = std::stod(req.body);

        Entry e;
        e.tx_id = tx_id;
        e.ref_id = 13;
        e.amount = amount;
        e.dir = Dir::CREDIT;
        e.status = Status::PENDING;
        e.ts = std::to_string(time(0));

        storage.appendLedger(e);
        res.set_content("OK", "text/plain");
    });

    svr.Get("/balance", [&](const httplib::Request &req, httplib::Response &res){
        auto key = req.get_header_value("API-Key");
        if(!auth.validate(key)){
            res.status = 401; return;
        }
        if(!limiter.allow(key)){
            res.status = 429; return;
        }

        sqlite3_stmt* stmt;
        const char* query = "SELECT available,pending FROM balances WHERE ref_id=13;";
        sqlite3_prepare_v2(storage.db, query, -1, &stmt, nullptr);
        if(sqlite3_step(stmt) == SQLITE_ROW){
            double avail = sqlite3_column_double(stmt,0);
            double pend = sqlite3_column_double(stmt,1);
            res.set_content("available: " + std::to_string(avail) + " pending: " + std::to_string(pend),
"text/plain");
        }
        sqlite3_finalize(stmt);
    });

    svr.listen("0.0.0.0", 8080);
```

```
}
EOF

echo "Creating main.cpp..."
cat > "$BASE/main.cpp" << 'EOF'
#include <thread>
#include "interface/cli.cpp"
#include "interface/http_server.cpp"
#include "core/storage_sqlite.h"

int main(){
    StorageSQLite storage("storage/jchain.db");
    std::thread http_thread(runHttp, std::ref(storage));

    runCLI();

    http_thread.join();
    return 0;
}
EOF

echo "Setup complete! Your JChain/JBits repo is ready."
```