?????????? ??????????: ?????????? ?????- ?????????? ??? ERACOIN

- ПОШАГОВАЯ ИНСТРУКЦИЯ: ПРИМЕНЕНИЕ СМАРТ-КОНТРАКТА ЦФА ERACOIN

?????????? ???????????:
??????????? ?????-??????????
??? ERACOIN

==========

==========

==========

?????????? ???????????:
??????????? ?????-??????????
??? ERACOIN

? ?????????? ???????
(ERACHAIN)

==========

==========

==========

API ?????????????: https://app.swaggerhub.com/apis-docs/Erachain/era-api/1.0.0-oas3

NPM ?????: erachain-js-api

GitHub: https://github.com/erachain

==========

==========

==========

??

??? 0. ?????????? ?????????

???

0.1. ????????? ????????????

```
# Создаём проект
mkdir eracoin-cfa-project
cd eracoin-cfa-project
npm init -y

# Устанавливаем официальный API-клиент Эрачейн
npm install erachain-js-api

# Дополнительные пакеты
npm install axios crypto-js
```

0.2. ????????? ???????

```
eracoin-cfa-project/
├── src/
│   ├── contracts/
│   │   └── eracoin_contract.js      # Смарт-контракт
│   ├── config/
│   │   └── network.js               # Настройки сети
│   ├── utils/
│   │   └── crypto.js                # Криптографические утилиты
│   └── deploy.js                    # Скрипт деплоя
├── test/
│   └── contract.test.js             # Тесты
├── .env                             # Переменные окружения
└── package.json
```

0.3. ???? ????????????? ???? (src/config/network.js)

```
// ================================================================
// КОНФИГУРАЦИЯ СЕТЕЙ ЭРАЧЕЙН
// ================================================================

const NETWORKS = {
    // TESTNET — для разработки и тестирования
    testnet: {
        name: 'Erachain Testnet',
        baseUrl: 'https://testnet.erachain.org/api',  // API endpoint
        explorer: 'https://testnet.erachain.org',
        chainId: 'testnet',
        // Параметры ноды
        node: {
            host: 'testnet.erachain.org',
            port: 9067,           // Порт API
            ssl: true
        },
        // Параметры блокчейна
        blockTime: 30,            // 30 секунд на блок
        confirmations: 1,         // Подтверждений для финальности
    },

    // MAINNET — боевой запуск
    mainnet: {
        name: 'Erachain Mainnet',
        baseUrl: 'https://mainnet.erachain.org/api',
        explorer: 'https://mainnet.erachain.org',
        chainId: 'mainnet',
        node: {
            host: 'mainnet.erachain.org',
            port: 9067,
            ssl: true
        },
        blockTime: 30,
        confirmations: 3,
```

```
    },

    // ЛОКАЛЬНАЯ НОДА (для разработки)
    local: {
        name: 'Local Erachain Node',
        baseUrl: 'http://localhost:9067/api',
        explorer: 'http://localhost:9067',
        chainId: 'local',
        node: {
            host: 'localhost',
            port: 9067,
            ssl: false
        },
        blockTime: 30,
        confirmations: 1,
    }
};

module.exports = { NETWORKS };
```

??

??? 1. ??????????? ?????-?????????

??

??????????

1.1. ?????????????? ???????? ??? ??????? ? ???? script

```
// ================================================================
// СМАРТ-КОНТРАКТ ЦФА "ERACOIN" — ВЕРСИЯ ДЛЯ ВСТАВКИ В API
// ================================================================
// Этот код вставляется в поле "script" при создании актива через API

const ERACOIN_CONTRACT_SCRIPT = `
// === КОНФИГУРАЦИЯ ===
const CFG = {
    NAME: "Цифровые облигации Eracoin серия 001",
    TICKER: "ERACOIN001",
    NOMINAL: 1000,
    COUPON_RATE: 0.14,
    COUPON_PERIOD: 3,
    MATURITY_MONTHS: 24,
    MIN_INVEST: 100000,
    MAX_INVEST: 10000000,
    ONLY_QUALIFIED: true,
    EMITTER: "EMITTER_ADDRESS_PLACEHOLDER",
    SETTLEMENT: "SETTLEMENT_ACCOUNT_PLACEHOLDER"
};

// === СОСТОЯНИЕ ===
const ST = {
    holders: {},
    coupons: [],
    status: "CREATED",
    issued: 0,
    burned: 0,
    totalPaid: 0,
    frozen: {},
    votes: {}
```

```
};

// === МОДИФИКАТОРЫ ===
function onlyEmitter() {
    if (msg.sender !== CFG.EMITTER) throw "ACCESS_DENIED";
}
function onlyOperator() {
    if (!isAuthOp(msg.sender)) throw "NOT_OPERATOR";
}
function onlyActive() {
    if (ST.status !== "ACTIVE" && ST.status !== "COUPON_PAID") throw "NOT_ACTIVE";
}
function notFrozen(addr) {
    if (ST.frozen[addr] && ST.frozen[addr].until > block.timestamp) throw "FROZEN";
}
function onlyQualified(addr) {
    if (CFG.ONLY_QUALIFIED && !ST.holders[addr]?.qualified) throw "NOT_QUALIFIED";
}

// === KYC / РЕЕСТР ===
function registerInvestor(addr, kyc) {
    onlyOperator();
    if (ST.holders[addr]) throw "ALREADY_REGISTERED";
    ST.holders[addr] = {
        balance: 0,
        kycVerified: true,
        qualified: kyc.qualified || false,
        regDate: block.timestamp,
        totalReceived: 0,
        lastCoupon: null
    };
    emit("INVESTOR_REGISTERED", {addr, qualified: kyc.qualified});
    return {success: true};
}

function updateQualification(addr, isQ) {
    onlyOperator();
    if (!ST.holders[addr]) throw "NOT_FOUND";
    ST.holders[addr].qualified = isQ;
    emit("QUALIFICATION_UPDATED", {addr, qualified: isQ});
```

```
    return {success: true};
}

// === ЭМИССИЯ ===
function primaryPlacement(recipients) {
    onlyEmitter();
    if (ST.status !== "CREATED") throw "ALREADY_PLACED";
    let total = 0;
    for (const r of recipients) {
        const {address, amount} = r;
        if (!ST.holders[address]) throw "NOT_REGISTERED";
        onlyQualified(address);
        notFrozen(address);
        const newBal = ST.holders[address].balance + amount;
        const invest = newBal * CFG.NOMINAL;
        if (invest < CFG.MIN_INVEST) throw "MIN_INVESTMENT";
        if (invest > CFG.MAX_INVEST) throw "MAX_INVESTMENT";
        ST.holders[address].balance = newBal;
        total += amount;
    }
    if (total > 50000) throw "OVER_ISSUE";
    ST.issued = total;
    ST.status = "ACTIVE";
    emit("PLACEMENT_COMPLETE", {totalIssued: total});
    return {success: true, totalIssued: total};
}

// === ПЕРЕВОДЫ ===
function transfer(from, to, amount) {
    onlyActive();
    if (msg.sender !== from && msg.sender !== CFG.EMITTER) throw "ACCESS_DENIED";
    notFrozen(from);
    notFrozen(to);
    if (!ST.holders[to]) throw "NOT_REGISTERED";
    onlyQualified(to);
    if (ST.holders[from].balance < amount) throw "INSUFFICIENT";
    const newTo = ST.holders[to].balance + amount;
    if (newTo * CFG.NOMINAL > CFG.MAX_INVEST) throw "MAX_INVEST";
    ST.holders[from].balance -= amount;
    ST.holders[to].balance = newTo;
```

```
    emit("TRANSFER", {from, to, amount});
    return {success: true};
}

// === КУПОНЫ ===
function calcCoupon(addr) {
    const h = ST.holders[addr];
    if (!h || h.balance <= 0) return 0;
    return Math.floor(h.balance * CFG.NOMINAL * CFG.COUPON_RATE / 4 * 100) / 100;
}

function payCoupons() {
    onlyEmitter();
    onlyActive();
    const now = block.timestamp;
    const last = ST.coupons.length > 0 ? ST.coupons[ST.coupons.length - 1].date :
CFG.ISSUE_DATE;
    const days = (now - last) / 86400;
    if (days < 85) throw "TOO_EARLY";
    let totalPaid = 0, count = 0;
    const details = [];
    for (const addr in ST.holders) {
        const h = ST.holders[addr];
        if (h.balance <= 0) continue;
        const amt = calcCoupon(addr);
        if (amt <= 0) continue;
        h.totalReceived += amt;
        h.lastCoupon = now;
        totalPaid += amt;
        count++;
        details.push({addr, amt, bal: h.balance});
    }
    ST.coupons.push({date: now, amount: totalPaid, holders: count, details, tx: tx.hash});
    ST.totalPaid += totalPaid;
    ST.status = "COUPON_PAID";
    emit("COUPONS_PAID", {total: totalPaid, count});
    return {success: true, totalPaid, count};
}

function onSchedule(date) {
```

```
    const next = getNextCouponDate();
    if (Math.abs(date - next) < 86400) {
        try { return payCoupons(); }
        catch(e) { emit("SCHEDULED_FAIL", {error: e}); return {success: false, error: e}; }
    }
    return {success: true, msg: "Not time"};
}

function getNextCouponDate() {
    const now = block.timestamp;
    const issue = CFG.ISSUE_DATE;
    const period = CFG.COUPON_PERIOD * 30 * 86400;
    let next = issue + period;
    while (next < now) next += period;
    return next;
}

// === ПОГАШЕНИЕ ===
function earlyRedemption(addr, amount) {
    onlyEmitter();
    onlyActive();
    notFrozen(addr);
    const h = ST.holders[addr];
    if (!h || h.balance < amount) throw "INSUFFICIENT";
    const nominal = amount * CFG.NOMINAL;
    const accrued = calcAccrued(addr);
    const total = nominal + accrued;
    h.balance -= amount;
    ST.burned += amount;
    emit("EARLY_REDEMPTION", {addr, amount, total});
    return {success: true, total};
}

function onMaturity() {
    const now = block.timestamp;
    if (now < CFG.MATURITY_DATE) throw "NOT_MATURED";
    if (ST.status === "MATURED") throw "ALREADY_MATURED";
    let total = 0, count = 0;
    for (const addr in ST.holders) {
        const h = ST.holders[addr];
```

```
        if (h.balance <= 0) continue;
        const amt = h.balance;
        const nominal = amt * CFG.NOMINAL;
        const finalCoupon = calcCoupon(addr);
        const payment = nominal + finalCoupon;
        h.balance = 0;
        ST.burned += amt;
        total += payment;
        count++;
        emit("MATURITY_REDEMPTION", {addr, amount: amt, payment});
    }
    ST.status = "MATURED";
    emit("MATURITY_COMPLETE", {total, count});
    return {success: true, total, count};
}

function calcAccrued(addr) {
    const h = ST.holders[addr];
    if (!h || h.balance <= 0) return 0;
    const days = (block.timestamp - (h.lastCoupon || CFG.ISSUE_DATE)) / 86400;
    return Math.floor(h.balance * CFG.NOMINAL * CFG.COUPON_RATE * days / 365 * 100) / 100;
}

// === ЗАМОРОЗКА ===
function freezeAddress(addr, until, reason) {
    onlyEmitter();
    ST.frozen[addr] = {until, reason, at: block.timestamp};
    emit("FROZEN", {addr, until, reason});
    return {success: true};
}
function unfreezeAddress(addr) {
    onlyEmitter();
    delete ST.frozen[addr];
    emit("UNFROZEN", {addr});
    return {success: true};
}

// === ГОЛОСОВАНИЯ ===
function createVote(vid, topic, options, deadline) {
    onlyEmitter();
```

```
    if (ST.votes[vid]) throw "VOTE_EXISTS";
    ST.votes[vid] = {topic, options, votes: {}, deadline, created: block.timestamp, status:
"ACTIVE"};
    emit("VOTE_CREATED", {vid, topic, deadline});
    return {success: true};
}
function castVote(vid, optIdx) {
    onlyActive();
    notFrozen(msg.sender);
    const v = ST.votes[vid];
    if (!v) throw "VOTE_NOT_FOUND";
    if (block.timestamp > v.deadline) throw "VOTE_CLOSED";
    const h = ST.holders[msg.sender];
    if (!h || h.balance <= 0) throw "NO_BALANCE";
    if (optIdx < 0 || optIdx >= v.options.length) throw "INVALID_OPTION";
    v.votes[msg.sender] = {optIdx, weight: h.balance, at: block.timestamp};
    emit("VOTE_CAST", {vid, voter: msg.sender, optIdx, weight: h.balance});
    return {success: true};
}
function tallyVotes(vid) {
    onlyEmitter();
    const v = ST.votes[vid];
    if (!v) throw "VOTE_NOT_FOUND";
    if (block.timestamp <= v.deadline) throw "VOTE_OPEN";
    const res = {};
    for (let i = 0; i < v.options.length; i++) res[i] = 0;
    let total = 0;
    for (const voter in v.votes) {
        const vote = v.votes[voter];
        res[vote.optIdx] += vote.weight;
        total += vote.weight;
    }
    v.status = "CLOSED";
    v.results = res;
    v.totalVotes = total;
    emit("VOTE_TALLIED", {vid, results: res, total});
    return {success: true, results: res, total};
}

// === VIEW ФУНКЦИИ ===
```

```
function getBalance(addr) {
    return ST.holders[addr]?.balance || 0;
}
function getHolderInfo(addr) {
    const h = ST.holders[addr];
    if (!h) return null;
    return {
        balance: h.balance,
        qualified: h.qualified,
        totalReceived: h.totalReceived,
        currentCoupon: calcCoupon(addr),
        accrued: calcAccrued(addr),
        frozen: ST.frozen[addr] ? true : false
    };
}
function getAssetInfo() {
    return {
        name: CFG.NAME,
        ticker: CFG.TICKER,
        nominal: CFG.NOMINAL,
        quantity: 50000,
        issued: ST.issued,
        burned: ST.burned,
        circulating: ST.issued - ST.burned,
        couponRate: CFG.COUPON_RATE,
        status: ST.status,
        totalPaid: ST.totalPaid,
        nextCoupon: getNextCouponDate()
    };
}

// === ВСПОМОГАТЕЛЬНЫЕ ===
function isAuthOp(addr) {
    // Проверка авторизации оператора ИС
    return true; // Заменить на реальную проверку
}
function emit(event, data) {
    log("EracoinEvent", {event, data, block: block.number, time: block.timestamp});
}
```

```
// === ОБРАБОТЧИКИ СОБЫТИЙ ===
function onBlock() {
    const now = block.timestamp;
    if (now >= CFG.MATURITY_DATE && ST.status !== "MATURED") {
        try { onMaturity(); } catch(e) { emit("MATURITY_FAIL", {error: e}); }
    }
    const next = getNextCouponDate();
    if (Math.abs(now - next) < 86400 && ST.status === "ACTIVE") {
        try { payCoupons(); } catch(e) { emit("COUPON_FAIL", {error: e}); }
    }
    for (const addr in ST.frozen) {
        if (ST.frozen[addr].until <= now) {
            delete ST.frozen[addr];
            emit("AUTO_UNFROZEN", {addr});
        }
    }
}

// === ИНИЦИАЛИЗАЦИЯ ===
function init(params) {
    CFG.ISSUE_DATE = block.timestamp;
    CFG.MATURITY_DATE = block.timestamp + 730 * 86400;
    if (params.emitter) CFG.EMITTER = params.emitter;
    if (params.settlement) CFG.SETTLEMENT = params.settlement;
    emit("ASSET_CREATED", {name: CFG.NAME, ticker: CFG.TICKER});
    return {success: true, assetId: CFG.TICKER + "_" + block.number};
}

module.exports = {init, registerInvestor, updateQualification, primaryPlacement, transfer,
payCoupons, onSchedule, earlyRedemption, onMaturity, freezeAddress, unfreezeAddress,
createVote, castVote, tallyVotes, getBalance, getHolderInfo, getAssetInfo, onBlock};
`;

module.exports = { ERACOIN_CONTRACT_SCRIPT };
```

??????????????????????????
??????????????????????????

??????????

??? 2. ?????? ?????? ????? API ???????

???

2.1. ???????? ?????? ?????? (src/deploy.js)

```
// ================================================================
// СКРИПТ ДЕПЛОЯ ЦФА ERACOIN В БЛОКЧЕЙН ЭРАЧЕЙН
// ================================================================

const axios = require('axios');
const crypto = require('crypto-js');
const { NETWORKS } = require('./config/network');
const { ERACOIN_CONTRACT_SCRIPT } = require('./contracts/eracoin_contract');

// === КОНФИГУРАЦИЯ ===
const CONFIG = {
    // Выбор сети: 'testnet' | 'mainnet' | 'local'
    network: process.env.ERACHAIN_NETWORK || 'testnet',

    // Ключи эмитента (хранить в .env!)
    emitterSeed: process.env.EMITTER_SEED,
    emitterAddress: process.env.EMITTER_ADDRESS,
```

```
    // Параметры актива
    asset: {
        name: "Цифровые облигации Eracoin серия 001",
        ticker: "ERACOIN001",
        quantity: 50000,
        scale: 2,
        description: "Проспект: https://example.com/eracoin-prospectus | Купон 14% годовых |
Срок 24 мес.",
        movable: true,
        divisible: true
    }
};

// === КЛАСС API КЛИЕНТА ===
class ErachainAPI {
    constructor(networkConfig) {
        this.baseUrl = networkConfig.baseUrl;
        this.node = networkConfig.node;
    }

    /**
     * Получение информации о ноде
     */
    async getNodeInfo() {
        const response = await axios.get(`${this.baseUrl}/node/info`);
        return response.data;
    }

    /**
     * Получение баланса адреса
     */
    async getBalance(address) {
        const response = await axios.get(`${this.baseUrl}/addresses/balance/${address}`);
        return response.data;
    }

    /**
     * СОЗДАНИЕ АКТИВА (Asset) со смарт-контрактом
     * POST /api/assets
```

```
     */
    async createAsset(assetParams, seed) {
        const payload = {
            // Основные параметры актива
            creator: assetParams.emitterAddress,
            name: assetParams.name,
            ticker: assetParams.ticker,
            quantity: assetParams.quantity,
            scale: assetParams.scale,
            description: assetParams.description,
            movable: assetParams.movable,
            divisible: assetParams.divisible,

            // === СМАРТ-КОНТРАКТ ===
            // Вставляем JS-код в поле script
            script: assetParams.script,

            // Параметры инициализации контракта
            scriptParams: {
                emitter: assetParams.emitterAddress,
                settlement: assetParams.settlementAccount
            },

            // Комиссия
            fee: 1000000,  // В минимальных единицах ERACHAIN

            // Подпись
            timestamp: Date.now()
        };

        // Подпись транзакции (пример — реальная реализация зависит от криптографии Эрачейн)
        const signature = this.signTransaction(payload, seed);
        payload.signature = signature;

        const response = await axios.post(`${this.baseUrl}/assets`, payload, {
            headers: {
                'Content-Type': 'application/json',
                'X-API-Key': process.env.ERACHAIN_API_KEY
            }
        });
```

```
    return response.data;
}

/**
 * Вызов функции смарт-контракта
 * POST /api/assets/{assetId}/call
 */
async callContract(assetId, functionName, params, seed) {
    const payload = {
        assetId: assetId,
        function: functionName,
        params: params,
        caller: CONFIG.emitterAddress,
        timestamp: Date.now()
    };

    const signature = this.signTransaction(payload, seed);
    payload.signature = signature;

    const response = await axios.post(
        `${this.baseUrl}/assets/${assetId}/call`,
        payload,
        { headers: { 'Content-Type': 'application/json' } }
    );

    return response.data;
}

/**
 * Получение информации об активе
 * GET /api/assets/{assetId}
 */
async getAsset(assetId) {
    const response = await axios.get(`${this.baseUrl}/assets/${assetId}`);
    return response.data;
}

/**
 * Получение состояния смарт-контракта
```

```
     * GET /api/assets/{assetId}/state
     */
    async getContractState(assetId) {
        const response = await axios.get(`${this.baseUrl}/assets/${assetId}/state`);
        return response.data;
    }

    /**
     * Подпись транзакции (заглушка — заменить на реальную криптографию Эрачейн)
     */
    signTransaction(payload, seed) {
        // В реальности используется Ed25519 или аналогичный алгоритм
        const data = JSON.stringify(payload);
        return crypto.SHA256(data + seed).toString();
    }
}

// === ОСНОВНАЯ ЛОГИКА ДЕПЛОЯ ===
async function deployEracoin() {
    console.log("╔═══════════════════════════════════════════════════════════════╗");
    console.log("║     ДЕПЛОЙ ЦФА ERACOIN В БЛОКЧЕЙН ЭРАЧЕЙН                     ║");
    console.log("╚═══════════════════════════════════════════════════════════════╝\n");

    // 1. Инициализация API
    const network = NETWORKS[CONFIG.network];
    const api = new ErachainAPI(network);

    console.log(`[1/8] Подключение к сети: ${network.name}`);
    console.log(`       URL: ${network.baseUrl}`);

    // 2. Проверка подключения
    try {
        const nodeInfo = await api.getNodeInfo();
        console.log(`[2/8] ✓ Нода доступна. Версия: ${nodeInfo.version}`);
        console.log(`       Высота блока: ${nodeInfo.height}`);
        console.log(`       Время блока: ${nodeInfo.blockTime}s`);
    } catch (e) {
        console.error(`[2/8] ✗ Ошибка подключения: ${e.message}`);
        process.exit(1);
    }
```

```
// 3. Проверка баланса эмитента
console.log(`[3/8] Проверка баланса эмитента: ${CONFIG.emitterAddress}`);
try {
    const balance = await api.getBalance(CONFIG.emitterAddress);
    console.log(`       Баланс: ${balance.balance} ERACHAIN`);
    if (balance.balance < 1000000) {
        console.error(`       ✗ Недостаточно средств для комиссии`);
        process.exit(1);
    }
    console.log(`       ✓ Баланс достаточен`);
} catch (e) {
    console.error(`       ✗ Ошибка: ${e.message}`);
    process.exit(1);
}

// 4. Подготовка смарт-контракта
console.log(`[4/8] Подготовка смарт-контракта...`);
const script = ERACOIN_CONTRACT_SCRIPT
    .replace('EMITTER_ADDRESS_PLACEHOLDER', CONFIG.emitterAddress)
    .replace('SETTLEMENT_ACCOUNT_PLACEHOLDER', process.env.SETTLEMENT_ACCOUNT || '');
console.log(`       ✓ Контракт подготовлен (${script.length} символов)`);

// 5. Создание актива
console.log(`[5/8] Создание актива в блокчейне...`);
console.log(`       Название: ${CONFIG.asset.name}`);
console.log(`       Тикер: ${CONFIG.asset.ticker}`);
console.log(`       Объём: ${CONFIG.asset.quantity} шт.`);
console.log(`       Номинал: 1000 RUB`);
console.log(`       Купон: 14% годовых`);

let assetResult;
try {
    assetResult = await api.createAsset({
        ...CONFIG.asset,
        emitterAddress: CONFIG.emitterAddress,
        settlementAccount: process.env.SETTLEMENT_ACCOUNT,
        script: script
    }, CONFIG.emitterSeed);
```

```
    console.log(`       ✓ Актив создан!`);
    console.log(`       Asset ID: ${assetResult.assetId}`);
    console.log(`       Tx Hash: ${assetResult.txHash}`);
    console.log(`       Block: ${assetResult.blockNumber}`);
} catch (e) {
    console.error(`       ✗ Ошибка создания: ${e.message}`);
    if (e.response) {
        console.error(`       Детали: ${JSON.stringify(e.response.data)}`);
    }
    process.exit(1);
}

// 6. Ожидание подтверждения
console.log(`[6/8] Ожидание подтверждения (${network.confirmations} блоков)...`);
await new Promise(r => setTimeout(r, network.blockTime * network.confirmations * 1000));
console.log(`       ✓ Подтверждено`);

// 7. Проверка актива
console.log(`[7/8] Проверка актива в блокчейне...`);
try {
    const assetInfo = await api.getAsset(assetResult.assetId);
    console.log(`       ✓ Актив найден`);
    console.log(`       Статус: ${assetInfo.status}`);
    console.log(`       Скрипт: ${assetInfo.hasScript ? 'присутствует' : 'отсутствует'}`);

    const state = await api.getContractState(assetResult.assetId);
    console.log(`       Состояние контракта: ${JSON.stringify(state, null, 2)}`);
} catch (e) {
    console.error(`       ✗ Ошибка проверки: ${e.message}`);
}

// 8. Инициализация контракта
console.log(`[8/8] Инициализация смарт-контракта...`);
try {
    const initResult = await api.callContract(
        assetResult.assetId,
        'init',
        {
            emitter: CONFIG.emitterAddress,
            settlement: process.env.SETTLEMENT_ACCOUNT
```

```
            },
            CONFIG.emitterSeed
        );
        console.log(`       ✓ Контракт инициализирован`);
        console.log(`       Результат: ${JSON.stringify(initResult)}`);
    } catch (e) {
        console.error(`       ✗ Ошибка инициализации: ${e.message}`);
    }

    // Итог
    console.log("\n╔══════════════════════════════════════════════════════════════╗");
    console.log("║                    ДЕПЛОЙ ЗАВЕРШЁН                           ║");
    console.log("╠══════════════════════════════════════════════════════════════╣");
    console.log(`║  Asset ID:    ${assetResult.assetId.padEnd(47)} ║`);
    console.log(`║  Tx Hash:     ${assetResult.txHash.padEnd(47)} ║`);
    console.log(`║  Network:     ${CONFIG.network.padEnd(47)} ║`);
    console.log(`║  Explorer:    ${(network.explorer + '/tx/' +
assetResult.txHash).padEnd(47)} ║`);
    console.log("╚══════════════════════════════════════════════════════════════╝");

    return assetResult;
}

// Запуск
if (require.main === module) {
    deployEracoin().catch(console.error);
}

module.exports = { deployEracoin, ErachainAPI };
```

??

??? 3. ??????????? ????????? ? ???????????

??

3.1. ??????????? ??????????? (KYC)

```
// ================================================================
// РЕГИСТРАЦИЯ ИНВЕСТОРОВ (вызывается оператором ИС)
// ================================================================

async function registerInvestors(api, assetId, operatorSeed) {
    const investors = [
        {
            address: "7NhZBb8CeLtcD1aBcDeFgHiJkLmNoPqRsTuVwXyZ",
            kyc: {
                name: "Иванов Иван Иванович",
                passport: "4515 123456",
                inn: "770123456789",
                qualified: true  // Квалифицированный инвестор
            }
        },
        {
            address: "8MiACc9DfMueE2bCdEfGhIjKlMnOpQrStUvWxYz",
            kyc: {
                name: "Петрова Мария Сергеевна",
                passport: "4515 654321",
                inn: "770987654321",
                qualified: true
```

```
            }
        }
    ];

    for (const investor of investors) {
        console.log(`Регистрация инвестора: ${investor.kyc.name}`);

        const result = await api.callContract(
            assetId,
            'registerInvestor',
            {
                address: investor.address,
                kycData: investor.kyc
            },
            operatorSeed
        );

        console.log(`  Результат: ${result.success ? '✓' : '✗'}`);
    }
}
```

3.2. ?????????? ??????????

```
// ================================================================
// ПЕРВИЧНОЕ РАЗМЕЩЕНИЕ (вызывается эмитентом)
// ================================================================

async function primaryPlacement(api, assetId, emitterSeed) {
    const recipients = [
        {
            address: "7NhZBb8CeLtcD1aBcDeFgHiJkLmNoPqRsTuVwXyZ",
            amount: 10000  // 10 000 шт. × 1000 RUB = 10 000 000 RUB
        },
        {
            address: "8MiACc9DfMueE2bCdEfGhIjKlMnOpQrStUvWxYz",
            amount: 5000   // 5 000 шт. × 1000 RUB = 5 000 000 RUB
        }
    ];
```

```
    console.log("Первичное размещение ЦФА Eracoin...");

    const result = await api.callContract(
        assetId,
        'primaryPlacement',
        { recipients: recipients },
        emitterSeed
    );

    console.log(`Размещено: ${result.totalIssued} шт.`);
    console.log(`Статус актива: ${result.status}`);

    return result;
}
```

3.3. ??????? ???????

```
// ================================================================
// ВЫПЛАТА КУПОНОВ (автоматическая или ручная)
// ================================================================

async function payCoupons(api, assetId, emitterSeed) {
    console.log("Выплата купонов...");

    const result = await api.callContract(
        assetId,
        'payCoupons',
        {},
        emitterSeed
    );

    console.log(`Выплачено: ${result.totalPaid} RUB`);
    console.log(`Инвесторов: ${result.count}`);
    console.log(`Детали: ${JSON.stringify(result.paymentRecord, null, 2)}`);

    return result;
}
```

3.4. ???????? ??????????? ?? ??????

```
// ================================================================
// ПРОВЕРКА СОСТОЯНИЯ АКТИВА
// ================================================================

async function checkAssetInfo(api, assetId) {
    // Информация об активе
    const assetInfo = await api.getAsset(assetId);
    console.log("Информация об активе:", assetInfo);

    // Состояние контракта
    const state = await api.getContractState(assetId);
    console.log("Состояние контракта:", state);

    // Информация о конкретном владельце
    const holderInfo = await api.callContract(
        assetId,
        'getHolderInfo',
        { address: "7NhZBb8CeLtcD1aBcDeFgHiJkLmNoPqRsTuVwXyZ" },
        null  // view функция не требует подписи
    );
    console.log("Информация о владельце:", holderInfo);

    return { assetInfo, state, holderInfo };
}
```

??

??? 4. ???????? ????? EXPLORER / API

???

4.1. ???????? ????? Explorer

```
https://testnet.erachain.org/tx/{TX_HASH}
https://testnet.erachain.org/asset/{ASSET_ID}
```

4.2. ???????? ????? REST API

```
# Информация об активе
curl -X GET "https://testnet.erachain.org/api/assets/ERACOIN001"   -H "Content-Type:
application/json"

# Состояние смарт-контракта
curl -X GET "https://testnet.erachain.org/api/assets/ERACOIN001/state"   -H "Content-Type:
application/json"

# История транзакций актива
curl -X GET "https://testnet.erachain.org/api/assets/ERACOIN001/transactions"   -H "Content-
Type: application/json"

# Баланс владельца
curl -X GET
"https://testnet.erachain.org/api/addresses/balance/7NhZBb8CeLtcD1aBcDeFgHiJkLmNoPqRsTuVwXyZ"
  -H "Content-Type: application/json"
```

??

??? 5. ???????????? ?? TESTNET

??

5.1. ???????? ??????

```
// ============================================================
// ТЕСТИРОВАНИЕ КОНТРАКТА НА TESTNET
// ============================================================

const { deployEracoin, ErachainAPI } = require('./src/deploy');
const { NETWORKS } = require('./src/config/network');

async function runTests() {
    const network = NETWORKS.testnet;
    const api = new ErachainAPI(network);

    console.log("=== ТЕСТИРОВАНИЕ ЦФА ERACOIN ===\n");

    // Тест 1: Деплой
```

```
    console.log("Тест 1: Деплой контракта");
    const asset = await deployEracoin();

    // Тест 2: Регистрация инвестора
    console.log("\nТест 2: Регистрация инвестора");
    const regResult = await api.callContract(
        asset.assetId,
        'registerInvestor',
        {
            address: "TEST_ADDRESS_1",
            kycData: { qualified: true }
        },
        process.env.OPERATOR_SEED
    );
    console.log(regResult.success ? "✓ Инвестор зарегистрирован" : "✗ Ошибка");

    // Тест 3: Первичное размещение
    console.log("\nТест 3: Первичное размещение");
    const placeResult = await api.callContract(
        asset.assetId,
        'primaryPlacement',
        {
            recipients: [{ address: "TEST_ADDRESS_1", amount: 1000 }]
        },
        process.env.EMITTER_SEED
    );
    console.log(placeResult.success ? `✓ Размещено ${placeResult.totalIssued} шт.` : "✗
Ошибка");

    // Тест 4: Проверка баланса
    console.log("\nТест 4: Проверка баланса");
    const balance = await api.callContract(
        asset.assetId,
        'getBalance',
        { address: "TEST_ADDRESS_1" },
        null
    );
    console.log(`✓ Баланс: ${balance} ERACOIN`);

    // Тест 5: Расчёт купона
```

```
    console.log("\nТест 5: Расчёт купона");
    const coupon = await api.callContract(
        asset.assetId,
        'calculateCoupon',
        { address: "TEST_ADDRESS_1" },
        null
    );
    console.log(`✓ Купон: ${coupon} RUB`);

    console.log("\n=== ВСЕ ТЕСТЫ ЗАВЕРШЕНЫ ===");
}

runTests().catch(console.error);
```

??

??? 6. ??????? ?? MAINNET

??

6.1. ???-???? ????? mainnet

```
□ Пройден аудит смарт-контракта (внешний аудитор)
□ Тесты на testnet пройдены успешно (100%)
```

```
□ Подготовлены все юридические документы (проспект, договор с ОИС)
□ Получена лицензия ЦБ РФ на выпуск ЦФА
□ Выбран и зарегистрирован оператор ИС в реестре ЦБ РФ
□ Подготовлен расчётный счёт для купонных выплат
□ Настроен KYC/AML процесс
□ Подготовлена инфраструктура для инвесторов
□ Создана резервная копия приватных ключей
□ Настроен мониторинг и алерты
```

6.2. ??????? ??? mainnet

```
# Установка переменных окружения
export ERACHAIN_NETWORK=mainnet
export EMITTER_SEED="your-secret-seed-phrase"
export EMITTER_ADDRESS="your-emitter-address"
export SETTLEMENT_ACCOUNT="40802810..."
export ERACHAIN_API_KEY="your-api-key"

# Запуск деплоя
node src/deploy.js
```

??

??????????: API ENDPOINTS ??????? (SWAGGER)

??

???????? endpoints (?? ???????????? Swagger):

Method	Endpoint	Описание
GET	`/api/node/info`	Информация о ноде
GET	`/api/addresses/balance/{address}`	Баланс адреса
POST	`/api/assets`	Создание актива (с контрактом)
GET	`/api/assets/{assetId}`	Информация об активе
GET	`/api/assets/{assetId}/state`	Состояние контракта
POST	`/api/assets/{assetId}/call`	Вызов функции контракта
GET	`/api/assets/{assetId}/transactions`	История транзакций
POST	`/api/transactions`	Отправка транзакции
GET	`/api/blocks/{height}`	Информация о блоке
GET	`/api/blocks/last`	Последний блок

?????? ?????? API:

```
{
  "success": true,
  "data": { ... },
  "timestamp": 1717189200000,
  "blockHeight": 1234567
}
```