Plano de Evolução da Arquitetura de Software: Mark v1.1

Autor: Giovanni Lemos Barcelos
Status: Rascunho de Definição
Objetivo: Mitigar riscos críticos do MVP e implementar funcionalidades de escala e governança.

🛑 Módulo 0: Hardening do Core (Segurança e Atomicidade)

Prioridade Máxima. Nenhum recurso novo deve ser construído sobre uma fundação instável.

0.1. Refatoração da Autenticação (Security-First)

O Problema: Criptografia manual em JS no n8n é propensa a erros (ex: entropia fraca).
A Solução: Padronização criptográfica utilizando bibliotecas nativas do Node.js com parâmetros NIST.

Especificação Técnica:

1. **Algoritmo de Hashing:** Migrar para **PBKDF2** (Password-Based Key Derivation Function 2).
    - *Iterações:* Mínimo 100.000.
    - *Keylen:* 64 bytes.
    - *Digest:* sha512.
    - *Salt:* Gerado via crypto.randomBytes(16) para cada usuário.
2.
3. **Assinatura de JWT:**
    - Implementar rotação de chaves. Suportar JWT_SECRET_V1 e JWT_SECRET_V2 no ambiente para permitir troca de chaves sem deslogar todos os usuários abruptamente (grace period).
    - Validação estrita de exp (expiration) e iss (issuer).
4.
5. **Rate Limiting:**
    - Implementar no Nginx (camada 7) limitação de requisições no endpoint /auth/login para mitigar Brute Force (ex: 5 req/min por IP).
6.

0.2. Blindagem Transacional (Atomicidade)

O Problema: Orquestração de transações financeiras via múltiplos nós HTTP/n8n corre risco de inconsistência (estado zumbi).
A Solução: Delegar a integridade para a engine do PostgreSQL via **Stored Procedures**.

Nova Arquitetura de Resgate (redeem_voucher):

Criar uma função PL/pgSQL armazenada no banco que encapsula todo o ciclo financeiro. O n8n fará apenas **uma** chamada ao banco.

code SQL
downloadcontent_copy
expand_less

```
  -- DDL Sugerida para Stored Procedure
CREATE OR REPLACE FUNCTION process_redemption(
    p_student_id UUID,
    p_voucher_id UUID,
    p_cost INT
) RETURNS JSONB AS $$
DECLARE
    v_balance INT;
    v_redemption_id UUID;
BEGIN
    -- 1. Lock na linha do aluno para evitar Race Condition (Double Spending)
    SELECT marks_balance INTO v_balance FROM students WHERE id = p_student_id
FOR UPDATE;

    -- 2. Validação de Saldo
    IF v_balance < p_cost THEN
        RAISE EXCEPTION 'Saldo insuficiente';
    END IF;

    -- 3. Início da Transação
    -- Debitar
    UPDATE students SET marks_balance = marks_balance - p_cost WHERE id =
p_student_id;

    -- Registrar Voucher PENDING
    INSERT INTO redeemed_vouchers (student_id, voucher_catalog_id, status, cost)
    VALUES (p_student_id, p_voucher_id, 'PENDING', p_cost)
    RETURNING id INTO v_redemption_id;

    -- Registrar Ledger
    INSERT INTO ledger_transactions (student_id, type, amount, description,
source_redemption_id)
    VALUES (p_student_id, 'DEBIT', p_cost, 'Resgate de Voucher', v_redemption_id);

    -- Retorno Seguro
    RETURN jsonb_build_object('status', 'SUCCESS', 'redemptionId', v_redemption_id);
```

```
EXCEPTION WHEN OTHERS THEN
    RETURN jsonb_build_object('status', 'ERROR', 'message', SQLERRM);
END;
$$ LANGUAGE plpgsql;
```

0.3. Otimização de Latência

A Solução: "Shift-Left" de performance.

1. **Query Plan Audit:** Rodar EXPLAIN ANALYZE em todas as queries dos dashboards.
2. **Índices Compostos:** Adicionar índices específicos para as queries mais pesadas (ex: CREATE INDEX idx_ledger_student_date ON ledger_transactions (student_id, created_at DESC)).
3. **N8N Lean Mode:** Configurar n8n para não salvar dados de execução de sucesso (EXECUTIONS_DATA_SAVE_ON_SUCCESS=none) para reduzir I/O de disco e CPU.

🚀 Módulo 1: Gestão de Alunos em Escala (Onboarding)

Endereçando a lacuna "A" - Entrada de dados.

1.1. Novo Endpoint: POST /students/batch-import

Objetivo: Permitir que o Admin faça upload de uma lista de alunos, eliminando a necessidade de inserção manual via banco.

Fluxo Lógico (Workflow n8n):

1. **Input:** Recebe JSON (Array de Objetos) ou Arquivo CSV (convertido para JSON no n8n).
    - *Campos:* Nome Completo, Email do Responsável (opcional), Turma/Série, Matrícula Escolar.
2.
3. **Validação (Function Node):**
    - Verifica duplicidade de e-mails dentro do arquivo.
    - Valida formato de e-mail.
4.
5. **Processamento em Lote (Postgres):**
    - Utilizar operação INSERT INTO ... ON CONFLICT DO NOTHING (Upsert) para evitar erros se o aluno já existir.
    - Inserção deve ocorrer em blocos (ex: 100 alunos por query) para não estourar a memória do workflow.

6.
7. **Criação de Credenciais:**
    - Para cada novo aluno, gerar uma senha temporária aleatória.
    - Registrar hash na tabela users.
8.
9. **Output:** Relatório de sucesso/falha (ex: "48 alunos importados, 2 erros").

Atualização de Schema (DDL):

Adicionar campo enrollment_id (Matrícula) na tabela students para facilitar a conciliação com o ERP da escola.

🔥 Módulo 2: Governança Econômica (Burn Policy)

Endereçando a lacuna "B" - Controle de Inflação.

2.1. Motor de Expiração (Cron Job)

Objetivo: Implementar a política de expiração de Marks para garantir a saúde financeira da economia tokenizada.

Definição da Regra (Exemplo):

- *Política:* Marks expiram 365 dias após a data de ganho (FIFO - First In, First Out) OU em uma data fixa anual (ex: 31 de Dezembro).
- *Decisão para v1.1:* **Expiração Anual Fixa (Hard Reset)**. É computacionalmente mais leve e pedagogicamente mais fácil de explicar ("Ano novo, vida nova").

Implementação Técnica:

1. **Novo Workflow Agendado (n8n Trigger: Cron):**
    - Execução: 00:00 do dia 01/Jan (ou data configurável por escola).
2.
3. **Stored Procedure de Queima (expire_school_balances):**
    - Itera sobre todos os alunos da escola alvo.
    - Lê o saldo atual.
    - Se saldo > 0:
        1. Gera transação no Ledger: type: 'DEBIT', description: 'Expiração Anual de Pontos'.
        2. Zera (ou reduz) o saldo na tabela students.
    -
4.
5. **Notificação:** Envia e-mail para o Admin da escola com o relatório de "Total Queimado" (Deflation Report).

📊 Módulo 3: Inteligência de Dados (Dashboards)

Endereçando a lacuna "C" - Visibilidade Gerencial.

3.1. Arquitetura de Dados para Analytics

O Desafio: Rodar queries de agregação (SUM, COUNT, AVG) na tabela transacional (ledger) em tempo real deixa o sistema lento.
A Solução: Views Materializadas (PostgreSQL Materialized Views).

DDL das Views:

code SQL
downloadcontent_copy
expand_less

```
  -- View: Engajamento Financeiro (Atualizada diariamente)
CREATE MATERIALIZED VIEW analytics_school_engagement AS
SELECT
    s.school_id,
    COUNT(DISTINCT l.student_id) as active_students_30d,
    SUM(CASE WHEN l.type = 'CREDIT' THEN l.amount ELSE 0 END) as total_minted,
    SUM(CASE WHEN l.type = 'DEBIT' AND l.source_redemption_id IS NOT NULL THEN
l.amount ELSE 0 END) as total_redeemed,
    DATE_TRUNC('day', l.created_at) as ref_date
FROM ledger_transactions l
JOIN students s ON l.student_id = s.id
WHERE l.created_at > NOW() - INTERVAL '30 days'
GROUP BY s.school_id, ref_date;
```

3.2. Novos Endpoints de BI

1. **GET /schools/analytics/financial**:
    - Retorna: Total emitido vs. Total resgatado (Burn Rate), Saldo circulante total (Liability da escola).
    - Fonte: Consulta direta na View Materializada.
2.
3. **GET /schools/analytics/performance**:
    - Retorna: Ranking de turmas mais engajadas, Regras mais acionadas ("Top Conquistas").
    - Uso: Diretor pedagógico ajusta comportamentos baseados nesses dados.
4.

📅 Roteiro de Execução Sugerido

Este plano deve ser executado na seguinte ordem para garantir estabilidade:

1. **Semana 1 - Fundação (Módulo 0):**
    - Implementar PBKDF2 nos workflows de Auth.
    - Escrever e aplicar as Stored Procedures de Resgate no PostgreSQL.
    - Configurar índices de banco.
    - *Milestone:* Sistema seguro e atômico.
2.
3. **Semana 2 - Operação (Módulo 1):**
    - Criar endpoint de Batch Import.
    - Testar carga com CSV de 500 alunos.
    - *Milestone:* Escola consegue fazer onboarding autônomo.
4.
5. **Semana 3 - Inteligência (Módulo 3):**
    - Criar Views Materializadas.
    - Criar endpoints de Analytics.
    - *Milestone:* Diretores veem valor gráfico no produto.
6.
7. **Semana 4 - Governança (Módulo 2):**
    - Configurar regras de Expiração.
    - Testar workflow de Cron em ambiente de Staging.
    - *Milestone:* Ciclo econômico fechado (Emissão -> Uso -> Expiração).
8.

📝 Artifacts & Deliverables

Ao final deste desenvolvimento, o repositório do projeto deverá conter:

1. /db/migrations/v2_hardening.sql: Stored Procedures e Índices.
2. /db/migrations/v3_views.sql: Materialized Views para Analytics.
3. /n8n/workflows/admin_onboarding.json: Workflow de importação de alunos.
4. /n8n/workflows/cron_expiration.json: Workflow de expiração de pontos.
5. SECURITY.md: Documentação atualizada dos novos padrões de hash e rotação de chaves.