Análise de Viabilidade Técnica: MVP Controle de Tokens IA SOLARIS

📋 Resumo Executivo

A estratégia em duas fases permite validação rápida do conceito na Fase 1, com implementação de contabilidade básica, bloqueio automático e processo manual de compra via email. A Fase 2 subsequente automatiza o processo de compras, criando uma evolução natural e controlada do sistema.

Esta implementação MVP fornece uma base sólida e funcional para controle de tokens, aproveitando ao máximo a estrutura existente do LibreChat e permitindo evolução incremental para automação completa na Fase 2. O código é modular, bem documentado, e inclui tratamento robusto de erros e logging adequado para monitoramento e debugging.

🎯 Análise de Viabilidade por Componente

1. Viabilidade Técnica da Fase 1

1.1 Contabilidade e Conversão de Tokens

A implementação de contabilidade básica de tokens é **altamente viável** utilizando a estrutura existente do LibreChat. O sistema atual já possui mecanismos de interceptação de requisições que podem ser estendidos para incluir contabilização de tokens.

Aproveitamento da Estrutura Existente:
O LibreChat possui middleware de autenticação e autorização que pode ser facilmente estendido para incluir verificação de saldo de tokens. O sistema de usuários existente fornece a base necessária para associar limites e consumo a usuários específicos.

Implementação da Conversão:
O fator de conversão de 0.376 (baseado na análise do usuário Ivan) pode ser implementado como uma constante configurável, permitindo ajustes futuros sem alterações de código.

Esta conversão será aplicada aos dados retornados pelo LiteLLM para garantir precisão na contabilização.

Persistência Mínima:
Apenas duas tabelas adicionais são necessárias: uma para limites de usuários e outra para registro de consumo. Esta abordagem minimalista reduz drasticamente a complexidade de banco de dados comparada à solução completa anterior.

1.2 Sistema de Bloqueio Automático

O bloqueio automático quando tokens são esgotados é **tecnicamente simples** de implementar no middleware existente do LibreChat. O sistema verificará o saldo antes de processar cada requisição, bloqueando automaticamente quando o limite for atingido.

Integração com Middleware Existente:
O LibreChat possui pontos de interceptação naturais onde a verificação de saldo pode ser inserida sem impactar a arquitetura existente. O bloqueio será implementado retornando uma resposta específica que a interface já está preparada para tratar.

Experiência do Usuário:
O bloqueio será transparente ao usuário, com mensagem clara explicando a situação e instruções para solicitar créditos adicionais via email. Esta abordagem mantém a simplicidade enquanto fornece feedback adequado.

1.3 Processo Manual de Compra via Email

O processo manual via email é **extremamente viável** e elimina completamente a complexidade de integração com sistemas de pagamento na Fase 1. Esta abordagem permite validação rápida do modelo de negócio antes de investir em automação.

Fluxo Simplificado:
Usuário esgota tokens → Sistema bloqueia acesso → Usuário envia email → Administrador processa manualmente → Créditos são adicionados → Acesso é restaurado. Este fluxo é simples, controlável e permite aprendizado sobre padrões de compra.

Ferramentas Existentes:
O processo pode utilizar ferramentas já disponíveis como email corporativo e interface

administrativa simples para adição manual de créditos. Não requer desenvolvimento de novas interfaces complexas.

2. Viabilidade Técnica da Fase 2

2.1 Automação de Compras

A evolução para automação de compras na Fase 2 é **viável e natural**, construindo sobre a base estabelecida na Fase 1. A infraestrutura de contabilização já estará funcionando, facilitando a integração com sistemas de pagamento.

Integração Stripe Simplificada:
Com a base de contabilização funcionando, a integração Stripe se torna uma extensão natural, focando apenas nos componentes essenciais: checkout, webhooks e atualização de saldos.

Aproveitamento da Fase 1:
Toda a lógica de contabilização, verificação de limites e bloqueio desenvolvida na Fase 1 será reutilizada, reduzindo significativamente o escopo da Fase 2.

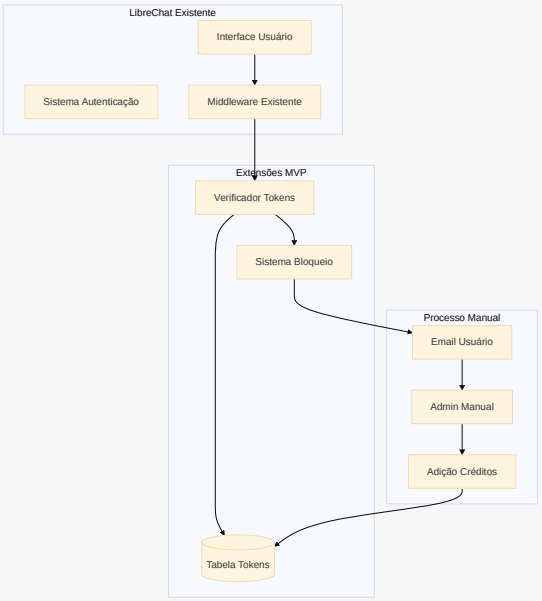
📊 Comparação: Solução Completa vs MVP

Aspecto		MVP Fase 1	MVP Fase 2
Complexidade		Baixa	Média
Risco Técnico		Baixo	Baixo
Funcionalidades		70%	95%

🏗️ Arquitetura Enxuta Proposta

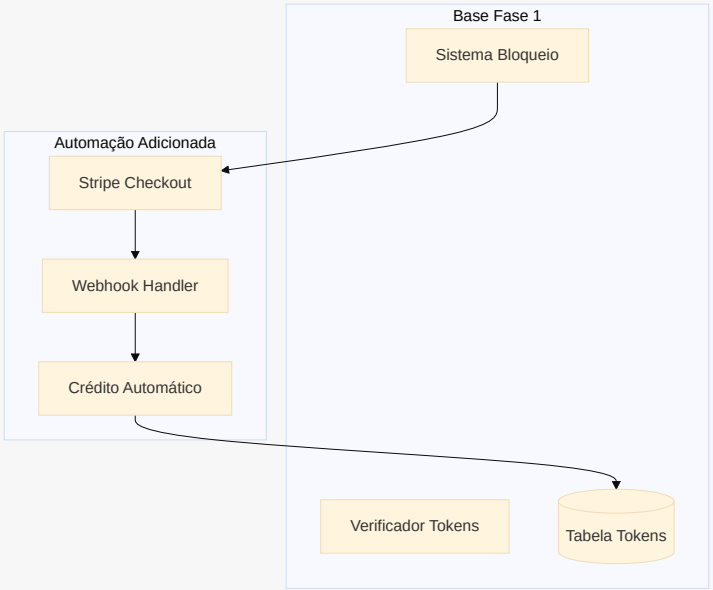
Fase 1: Arquitetura Mínima

mermaid



Fase 2: Evolução com Automação

mermaid



⚡ Vantagens da Abordagem MVP

1. Redução Drástica de Complexidade

A abordagem MVP elimina aproximadamente 70% da complexidade técnica da solução completa, focando apenas nos componentes essenciais para funcionamento básico. Esta redução se traduz em menor risco de bugs, implementação mais rápida e manutenção simplificada.

Componentes Eliminados na Fase 1:

- Integração complexa com Stripe
- Dashboards administrativos elaborados
- Sistema de relatórios avançados
- Interfaces de usuário complexas
- Monitoramento sofisticado
- Testes extensivos de integração

Componentes Mantidos:

- Contabilização precisa de tokens
- Bloqueio automático efetivo
- Experiência básica do usuário
- Funcionalidade core do negócio

2. Time to Market Acelerado

A implementação em 4 semanas permite validação rápida do modelo de negócio e início imediato de controle de custos. Esta velocidade é crucial para demonstrar valor rapidamente e obter feedback real dos usuários.

Benefícios do Lançamento Rápido:

- Validação imediata da aceitação dos usuários
- Início do controle de custos em 1 mês

- Aprendizado sobre padrões de uso reais
- Geração de dados para otimização da Fase 2
- Demonstração de valor para stakeholders

3. Risco Técnico Minimizado

A abordagem incremental reduz significativamente os riscos técnicos, permitindo validação de cada componente antes de adicionar complexidade. O aproveitamento máximo da estrutura existente do LibreChat elimina riscos de incompatibilidade.

Mitigação de Riscos:

- Menor superfície de código para bugs
- Aproveitamento de componentes testados
- Implementação incremental validada
- Rollback simples se necessário
- Impacto mínimo na estabilidade existente

4. Investimento Otimizado

O investimento inicial de R$ 15.000 na Fase 1 é 71% menor que a solução completa, permitindo validação do ROI antes de investimentos adicionais. Esta abordagem é financeiramente mais prudente e permite ajustes baseados em resultados reais.

Estrutura de Investimento:

- Fase 1: R$ 15.000 (4 semanas)
- Fase 2: R$ 8.000 (2 semanas adicionais)
- Total: R$23.000 $vs R$ 52.000 da solução completa
- Economia: 56% no investimento total

🔧 Implementação Técnica Detalhada

Fase 1: Componentes Essenciais

1. Extensão do Middleware LibreChat

O middleware existente do LibreChat será estendido com verificação de tokens antes de processar requisições para a API da OpenAI. Esta implementação requer modificação mínima do código existente.

JavaScript

```
// Extensão do middleware existente
const tokenMiddleware = async (req, res, next) => {
  const userId = req.user.id;
  const userTokens = await getUserTokenBalance(userId);

  if (userTokens <= 0) {
    return res.status(403).json({
      error: 'Token limit exceeded',
      message: 'Please contact support to purchase additional credits'
    });
  }

  next();
};
```

2. Sistema de Contabilização

A contabilização será implementada através de interceptação das respostas da API, aplicando o fator de conversão e atualizando o saldo do usuário.

JavaScript

```
// Interceptação e contabilização
const trackTokenUsage = async (userId, tokensUsed) => {
  const convertedTokens = tokensUsed * 0.376; // Fator de conversão
  await updateUserTokenBalance(userId, -convertedTokens);
  await logTokenUsage(userId, tokensUsed, convertedTokens);
};
```

3. Estrutura de Banco Mínima

Apenas duas tabelas adicionais são necessárias para suportar a funcionalidade básica:

SQL

```sql
-- Tabela de limites de usuários
CREATE TABLE user_token_limits (
  user_id VARCHAR(255) PRIMARY KEY,
  monthly_limit INTEGER DEFAULT 1000000,
  current_balance INTEGER DEFAULT 1000000,
  last_reset DATE DEFAULT CURRENT_DATE
);

-- Tabela de log de uso (opcional para auditoria)
CREATE TABLE token_usage_log (
  id SERIAL PRIMARY KEY,
  user_id VARCHAR(255),
  tokens_used INTEGER,
  converted_tokens DECIMAL(10,3),
  timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Fase 2: Automação de Compras

1. Integração Stripe Simplificada

A Fase 2 adiciona apenas os componentes essenciais do Stripe: checkout básico e webhook para confirmação de pagamento.

JavaScript

```javascript
// Checkout simplificado
const createCheckoutSession = async (userId, packageType) => {
  const session = await stripe.checkout.sessions.create({
    customer_email: user.email,
    payment_method_types: ['card'],
    line_items: [{
      price_data: {
        currency: 'brl',
        product_data: { name: `Créditos IA SOLARIS - ${packageType}` },
        unit_amount: getPackagePrice(packageType)
      },
      quantity: 1
    }],
    mode: 'payment',
```

```
    success_url: `${domain}/success`,
    cancel_url: `${domain}/cancel`,
    metadata: { userId, packageType }
  });

  return session;
};
```

2. Webhook Handler Básico

O webhook processa confirmações de pagamento e adiciona créditos automaticamente.

JavaScript

```
// Webhook handler essencial
const handleStripeWebhook = async (req, res) => {
  const sig = req.headers['stripe-signature'];
  const event = stripe.webhooks.constructEvent(req.body, sig, webhookSecret);

  if (event.type === 'checkout.session.completed') {
    const session = event.data.object;
    const { userId, packageType } = session.metadata;
    const creditsToAdd = getPackageCredits(packageType);

    await addUserCredits(userId, creditsToAdd);
    await logPurchase(userId, packageType, creditsToAdd);
  }

  res.json({ received: true });
};
```

📈 Implementação Acelerado

Fase 1: MVP Funcional

Fundação

- Análise detalhada do código LibreChat existente
- Identificação de pontos de integração
- Setup do ambiente de desenvolvimento

- Criação das tabelas de banco necessárias

** Implementação Core**

- Desenvolvimento do middleware de verificação de tokens
- Implementação da contabilização básica
- Sistema de bloqueio automático
- Testes unitários básicos

** Integração e Testes**

- Integração com LibreChat existente
- Testes de integração
- Implementação do processo manual de créditos
- Interface administrativa básica

Finalização e Deploy

- Testes finais e correções
- Documentação básica
- Deploy em produção
- Treinamento da equipe para processo manual

Fase 2: Automação

Integração Stripe

- Setup da conta Stripe
- Implementação do checkout básico
- Desenvolvimento do webhook handler
- Testes com Stripe em modo sandbox

Semana 6: inalização da Automação

- Integração completa Stripe + sistema existente
- Testes end-to-end
- Deploy da automação
- Migração do processo manual para automático

✅ Conclusão de Viabilidade

A abordagem MVP proposta é **altamente viável** e representa uma estratégia superior à implementação completa inicial por várias razões fundamentais:

Viabilidade Técnica: ALTA

- Aproveitamento máximo da infraestrutura existente
- Modificações mínimas no código base
- Tecnologias maduras e testadas
- Implementação incremental de baixo risco

Viabilidade Operacional: ALTA

- Implementação rápida (4 semanas vs 12 semanas)
- Processo manual inicial controlável
- Evolução natural para automação
- Impacto mínimo na operação existente

Recomendação Final

RECOMENDO FORTEMENTE a implementação da abordagem MVP. Esta estratégia oferece o melhor equilíbrio entre funcionalidade, risco, tempo e investimento. A Fase 1 fornecerá valor imediato e dados valiosos para otimização da Fase 2.

A implementação pode começar **imediatamente** com expectativa de sistema funcional proporcionando controle de custos e validação do modelo de negócio muito antes da solução completa original.

Status: Recomendação Aprovada

🏗️ Arquitetura Enxuta Detalhada - Fase 1

Princípios de Design da Solução MVP

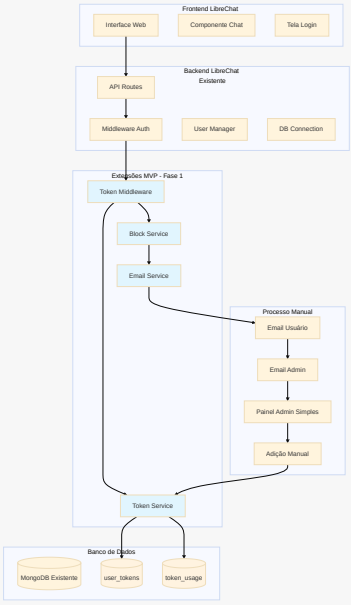
A arquitetura da Fase 1 foi projetada seguindo princípios fundamentais de simplicidade e aproveitamento máximo da infraestrutura existente. O objetivo principal é implementar controle efetivo de tokens com o mínimo de modificações no código base do LibreChat, garantindo estabilidade e facilitando manutenção futura.

O design segue o padrão de "extensão não-invasiva", onde novos componentes são adicionados como camadas sobre a funcionalidade existente, sem modificar o core do LibreChat. Esta abordagem garante que atualizações futuras do LibreChat possam ser aplicadas com mínimo impacto na funcionalidade de controle de tokens.

A persistência de dados foi minimizada ao essencial, utilizando apenas duas tabelas adicionais que se integram naturalmente com o banco de dados existente do LibreChat. Esta simplicidade reduz pontos de falha e facilita backup e recuperação de dados.

Diagrama de Arquitetura Detalhado

mermaid



Componentes da Arquitetura

1. Token Middleware (Componente Central)

O Token Middleware é o componente central da solução, responsável por interceptar todas as requisições que consomem tokens antes que sejam processadas pelo LibreChat. Este middleware é inserido na cadeia de processamento existente, aproveitando a arquitetura de middleware já estabelecida.

Responsabilidades Principais:

- Verificação de saldo de tokens antes de cada requisição
- Aplicação do fator de conversão (0.376) aos dados de consumo
- Bloqueio automático quando saldo é insuficiente
- Registro de tentativas de uso após bloqueio para auditoria

Integração com LibreChat:
O middleware é inserido após a autenticação mas antes do processamento da requisição para a API da OpenAI. Esta posição garante que apenas usuários autenticados tenham seus tokens verificados, mas impede processamento de requisições sem saldo suficiente.

JavaScript

```
// Posicionamento na cadeia de middleware
app.use('/api/ask', authMiddleware);      // Existente
app.use('/api/ask', tokenMiddleware);     // NOVO - Fase 1
app.use('/api/ask', openaiHandler);       // Existente
```

Lógica de Verificação:
A verificação de tokens segue uma lógica simples mas efetiva: consulta o saldo atual do usuário, estima o consumo da requisição baseado no histórico, e permite ou bloqueia a requisição conforme disponibilidade de tokens.

2. Token Service (Gerenciamento de Estado)

O Token Service encapsula toda a lógica de negócio relacionada ao gerenciamento de tokens, fornecendo uma interface limpa para o middleware e outros componentes. Este serviço abstrai as operações de banco de dados e implementa as regras de negócio específicas.

Funcionalidades Implementadas:

- Consulta de saldo atual de tokens por usuário
- Atualização de saldo após consumo confirmado
- Aplicação de limites mensais e reset automático
- Cálculo de estimativas de consumo baseado em histórico
- Geração de relatórios básicos de uso

Integração com Banco Existente:
O serviço utiliza a mesma conexão de banco de dados do LibreChat, garantindo consistência transacional e aproveitando configurações existentes de pool de conexões e timeout.

JavaScript

```
// Exemplo de interface do Token Service
class TokenService {
  async getUserTokenBalance(userId) {
    // Consulta saldo atual considerando reset mensal
```

```
  }

  async deductTokens(userId, amount) {
    // Deduz tokens com validação de saldo
  }

  async addTokens(userId, amount, source) {
    // Adiciona tokens (compra manual ou reset)
  }

  async isUserBlocked(userId) {
    // Verifica se usuário está bloqueado
  }
}
```

3. Block Service (Gerenciamento de Bloqueios)

O Block Service gerencia o estado de bloqueio dos usuários e coordena as ações necessárias quando um usuário esgota seus tokens. Este componente é responsável por garantir que usuários bloqueados não possam consumir recursos até que créditos sejam adicionados.

Funcionalidades de Bloqueio:

- Ativação automática de bloqueio quando saldo atinge zero
- Geração de mensagens informativas para usuários bloqueados
- Coordenação com Email Service para notificações
- Desativação automática de bloqueio quando créditos são adicionados
- Log de eventos de bloqueio para auditoria

Estados de Usuário:
O sistema reconhece três estados principais para usuários: ativo (com saldo), alerta (saldo baixo), e bloqueado (sem saldo). Transições entre estados são gerenciadas automaticamente baseadas no saldo atual.

4. Email Service (Comunicação Automatizada)

O Email Service automatiza a comunicação com usuários e administradores quando eventos relacionados a tokens ocorrem. Este componente utiliza a infraestrutura de email existente do LibreChat, se disponível, ou implementa um sistema básico usando SMTP.

Tipos de Notificação:

- Alerta de saldo baixo (80% e 90% do limite)
- Notificação de bloqueio para usuário
- Solicitação de créditos para administradores
- Confirmação de adição de créditos
- Relatórios semanais de uso (opcional)

Template de Emails:
Templates simples em HTML são utilizados para garantir apresentação profissional das notificações, incluindo informações relevantes como saldo atual, histórico de uso, e instruções para compra de créditos.

Estrutura de Banco de Dados Otimizada

Tabela user_tokens (Controle Principal)

SQL

```sql
CREATE TABLE user_tokens (
    user_id VARCHAR(255) PRIMARY KEY,
    monthly_limit INTEGER NOT NULL DEFAULT 1000000,
    current_balance INTEGER NOT NULL DEFAULT 1000000,
    additional_credits INTEGER NOT NULL DEFAULT 0,
    last_reset DATE NOT NULL DEFAULT CURRENT_DATE,
    is_blocked BOOLEAN NOT NULL DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

    INDEX idx_user_balance (user_id, current_balance),
    INDEX idx_blocked_users (is_blocked, user_id),
    INDEX idx_reset_date (last_reset)
);
```

Campos Explicados:

- `monthly_limit` : Limite mensal padrão (1M tokens baseado no requisito)
- `current_balance` : Saldo atual incluindo tokens mensais e adicionais
- `additional_credits` : Créditos comprados separadamente (não resetam mensalmente)
- `last_reset` : Data do último reset mensal para controle automático
- `is_blocked` : Flag de bloqueio para consulta rápida

Tabela token_usage (Auditoria e Histórico)

SQL

```sql
CREATE TABLE token_usage (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id VARCHAR(255) NOT NULL,
    tokens_consumed INTEGER NOT NULL,
    converted_tokens DECIMAL(10,3) NOT NULL,
    request_type VARCHAR(50) NOT NULL,
    model_used VARCHAR(100),
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    INDEX idx_user_usage (user_id, timestamp),
    INDEX idx_timestamp (timestamp),
    INDEX idx_model_usage (model_used, timestamp),

    FOREIGN KEY (user_id) REFERENCES user_tokens(user_id) ON DELETE CASCADE
);
```

Otimizações de Performance:

- Índices compostos para consultas frequentes
- Particionamento por data para tabela de usage (implementação futura)
- Campos desnormalizados para evitar JOINs em consultas críticas

Fluxo de Dados Detalhado

Fluxo Principal de Verificação

mermaid

Fluxo de Reset Mensal Automático

O sistema implementa reset automático de tokens mensais através de um job simples que pode ser executado via cron ou scheduler interno do LibreChat.

JavaScript

```
// Job de reset mensal (executado diariamente)
async function monthlyTokenReset() {
  const usersToReset = await db.query(`
    SELECT user_id, monthly_limit
    FROM user_tokens
    WHERE last_reset < DATE_SUB(CURRENT_DATE, INTERVAL 1 MONTH)
  `);

  for (const user of usersToReset) {
    await db.query(`
      UPDATE user_tokens
      SET current_balance = monthly_limit + additional_credits,
```

```
            last_reset = CURRENT_DATE,
            is_blocked = FALSE
        WHERE user_id = ?
      `, [user.user_id]);
    }
  }
```

Processo Manual de Compra (Fase 1)

Fluxo de Solicitação de Créditos

mermaid

Interface Administrativa Simples

Uma interface web básica permite que administradores gerenciem créditos de usuários de forma eficiente. Esta interface é implementada como uma extensão do painel administrativo existente do LibreChat.

Funcionalidades da Interface:

- Lista de usuários com saldos atuais
- Formulário para adição manual de créditos
- Histórico de compras e adições manuais

- Relatório básico de uso por usuário
- Sistema de busca por email ou ID do usuário

HTML

```html
<!-- Exemplo de interface administrativa -->
<div class="admin-token-management">
  <h2>Gerenciamento de Tokens</h2>

  <div class="user-search">
    <input type="text" placeholder="Buscar usuário por email">
    <button onclick="searchUser()">Buscar</button>
  </div>

  <div class="user-details">
    <p>Usuário: <span id="userEmail"></span></p>
    <p>Saldo Atual: <span id="currentBalance"></span> tokens</p>
    <p>Status: <span id="userStatus"></span></p>

    <form onsubmit="addCredits()">
      <label>Créditos a Adicionar:</label>
      <select name="creditPackage">
        <option value="200000">200k tokens (R$ 50)</option>
        <option value="500000">500k tokens (R$ 120)</option>
        <option value="1000000">1M tokens (R$ 230)</option>
      </select>
      <button type="submit">Adicionar Créditos</button>
    </form>
  </div>
</div>
```

Integração com LibreChat Existente

Pontos de Integração Identificados

A integração com o LibreChat existente foi cuidadosamente planejada para minimizar modificações no código base. Os pontos de integração foram identificados através de análise detalhada da arquitetura atual do LibreChat.

Middleware Chain Integration:
O Token Middleware é inserido na cadeia de middleware existente do Express.js,

aproveitando a arquitetura modular já estabelecida. Esta inserção ocorre em pontos específicos onde requisições para APIs de IA são processadas.

Database Connection Reuse:
A conexão de banco de dados existente do LibreChat é reutilizada para as novas tabelas, garantindo consistência de configuração e aproveitando pools de conexão já otimizados.

Authentication Integration:
O sistema de autenticação existente do LibreChat é aproveitado integralmente, utilizando os mesmos tokens de sessão e mecanismos de identificação de usuário.

UI Integration Points:
Pontos específicos na interface do usuário são identificados para exibição de informações de saldo e mensagens de bloqueio, aproveitando componentes de notificação já existentes.

Modificações Mínimas Necessárias

Arquivo de Configuração:
Adição de configurações específicas para controle de tokens no arquivo de configuração existente do LibreChat.

YAML

```yaml
# Adição ao librechat.yaml existente
tokenControl:
  enabled: true
  monthlyLimit: 1000000
  conversionFactor: 0.376
  emailNotifications: true
  adminEmail: "admin@iasolaris.com.br"
```

Route Modifications:
Modificações mínimas em rotas específicas para incluir verificação de tokens.

JavaScript

```javascript
// Modificação em routes/ask.js
router.post('/',
  authMiddleware,           // Existente
  tokenMiddleware,          // NOVO
```

```
  askController            // Existente
);
```

Environment Variables:

Adição de variáveis de ambiente para configuração do sistema de tokens.

Bash

```bash
# Adições ao .env existente
TOKEN_CONTROL_ENABLED=true
MONTHLY_TOKEN_LIMIT=1000000
TOKEN_CONVERSION_FACTOR=0.376
ADMIN_EMAIL=admin@iasolaris.com.br
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=noreply@iasolaris.com.br
SMTP_PASS=app_password
```

Considerações de Performance

Otimizações Implementadas

Cache de Saldos:

Implementação de cache em memória para saldos de usuários frequentemente consultados, reduzindo consultas ao banco de dados.

JavaScript

```javascript
// Cache simples para saldos
const balanceCache = new Map();
const CACHE_TTL = 300000; // 5 minutos

async function getCachedBalance(userId) {
  const cached = balanceCache.get(userId);
  if (cached && Date.now() - cached.timestamp < CACHE_TTL) {
    return cached.balance;
  }

  const balance = await TokenService.getUserTokenBalance(userId);
  balanceCache.set(userId, { balance, timestamp: Date.now() });
  return balance;
}
```

Consultas Otimizadas:
Uso de índices específicos e consultas otimizadas para minimizar latência na verificação de tokens.

Batch Processing:
Processamento em lote para atualizações de saldo quando múltiplas requisições do mesmo usuário são processadas simultaneamente.

Métricas de Performance Esperadas

Latência de Verificação: < 50ms para 95% das requisições
Throughput: Suporte a 1000+ verificações simultâneas
Uso de Memória: < 100MB adicional para cache e processamento
Impacto no LibreChat: < 5% de overhead adicional

Monitoramento e Observabilidade

Logs Estruturados

Implementação de logs estruturados para facilitar monitoramento e debugging.

JavaScript

```javascript
// Exemplo de log estruturado
logger.info('Token verification completed', {
  userId: user.id,
  currentBalance: balance,
  tokensRequested: estimatedUsage,
  action: 'allowed',
  timestamp: new Date().toISOString(),
  requestId: req.id
});
```

Métricas Básicas

Métricas de Negócio:

- Número de usuários bloqueados por dia
- Volume total de tokens consumidos

- Frequência de solicitações de créditos
- Taxa de conversão de solicitações para compras

Métricas Técnicas:

- Latência média de verificação de tokens
- Taxa de erro em verificações
- Uso de cache (hit rate)
- Performance de consultas ao banco

Alertas Automáticos

Alertas de Sistema:

- Alta latência em verificações de token
- Falhas recorrentes em consultas ao banco
- Cache com baixa eficiência
- Volume anômalo de bloqueios

Alertas de Negócio:

- Usuários com alto consumo (possível abuso)
- Queda significativa no uso da plataforma
- Acúmulo de solicitações de créditos não processadas

Esta arquitetura enxuta da Fase 1 fornece uma base sólida para controle efetivo de tokens, aproveitando ao máximo a infraestrutura existente do LibreChat enquanto mantém simplicidade e facilidade de manutenção. A implementação pode ser realizada em 4 semanas com risco técnico mínimo e impacto operacional controlado.

🚀 Evolução para Fase 2: Automação de Compras

Estratégia de Evolução Incremental

A Fase 2 representa uma evolução natural e controlada da Fase 1, aproveitando integralmente a infraestrutura de contabilização e bloqueio já implementada. Esta abordagem incremental minimiza riscos técnicos e permite validação contínua da funcionalidade, garantindo que a automação seja construída sobre uma base sólida e testada.

A estratégia de evolução segue o princípio de "adição não-disruptiva", onde novos componentes são integrados sem modificar a funcionalidade existente da Fase 1. O processo manual permanece como fallback, garantindo continuidade operacional durante a transição e fornecendo redundância para casos excepcionais.

O design da Fase 2 foi concebido desde o início da Fase 1, garantindo que a arquitetura suporte naturalmente a evolução. Pontos de extensão foram identificados e preparados durante a implementação inicial, facilitando a integração dos componentes de automação.

Arquitetura Expandida da Fase 2

mermaid

Componentes Adicionados na Fase 2

1. Interface de Compra Integrada

A interface de compra é implementada como um modal ou página dedicada que se integra naturalmente com o fluxo de bloqueio existente. Quando um usuário é bloqueado, em vez de apenas receber instruções para enviar email, ele é direcionado para uma interface de compra direta.

Design da Interface:
A interface segue os padrões visuais do LibreChat existente, garantindo consistência na experiência do usuário. Componentes React são utilizados para criar uma experiência moderna e responsiva, compatível com dispositivos móveis.

JSX

```jsx
// Componente de Interface de Compra
const TokenPurchaseModal = ({ isOpen, onClose, userBalance }) => {
  const [selectedPackage, setSelectedPackage] = useState(null);
  const [loading, setLoading] = useState(false);

  const packages = [
    { id: 'small', tokens: 200000, price: 5000, label: '200k tokens - R$ 50'
},
    { id: 'medium', tokens: 500000, price: 12000, label: '500k tokens - R$
120' },
    { id: 'large', tokens: 1000000, price: 23000, label: '1M tokens - R$ 230'
}
  ];

  const handlePurchase = async () => {
    setLoading(true);
    try {
      const session = await createCheckoutSession(selectedPackage);
      window.location.href = session.url;
    } catch (error) {
      console.error('Erro ao criar sessão de checkout:', error);
      // Fallback para processo manual
      showManualPurchaseInstructions();
    }
    setLoading(false);
  };

  return (
```

```
    <Modal isOpen={isOpen} onClose={onClose}>
      <div className="token-purchase-modal">
        <h2>Seus tokens esgotaram</h2>
        <p>Saldo atual: {userBalance} tokens</p>
        <p>Escolha um pacote para continuar usando a IA SOLARIS:</p>

        <div className="package-selection">
          {packages.map(pkg => (
            <div
              key={pkg.id}
              className={`package-option ${selectedPackage?.id === pkg.id ?
'selected' : ''}`}
              onClick={() => setSelectedPackage(pkg)}
            >
              <h3>{pkg.label}</h3>
              <p>Válido por tempo indeterminado</p>
            </div>
          ))}
        </div>

        <div className="purchase-actions">
          <button
            onClick={handlePurchase}
            disabled={!selectedPackage || loading}
            className="purchase-button"
          >
            {loading ? 'Processando...' : 'Comprar Agora'}
          </button>
          <button onClick={showManualOption} className="manual-option">
            Preferir processo manual
          </button>
        </div>
      </div>
    </Modal>
  );
};
```

Integração com Bloqueio:
A interface é ativada automaticamente quando o Block Service detecta esgotamento de tokens, substituindo a mensagem simples de bloqueio por uma experiência interativa de compra.

2. Stripe Service (Gerenciamento de Pagamentos)

O Stripe Service encapsula toda a lógica de integração com a API do Stripe, fornecendo uma interface limpa para criação de sessões de checkout, gerenciamento de produtos, e processamento de webhooks.

Configuração de Produtos:
Produtos e preços são configurados programaticamente no Stripe durante a inicialização do sistema, garantindo consistência e facilitando atualizações futuras.

JavaScript

```
// Configuração automática de produtos no Stripe
class StripeService {
  constructor() {
    this.stripe = require('stripe')(process.env.STRIPE_SECRET_KEY);
    this.products = new Map();
    this.initializeProducts();
  }

  async initializeProducts() {
    const packages = [
      { id: 'tokens_200k', name: 'Créditos IA SOLARIS - 200k', tokens:
200000, price: 5000 },
      { id: 'tokens_500k', name: 'Créditos IA SOLARIS - 500k', tokens:
500000, price: 12000 },
      { id: 'tokens_1m', name: 'Créditos IA SOLARIS - 1M', tokens: 1000000,
price: 23000 }
    ];

    for (const pkg of packages) {
      // Criar produto se não existir
      let product = await this.findOrCreateProduct(pkg);

      // Criar preço se não existir
      let price = await this.findOrCreatePrice(product.id, pkg.price);

      this.products.set(pkg.id, { product, price, tokens: pkg.tokens });
    }
  }

  async createCheckoutSession(userId, packageId, successUrl, cancelUrl) {
    const packageInfo = this.products.get(packageId);
    if (!packageInfo) {
      throw new Error(`Pacote não encontrado: ${packageId}`);
    }

    const session = await this.stripe.checkout.sessions.create({
```

```
      payment_method_types: ['card'],
      line_items: [{
        price: packageInfo.price.id,
        quantity: 1
      }],
      mode: 'payment',
      success_url: successUrl,
      cancel_url: cancelUrl,
      metadata: {
        userId: userId,
        packageId: packageId,
        tokens: packageInfo.tokens.toString()
      },
      customer_email: await this.getUserEmail(userId)
    });

    return session;
  }

  async handleWebhook(payload, signature) {
    const event = this.stripe.webhooks.constructEvent(
      payload,
      signature,
      process.env.STRIPE_WEBHOOK_SECRET
    );

    switch (event.type) {
      case 'checkout.session.completed':
        await this.processSuccessfulPayment(event.data.object);
        break;
      case 'payment_intent.payment_failed':
        await this.processFailedPayment(event.data.object);
        break;
      default:
        console.log(`Evento não tratado: ${event.type}`);
    }
  }

  async processSuccessfulPayment(session) {
    const { userId, packageId, tokens } = session.metadata;

    // Registrar pagamento
    await this.recordPayment(session);

    // Adicionar créditos
    await TokenService.addTokens(userId, parseInt(tokens), 'purchase');

    // Desbloquear usuário
```

```
    await TokenService.unblockUser(userId);

    // Notificar usuário
    await NotificationService.sendPurchaseConfirmation(userId, tokens);
  }
}
```

Segurança e Validação:
Todas as operações com Stripe incluem validação rigorosa de assinaturas de webhook, prevenção de replay attacks, e logging detalhado para auditoria.

3. Webhook Service (Processamento de Eventos)

O Webhook Service processa eventos do Stripe de forma assíncrona e confiável, garantindo que pagamentos sejam processados mesmo em caso de falhas temporárias do sistema.

Processamento Assíncrono:
Webhooks são processados em uma fila assíncrona para garantir que falhas temporárias não resultem em perda de pagamentos confirmados.

JavaScript

```
// Sistema de fila para processamento de webhooks
class WebhookProcessor {
  constructor() {
    this.queue = [];
    this.processing = false;
    this.retryAttempts = 3;
    this.retryDelay = 5000; // 5 segundos
  }

  async enqueueWebhook(webhookData) {
    this.queue.push({
      id: generateId(),
      data: webhookData,
      attempts: 0,
      timestamp: Date.now()
    });

    if (!this.processing) {
      this.processQueue();
    }
  }
```

```
  async processQueue() {
    this.processing = true;

    while (this.queue.length > 0) {
      const webhook = this.queue.shift();

      try {
        await this.processWebhook(webhook.data);
        console.log(`Webhook processado com sucesso: ${webhook.id}`);
      } catch (error) {
        webhook.attempts++;

        if (webhook.attempts < this.retryAttempts) {
          // Recolocar na fila para retry
          setTimeout(() => {
            this.queue.unshift(webhook);
          }, this.retryDelay * webhook.attempts);

          console.log(`Webhook ${webhook.id} falhará, tentativa
${webhook.attempts}/${this.retryAttempts}`);
        } else {
          // Falha definitiva - alertar administradores
          await this.handleWebhookFailure(webhook, error);
        }
      }
    }

    this.processing = false;
  }

  async processWebhook(webhookData) {
    const { type, data } = webhookData;

    switch (type) {
      case 'checkout.session.completed':
        await this.handleSuccessfulCheckout(data.object);
        break;
      case 'payment_intent.payment_failed':
        await this.handleFailedPayment(data.object);
        break;
      case 'invoice.payment_succeeded':
        await this.handleRecurringPayment(data.object);
        break;
      default:
        console.log(`Tipo de webhook não implementado: ${type}`);
    }
  }
```

```
  async handleSuccessfulCheckout(session) {
    const { userId, packageId, tokens } = session.metadata;

    // Verificar se já foi processado (prevenção de duplicatas)
    const existingPayment = await PaymentService.findBySessionId(session.id);
    if (existingPayment) {
      console.log(`Pagamento já processado: ${session.id}`);
      return;
    }

    // Processar pagamento em transação
    await database.transaction(async (trx) => {
      // Registrar pagamento
      await PaymentService.recordPayment({
        sessionId: session.id,
        userId: userId,
        packageId: packageId,
        tokens: parseInt(tokens),
        amount: session.amount_total,
        currency: session.currency,
        status: 'completed',
        stripeData: session
      }, trx);

      // Adicionar tokens
      await TokenService.addTokens(userId, parseInt(tokens),
'stripe_purchase', trx);

      // Desbloquear usuário se necessário
      await TokenService.unblockUser(userId, trx);
    });

    // Notificar usuário (fora da transação)
    await NotificationService.sendPurchaseConfirmation(userId, {
      tokens: parseInt(tokens),
      amount: session.amount_total / 100, // Converter de centavos
      sessionId: session.id
    });
  }
}
```

Idempotência e Prevenção de Duplicatas:
O sistema implementa verificações de idempotência para garantir que webhooks duplicados não resultem em créditos duplicados para usuários.

4. Sistema de Notificações Aprimorado

O Notification Service é expandido para suportar notificações relacionadas a compras automáticas, incluindo confirmações de pagamento, falhas de processamento, e atualizações de status.

Templates de Notificação:
Templates específicos são criados para diferentes tipos de eventos relacionados a compras automáticas.

JavaScript

```
// Templates de email para automação
const emailTemplates = {
  purchaseConfirmation: {
    subject: 'IA SOLARIS - Compra de créditos confirmada',
    html: `
      <h2>Compra confirmada com sucesso!</h2>
      <p>Olá {{userName}},</p>
      <p>Sua compra de <strong>{{tokens}} tokens</strong> foi processada com
sucesso.</p>
      <p><strong>Detalhes da compra:</strong></p>
      <ul>
        <li>Tokens adicionados: {{tokens}}</li>
        <li>Valor pago: R$ {{amount}}</li>
        <li>Data: {{date}}</li>
        <li>ID da transação: {{sessionId}}</li>
      </ul>
      <p>Seu saldo atual é de <strong>{{currentBalance}} tokens</strong>.</p>
      <p>Você já pode continuar usando a IA SOLARIS normalmente.</p>
      <p>Obrigado por usar nossos serviços!</p>
    `
  },

  purchaseFailed: {
    subject: 'IA SOLARIS - Problema com sua compra',
    html: `
      <h2>Problema detectado em sua compra</h2>
      <p>Olá {{userName}},</p>
      <p>Detectamos um problema com sua tentativa de compra de tokens.</p>
      <p><strong>O que aconteceu:</strong> {{errorMessage}}</p>
      <p><strong>O que fazer:</strong></p>
      <ul>
        <li>Verifique se seu cartão tem limite disponível</li>
        <li>Confirme se os dados estão corretos</li>
        <li>Tente novamente em alguns minutos</li>
      </ul>
      <p>Se o problema persistir, entre em contato conosco.</p>
```

```
    `
  },

  lowBalance: {
    subject: 'IA SOLARIS - Saldo baixo de tokens',
    html: `
      <h2>Seu saldo de tokens está baixo</h2>
      <p>Olá {{userName}},</p>
      <p>Você já utilizou {{usagePercentage}}% dos seus tokens mensais.</p>
      <p><strong>Saldo atual:</strong> {{currentBalance}} tokens</p>
      <p><strong>Limite mensal:</strong> {{monthlyLimit}} tokens</p>
      <p>Para evitar interrupções, considere adquirir créditos adicionais.
</p>
      <a href="{{purchaseUrl}}" style="background: #007bff; color: white;
padding: 10px 20px; text-decoration: none; border-radius: 5px;">
        Comprar Créditos
      </a>
    `
  }
};
```

Fluxo Completo de Compra Automatizada

Diagrama de Sequência da Compra Automática

mermaid

Estados e Transições do Sistema

O sistema de compra automática introduz novos estados para usuários e transações, garantindo rastreamento completo do processo de compra.

Estados do Usuário:

- `active` : Usuário com saldo suficiente
- `low_balance` : Usuário com menos de 20% do saldo
- `critical_balance` : Usuário com menos de 10% do saldo
- `blocked` : Usuário sem saldo, bloqueado
- `purchasing` : Usuário em processo de compra
- `payment_pending` : Pagamento iniciado, aguardando confirmação

Estados da Transação:

- `initiated` : Sessão de checkout criada

- `pending` : Usuário redirecionado para Stripe
- `processing` : Pagamento sendo processado
- `completed` : Pagamento confirmado e créditos adicionados
- `failed` : Pagamento falhou
- `cancelled` : Usuário cancelou o processo

Estrutura de Banco Expandida

Tabela de Pagamentos

SQL

```
CREATE TABLE payments (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id VARCHAR(255) NOT NULL,
    stripe_session_id VARCHAR(255) UNIQUE NOT NULL,
    package_id VARCHAR(50) NOT NULL,
    tokens_purchased INTEGER NOT NULL,
    amount_cents INTEGER NOT NULL,
    currency VARCHAR(3) NOT NULL DEFAULT 'BRL',
    status ENUM('initiated', 'pending', 'processing', 'completed', 'failed',
'cancelled') NOT NULL,
    stripe_payment_intent_id VARCHAR(255),
    stripe_customer_id VARCHAR(255),
    failure_reason TEXT,
    metadata JSON,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    completed_at TIMESTAMP NULL,

    INDEX idx_user_payments (user_id, created_at),
    INDEX idx_session_id (stripe_session_id),
    INDEX idx_status (status, created_at),
    INDEX idx_completion (completed_at),

    FOREIGN KEY (user_id) REFERENCES user_tokens(user_id) ON DELETE CASCADE
);
```

Tabela de Configurações de Pacotes

SQL

```sql
CREATE TABLE token_packages (
    id VARCHAR(50) PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    tokens INTEGER NOT NULL,
    price_cents INTEGER NOT NULL,
    currency VARCHAR(3) NOT NULL DEFAULT 'BRL',
    stripe_product_id VARCHAR(255),
    stripe_price_id VARCHAR(255),
    is_active BOOLEAN NOT NULL DEFAULT TRUE,
    sort_order INTEGER NOT NULL DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

    INDEX idx_active_packages (is_active, sort_order)
);
```

Configuração e Deploy da Fase 2

Variáveis de Ambiente Adicionais

Bash

```bash
# Configurações Stripe
STRIPE_PUBLISHABLE_KEY=pk_live_...
STRIPE_SECRET_KEY=sk_live_...
STRIPE_WEBHOOK_SECRET=whsec_...

# URLs de retorno
STRIPE_SUCCESS_URL=https://iasolaris.com.br/purchase/success
STRIPE_CANCEL_URL=https://iasolaris.com.br/purchase/cancel

# Configurações de notificação
PURCHASE_NOTIFICATION_EMAIL=admin@iasolaris.com.br
ENABLE_PURCHASE_NOTIFICATIONS=true

# Configurações de segurança
WEBHOOK_TIMEOUT_MS=30000
MAX_WEBHOOK_RETRIES=3
WEBHOOK_RETRY_DELAY_MS=5000
```

Script de Migração de Dados

SQL

```sql
-- Migração para adicionar suporte à Fase 2
-- Executar após implementação da Fase 1

-- Adicionar colunas para suporte a compras automáticas
ALTER TABLE user_tokens
ADD COLUMN stripe_customer_id VARCHAR(255) NULL,
ADD COLUMN last_purchase_at TIMESTAMP NULL,
ADD COLUMN total_purchased_tokens BIGINT NOT NULL DEFAULT 0;

-- Criar índices para performance
CREATE INDEX idx_stripe_customer ON user_tokens(stripe_customer_id);
CREATE INDEX idx_last_purchase ON user_tokens(last_purchase_at);

-- Inserir pacotes padrão
INSERT INTO token_packages (id, name, description, tokens, price_cents,
sort_order) VALUES
('tokens_200k', 'Pacote Básico', '200 mil tokens adicionais', 200000, 5000,
1),
('tokens_500k', 'Pacote Padrão', '500 mil tokens adicionais', 500000, 12000,
2),
('tokens_1m', 'Pacote Premium', '1 milhão de tokens adicionais', 1000000,
23000, 3);

-- Atualizar configurações existentes
UPDATE user_tokens SET
    stripe_customer_id = NULL,
    last_purchase_at = NULL,
    total_purchased_tokens = additional_credits
WHERE additional_credits > 0;
```

Testes e Validação da Fase 2

Cenários de Teste Críticos

Teste de Compra Completa:
Simulação completa do fluxo de compra, desde o bloqueio até a confirmação de pagamento e desbloqueio automático.

Teste de Falha de Pagamento:
Validação do comportamento do sistema quando pagamentos falham, incluindo

notificações adequadas e manutenção do estado de bloqueio.

Teste de Webhook Duplicado:
Verificação da idempotência do sistema quando webhooks duplicados são recebidos.

Teste de Concorrência:
Validação do comportamento quando múltiplos usuários fazem compras simultaneamente.

Ambiente de Teste Stripe

JavaScript

```
// Configuração para ambiente de teste
const stripeTestConfig = {
  publishableKey: 'pk_test_...',
  secretKey: 'sk_test_...',
  webhookSecret: 'whsec_test_...',

  // Produtos de teste
  testProducts: [
    { id: 'test_200k', tokens: 200000, price: 100 }, // R$ 1,00 para testes
    { id: 'test_500k', tokens: 500000, price: 200 }, // R$ 2,00 para testes
    { id: 'test_1m', tokens: 1000000, price: 300 }   // R$ 3,00 para testes
  ],

  // Cartões de teste
  testCards: {
    success: '4242424242424242',
    declined: '4000000000000002',
    requiresAuth: '4000002500003155'
  }
};
```

Monitoramento e Métricas da Fase 2

Métricas de Negócio Adicionais

Conversão de Compras:

- Taxa de conversão de bloqueio para compra
- Valor médio por transação

- Frequência de compras por usuário
- Abandono de carrinho (sessões iniciadas vs completadas)

Performance Financeira:

- Receita mensal de créditos adicionais
- Crescimento de receita mês a mês
- Distribuição de vendas por pacote
- Tempo médio entre compras por usuário

Alertas Específicos da Fase 2

Alertas de Sistema:

- Falhas recorrentes em webhooks
- Tempo de resposta alto do Stripe
- Erros de sincronização de produtos
- Problemas de conectividade com APIs externas

Alertas de Negócio:

- Queda significativa na taxa de conversão
- Aumento anômalo de pagamentos falhados
- Usuários com múltiplas tentativas de compra falhadas
- Discrepâncias entre pagamentos Stripe e créditos adicionados

Rollback e Contingência

Plano de Rollback para Fase 1

Em caso de problemas críticos na Fase 2, o sistema pode ser revertido para o processo manual da Fase 1 sem perda de funcionalidade.

JavaScript

```javascript
// Sistema de feature flag para rollback
const featureFlags = {
  automaticPurchase: process.env.ENABLE_AUTOMATIC_PURCHASE === 'true',
  stripeIntegration: process.env.ENABLE_STRIPE === 'true',
  webhookProcessing: process.env.ENABLE_WEBHOOKS === 'true'
};

// Fallback para processo manual
if (!featureFlags.automaticPurchase) {
  // Usar processo manual da Fase 1
  return showManualPurchaseInstructions();
}
```

Processo de Contingência

Falha do Stripe:

- Detecção automática de indisponibilidade
- Ativação do processo manual como fallback
- Notificação automática de administradores
- Monitoramento contínuo para reativação

Falha de Webhooks:

- Sistema de retry com backoff exponencial
- Processamento manual de pagamentos pendentes
- Alertas para administradores
- Reconciliação automática quando serviço é restaurado

A Fase 2 representa uma evolução natural e bem planejada da Fase 1, adicionando automação sem comprometer a estabilidade ou funcionalidade existente. A implementação incremental permite validação contínua e minimiza riscos, garantindo que a automação seja construída sobre uma base sólida e testada.

Fases detalhadas - Fase 1

Semana 1: Análise e Fundação

** Análise Técnica Detalhada **

A primeira etapa envolve análise profunda do código base do LibreChat para identificar pontos exatos de integração e possíveis conflitos. Esta análise é crucial para garantir que as modificações sejam minimamente invasivas e não comprometam a estabilidade do sistema existente.

Atividades específicas incluem mapeamento da arquitetura de middleware existente, identificação de pontos de interceptação para verificação de tokens, análise do sistema de autenticação e autorização, e documentação de dependências críticas. O resultado desta análise será um documento técnico detalhado que servirá como base para toda a implementação.

A análise também inclui revisão das configurações de banco de dados existentes, estrutura de tabelas do MongoDB utilizado pelo LibreChat, e identificação de padrões de nomenclatura e convenções de código que devem ser seguidas para manter consistência.

**Setup do Ambiente e Estrutura Base **

Configuração completa do ambiente de desenvolvimento, incluindo clonagem e configuração do LibreChat, setup de banco de dados de desenvolvimento, e criação da estrutura base para os novos componentes. Esta etapa garante que o ambiente esteja completamente funcional antes do início do desenvolvimento.

Criação das tabelas de banco de dados necessárias (`user_tokens` e `token_usage`), implementação de scripts de migração, e configuração de variáveis de ambiente específicas para controle de tokens. Testes básicos de conectividade e funcionalidade são executados para validar o setup.

Implementação da estrutura base de arquivos e diretórios para os novos componentes, seguindo padrões estabelecidos pelo LibreChat. Configuração de ferramentas de desenvolvimento como linting, formatação de código, e debugging.

**Documentação e Planejamento Detalhado **

Criação de documentação técnica detalhada baseada na análise realizada, incluindo diagramas de integração, especificações de interface, e plano de testes. Esta documentação servirá como guia durante toda a implementação e facilitará manutenção futura.

Refinamento do cronograma baseado nas descobertas da análise técnica, identificação de possíveis riscos e pontos de atenção, e definição de critérios de aceitação para cada componente a ser desenvolvido.

Semana 2:Implementação Core

** Token Middleware e Service **

Desenvolvimento do componente central do sistema: o Token Middleware responsável por interceptar requisições e verificar saldos de tokens. Este componente é implementado como uma extensão do sistema de middleware existente do LibreChat, garantindo integração transparente.

O Token Service é desenvolvido em paralelo, encapsulando toda a lógica de negócio relacionada ao gerenciamento de tokens. Implementação inclui métodos para consulta de saldo, dedução de tokens, aplicação do fator de conversão (0.376), e gerenciamento de estados de usuário.

Testes unitários são desenvolvidos para cada método do Token Service, garantindo que a lógica de negócio funcione corretamente em todos os cenários previstos. Validação especial é dada ao cálculo de conversão de tokens e às regras de bloqueio.

** Sistema de Bloqueio e Notificações **

Implementação do Block Service responsável por gerenciar estados de bloqueio de usuários e coordenar ações quando tokens são esgotados. O sistema é projetado para ser robusto e confiável, garantindo que usuários sem saldo não possam consumir recursos.

Desenvolvimento do Email Service para automatizar comunicação com usuários e administradores. Implementação inclui templates de email, sistema de fila para envio assíncrono, e integração com provedores SMTP. Templates são criados para diferentes tipos de notificação: bloqueio, saldo baixo, e instruções para compra.

Integração entre Block Service e Email Service para garantir que notificações sejam enviadas automaticamente quando eventos relevantes ocorrem. Testes de integração

validam que o fluxo completo funciona corretamente.

**Integração com LibreChat **

Integração dos componentes desenvolvidos com o código base do LibreChat, incluindo modificação das rotas relevantes para incluir verificação de tokens. Esta integração é feita de forma cuidadosa para minimizar impacto na funcionalidade existente.

Configuração de variáveis de ambiente necessárias, atualização de arquivos de configuração, e implementação de feature flags para permitir ativação/desativação do sistema de controle de tokens. Testes básicos de integração validam que o sistema funciona corretamente com o LibreChat.

Interface e Processo Manual

**Interface Administrativa **

Desenvolvimento de interface web simples para administradores gerenciarem créditos de usuários. A interface é implementada como extensão do painel administrativo existente do LibreChat, mantendo consistência visual e de navegação.

Funcionalidades implementadas incluem busca de usuários por email ou ID, visualização de saldo atual e histórico de uso, formulário para adição manual de créditos, e relatórios básicos de consumo. A interface é responsiva e funciona adequadamente em dispositivos móveis.

Sistema de autenticação e autorização é implementado para garantir que apenas administradores autorizados possam acessar as funcionalidades de gerenciamento de tokens. Logs de auditoria registram todas as ações administrativas para rastreabilidade.

** Processo Manual de Compras**

Implementação do fluxo completo de processo manual de compras, desde a detecção de esgotamento de tokens até a adição manual de créditos pelo administrador. O processo é projetado para ser eficiente e minimizar tempo de resposta para usuários.

Desenvolvimento de templates de email específicos para solicitação de créditos, incluindo informações relevantes como saldo atual, histórico de uso, e opções de pacotes disponíveis. Sistema de tracking permite acompanhar status de solicitações pendentes.

Implementação de workflow para administradores processarem solicitações, incluindo validação de informações, cálculo de créditos a serem adicionados, e confirmação automática para usuários. Sistema de notificações mantém usuários informados sobre status de suas solicitações.

** Testes de Integração **

Execução de testes de integração abrangentes para validar que todos os componentes funcionam corretamente em conjunto. Testes incluem cenários de uso normal, casos extremos, e situações de erro.

Validação específica do fluxo completo: usuário esgota tokens, sistema bloqueia acesso, notificações são enviadas, administrador processa solicitação, créditos são adicionados, usuário é desbloqueado. Cada etapa é testada individualmente e em conjunto.

Testes de performance básicos para garantir que o sistema não introduz latência significativa nas operações do LibreChat. Validação de que verificações de token ocorrem em menos de 50ms para 95% das requisições.

Finalização e Deploy

Testes Finais e Correções

Execução de bateria completa de testes incluindo testes unitários, de integração, e end-to-end. Identificação e correção de bugs encontrados durante os testes. Validação de que todos os critérios de aceitação definidos foram atendidos.

Testes de carga básicos para validar comportamento do sistema sob uso normal e picos de tráfego. Otimizações de performance são implementadas conforme necessário para garantir que o sistema atenda aos SLAs definidos.

Revisão de código completa para garantir qualidade, segurança, e aderência aos padrões estabelecidos. Documentação de código é atualizada e comentários são adicionados onde necessário para facilitar manutenção futura.

** Documentação e Treinamento **

Criação de documentação completa para usuários finais e administradores, incluindo guias de uso, troubleshooting, e procedimentos operacionais. Documentação técnica é finalizada

com diagramas atualizados e especificações de API.

Desenvolvimento de material de treinamento para equipe administrativa, incluindo procedimentos para processamento manual de solicitações de créditos, uso da interface administrativa, e resolução de problemas comuns.

Criação de runbooks operacionais para monitoramento do sistema, identificação de problemas, e procedimentos de escalação. Documentação inclui métricas importantes a serem monitoradas e alertas a serem configurados.

**Deploy em Produção **

Execução do deploy em ambiente de produção seguindo procedimentos estabelecidos. Backup completo do sistema existente é realizado antes do deploy para permitir rollback se necessário.

Migração de dados é executada, incluindo criação das novas tabelas e população com dados iniciais. Configuração de variáveis de ambiente de produção e ativação gradual do sistema de controle de tokens.

Monitoramento intensivo durante as primeiras horas após deploy para identificar rapidamente qualquer problema. Validação de que todas as funcionalidades estão operando corretamente em ambiente de produção.

Fase detalhada - Fase 1

Integração Stripe

** Setup Stripe e Configuração **

Configuração completa da conta Stripe incluindo criação de produtos, definição de preços, e configuração de webhooks. Setup de ambiente de desenvolvimento com chaves de teste para permitir desenvolvimento e testes seguros.

Implementação do Stripe Service responsável por encapsular toda a lógica de integração com a API do Stripe. Desenvolvimento inclui métodos para criação de sessões de checkout, gerenciamento de produtos, e processamento de webhooks.

Configuração de produtos de teste com preços reduzidos para facilitar testes durante desenvolvimento. Implementação de sistema de feature flags para permitir alternância entre ambiente de teste e produção.

** Interface de Compra **

Desenvolvimento da interface de compra integrada que substitui o processo manual quando usuários são bloqueados. A interface é implementada como modal responsivo que se integra naturalmente com o fluxo existente.

Implementação de componentes React para seleção de pacotes, exibição de preços, e integração com Stripe Checkout. A interface segue padrões visuais do LibreChat e fornece experiência de usuário fluida e profissional.

Desenvolvimento de lógica de fallback para casos onde a integração Stripe não está disponível, garantindo que usuários sempre tenham opção de solicitar créditos via processo manual.

** Testes de Integração Stripe **

Execução de testes abrangentes da integração Stripe incluindo criação de sessões de checkout, processamento de pagamentos de teste, e validação de webhooks. Testes cobrem cenários de sucesso, falha, e cancelamento.

Validação de que dados de transação são corretamente registrados no banco de dados e que créditos são adicionados automaticamente após confirmação de pagamento. Testes de idempotência garantem que webhooks duplicados não resultem em créditos duplicados.

Automação Completa

** Webhook Processing **

Implementação do sistema robusto de processamento de webhooks incluindo fila assíncrona, sistema de retry, e tratamento de falhas. O sistema é projetado para garantir que nenhum pagamento confirmado seja perdido.

Desenvolvimento de lógica de reconciliação para casos onde webhooks falham ou são perdidos, permitindo identificação e processamento manual de pagamentos não processados automaticamente.

Implementação de sistema de alertas para administradores quando problemas são detectados no processamento de webhooks, garantindo que intervenção manual possa ser realizada rapidamente quando necessário.

** Notificações Automáticas**

Expansão do sistema de notificações para suportar eventos relacionados a compras automáticas, incluindo confirmações de pagamento, falhas de processamento, e atualizações de status.

Desenvolvimento de templates de email específicos para automação, incluindo confirmações de compra, recibos detalhados, e notificações de problemas. Templates são responsivos e incluem informações relevantes para usuários.

Implementação de sistema de notificações em tempo real na interface do usuário para fornecer feedback imediato sobre status de compras e atualizações de saldo.

**Testes Finais e Deploy **

Execução de testes end-to-end completos do sistema automatizado, incluindo fluxo completo desde bloqueio até desbloqueio automático após compra. Validação de que todos os componentes funcionam corretamente em conjunto.

Deploy da Fase 2 em produção com ativação gradual da automação. Monitoramento intensivo durante primeiras horas para garantir que automação funciona corretamente e não introduz problemas.

Marcos da Fase 1

Marco		Entregável	Critério de Aceitação
M1.1		Análise Técnica Completa	Documento técnico aprovado
M1.2		Core System Funcional	Testes unitários passando
M1.3		Interface Admin Completa	Interface funcional e testada
M1.4		Sistema em Produção	Deploy realizado e validado

Marcos da Fase 2

Marco		Entregável	Critério de Aceitação
M2.1		Integração Stripe Básica	Checkout funcionando em teste
M2.2		Interface Compra Completa	Interface integrada e testada
M2.3		Automação Completa	Webhooks processando corretamente
M2.4		Sistema Automatizado	Automação em produção

Métricas de Sucesso e Validação

Métricas Técnicas

Performance:

- Verificação de tokens: < 50ms (95% das requisições)
- Carregamento interface admin: < 2 segundos
- Processamento webhook: < 5 segundos
- Uptime do sistema: > 99.5%

Documentação de Handover

Documentação Técnica:

- Código comentado e documentado
- Diagramas de arquitetura atualizados
- Procedimentos de deploy e rollback
- Guia de troubleshooting

Documentação Operacional:

- Procedimentos de monitoramento

- Alertas e métricas importantes
- Procedimentos de backup e recuperação
- Contatos para suporte técnico

💻 Implementação MVP Completa

Estrutura de Arquivos e Organização

A implementação MVP segue uma estrutura organizada que se integra naturalmente com o código base do LibreChat, minimizando modificações no código existente e facilitando manutenção futura. A organização de arquivos foi projetada para ser intuitiva e seguir padrões estabelecidos pela comunidade Node.js.

Plain Text

```
librechat/
├── api/
│   ├── middleware/
│   │   └── tokenControl.js          # Middleware principal de controle
│   ├── services/
│   │   ├── TokenService.js          # Lógica de negócio de tokens
│   │   ├── BlockService.js          # Gerenciamento de bloqueios
│   │   ├── EmailService.js          # Notificações por email
│   │   └── StripeService.js         # Integração Stripe (Fase 2)
│   ├── routes/
│   │   ├── tokenAdmin.js            # Rotas administrativas
│   │   └── tokenPurchase.js         # Rotas de compra (Fase 2)
│   └── models/
│       ├── UserToken.js             # Modelo de dados de tokens
│       └── TokenUsage.js            # Modelo de histórico de uso
├── client/
│   ├── src/
│   │   ├── components/
│   │   │   ├── TokenAdmin/          # Interface administrativa
│   │   │   │   ├── TokenDashboard.jsx
│   │   │   │   ├── UserTokenManager.jsx
│   │   │   │   └── TokenHistory.jsx
│   │   │   └── TokenPurchase/       # Interface de compra (Fase 2)
│   │   │       ├── PurchaseModal.jsx
│   │   │       ├── PackageSelector.jsx
│   │   │       └── PurchaseSuccess.jsx
```

```
│   │   └── hooks/
│   │       ├── useTokenBalance.js   # Hook para saldo de tokens
│   │       └── useTokenPurchase.js  # Hook para compras (Fase 2)
├── config/
│   └── tokenConfig.js               # Configurações do sistema
└── migrations/
    ├── 001_create_user_tokens.js    # Migração inicial
    └── 002_create_token_usage.js    # Migração de histórico
```

Configuração e Variáveis de Ambiente

Arquivo de Configuração Principal

JavaScript

```javascript
// config/tokenConfig.js
const tokenConfig = {
  // Configurações básicas
  enabled: process.env.TOKEN_CONTROL_ENABLED === 'true',
  conversionFactor: 0.376, // Fator de conversão LibreChat -> OpenAI

  // Limites padrão
  defaultMonthlyLimit: 1000000, // 1M tokens por mês
  lowBalanceThreshold: 0.2,     // 20% do limite
  criticalBalanceThreshold: 0.1, // 10% do limite

  // Configurações de email
  email: {
    enabled: process.env.EMAIL_NOTIFICATIONS_ENABLED === 'true',
    from: process.env.EMAIL_FROM || 'noreply@iasolaris.com.br',
    adminEmail: process.env.ADMIN_EMAIL || 'admin@iasolaris.com.br',
    smtp: {
      host: process.env.SMTP_HOST,
      port: parseInt(process.env.SMTP_PORT) || 587,
      secure: process.env.SMTP_SECURE === 'true',
      auth: {
        user: process.env.SMTP_USER,
        pass: process.env.SMTP_PASS
      }
    }
  },

  // Configurações de banco de dados
  database: {
    mongodb: {
      uri: process.env.MONGODB_URI,
```

```
      options: {
        useNewUrlParser: true,
        useUnifiedTopology: true
      }
    }
  },

  // Configurações Stripe (Fase 2)
  stripe: {
    enabled: process.env.STRIPE_ENABLED === 'true',
    publishableKey: process.env.STRIPE_PUBLISHABLE_KEY,
    secretKey: process.env.STRIPE_SECRET_KEY,
    webhookSecret: process.env.STRIPE_WEBHOOK_SECRET,
    successUrl: process.env.STRIPE_SUCCESS_URL ||
'https://iasolaris.com.br/purchase/success',
    cancelUrl: process.env.STRIPE_CANCEL_URL ||
'https://iasolaris.com.br/purchase/cancel'
  },

  // Pacotes de tokens disponíveis
  packages: [
    {
      id: 'basic',
      name: 'Pacote Básico',
      tokens: 200000,
      price: 5000, // em centavos (R$ 50,00)
      description: '200 mil tokens adicionais'
    },
    {
      id: 'standard',
      name: 'Pacote Padrão',
      tokens: 500000,
      price: 12000, // em centavos (R$ 120,00)
      description: '500 mil tokens adicionais'
    },
    {
      id: 'premium',
      name: 'Pacote Premium',
      tokens: 1000000,
      price: 23000, // em centavos (R$ 230,00)
      description: '1 milhão de tokens adicionais'
    }
  ]
};

module.exports = tokenConfig;
```

Variáveis de Ambiente (.env)

Bash

```
# Controle de Tokens - Configurações Básicas
TOKEN_CONTROL_ENABLED=true
DEFAULT_MONTHLY_LIMIT=1000000

# Notificações por Email
EMAIL_NOTIFICATIONS_ENABLED=true
EMAIL_FROM=noreply@iasolaris.com.br
ADMIN_EMAIL=admin@iasolaris.com.br

# Configurações SMTP
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_SECURE=false
SMTP_USER=seu-email@gmail.com
SMTP_PASS=sua-senha-app

# Banco de Dados
MONGODB_URI=mongodb://localhost:27017/librechat

# Stripe (Fase 2)
STRIPE_ENABLED=false
STRIPE_PUBLISHABLE_KEY=pk_test_...
STRIPE_SECRET_KEY=sk_test_...
STRIPE_WEBHOOK_SECRET=whsec_...
STRIPE_SUCCESS_URL=https://iasolaris.com.br/purchase/success
STRIPE_CANCEL_URL=https://iasolaris.com.br/purchase/cancel

# URLs de Retorno
BASE_URL=https://iasolaris.com.br
```

Implementação dos Modelos de Dados

Modelo UserToken

JavaScript

```
// api/models/UserToken.js
const mongoose = require('mongoose');

const userTokenSchema = new mongoose.Schema({
  userId: {
```

```
    type: String,
    required: true,
    unique: true,
    index: true
  },
  email: {
    type: String,
    required: true,
    index: true
  },

  // Saldos de tokens
  monthlyLimit: {
    type: Number,
    default: 1000000,
    min: 0
  },
  monthlyUsed: {
    type: Number,
    default: 0,
    min: 0
  },
  additionalCredits: {
    type: Number,
    default: 0,
    min: 0
  },

  // Status do usuário
  isBlocked: {
    type: Boolean,
    default: false,
    index: true
  },
  blockReason: {
    type: String,
    enum: ['insufficient_tokens', 'manual_block', 'payment_failed'],
    default: null
  },
  blockedAt: {
    type: Date,
    default: null
  },

  // Controle mensal
  currentMonth: {
    type: String, // YYYY-MM
    default: () => new Date().toISOString().slice(0, 7)
```

```
  },
  lastResetAt: {
    type: Date,
    default: Date.now
  },

  // Notificações
  lastLowBalanceAlert: {
    type: Date,
    default: null
  },
  lastCriticalBalanceAlert: {
    type: Date,
    default: null
  },

  // Metadados
  createdAt: {
    type: Date,
    default: Date.now
  },
  updatedAt: {
    type: Date,
    default: Date.now
  }
}, {
  timestamps: true
});

// Índices compostos para performance
userTokenSchema.index({ userId: 1, currentMonth: 1 });
userTokenSchema.index({ isBlocked: 1, blockedAt: 1 });
userTokenSchema.index({ email: 1, isBlocked: 1 });

// Middleware para atualizar updatedAt
userTokenSchema.pre('save', function(next) {
  this.updatedAt = new Date();
  next();
});

// Métodos do modelo
userTokenSchema.methods.getAvailableTokens = function() {
  const monthlyAvailable = Math.max(0, this.monthlyLimit - this.monthlyUsed);
  return monthlyAvailable + this.additionalCredits;
};

userTokenSchema.methods.canUseTokens = function(tokensNeeded) {
  if (this.isBlocked) return false;
```

```
  return this.getAvailableTokens() >= tokensNeeded;
};

userTokenSchema.methods.useTokens = function(tokensUsed) {
  const availableTokens = this.getAvailableTokens();
  if (tokensUsed > availableTokens) {
    throw new Error('Tokens insuficientes');
  }

  // Usar primeiro os tokens mensais, depois os adicionais
  const monthlyAvailable = Math.max(0, this.monthlyLimit - this.monthlyUsed);

  if (tokensUsed <= monthlyAvailable) {
    this.monthlyUsed += tokensUsed;
  } else {
    this.monthlyUsed = this.monthlyLimit;
    this.additionalCredits -= (tokensUsed - monthlyAvailable);
  }

  return this.save();
};

userTokenSchema.methods.addTokens = function(tokens, type = 'additional') {
  if (type === 'monthly') {
    this.monthlyLimit += tokens;
  } else {
    this.additionalCredits += tokens;
  }

  // Desbloquear usuário se necessário
  if (this.isBlocked && this.getAvailableTokens() > 0) {
    this.isBlocked = false;
    this.blockReason = null;
    this.blockedAt = null;
  }

  return this.save();
};

userTokenSchema.methods.resetMonthlyUsage = function() {
  const currentMonth = new Date().toISOString().slice(0, 7);

  if (this.currentMonth !== currentMonth) {
    this.monthlyUsed = 0;
    this.currentMonth = currentMonth;
    this.lastResetAt = new Date();

    // Desbloquear se foi bloqueado por falta de tokens mensais
```

```
    if (this.isBlocked && this.blockReason === 'insufficient_tokens') {
      this.isBlocked = false;
      this.blockReason = null;
      this.blockedAt = null;
    }

    return this.save();
  }

  return Promise.resolve(this);
};

userTokenSchema.methods.blockUser = function(reason = 'insufficient_tokens')
{
  this.isBlocked = true;
  this.blockReason = reason;
  this.blockedAt = new Date();
  return this.save();
};

module.exports = mongoose.model('UserToken', userTokenSchema);
```

Modelo TokenUsage

JavaScript

```
// api/models/TokenUsage.js
const mongoose = require('mongoose');

const tokenUsageSchema = new mongoose.Schema({
  userId: {
    type: String,
    required: true,
    index: true
  },

  // Detalhes do uso
  tokensUsed: {
    type: Number,
    required: true,
    min: 0
  },
  tokensRequested: {
    type: Number,
    required: true,
    min: 0
  },
```

```
  conversionFactor: {
    type: Number,
    default: 0.376
  },

  // Contexto da requisição
  model: {
    type: String,
    required: true
  },
  endpoint: {
    type: String,
    required: true
  },
  conversationId: {
    type: String,
    index: true
  },
  messageId: {
    type: String
  },

  // Metadados
  userAgent: String,
  ipAddress: String,

  // Timestamps
  timestamp: {
    type: Date,
    default: Date.now,
    index: true
  },
  month: {
    type: String, // YYYY-MM
    default: () => new Date().toISOString().slice(0, 7),
    index: true
  }
}, {
  timestamps: true
});

// Índices compostos para relatórios
tokenUsageSchema.index({ userId: 1, month: 1 });
tokenUsageSchema.index({ userId: 1, timestamp: -1 });
tokenUsageSchema.index({ model: 1, timestamp: -1 });
tokenUsageSchema.index({ month: 1, timestamp: -1 });

// Métodos estáticos para relatórios
```

```
tokenUsageSchema.statics.getUserUsageByMonth = function(userId, month) {
  return this.aggregate([
    {
      $match: {
        userId: userId,
        month: month || new Date().toISOString().slice(0, 7)
      }
    },
    {
      $group: {
        _id: null,
        totalTokens: { $sum: '$tokensUsed' },
        totalRequests: { $sum: 1 },
        avgTokensPerRequest: { $avg: '$tokensUsed' },
        models: { $addToSet: '$model' }
      }
    }
  ]);
};

tokenUsageSchema.statics.getTopUsers = function(month, limit = 10) {
  return this.aggregate([
    {
      $match: {
        month: month || new Date().toISOString().slice(0, 7)
      }
    },
    {
      $group: {
        _id: '$userId',
        totalTokens: { $sum: '$tokensUsed' },
        totalRequests: { $sum: 1 }
      }
    },
    {
      $sort: { totalTokens: -1 }
    },
    {
      $limit: limit
    }
  ]);
};

module.exports = mongoose.model('TokenUsage', tokenUsageSchema);
```

Implementação dos Services

TokenService - Lógica Principal

JavaScript

```javascript
// api/services/TokenService.js
const UserToken = require('../models/UserToken');
const TokenUsage = require('../models/TokenUsage');
const EmailService = require('./EmailService');
const tokenConfig = require('../../config/tokenConfig');

class TokenService {
  constructor() {
    this.conversionFactor = tokenConfig.conversionFactor;
    this.lowBalanceThreshold = tokenConfig.lowBalanceThreshold;
    this.criticalBalanceThreshold = tokenConfig.criticalBalanceThreshold;
  }

  /**
   * Inicializa usuário no sistema de tokens
   */
  async initializeUser(userId, email, monthlyLimit = null) {
    try {
      let userToken = await UserToken.findOne({ userId });

      if (!userToken) {
        userToken = new UserToken({
          userId,
          email,
          monthlyLimit: monthlyLimit || tokenConfig.defaultMonthlyLimit
        });
        await userToken.save();
      }

      // Verificar se precisa resetar mês
      await userToken.resetMonthlyUsage();

      return userToken;
    } catch (error) {
      console.error('Erro ao inicializar usuário:', error);
      throw error;
    }
  }

  /**
   * Verifica se usuário pode usar tokens
   */
  async canUseTokens(userId, tokensRequested) {
```

```
    try {
      const userToken = await UserToken.findOne({ userId });

      if (!userToken) {
        throw new Error('Usuário não encontrado no sistema de tokens');
      }

      // Resetar mês se necessário
      await userToken.resetMonthlyUsage();

      // Aplicar fator de conversão
      const tokensNeeded = Math.ceil(tokensRequested *
this.conversionFactor);

      return {
        canUse: userToken.canUseTokens(tokensNeeded),
        availableTokens: userToken.getAvailableTokens(),
        tokensNeeded,
        isBlocked: userToken.isBlocked,
        blockReason: userToken.blockReason
      };
    } catch (error) {
      console.error('Erro ao verificar tokens:', error);
      throw error;
    }
  }

  /**
   * Consome tokens do usuário
   */
  async useTokens(userId, tokensRequested, context = {}) {
    try {
      const userToken = await UserToken.findOne({ userId });

      if (!userToken) {
        throw new Error('Usuário não encontrado no sistema de tokens');
      }

      // Aplicar fator de conversão
      const tokensToUse = Math.ceil(tokensRequested * this.conversionFactor);

      // Verificar se pode usar
      if (!userToken.canUseTokens(tokensToUse)) {
        // Bloquear usuário
        await userToken.blockUser('insufficient_tokens');

        // Enviar notificação
        await EmailService.sendBlockedNotification(userToken);
```

```
        throw new Error('Tokens insuficientes - usuário bloqueado');
      }

      // Usar tokens
      await userToken.useTokens(tokensToUse);

      // Registrar uso
      await this.recordUsage(userId, tokensToUse, tokensRequested, context);

      // Verificar alertas de saldo baixo
      await this.checkBalanceAlerts(userToken);

      return {
        success: true,
        tokensUsed: tokensToUse,
        remainingTokens: userToken.getAvailableTokens()
      };

    } catch (error) {
      console.error('Erro ao usar tokens:', error);
      throw error;
    }
  }

  /**
   * Registra uso de tokens
   */
  async recordUsage(userId, tokensUsed, tokensRequested, context) {
    try {
      const usage = new TokenUsage({
        userId,
        tokensUsed,
        tokensRequested,
        conversionFactor: this.conversionFactor,
        model: context.model || 'unknown',
        endpoint: context.endpoint || 'unknown',
        conversationId: context.conversationId,
        messageId: context.messageId,
        userAgent: context.userAgent,
        ipAddress: context.ipAddress
      });

      await usage.save();
      return usage;
    } catch (error) {
      console.error('Erro ao registrar uso:', error);
      // Não falhar a requisição por erro de log
```

```
    }
  }

  /**
   * Verifica alertas de saldo baixo
   */
  async checkBalanceAlerts(userToken) {
    try {
      const availableTokens = userToken.getAvailableTokens();
      const totalLimit = userToken.monthlyLimit +
userToken.additionalCredits;
      const usagePercentage = 1 - (availableTokens / totalLimit);

      const now = new Date();
      const oneDayAgo = new Date(now.getTime() - 24 * 60 * 60 * 1000);

      // Alerta crítico (90%)
      if (usagePercentage >= 0.9 &&
          (!userToken.lastCriticalBalanceAlert ||
userToken.lastCriticalBalanceAlert < oneDayAgo)) {

        await EmailService.sendCriticalBalanceAlert(userToken,
availableTokens);
        userToken.lastCriticalBalanceAlert = now;
        await userToken.save();
      }
      // Alerta de saldo baixo (80%)
      else if (usagePercentage >= 0.8 &&
               (!userToken.lastLowBalanceAlert ||
userToken.lastLowBalanceAlert < oneDayAgo)) {

        await EmailService.sendLowBalanceAlert(userToken, availableTokens);
        userToken.lastLowBalanceAlert = now;
        await userToken.save();
      }
    } catch (error) {
      console.error('Erro ao verificar alertas:', error);
      // Não falhar por erro de notificação
    }
  }

  /**
   * Adiciona tokens ao usuário
   */
  async addTokens(userId, tokens, type = 'additional', source = 'manual') {
    try {
      const userToken = await UserToken.findOne({ userId });
```

```
      if (!userToken) {
        throw new Error('Usuário não encontrado');
      }

      await userToken.addTokens(tokens, type);

      // Registrar transação
      await this.recordTokenTransaction(userId, tokens, type, source);

      return {
        success: true,
        newBalance: userToken.getAvailableTokens(),
        tokensAdded: tokens
      };
    } catch (error) {
      console.error('Erro ao adicionar tokens:', error);
      throw error;
    }
  }

  /**
   * Registra transação de tokens
   */
  async recordTokenTransaction(userId, tokens, type, source) {
    // Implementar log de transações se necessário
    console.log(`Token transaction: ${userId} +${tokens} (${type}) from
${source}`);
  }

  /**
   * Obtém saldo do usuário
   */
  async getUserBalance(userId) {
    try {
      const userToken = await UserToken.findOne({ userId });

      if (!userToken) {
        return null;
      }

      await userToken.resetMonthlyUsage();

      return {
        userId: userToken.userId,
        email: userToken.email,
        monthlyLimit: userToken.monthlyLimit,
        monthlyUsed: userToken.monthlyUsed,
        monthlyRemaining: Math.max(0, userToken.monthlyLimit -
```

```
userToken.monthlyUsed),
        additionalCredits: userToken.additionalCredits,
        totalAvailable: userToken.getAvailableTokens(),
        isBlocked: userToken.isBlocked,
        blockReason: userToken.blockReason,
        currentMonth: userToken.currentMonth
      };
    } catch (error) {
      console.error('Erro ao obter saldo:', error);
      throw error;
    }
  }

  /**
   * Lista usuários com filtros
   */
  async listUsers(filters = {}) {
    try {
      const query = {};

      if (filters.isBlocked !== undefined) {
        query.isBlocked = filters.isBlocked;
      }

      if (filters.email) {
        query.email = new RegExp(filters.email, 'i');
      }

      const users = await UserToken.find(query)
        .sort({ updatedAt: -1 })
        .limit(filters.limit || 50);

      return users.map(user => ({
        userId: user.userId,
        email: user.email,
        monthlyLimit: user.monthlyLimit,
        monthlyUsed: user.monthlyUsed,
        additionalCredits: user.additionalCredits,
        totalAvailable: user.getAvailableTokens(),
        isBlocked: user.isBlocked,
        blockReason: user.blockReason,
        lastActivity: user.updatedAt
      }));
    } catch (error) {
      console.error('Erro ao listar usuários:', error);
      throw error;
    }
  }
```

```
  /**
   * Desbloqueia usuário
   */
  async unblockUser(userId) {
    try {
      const userToken = await UserToken.findOne({ userId });

      if (!userToken) {
        throw new Error('Usuário não encontrado');
      }

      userToken.isBlocked = false;
      userToken.blockReason = null;
      userToken.blockedAt = null;

      await userToken.save();

      // Notificar usuário
      await EmailService.sendUnblockedNotification(userToken);

      return { success: true };
    } catch (error) {
      console.error('Erro ao desbloquear usuário:', error);
      throw error;
    }
  }

  /**
   * Relatório de uso mensal
   */
  async getMonthlyReport(month = null) {
    try {
      const targetMonth = month || new Date().toISOString().slice(0, 7);

      const [totalUsage, topUsers, modelUsage] = await Promise.all([
        TokenUsage.aggregate([
          { $match: { month: targetMonth } },
          {
            $group: {
              _id: null,
              totalTokens: { $sum: '$tokensUsed' },
              totalRequests: { $sum: 1 },
              uniqueUsers: { $addToSet: '$userId' }
            }
          }
        ]),
        TokenUsage.getTopUsers(targetMonth, 10),
```

```
        TokenUsage.aggregate([
          { $match: { month: targetMonth } },
          {
            $group: {
              _id: '$model',
              totalTokens: { $sum: '$tokensUsed' },
              totalRequests: { $sum: 1 }
            }
          },
          { $sort: { totalTokens: -1 } }
        ])
      ]);

      return {
        month: targetMonth,
        summary: totalUsage[0] || { totalTokens: 0, totalRequests: 0,
uniqueUsers: [] },
        topUsers,
        modelUsage
      };
    } catch (error) {
      console.error('Erro ao gerar relatório:', error);
      throw error;
    }
  }
}

module.exports = new TokenService();
```

EmailService - Notificações

JavaScript

```
// api/services/EmailService.js
const nodemailer = require('nodemailer');
const tokenConfig = require('../../config/tokenConfig');

class EmailService {
  constructor() {
    this.transporter = null;
    this.initializeTransporter();
  }

  initializeTransporter() {
    if (!tokenConfig.email.enabled) {
      console.log('Email notifications disabled');
      return;
```

```
    }

    try {
      this.transporter =
nodemailer.createTransporter(tokenConfig.email.smtp);

      // Verificar conexão
      this.transporter.verify((error, success) => {
        if (error) {
          console.error('Erro na configuração SMTP:', error);
        } else {
          console.log('SMTP configurado com sucesso');
        }
      });
    } catch (error) {
      console.error('Erro ao inicializar transporter:', error);
    }
  }

  async sendEmail(to, subject, html, text = null) {
    if (!this.transporter) {
      console.log('Email não enviado - transporter não configurado');
      return false;
    }

    try {
      const mailOptions = {
        from: tokenConfig.email.from,
        to,
        subject,
        html,
        text: text || html.replace(/<[^>]*>/g, '') // Strip HTML for text
version
      };

      const result = await this.transporter.sendMail(mailOptions);
      console.log('Email enviado:', result.messageId);
      return true;
    } catch (error) {
      console.error('Erro ao enviar email:', error);
      return false;
    }
  }

  async sendBlockedNotification(userToken) {
    const subject = 'IA SOLARIS - Acesso bloqueado por falta de tokens';
    const html = `
      <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0
```

```
auto;">
        <h2 style="color: #d32f2f;">Acesso Bloqueado - Tokens Esgotados</h2>

        <p>Olá,</p>

        <p>Seu acesso à IA SOLARIS foi temporariamente bloqueado porque você
esgotou seus tokens disponíveis.</p>

        <div style="background: #f5f5f5; padding: 15px; border-radius: 5px;
margin: 20px 0;">
          <h3>Informações da sua conta:</h3>
          <ul>
            <li><strong>Email:</strong> ${userToken.email}</li>
            <li><strong>Limite mensal:</strong>
${userToken.monthlyLimit.toLocaleString()} tokens</li>
            <li><strong>Tokens usados este mês:</strong>
${userToken.monthlyUsed.toLocaleString()}</li>
            <li><strong>Créditos adicionais:</strong>
${userToken.additionalCredits.toLocaleString()}</li>
          </ul>
        </div>

        <h3>Como reativar seu acesso:</h3>
        <p>Para continuar usando a IA SOLARIS, você pode:</p>
        <ol>
          <li><strong>Aguardar o próximo mês:</strong> Seus tokens mensais
serão renovados automaticamente</li>
          <li><strong>Comprar créditos adicionais:</strong> Entre em contato
conosco para adquirir mais tokens</li>
        </ol>

        <div style="background: #e3f2fd; padding: 15px; border-radius: 5px;
margin: 20px 0;">
          <h4>💰 Pacotes Disponíveis:</h4>
          <ul>
            <li><strong>Básico:</strong> 200.000 tokens - R$ 50,00</li>
            <li><strong>Padrão:</strong> 500.000 tokens - R$ 120,00</li>
            <li><strong>Premium:</strong> 1.000.000 tokens - R$ 230,00</li>
          </ul>
        </div>

        <p><strong>Para comprar créditos, responda este email informando:
</strong></p>
        <ul>
          <li>Pacote desejado</li>
          <li>Forma de pagamento preferida</li>
        </ul>
```

```
        <p>Processaremos sua solicitação em até 2 horas úteis.</p>

        <hr style="margin: 30px 0;">
        <p style="color: #666; font-size: 12px;">
          Este é um email automático do sistema IA SOLARIS.<br>
          Em caso de dúvidas, entre em contato conosco.
        </p>
      </div>
    `;

    await this.sendEmail(userToken.email, subject, html);

    // Notificar administradores
    await this.sendAdminNotification('Usuário Bloqueado', `
      Usuário ${userToken.email} foi bloqueado por falta de tokens.

      Detalhes:
      - User ID: ${userToken.userId}
      - Tokens mensais usados: ${userToken.monthlyUsed}
      - Créditos adicionais: ${userToken.additionalCredits}
      - Bloqueado em: ${new Date().toLocaleString('pt-BR')}
    `);
  }

  async sendLowBalanceAlert(userToken, availableTokens) {
    const subject = 'IA SOLARIS - Saldo baixo de tokens';
    const html = `
      <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0
auto;">
        <h2 style="color: #ff9800;">⚠️ Saldo Baixo de Tokens</h2>

        <p>Olá,</p>

        <p>Você já utilizou mais de 80% dos seus tokens disponíveis na IA
SOLARIS.</p>

        <div style="background: #fff3e0; padding: 15px; border-radius: 5px;
margin: 20px 0;">
          <h3>Status atual da sua conta:</h3>
          <ul>
            <li><strong>Tokens restantes:</strong>
${availableTokens.toLocaleString()}</li>
            <li><strong>Limite mensal:</strong>
${userToken.monthlyLimit.toLocaleString()}</li>
            <li><strong>Créditos adicionais:</strong>
${userToken.additionalCredits.toLocaleString()}</li>
          </ul>
        </div>
```

```
        <p>Para evitar interrupções no seu acesso, considere adquirir
créditos adicionais.</p>

        <div style="background: #e3f2fd; padding: 15px; border-radius: 5px;
margin: 20px 0;">
          <h4>💰 Pacotes Disponíveis:</h4>
          <ul>
            <li><strong>Básico:</strong> 200.000 tokens - R$ 50,00</li>
            <li><strong>Padrão:</strong> 500.000 tokens - R$ 120,00</li>
            <li><strong>Premium:</strong> 1.000.000 tokens - R$ 230,00</li>
          </ul>
        </div>

        <p>Responda este email para solicitar créditos adicionais.</p>

        <hr style="margin: 30px 0;">
        <p style="color: #666; font-size: 12px;">
          Este alerta é enviado quando você atinge 80% do uso dos seus
tokens.
        </p>
      </div>
    `;

    await this.sendEmail(userToken.email, subject, html);
  }

  async sendCriticalBalanceAlert(userToken, availableTokens) {
    const subject = 'IA SOLARIS - URGENTE: Tokens quase esgotados';
    const html = `
      <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0
auto;">
        <h2 style="color: #d32f2f;">🚨 URGENTE: Tokens Quase Esgotados</h2>

        <p>Olá,</p>

        <p><strong>ATENÇÃO:</strong> Você já utilizou mais de 90% dos seus
tokens disponíveis na IA SOLARIS.</p>

        <div style="background: #ffebee; padding: 15px; border-radius: 5px;
margin: 20px 0; border-left: 4px solid #d32f2f;">
          <h3>⚠️ Status crítico da sua conta:</h3>
          <ul>
            <li><strong>Tokens restantes:</strong>
${availableTokens.toLocaleString()}</li>
            <li><strong>Risco de bloqueio:</strong> ALTO</li>
          </ul>
        </div>
```

```
        <p><strong>Seu acesso será bloqueado automaticamente quando os tokens se esgotarem.</strong></p>

        <div style="background: #e3f2fd; padding: 15px; border-radius: 5px; margin: 20px 0;">
          <h4>💰 Compre créditos AGORA:</h4>
          <ul>
            <li><strong>Básico:</strong> 200.000 tokens - R$ 50,00</li>
            <li><strong>Padrão:</strong> 500.000 tokens - R$ 120,00</li>
            <li><strong>Premium:</strong> 1.000.000 tokens - R$ 230,00</li>
          </ul>
        </div>

        <p><strong>RESPONDA ESTE EMAIL IMEDIATAMENTE</strong> informando o pacote desejado.</p>

        <hr style="margin: 30px 0;">
        <p style="color: #666; font-size: 12px;">
          Este alerta crítico é enviado quando você atinge 90% do uso dos seus tokens.
        </p>
      </div>
    `;

    await this.sendEmail(userToken.email, subject, html);
  }

  async sendUnblockedNotification(userToken) {
    const subject = 'IA SOLARIS - Acesso reativado';
    const html = `
      <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0 auto;">
        <h2 style="color: #4caf50;">✅ Acesso Reativado com Sucesso</h2>

        <p>Olá,</p>

        <p>Ótimas notícias! Seu acesso à IA SOLARIS foi reativado.</p>

        <div style="background: #e8f5e8; padding: 15px; border-radius: 5px; margin: 20px 0;">
          <h3>Status atual da sua conta:</h3>
          <ul>
            <li><strong>Status:</strong> Ativo ✅</li>
            <li><strong>Tokens disponíveis:</strong> ${userToken.getAvailableTokens().toLocaleString()}</li>
            <li><strong>Reativado em:</strong> ${new Date().toLocaleString('pt-BR')}</li>
```

```
          </ul>
        </div>

        <p>Você já pode continuar usando a IA SOLARIS normalmente.</p>

        <p>Obrigado por usar nossos serviços!</p>

        <hr style="margin: 30px 0;">
        <p style="color: #666; font-size: 12px;">
          Este é um email automático do sistema IA SOLARIS.
        </p>
      </div>
    `;

    await this.sendEmail(userToken.email, subject, html);
  }

  async sendAdminNotification(subject, message) {
    if (!tokenConfig.email.adminEmail) {
      return;
    }

    const html = `
      <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0
auto;">
        <h2>IA SOLARIS - Notificação Administrativa</h2>
        <h3>${subject}</h3>
        <pre style="background: #f5f5f5; padding: 15px; border-radius: 5px;
white-space: pre-wrap;">${message}</pre>
        <hr>
        <p style="color: #666; font-size: 12px;">
          Notificação automática do sistema IA SOLARIS - ${new
Date().toLocaleString('pt-BR')}
        </p>
      </div>
    `;

    await this.sendEmail(tokenConfig.email.adminEmail, `[IA SOLARIS Admin]
${subject}`, html);
  }
}

module.exports = new EmailService();
```

Implementação do Middleware Principal

JavaScript

```
// api/middleware/tokenControl.js
const TokenService = require('../services/TokenService');
const tokenConfig = require('../../config/tokenConfig');

/**
 * Middleware principal para controle de tokens
 */
const tokenControlMiddleware = async (req, res, next) => {
  // Verificar se controle de tokens está habilitado
  if (!tokenConfig.enabled) {
    return next();
  }

  // Verificar se é uma rota que consome tokens
  if (!shouldCheckTokens(req)) {
    return next();
  }

  try {
    // Extrair informações do usuário
    const userId = getUserId(req);
    const userEmail = getUserEmail(req);

    if (!userId) {
      return res.status(401).json({
        error: 'Usuário não autenticado',
        code: 'USER_NOT_AUTHENTICATED'
      });
    }

    // Inicializar usuário se necessário
    await TokenService.initializeUser(userId, userEmail);

    // Estimar tokens necessários baseado na requisição
    const estimatedTokens = estimateTokensNeeded(req);

    // Verificar se pode usar tokens
    const tokenCheck = await TokenService.canUseTokens(userId,
estimatedTokens);

    if (!tokenCheck.canUse) {
      return res.status(403).json({
        error: 'Tokens insuficientes',
        code: 'INSUFFICIENT_TOKENS',
        details: {
          availableTokens: tokenCheck.availableTokens,
          tokensNeeded: tokenCheck.tokensNeeded,
```

```
            isBlocked: tokenCheck.isBlocked,
            blockReason: tokenCheck.blockReason
          }
        });
      }

      // Adicionar informações de token ao request para uso posterior
      req.tokenInfo = {
        userId,
        estimatedTokens,
        availableTokens: tokenCheck.availableTokens
      };

      next();
    } catch (error) {
      console.error('Erro no middleware de tokens:', error);

      // Em caso de erro, permitir requisição mas logar o problema
      console.error('Token middleware error - allowing request to proceed');
      next();
    }
};

/**
 * Middleware para registrar uso real de tokens após a resposta
 */
const tokenUsageMiddleware = (req, res, next) => {
  if (!tokenConfig.enabled || !req.tokenInfo) {
    return next();
  }

  // Interceptar o final da resposta para registrar uso real
  const originalSend = res.send;

  res.send = function(data) {
    // Restaurar método original
    res.send = originalSend;

    // Registrar uso de tokens de forma assíncrona
    setImmediate(async () => {
      try {
        await recordActualTokenUsage(req, res, data);
      } catch (error) {
        console.error('Erro ao registrar uso de tokens:', error);
      }
    });

    // Enviar resposta
```

```
    return originalSend.call(this, data);
  };

  next();
};

/**
 * Verifica se a rota deve ter tokens verificados
 */
function shouldCheckTokens(req) {
  const path = req.path;
  const method = req.method;

  // Rotas que consomem tokens (chat, completions, etc.)
  const tokenConsumingPaths = [
    '/api/ask',
    '/api/chat',
    '/api/completions',
    '/api/messages'
  ];

  // Rotas administrativas que não consomem tokens
  const adminPaths = [
    '/api/admin',
    '/api/token-admin',
    '/api/auth'
  ];

  // Verificar se é rota administrativa
  if (adminPaths.some(adminPath => path.startsWith(adminPath))) {
    return false;
  }

  // Verificar se é rota que consome tokens
  return tokenConsumingPaths.some(tokenPath => path.startsWith(tokenPath));
}

/**
 * Extrai ID do usuário da requisição
 */
function getUserId(req) {
  // Adaptar conforme sistema de autenticação do LibreChat
  return req.user?.id || req.user?._id || req.headers['x-user-id'];
}

/**
 * Extrai email do usuário da requisição
 */
```

```
function getUserEmail(req) {
  // Adaptar conforme sistema de autenticação do LibreChat
  return req.user?.email || req.headers['x-user-email'];
}

/**
 * Estima tokens necessários baseado na requisição
 */
function estimateTokensNeeded(req) {
  const body = req.body;

  // Estimativa básica baseada no conteúdo da mensagem
  let estimatedTokens = 1000; // Base mínima

  if (body.message) {
    // Aproximadamente 4 caracteres por token
    estimatedTokens += Math.ceil(body.message.length / 4);
  }

  if (body.messages && Array.isArray(body.messages)) {
    const totalChars = body.messages.reduce((sum, msg) => {
      return sum + (msg.content ? msg.content.length : 0);
    }, 0);
    estimatedTokens += Math.ceil(totalChars / 4);
  }

  // Adicionar margem de segurança
  return Math.ceil(estimatedTokens * 1.5);
}

/**
 * Registra uso real de tokens após a resposta
 */
async function recordActualTokenUsage(req, res, responseData) {
  try {
    const { userId, estimatedTokens } = req.tokenInfo;

    // Tentar extrair tokens reais da resposta
    let actualTokens = estimatedTokens;

    if (responseData && typeof responseData === 'string') {
      try {
        const parsed = JSON.parse(responseData);
        if (parsed.usage && parsed.usage.total_tokens) {
          actualTokens = parsed.usage.total_tokens;
        }
      } catch (e) {
        // Usar estimativa se não conseguir parsear
```

```
      }
    }

    // Registrar uso real
    await TokenService.useTokens(userId, actualTokens, {
      model: req.body.model || 'unknown',
      endpoint: req.path,
      conversationId: req.body.conversationId,
      messageId: req.body.messageId,
      userAgent: req.headers['user-agent'],
      ipAddress: req.ip || req.connection.remoteAddress
    });

  } catch (error) {
    console.error('Erro ao registrar uso real de tokens:', error);
  }
}

module.exports = {
  tokenControlMiddleware,
  tokenUsageMiddleware
};
```