Guia de Implementação Perplexity API - Versão Lean

📋 Checklist de Implementação

- [ ] Configurar API Key
- [ ] Criar PerplexityService
- [ ] Implementar Rate Limiter
- [ ] Adicionar Source Validator
- [ ] Integrar no fluxo existente
- [ ] Adicionar toggle na UI
- [ ] Testar end-to-end
- [ ] Deploy com feature flag

🔧 Passo 1: Configuração Inicial

1.1 Adicionar ao `.env`

bash

```bash
# Perplexity API Configuration
PERPLEXITY_API_KEY=your_api_key_here
PERPLEXITY_ENABLED=false  # Feature flag
```

1.2 Atualizar tipos de ambiente

typescript

```typescript
// server/env.d.ts
declare global {
  namespace NodeJS {
    interface ProcessEnv {
      PERPLEXITY_API_KEY?: string;
      PERPLEXITY_ENABLED?: string;
    }
  }
}
```

📦 Passo 2: Criar o Serviço Perplexity

2.1 Criar arquivo `server/services/perplexity.ts`

typescript

```typescript
import { SimpleRateLimiter } from './rate-limiter';

export interface PerplexitySearchOptions {
  domains?: string[];
  recency?: 'day' | 'week' | 'month';
  maxResults?: number;
}

export interface PerplexitySearchResult {
  content: string;
  sources: string[];
  usage?: {
    prompt_tokens: number;
    completion_tokens: number;
  };
}

export class PerplexityService {
  private rateLimiter: SimpleRateLimiter;
  private apiKey: string;
  private baseUrl = 'https://api.perplexity.ai/chat/completions';

  constructor() {
    this.apiKey = process.env.PERPLEXITY_API_KEY || '';
    this.rateLimiter = new SimpleRateLimiter();

    if (!this.apiKey) {
      console.warn('Perplexity API key not configured');
    }
  }

  isEnabled(): boolean {
    return Boolean(this.apiKey && process.env.PERPLEXITY_ENABLED === 'true');
  }

  async search(
    query: string,
    options: PerplexitySearchOptions = {}
  ): Promise<PerplexitySearchResult> {
    if (!this.isEnabled()) {
      throw new Error('Perplexity service is not enabled');
    }

    // Rate limiting
    await this.rateLimiter.throttle();
```

```
  try {
    const response = await fetch(this.baseUrl, {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${this.apiKey}`,
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        model: 'llama-3.1-sonar-small-128k-online',
        messages: [{
          role: 'user',
          content: this.buildSearchQuery(query, options)
        }],
        search_domain_filter: options.domains,
        search_recency_filter: options.recency,
        return_citations: true,
        max_tokens: 1000,
        temperature: 0.1,
      }),
    });

    if (!response.ok) {
      const error = await response.text();
      throw new Error(`Perplexity API error: ${response.status} - ${error}`);
    }

    const data = await response.json();

    return {
      content: data.choices[0]?.message?.content || '',
      sources: data.citations || [],
      usage: data.usage,
    };
  } catch (error) {
    console.error('Perplexity search failed:', error);
    throw error;
  }
}

private buildSearchQuery(query: string, options: PerplexitySearchOptions): string {
  // Manter prompts simples e diretos
  let searchQuery = query;

  if (options.recency) {
    searchQuery = `${query} (focus on information from the last ${options.recency})`
  }
```

```
    if (options.maxResults) {
      searchQuery += ` Limit to ${options.maxResults} most relevant results.`;
    }

    return searchQuery;
  }
}

export const perplexityService = new PerplexityService();
```

📦 Passo 3: Implementar Rate Limiter Simples

3.1 Criar arquivo `server/services/rate-limiter.ts`

typescript

```typescript
export class SimpleRateLimiter {
  private lastRequestTime = 0;
  private requestCount = 0;
  private resetTime = 0;

  // Configuração conservadora para começar
  private readonly MAX_REQUESTS_PER_MINUTE = 20;
  private readonly MIN_INTERVAL_MS = 3000; // 3 segundos entre requests

  async throttle(): Promise<void> {
    const now = Date.now();

    // Reset contador a cada minuto
    if (now - this.resetTime > 60000) {
      this.requestCount = 0;
      this.resetTime = now;
    }

    // Verificar limite por minuto
    if (this.requestCount >= this.MAX_REQUESTS_PER_MINUTE) {
      const waitTime = 60000 - (now - this.resetTime);
      console.log(`Rate limit reached. Waiting ${waitTime}ms`);
      await this.sleep(waitTime);
      this.requestCount = 0;
      this.resetTime = Date.now();
    }

    // Verificar intervalo mínimo
    const timeSinceLastRequest = now - this.lastRequestTime;
    if (timeSinceLastRequest < this.MIN_INTERVAL_MS) {
      const waitTime = this.MIN_INTERVAL_MS - timeSinceLastRequest;
      await this.sleep(waitTime);
    }

    this.lastRequestTime = Date.now();
    this.requestCount++;
  }

  private sleep(ms: number): Promise<void> {
    return new Promise(resolve => setTimeout(resolve, ms));
  }
}
```

📦 Passo 4: Implementar Validador de Fontes

4.1 Criar arquivo `server/services/source-validator.ts`

typescript

```typescript
export interface ValidationResult {
  url: string;
  isValid: boolean;
  isAccessible: boolean;
  reason?: string;
}

export class SourceValidator {
  private blacklist = new Set([
    'example.com',
    'test.com',
    'localhost',
    'placeholder.com',
    '127.0.0.1',
    'fake-site.com'
  ]);

  async validate(url: string): Promise<ValidationResult> {
    const result: ValidationResult = {
      url,
      isValid: false,
      isAccessible: false,
    };

    try {
      // 1. Validar formato da URL
      const urlObj = new URL(url);

      // 2. Verificar blacklist
      if (this.blacklist.has(urlObj.hostname)) {
        result.reason = 'Domain is blacklisted';
        return result;
      }

      // 3. Verificar protocolo
      if (!['http:', 'https:'].includes(urlObj.protocol)) {
        result.reason = 'Invalid protocol';
        return result;
      }

      // 4. Verificar acessibilidade (HEAD request com timeout)
      const controller = new AbortController();
      const timeoutId = setTimeout(() => controller.abort(), 5000);

      try {
        const response = await fetch(url, {
```

```
          method: 'HEAD',
          signal: controller.signal,
          headers: {
            'User-Agent': 'ExpertPlanner/1.0'
          }
        });

        clearTimeout(timeoutId);

        result.isAccessible = response.ok;
        result.isValid = response.ok;

        if (!response.ok) {
          result.reason = `HTTP ${response.status}`;
        }
      } catch (fetchError) {
        result.reason = 'Not accessible';
      }

      return result;

    } catch (error) {
      result.reason = 'Invalid URL format';
      return result;
    }
  }

  async validateBatch(urls: string[]): Promise<ValidationResult[]> {
    // Validar em paralelo mas com limite de concorrência
    const results: ValidationResult[] = [];
    const batchSize = 5;

    for (let i = 0; i < urls.length; i += batchSize) {
      const batch = urls.slice(i, i + batchSize);
      const batchResults = await Promise.all(
        batch.map(url => this.validate(url))
      );
      results.push(...batchResults);
    }

    return results;
  }
}

export const sourceValidator = new SourceValidator();
```

📦 Passo 5: Integrar no Fluxo Existente

5.1 Atualizar `server/anthropic.ts`

typescript

```typescript
// Adicionar imports no topo
import { perplexityService } from './services/perplexity';
import { sourceValidator } from './services/source-validator';

// Adicionar interface para opções
export interface ContentGenerationOptions {
  usePerplexity?: boolean;
  includeRealTimeSources?: boolean;
}

// Modificar a função generateContentIdeas
export async function generateContentIdeas(
  params: ContentIdeaGenerationParams,
  options: ContentGenerationOptions = {}
): Promise<ContentIdeaResult[]> {

  // Se Perplexity está habilitado e foi solicitado
  if (options.usePerplexity && perplexityService.isEnabled()) {
    try {
      return await generateWithPerplexity(params);
    } catch (error) {
      console.error('Perplexity generation failed, falling back to Anthropic:', error)
      // Fallback automático para Anthropic
    }
  }

  // Fluxo original com Anthropic
  return await generateWithAnthropic(params);
}

// Nova função para gerar com Perplexity
async function generateWithPerplexity(
  params: ContentIdeaGenerationParams
): Promise<ContentIdeaResult[]> {
  const { topic, platform, viewpoints, expertiseKeywords } = params;

  // Construir query focada
  const searchQuery = `
    Generate ${platform} content ideas about "${topic}".
    Consider these viewpoints: ${viewpoints.slice(0, 3).join(', ')}.
    Target audience interested in: ${expertiseKeywords.slice(0, 5).join(', ')}.
    Include 2-3 actionable ideas with current examples and sources.
  `.trim();

  // Buscar com Perplexity
  const searchResult = await perplexityService.search(searchQuery, {
```

```
    recency: 'month',
    maxResults: 5
  });

  // Validar sources
  const validationResults = await sourceValidator.validateBatch(searchResult.sources);
  const validSources = validationResults
    .filter(r => r.isValid)
    .map(r => r.url);

  // Parse do conteúdo retornado
  try {
    // Perplexity às vezes retorna JSON, às vezes texto
    const ideas = parsePerplexityResponse(searchResult.content, platform);

    // Adicionar sources válidas a cada ideia
    return ideas.map(idea => ({
      ...idea,
      sources: validSources.slice(0, 3), // Máximo 3 sources por ideia
      generatedWith: 'perplexity'
    }));
  } catch (parseError) {
    console.error('Failed to parse Perplexity response:', parseError);
    throw new Error('Invalid response format from Perplexity');
  }
}

// Helper para parse da resposta
function parsePerplexityResponse(
  content: string,
  platform: string
): ContentIdeaResult[] {
  // Tentar parse como JSON primeiro
  try {
    const parsed = JSON.parse(content);
    if (parsed.contentIdeas) {
      return parsed.contentIdeas;
    }
  } catch {
    // Se não for JSON, fazer parse manual
  }

  // Parse manual básico do texto
  const ideas: ContentIdeaResult[] = [];
  const sections = content.split(/\d+\./);

  sections.slice(1, 4).forEach((section, index) => {
```

```
    const lines = section.trim().split('\n');
    ideas.push({
      title: lines[0]?.trim() || `${platform} Idea ${index + 1}`,
      description: lines[1]?.trim() || section.substring(0, 100),
      format: platform === 'linkedin' ? 'post' : 'thread',
      keyPoints: lines.slice(2, 5).map(l => l.trim()).filter(Boolean),
      sources: [] // Será preenchido depois
    });
  });

  return ideas;
}

// Manter a função original como generateWithAnthropic
async function generateWithAnthropic(
  params: ContentIdeaGenerationParams
): Promise<ContentIdeaResult[]> {
  // Código original da generateContentIdeas
  // ... (manter como está)
}
```

📦 Passo 6: Atualizar a API Routes

6.1 Modificar `server/routes.ts`

typescript

```typescript
// No endpoint /api/generate-content-ideas
app.post('/api/generate-content-ideas', async (req: Request, res: Response) => {
  try {
    const { topicId, platform, expertId, usePerplexity } = req.body;

    // Validações existentes...

    // Adicionar opção de usar Perplexity
    const options: ContentGenerationOptions = {
      usePerplexity: usePerplexity === true,
      includeRealTimeSources: true
    };

    // Chamar generateContentIdeas com options
    const result = await contentPipeline.generateContentWithScraping({
      topicId,
      platform,
      expertId
    }, options);

    res.status(201).json({
      ideas: result.ideas,
      metadata: {
        ...result.metadata,
        generatedWith: usePerplexity ? 'perplexity' : 'anthropic'
      }
    });
  } catch (err: any) {
    handleError(err, res);
  }
});
```

📦 Passo 7: Atualizar a UI

7.1 Modificar `ContentGenerator.tsx`

```typescript
// Adicionar state para Perplexity toggle
const [usePerplexity, setUsePerplexity] = useState(false);

// Adicionar ao JSX antes do botão Generate
<div className="flex items-center space-x-2 mb-4">
  <Switch
    id="use-perplexity"
    checked={usePerplexity}
    onCheckedChange={setUsePerplexity}
    disabled={generatingIdeas} // Desabilitar durante geração
  />
  <Label htmlFor="use-perplexity" className="flex items-center cursor-pointer">
    <Zap className="w-4 h-4 mr-1 text-yellow-500" />
    Use real-time search (Beta)
  </Label>
</div>

// Modificar a chamada da API
const generateIdeas = async () => {
  try {
    setGeneratingIdeas(true);
    const response = await fetch('/api/generate-content-ideas', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        topicId: selectedTopic,
        platform: selectedPlatform,
        expertId: expert.id,
        usePerplexity: usePerplexity // Adicionar esta linha
      }),
    });
    // ... resto do código
  } catch (error) {
    // ... tratamento de erro
  }
};
```

7.2 Modificar `ContentIdeaCard.tsx` para mostrar sources

typescript

```typescript
// Adicionar ao ContentIdeaCard
{idea.sources && idea.sources.length > 0 && (
  <div className="mt-3 pt-3 border-t border-gray-100">
    <p className="text-xs font-medium text-gray-700 mb-1">Sources:</p>
    <div className="space-y-1">
      {idea.sources.map((source, idx) => (
        <a
          key={idx}
          href={source}
          target="_blank"
          rel="noopener noreferrer"
          className="text-xs text-blue-600 hover:underline block truncate"
        >
          <ExternalLink className="w-3 h-3 inline mr-1" />
          {new URL(source).hostname}
        </a>
      ))}
    </div>
  </div>
)}

// Adicionar badge se foi gerado com Perplexity
{idea.generatedWith === 'perplexity' && (
  <Badge className="absolute top-2 right-2 bg-yellow-100 text-yellow-800">
    <Zap className="w-3 h-3 mr-1" />
    Live
  </Badge>
)}
```

🧪 Passo 8: Testes

8.1 Criar `tests/perplexity.test.ts`

typescript

```typescript
import { PerplexityService } from '../server/services/perplexity';
import { SimpleRateLimiter } from '../server/services/rate-limiter';
import { SourceValidator } from '../server/services/source-validator';

describe('Perplexity Integration', () => {
  describe('PerplexityService', () => {
    test('should respect rate limits', async () => {
      const service = new PerplexityService();
      const start = Date.now();

      // Fazer 2 requests seguidos
      await service.search('test query 1');
      await service.search('test query 2');

      const elapsed = Date.now() - start;
      expect(elapsed).toBeGreaterThanOrEqual(3000); // Mínimo 3s entre requests
    });

    test('should handle API errors gracefully', async () => {
      const service = new PerplexityService();
      // Mock de API key inválida
      process.env.PERPLEXITY_API_KEY = 'invalid_key';

      await expect(service.search('test')).rejects.toThrow();
    });
  });

  describe('SourceValidator', () => {
    test('should reject blacklisted domains', async () => {
      const validator = new SourceValidator();
      const result = await validator.validate('https://example.com/article');

      expect(result.isValid).toBe(false);
      expect(result.reason).toBe('Domain is blacklisted');
    });

    test('should validate real URLs', async () => {
      const validator = new SourceValidator();
      const result = await validator.validate('https://www.google.com');

      expect(result.isValid).toBe(true);
      expect(result.isAccessible).toBe(true);
    });
  });
});
```

🚀 Passo 9: Deployment

9.1 Checklist de Deploy

bash

```bash
# 1. Adicionar API key ao ambiente de produção
# No Replit Secrets ou .env de produção
PERPLEXITY_API_KEY=pk-xxxxx
PERPLEXITY_ENABLED=false  # Começar desabilitado

# 2. Deploy do código
git add .
git commit -m "feat: add Perplexity API integration (disabled by default)"
git push

# 3. Testar em produção com flag desabilitado
# Verificar que nada quebrou

# 4. Habilitar para grupo de teste
PERPLEXITY_ENABLED=true  # Apenas para subset de usuários

# 5. Monitorar métricas
# - Taxa de erro
# - Tempo de resposta
# - Qualidade das sources
```

9.2 Feature Flag para usuários específicos

typescript

```typescript
// Adicionar ao ExpertProfile ou criar tabela feature_flags
interface FeatureFlags {
  expertId: number;
  perplexityEnabled: boolean;
  enabledAt: Date;
}

// No frontend, verificar flag do usuário
const userCanUsePerplexity = expert.featureFlags?.perplexityEnabled || false;

// Mostrar toggle apenas se usuário tem acesso
{userCanUsePerplexity && (
  <div className="flex items-center space-x-2 mb-4">
    <Switch ... />
  </div>
)}
```

📊 Passo 10: Monitoramento

10.1 Adicionar logs básicos

typescript

```typescript
// Em PerplexityService
async search(query: string, options: PerplexitySearchOptions = {}): Promise<Perplexity
  const startTime = Date.now();

  try {
    // ... código de search ...

    // Log de sucesso
    console.log('Perplexity search completed', {
      duration: Date.now() - startTime,
      tokensUsed: result.usage?.total_tokens,
      sourcesFound: result.sources.length
    });

    return result;
  } catch (error) {
    // Log de erro
    console.error('Perplexity search failed', {
      duration: Date.now() - startTime,
      error: error.message,
      query: query.substring(0, 50) // Primeiros 50 chars
    });
    throw error;
  }
}
```

10.2 Métricas essenciais

typescript

```typescript
// Trackear no frontend
const trackPerplexityUsage = (event: string, data?: any) => {
  // Se tiver analytics (GA, Mixpanel, etc)
  if (window.analytics) {
    window.analytics.track(`perplexity_${event}`, {
      expertId: expert.id,
      ...data
    });
  }
};

// Uso:
trackPerplexityUsage('toggle_enabled', { enabled: true });
trackPerplexityUsage('generation_completed', {
  sourcesCount: ideas[0].sources.length,
  platform: selectedPlatform
});
```

✅ Critérios de Sucesso

1. **Funcional**
   - ☐ API key configurada e funcionando
   - ☐ Rate limiting previne 429 errors
   - ☐ Sources são validadas corretamente
   - ☐ Fallback para Anthropic funciona
2. **UX**
   - ☐ Toggle aparece e funciona
   - ☐ Sources são mostradas quando disponíveis
   - ☐ Sem aumento perceptível no tempo de geração
   - ☐ Mensagens de erro são claras
3. **Técnico**
   - ☐ Nenhum erro em produção
   - ☐ Logs capturados corretamente
   - ☐ Código testado com >80% coverage
   - ☐ Performance dentro do esperado (<5s)

🎯 Resultado Esperado

- Implementação funcional em **2-3 dias**
- Código total: **~300 linhas** (não 3000)
- Complexidade: **Baixa**

- Risco: **Mínimo** (feature flag + fallback)
- Valor: **A ser validado com usuários**

🚨 Troubleshooting

Erro 401 - Unauthorized

```bash
# Verificar API key
echo $PERPLEXITY_API_KEY
# Deve começar com "pplx-"
```

Erro 429 - Rate Limit

```typescript
// Aumentar intervalo no rate limiter
private readonly MIN_INTERVAL_MS = 5000; // De 3s para 5s
```

Sources não aparecem

```typescript
// Verificar response
console.log('Perplexity response:', data);
// citations pode vir em formato diferente
```

Timeout em validação

```typescript
// Reduzir timeout ou validar async
const timeoutId = setTimeout(() => controller.abort(), 3000); // De 5s para 3s
```

🎯 Lembre-se: KISS - Keep It Simple, Stupid!

Essa implementação lean entrega 90% do valor com 10% da complexidade. Podemos sempre adicionar mais features depois, baseado em feedback real dos usuários.