



# Como Funciona o Sistema de Recomendações com IA



## Visão Geral

O sistema de recomendações com IA usa o **Google Gemini 2.5 Flash** para gerar recomendações personalizadas baseadas no perfil e histórico de leitura do usuário.



## Fluxo Completo (Passo a Passo)

1. USUÁRIO CLICA NO BOTÃO "IA PREMIUM"



2. NAVEGADOR FAZ REQUEST PARA API  
GET /recommendations/api/recommendations/?algorithm=ai



3. BACKEND - views\_simple.py

- a) Verificar se usuário está autenticado
- b) Buscar histórico do usuário no banco:
  - Últimas 20 interações (UserBookInteraction)
  - Livros lidos, avaliados, na wishlist, etc.
- c) Preparar dados do histórico:

```
[{
  "title": "Os Filhos de Húrin",
  "author": "J.R.R. Tolkien",
  "categories": "Fantasia",
  "interaction_type": "read"
}, ...]
```



4. CHAMAR GeminiRecommendationEngine  
engine.generate\_recommendations(user, history\_data, n=6)



5. VERIFICAR CACHE (gemini\_ai.py)

Cache Key: "gemini\_rec:{user\_id}:6"

Timeout: 1 hora (3600 segundos)

- ☒ Se existe no cache → Retornar resultado imediato
- ☒ Se não existe → Continuar para API do Gemini



## 6. CONSTRUIR PROMPT PARA GEMINI (\_build\_prompt)

Prompt contém:

Você é um especialista em recomendação de livros.

PERFIL DO USUÁRIO:

- Nome: claud
- Livros lidos: Os Filhos de Húrin, Harry Potter...
- Gêneros favoritos: Fantasia, Ficção Científica
- Total de livros lidos: 8

INSTRUÇÕES:

1. Recomende 6 livros DIFERENTES
2. Para cada livro forneça:
  - Título completo
  - Autor
  - Justificativa personalizada

FORMATO: JSON

```
{
  "recommendations": [
    {
      "title": "Duna",
      "author": "Frank Herbert",
      "reason": "Como você gostou de... é perfeito"
    }
  ]
}
```



## 7. CHAMAR API DO GOOGLE GEMINI

Model: gemini-2.5-flash

API Key: AIzaSy...Db\_A (configurada em settings)

Request: model.generate\_content(prompt)

🕒 Tempo de resposta: ~3-5 segundos



#### 8. GEMINI RETORNA RESPOSTA JSON

```
{
  "recommendations": [
    {
      "title": "O Hobbit",
      "author": "J.R.R. Tolkien",
      "reason": "Claud já mostrou apreço pela riqueza do
                mundo de Tolkien com 'Os Filhos de Húrin'...
                porta de entrada perfeita para Terra Média"
    },
    {
      "title": "Duna",
      "author": "Frank Herbert",
      "reason": "Ficção científica épica com worldbuilding
                tão rico quanto Tolkien..."
    },
    ... (mais 4 livros)
  ]
}
```



#### 9. PARSE E VALIDAÇÃO (\_parse\_recommendations)

- Remove markdown (``json)
- Parse JSON
- Adiciona score padrão: 0.95 (95%)
- Retorna lista de dicts



#### 10. SALVAR NO CACHE

```
cache.set("gemini_rec:123:6", recommendations, timeout=3600)
```

☒ Próxima chamada na próxima hora será instantânea!



#### 11. VOLTAR PARA views\_simple.py

Agora temos 6 recomendações da IA:

```
[
  {"title": "O Hobbit", "author": "Tolkien", ...},
  {"title": "Duna", "author": "Herbert", ...},
  ...
]
```



## 12. BUSCAR LIVROS REAIS NO BANCO DE DADOS ⚠ CRÍTICO!

PROBLEMA: Gemini pode recomendar livros que NÃO EXISTEM no nosso catálogo!

SOLUÇÃO: Para cada recomendação da IA:

```
for rec in ai_recommendations:
    # Limpar título (remover subtítulos)
    book_title = rec['title'].split('(')[0].strip()

    # Buscar no banco (ICONTAINS = busca parcial)
    book = Book.objects.filter(
        title__icontains=book_title
    ).first()

    if book: # ☑ Livro EXISTE
        books_data.append({
            'id': book.id,
            'slug': book.slug,
            'title': book.title, # Título real do banco
            'author': str(book.author),
            'cover_image': book.cover_image.url, # ★ URL real
            'score': rec['score'], # 0.95
            'reason': rec['reason'] # Justificativa da IA
        })
    else: # ✗ Livro NÃO EXISTE → ignorar
        continue
```

RESULTADO:

- "O Hobbit" → ☑ Encontrado → Adiciona
- "Duna" → ✗ Não existe → Ignora
- "O Nome do Vento" → ☑ Encontrado → Adiciona
- ...

Final: 2 livros reais (de 6 recomendados)



## 13. FALLBACK SE NENHUM LIVRO ENCONTRADO

```
if not books_data: # Se nenhum livro da IA existe
    # Usar algoritmo HÍBRIDO como backup
    engine = HybridRecommendationSystem()
    recommendations = engine.recommend(user, n=6)

    # Adicionar nota na razão
    reason = "IA recomendou livros similares | " + ...
```



## 14. RETORNAR RESPOSTA JSON

```
return JsonResponse({
  'algorithm': 'ai',
  'count': 2,
  'recommendations': [
    {
      'id': 2,
      'slug': 'o-hobbit-edicao-ilustrada',
      'title': 'O Hobbit em quadrinhos',
      'author': 'J.R.R. Tolkien',
      'cover_image': 'https://supabase.co/.../hobbit',
      'score': 0.95,
      'reason': 'Claud já mostrou apreço...'
    },
    {
      'id': 45,
      'slug': 'o-nome-do-vento',
      'title': 'O Nome do Vento',
      'author': 'Patrick Rothfuss',
      'cover_image': 'https://supabase.co/.../vento',
      'score': 0.95,
      'reason': 'Fantasia moderna com prosa lírica...'
    }
  ]
})
```



## 15. NAVEGADOR RECEBE E RENDERIZA

JavaScript (recommendations\_section.html):

```
fetch('/recommendations/api/recommendations/?algorithm=ai')
  .then(response => response.json())
  .then(data => {
    renderRecommendations(data.recommendations)
  })
```

renderRecommendations() cria HTML:

```
<div class="card">
  <span class="score">95%</span>
  
  <h6>O Hobbit em quadrinhos</h6>
  <p>J.R.R. Tolkien</p>
  <p class="reason">Claud já mostrou apreço...</p>
  <a href="/livros/o-hobbit-edicao-ilustrada/">
    Ver livro
  </a>
</div>
```

16. USUÁRIO VÊ AS RECOMENDAÇÕES NA TELA! 🎉

## 🔑 Componentes Principais

### 1. **gemini\_ai.py** - Motor da IA

- **Classe:** `GeminiRecommendationEngine`
- **Modelo:** `gemini-2.5-flash` (Google)
- **Métodos principais:**
  - `generate_recommendations()` - Gera lista de livros
  - `_build_prompt()` - Cria prompt personalizado
  - `_parse_recommendations()` - Parse da resposta JSON

### 2. **views\_simple.py** - API Backend

- **Endpoint:** `/recommendations/api/recommendations/?algorithm=ai`
- **Função:** `get_recommendations_simple()`
- **Responsabilidades:**
  1. Buscar histórico do usuário
  2. Chamar Gemini AI
  3. Validar livros no banco
  4. Retornar JSON com livros reais

### 3. **recommendations\_section.html** - Frontend

- **JavaScript:** Faz fetch da API
- **Renderização:** Cria cards de livros dinamicamente
- **Fallback:** Mostra placeholder se imagem falhar

## 💾 Cache e Performance

Cache da IA (1 hora)

```
cache_key = f'gemini_rec:{user.id}:{n}'
cache.set(cache_key, recommendations, timeout=3600)
```

### Benefícios:

- ⚡ Primeira chamada: ~3-5 segundos (API Gemini)
- ⚡ Chamadas subsequentes: ~100ms (cache)
- 💰 Economia de custos da API
- 📄 Redução de requisições ao Gemini

## Diferença entre Algoritmos

Algoritmo	Como funciona	Dados usados	Velocidade
Híbrido	60% collaborative + 30% content + 10% trending	Interações de usuários + metadados dos livros	⚡ ⚡ ⚡ Rápido (~100ms)
IA Premium	Gemini analisa perfil e gera recomendações personalizadas	Histórico completo + análise de contexto	⚡ Moderado (~3s primeira vez, cache depois)
Similares	Collaborative filtering (usuários com gostos parecidos)	Interações de todos os usuários	⚡ ⚡ ⚡ Rápido (~100ms)
Conteúdo	Content-based (metadados dos livros)	Categorias, autores, descrições	⚡ ⚡ ⚡ Rápido (~100ms)

## Exemplo Real - Usuário "claud"

Entrada (Histórico do Usuário)

```
user_history = [  
  {  
    'title': 'Os Filhos de Húrin',  
    'author': 'J.R.R. Tolkien',  
    'categories': 'Fantasia',  
    'interaction_type': 'read'  
  },  
  {  
    'title': 'Harry Potter e o Prisioneiro de Azkaban',  
    'author': 'J.K. Rowling',  
    'categories': 'Fantasia',  
    'interaction_type': 'click'  
  },  
  {  
    'title': 'Jujutsu Kaisen, Vol. 1',  
    'author': 'Gege Akutami',  
    'categories': 'Mangá',  
    'interaction_type': 'review',  
    'rating': 4  
  }  
]
```

### Processamento Gemini

```
Prompt → "Usuário leu Tolkien, Harry Potter, Mangá..."  
↓  
Gemini analisa padrões:
```

```
- Gosta de fantasia épica
- Aprecia worldbuilding complexo
- Interesse em narrativas heroicas
  ↓
Recomenda: O Hobbit, Duna, A Odisseia, etc.
```

## Saída (Resposta Final)

```
{
  "algorithm": "ai",
  "count": 2,
  "recommendations": [
    {
      "title": "O Hobbit em quadrinhos",
      "author": "J.R.R. Tolkien",
      "slug": "o-hobbit-edicao-ilustrada",
      "cover_image": "https://supabase.co/.../hobbit.jpg",
      "score": 0.95,
      "reason": "Porta de entrada perfeita para Terra Média..."
    },
    {
      "title": "O Nome do Vento",
      "author": "Patrick Rothfuss",
      "slug": "o-nome-do-vento",
      "cover_image": "https://supabase.co/.../vento.jpg",
      "score": 0.95,
      "reason": "Fantasia moderna com prosa lírica..."
    }
  ]
}
```

---

## ⚠ Tratamento de Erros

### 1. Gemini API indisponível

```
if not engine.is_available():
    return JsonResponse({
        'error': 'Gemini AI não configurado',
        'status': 503
    })
```

### 2. Livros não existem no banco

```
if not books_data:
    # Fallback para algoritmo híbrido
```



```
engine = HybridRecommendationSystem()
recommendations = engine.recommend(user, n=limit)
```

### 3. Timeout da API

```
try:
    response = model.generate_content(prompt)
except Exception as e:
    logger.error(f"Gemini error: {e}")
    return [] # Retorna lista vazia
```

---

## Vantagens do Sistema IA

### 1. Justificativas Personalizadas

- Cada recomendação vem com explicação única
- Contexto baseado no perfil real do usuário

### 2. Análise de Contexto

- Entende padrões complexos (não só categorias)
- Considera evolução do gosto do usuário

### 3. Descoberta de Novos Livros

- Recomenda livros fora do padrão
- Expande horizontes de leitura

### 4. Cache Inteligente

- Performance similar aos outros algoritmos após cache
- Atualização automática a cada hora

---

## Arquivos Principais

- **Backend IA:** `recommendations/gemini_ai.py` (linhas 18-297)
- **API Endpoint:** `recommendations/views_simple.py` (linhas 53-134)
- **Frontend:** `templates/recommendations/recommendations_section.html` (linhas 113-228)
- **Configuração:** `cgbookstore/settings.py` (GEMINI\_API\_KEY)

---

## Resumo Técnico

### O sistema IA funciona assim:

1. ☒ Usuário clica "IA Premium"
2. ☒ Backend busca histórico de leitura (últimas 20 interações)

3. ☒ Monta prompt personalizado com perfil do usuário
4. ☒ Gemini 2.5 Flash analisa e retorna 6 livros (JSON)
5. ☒ Backend valida quais livros existem no catálogo
6. ☒ Retorna apenas livros reais com imagens válidas
7. ☒ Se nenhum existir, usa algoritmo híbrido como fallback
8. ☒ Cacheia resultado por 1 hora (performance)
9. ☒ Frontend renderiza cards com justificativas da IA

**Resultado:** Recomendações personalizadas com explicações detalhadas! 🐛