Teste Técnico Backend + Frontend - Plataforma de Comunidades

🎯 Objetivo

Desenvolver uma plataforma de gestão de comunidades com API REST, interface web e análise de sentimento básico usando IA.

🛠️ Stack Tecnológica

Você pode escolher:

Backend:

- **Ruby on Rails ⭐ (Diferencial - preferencial)**
- **Node.js (Express, NestJS, Fastify)**
- **Python (Django, FastAPI, Flask)**
- **PHP (Laravel, Symfony)**
- **Java (Spring Boot)**
- **Go (Gin, Echo)**
- **Outra de sua preferência**

Frontend:

- **Se escolher Rails: HAML + Stimulus + Turbo/Hotwire ⭐ (Diferencial)**
- **React / Vue / Angular / Svelte**
- **Server-side rendering (Next.js, Nuxt, etc)**
- **Templates nativos da linguagem escolhida**

Banco de Dados:

- **PostgreSQL (preferencial)**
- **MySQL / MariaDB**
- **MongoDB (se justificar a escolha)**

Requisitos Obrigatórios (qualquer stack):

- **✅ Testes automatizados (cobertura mínima 70%)**
- **✅ Linter/formatter configurado**
- **✅ Código no GitHub (repositório público)**
- **✅ Deploy funcionando**

- ✅ **README completo**

PARTE 1: API REST (Obrigatório - 60%)

Modelo de Dados

```
Users
- id (PK)
- username (string, unique, not null)
- created_at

Communities
- id (PK)
- name (string, unique, not null)
- description (text)
- created_at

Messages
- id (PK)
- user_id (FK, not null)
- community_id (FK, not null)
- parent_message_id (FK, nullable) // para comentários/respostas
- content (text, not null)
- user_ip (string, not null)
- ai_sentiment_score (float, nullable) // -1.0 a 1.0
- created_at

Reactions
- id (PK)
- message_id (FK, not null)
- user_id (FK, not null)
- reaction_type (string, not null) // 'like', 'love', 'insightful'
- created_at
- UNIQUE constraint em [message_id, user_id, reaction_type]
```

Endpoints Obrigatórios

1. POST /api/v1/messages

Criar nova mensagem ou comentário

Request:

json

```
{
  "username": "john_doe",
  "community_id": 1,
```

```
  "content": "Conteúdo da mensagem",
  "user_ip": "192.168.1.1",
  "parent_message_id": null  // opcional, apenas para comentários
}
```

Regras:

- Se usuário não existir, deve ser criado
- Calcular `ai_sentiment_score` automaticamente
- Validar campos obrigatórios

Response (201):

json

```
{
  "id": 1,
  "content": "Conteúdo da mensagem",
  "user": {
    "id": 1,
    "username": "john_doe"
  },
  "community_id": 1,
  "parent_message_id": null,
  "ai_sentiment_score": 0.75,
  "created_at": "2025-11-24T10:00:00Z"
}
```

2. POST /api/v1/reactions

Reagir a uma mensagem

Request:

json

```
{
  "message_id": 1,
  "user_id": 1,
  "reaction_type": "like"
}
```

Regras:

- Um usuário só pode adicionar UMA reação de cada tipo por mensagem
- **IMPORTANTE:** Deve lidar corretamente com requisições concorrentes
    - Use transactions, locks ou constraints do banco
    - Retornar erro apropriado se tentar duplicar

Response (200):

json

```
{
  "message_id": 1,
  "reactions": {
    "like": 15,
    "love": 8,
    "insightful": 3
  }
}
```

3. GET /api/v1/communities/:id/messages/top

Top N mensagens por engajamento

Query params:

- `limit` (default: 10, max: 50)

Critério de ranking:

- Engajamento = (número de reações * 1.5) + (número de respostas * 1.0)

Response (200):

json

```
{
  "messages": [
    {
      "id": 1,
      "content": "Conteúdo...",
      "user": {
        "id": 1,
        "username": "john_doe"
      },
      "ai_sentiment_score": 0.8,
      "reaction_count": 23,
      "reply_count": 5,
      "engagement_score": 39.5
    }
  ]
}
```

Performance esperada:

- Consulta otimizada (sem N+1)
- Usar índices apropriados

4. GET /api/v1/analytics/suspicious_ips

Detectar IPs usados por múltiplos usuários (possível fraude)

Query params:

- `min_users` (default: 3) - mínimo de usuários diferentes

Response (200):

json

```
{
  "suspicious_ips": [
    {
      "ip": "192.168.1.1",
      "user_count": 5,
      "usernames": ["user1", "user2", "user3", "user4", "user5"]
    }
  ]
}
```

PARTE 2: Interface Web (Obrigatório - 40%)

Páginas Obrigatórias

1. Home - Listagem de Comunidades

URL: `/` ou `/communities`

Deve mostrar:

- Lista/grid de todas as comunidades
- Para cada comunidade:
  - Nome
  - Descrição
  - Número de mensagens
  - Link para acessar

2. Timeline da Comunidade

URL: `/communities/:id`

Deve mostrar:

- Nome e descrição da comunidade
- Últimas 50 mensagens (ordenadas por data, mais recentes primeiro)

- Para cada mensagem:
    - Username do autor
    - Conteúdo
    - Indicador visual do sentiment score (emoji, cor, badge)
    - Botões de reação (like, love, insightful) com contador
    - Número de comentários/respostas
    - Data/hora
- Formulário para criar nova mensagem
    - Campo: username (text input)
    - Campo: conteúdo (textarea)
    - Botão: enviar

Comportamento esperado:

- Ao criar mensagem, adicionar na timeline **SEM reload da página**
- Mostrar o sentiment score calculado após criar
- Feedback visual de loading/sucesso/erro

3. Detalhes da Mensagem + Thread de Comentários

URL: `/messages/:id`

Deve mostrar:

- Mensagem principal (mesmas informações da timeline)
- Lista de comentários/respostas abaixo
- Formulário para adicionar comentário
- Indicação visual de hierarquia (comentários indentados ou com linha)

Requisitos Técnicos Frontend

Funcionalidades JavaScript obrigatórias:

- ✅ Botões de reação devem funcionar sem reload
    - Atualizar contador visualmente
    - Feedback visual quando já reagiu
    - Chamar API em background
- ✅ Criar mensagem sem reload da página
    - Adicionar mensagem na timeline dinamicamente
    - Limpar formulário após sucesso

Responsividade:

- ✅ Layout funcional em mobile e desktop
- ✅ Não precisa ser perfeito, mas deve ser usável

CSS/Styling:

- Use o que preferir: Tailwind, Bootstrap, CSS puro, styled-components

- Foco em funcionalidade, não em beleza

PARTE 3: Seeds & Deploy

Seeds Script

Criar script que popula o banco via **chamadas HTTP** aos endpoints da API:

Dados a gerar:

- 3-5 comunidades
- 50 usuários únicos
- 1000 mensagens:
  - 70% são posts principais
  - 30% são comentários/respostas
- 20 IPs únicos diferentes
- 80% das mensagens têm pelo menos uma reação

Tecnologias sugeridas:

- Ruby: usar `net/http`, `httparty` ou `faraday`
- Node: usar `axios` ou `fetch`
- Python: usar `requests`
- Bash: usar `curl`

Deploy

Plataformas gratuitas sugeridas:

- Render.com
- Railway.app
- Fly.io
- Heroku (se ainda tiver free tier)
- Vercel/Netlify (se separar frontend)

Requisitos:

- ✅ Aplicação acessível via URL pública
- ✅ Seeds executados (dados visíveis)
- ✅ Banco de dados funcionando

Análise de Sentimento (IA)

Opção 1: Simulação Simples (Aceita)

Implementar um analisador baseado em palavras-chave:

python
```python
# Exemplo em Python
POSITIVE_WORDS = ['ótimo', 'excelente', 'legal', 'bom', 'adorei', 'incrível']
NEGATIVE_WORDS = ['ruim', 'péssimo', 'horrível', 'terrível', 'odeio']

def analyze_sentiment(text):
    text_lower = text.lower()
    positive = sum(1 for word in POSITIVE_WORDS if word in text_lower)
    negative = sum(1 for word in NEGATIVE_WORDS if word in text_lower)

    total = positive + negative
    if total == 0:
        return 0.0


    return round((positive - negative) / total, 2)
```

Opção 2: Integração Real (Diferencial ⭐)

Integrar com serviço de NLP/IA:

- OpenAI API
- Hugging Face
- Google Cloud Natural Language
- AWS Comprehend
- Biblioteca local (NLTK, spaCy, etc)

Se usar API paga: pode usar modo mock/fake em produção e deixar documentado

---

📋 Checklist de Entrega

Copie e complete no seu README:

markdown
```markdown
## ✅ Checklist de Entrega - [SEU NOME]

### Repositório & Código
- [x ] Código no GitHub (público): [URL DO REPO]
- [ ] README com instruções completas
- [ ] `.env.example` ou similar com variáveis de ambiente
- [ ] Linter/formatter configurado
- [ ] Código limpo e organizado
```

Stack Utilizada
- [ ] Backend: [linguagem/framework]
- [ ] Frontend: [tecnologia]
- [ ] Banco de dados: [qual]
- [ ] Testes: [framework usado]

Deploy
- [ ] URL da aplicação: [URL]
- [ ] Seeds executados (dados de exemplo visíveis)

Funcionalidades - API
- [ ] POST /api/v1/messages (criar mensagem + sentiment)
- [ ] POST /api/v1/reactions (com proteção de concorrência)
- [ ] GET /api/v1/communities/:id/messages/top
- [ ] GET /api/v1/analytics/suspicious_ips
- [ ] Tratamento de erros apropriado
- [ ] Validações implementadas

Funcionalidades - Frontend
- [ ] Listagem de comunidades
- [ ] Timeline de mensagens
- [ ] Criar mensagem (sem reload)
- [ ] Reagir a mensagens (sem reload)
- [ ] Ver thread de comentários
- [ ] Responsivo (mobile + desktop)

Testes
- [ ] Cobertura mínima de 70%
- [ ] Testes passando
- [ ] Como rodar: [comando]

Documentação
- [ ] Setup local documentado
- [ ] Decisões técnicas explicadas
- [ ] Como rodar seeds
- [ ] Endpoints da API documentados
- [ ] Screenshot ou GIF da interface (opcional)

⏰ Entregue em: 01/12/2025

🎯 Critérios de Avaliação

Critério	Peso	Detalhes

API REST	35%	Endpoints funcionando, validações, performance, concorrência
Frontend	25%	Funcionalidade, UX, interatividade sem reload
Testes	20%	Cobertura, qualidade, casos de teste relevantes
Qualidade do Código	10%	Organização, patterns, clean code
Deploy & Docs	10%	Funcionando, bem documentado

Nota de corte: 70%

⭐ Diferenciadores (Bônus - não obrigatório)

Quer se destacar? Implemente 1-2 destes:

- **Ruby on Rails stack completo** (HAML + Stimulus + Turbo)
- Real-time updates (WebSockets/Action Cable/Socket.io)
- Filtros de mensagens (por sentiment, por data)
- Dashboard de moderação
- Paginação infinita (infinite scroll)
- Integração real com API de IA
- Testes end-to-end (Cypress, Playwright, Capybara)
- CI/CD configurado (GitHub Actions)
- Docker / Docker Compose
- UI polida com animações

💡 Dicas

1. **Comece pelo backend** - garanta a API funcionando primeiro
2. **Use bibliotecas consolidadas** - não reinvente a roda
3. **Priorize funcionalidade sobre beleza** - frontend bonito é bônus
4. **Teste enquanto desenvolve** - não deixe testes para o final
5. **Documente decisões técnicas** - explique seus trade-offs
6. **Deploy cedo** - não deixe para última hora
7. **Gerencie seu tempo:**
    - Dias 1-2: API básica
    - Dias 3-4: Frontend + testes
    - Dia 5: Seeds + deploy
    - Dias 6-7: Buffer e polimento

❓ Dúvidas?

Se tiver dúvidas sobre requisitos, entre em contato.

Boa sorte! 🚀