Avaliação Técnica

Engenharia de Software com IA Generativa

Resumo da avaliação

Esta avaliação técnica visa avaliar suas habilidades em:

1. Design e implementação de software seguindo princípios **SOLID**
2. Integração de IA generativa em aplicações práticas
3. Engenharia de prompts e otimização
4. Considerações éticas no uso de IA

Problema: Sistema de Recomendação com IA Generativa

Contexto

Você está desenvolvendo um componente de recomendação para uma plataforma de e-commerce que utiliza tanto análise de dados tradicional quanto IA generativa para melhorar a experiência do usuário.

Requisitos Funcionais

1. **Sistema de Recomendação Básico**
   - Implementar um algoritmo simples de recomendação baseado no histórico de compras
   - Desenvolver uma API para servir as recomendações
2. **Integração com IA Generativa**
   - Criar um componente que utiliza um LLM para gerar descrições personalizadas para as recomendações
   - Implementar engenharia de prompts para obter resultados relevantes e concisos
3. **Interface de Usuário**
   - Desenvolver uma interface simples somente para exibir as recomendações e descrições geradas

Requisitos Técnicos

1. **Arquitetura**
   - Aplicar princípios **SOLID** no design do código
   - Implementar pelo menos um padrão de design relevante
   - Documentar a arquitetura e justificar suas escolhas
2. **Backend**
   - Criar uma API RESTful para servir recomendações
   - Implementar tratamento adequado de erros e logging
3. **Integração com LLMs**
   - Implementar uma camada de abstração para interação com LLMs

- Desenvolver estratégias de fallback para quando o LLM não responder adequadamente
- Otimizar prompts para reduzir custos e melhorar a qualidade das respostas

4. **Considerações Éticas**
   - Implementar verificações básicas para garantir que o conteúdo gerado seja apropriado
   - Documentar potenciais problemas éticos e como você os abordaria em um sistema real

Tarefas Específicas

Parte 1: Design e Implementação

1. **Modelagem de Domínio**
   - Criar um modelo de domínio para o sistema de recomendação
   - Identificar entidades principais, serviços e repositórios
   - Implementar o código seguindo princípios **SOLID**
2. **Algoritmo de Recomendação**
   - Implementar um algoritmo de recomendação baseado em similaridade
   - Desenvolver uma interface que permita trocar facilmente o algoritmo (padrão Strategy)
3. **API RESTful**
   - Criar endpoints para obter recomendações para um usuário
   - Implementar paginação, filtragem e tratamento de erros
   - Documentar a API usando Swagger/OpenAPI

Parte 2: Integração com IA Generativa

1. **Serviço de IA Generativa**
   - Implementar um serviço que se comunica com um LLM (OpenAI GPT, Claude, etc.), Se preferir pode usar os simuladores aqui.
   - Desenvolver uma abstração que permita trocar facilmente de provedor
   - Implementar cache para reduzir chamadas à API
2. **Engenharia de Prompts**
   - Criar templates de prompts para gerar descrições personalizadas
   - Desenvolver uma estratégia para lidar com respostas inadequadas
3. **Interface com o Usuário**
   - Criar uma interface web simples para exibir recomendações e descrições

Estrutura do Projeto

Este repositório contém:

```
/
├── data/                     # Dados de exemplo
│   ├── users.json            # Dados de usuários com histórico de compras
│   ├── products.json         # Catálogo de produtos
│   └── recommendations.json  # Exemplos de recomendações
│
├── mock/                     # APIs simuladas
│   ├── llm_api_mock.js       # Simulador de API de LLM (JavaScript)
│   ├── llm_api_mock.py       # Simulador de API de LLM (Python)
│   └── LLMApiMock.java       # Simulador de API de LLM (Java)
│
└── README.md                 # Este arquivo
```

Configuração

1. Clone este repositório
2. Instale as dependências necessárias (você pode escolher as bibliotecas que preferir)
3. Para iniciar o desenvolvimento, você pode usar um dos simuladores de LLM fornecidos em JavaScript, Python ou Java, conforme sua preferência

Para JavaScript:

```
npm install   # ou yarn install
npm start     # ou yarn start
```

Para Python:

```
pip install asyncio   # Se necessário
python -m mock.llm_api_mock
```

Para Java:

```
javac -cp "lib/*" mock/LLMApiMock.java
java -cp ".:lib/*" mock.LLMApiMock
```

Usando o Simulador de API LLM

Para facilitar o desenvolvimento sem depender de uma conexão real com uma API de LLM, este projeto tem simuladores em JavaScript,Python e Java. Eles permitem testar a integração com LLMs sem consumir tokens de APIs reais.

Exemplo em JavaScript:

```
const LLMApiMock = require('./mock/llm_api_mock');
const llmApi = new LLMApiMock();

async function testLLM() {
  try {
```

```
    // Descrição genérica de produto
    const genericDesc = await llmApi.generateResponse({
      type: 'generic_description',
      product_id: 'p1005'
    });
    console.log('Descrição genérica:', genericDesc);

    // Descrição personalizada
    const personalizedDesc = await llmApi.generateResponse({
      type: 'personalized_description',
      user_id: 'u1001',
      product_id: 'p1005'
    });
    console.log('Descrição personalizada:', personalizedDesc);
  } catch (error) {
    console.error('Erro:', error.message);
  }
}

testLLM();
```

Exemplo em Python:

```
import asyncio
from mock.llm_api_mock import LLMApiMock

async def test_llm():
    llm_api = LLMApiMock()

    try:
        # Descrição genérica de produto
        generic_desc = await llm_api.generate_response({
            "type": "generic_description",
            "product_id": "p1005"
        })
        print("Descrição genérica:", generic_desc)

        # Descrição personalizada
        personalized_desc = await llm_api.generate_response({
            "type": "personalized_description",
            "user_id": "u1001",
            "product_id": "p1005"
        })
        print("Descrição personalizada:", personalized_desc)
    except Exception as e:
        print("Erro:", str(e))

# Executa o exemplo
asyncio.run(test_llm())
```

Exemplo em Java:

```java
import java.util.HashMap;
import java.util.Map;
import mock.LLMApiMock;

public class TestLLM {
    public static void main(String[] args) {
        LLMApiMock llmApi = new LLMApiMock();

        // Descrição genérica de produto
        Map<String, Object> params1 = new HashMap<>();
        params1.put("type", "generic_description");
        params1.put("product_id", "p1005");

        llmApi.generateResponse(params1).thenAccept(response -> {
            System.out.println("Descrição genérica: " + response);
        }).exceptionally(ex -> {
            System.err.println("Erro: " + ex.getMessage());
            return null;
        });

        // Descrição personalizada
        Map<String, Object> params2 = new HashMap<>();
        params2.put("type", "personalized_description");
        params2.put("user_id", "u1001");
        params2.put("product_id", "p1005");

        llmApi.generateResponse(params2).thenAccept(response -> {
            System.out.println("Descrição personalizada: " + response);
        }).exceptionally(ex -> {
            System.err.println("Erro: " + ex.getMessage());
            return null;
        });
    }
}
```

Os simuladores suportam diferentes tipos de respostas:

- `generic_description`: Descrição padrão de um produto
- `personalized_description`: Descrição adaptada ao perfil do usuário
- `recommendation_explanation`: Explicação de por que um produto foi recomendado
- `product_comparison`: Comparação entre produtos
- Prompts customizados: Use o parâmetro `prompt` para enviar um texto livre

Dados de Exemplo

Para facilitar o desenvolvimento, o teste inclui:

1. **Dataset**

    - Um JSON com usuários fictícios e seus históricos de compra
    - Catálogo de produtos com informações básicas
    - Exemplos de recomendações para referência
2. **API Mock**
    - Um simulador de API de LLM para testes locais
    - Exemplos de respostas para diferentes tipos de prompts

> Se preferir você pode criar seus próprios dados de exemplo

Entregáveis

1. **Código Fonte**
    - Repositório Git completo com histórico de commits
2. **Documentação**
    - Documentação da API (Swagger/OpenAPI)
    - Análise das considerações éticas
3. **Demonstração**
    - Instruções claras para executar o projeto
    - Exemplos de uso da API e da interface

Processo de Entrega

Um dos critérios que vamos avaliar é como são os commits. Abaixo estão as instruções de como enviar a sua solução.

Para enviar o seu projeto do git você deve gerar um git bundle. O nome do arquivo bundle deve ser `nome_sobrenome.bundle`. Para gerar o bundle basta executar o seguinte comando:

```
git bundle create nome_sobrenome.bundle HEAD <nome_branch>
```

Por exemplo, se os commits estiverem na branch main:

```
git bundle create nome_sobrenome.bundle HEAD main
```

Depois basta enviar o arquivo para o e-mail da pessoa que tiver te enviado o teste.

Se você quiser conferir que o bundle foi gerado com sucesso, basta copiá-lo para um nova pasta e executar:

```
git clone nome_sobrenome.bundle meu_projeto
```

Uma cópia do seu repositório vai ser feita na pasta meu_projeto.

Critérios de Avaliação

Qualidade do Código

- Princípios SOLID e padrões de design
- Organização e estrutura do código
- Tratamento de erros e edge cases

Integração com IA Generativa

- Estratégias de fallback e otimização
- Abstração e flexibilidade da integração
- Tratamento de limitações dos LLMs

Design de Sistema

- Arquitetura geral e modelagem de domínio
- Design da API e interfaces
- Facilidade de manutenção e extensão

Considerações Éticas e Documentação

- Identificação de questões éticas
- Estratégias para mitigar problemas
- Qualidade da documentação
- Clareza nas explicações de decisões de design

Perguntas de Reflexão

Inclua respostas breves às seguintes questões em sua documentação:

1. Como você equilibraria o uso de LLMs (que podem ser custosos) com métodos tradicionais de recomendação?
2. Quais são os principais desafios éticos de usar IA generativa em um sistema de recomendação?
3. Como você abordaria o problema de explicabilidade em um sistema que combina métodos estatísticos com outputs de LLMs?
4. Quais métricas você usaria para avaliar o sucesso de sua implementação em um ambiente de produção?

Recursos Permitidos

- Acesso à internet para consulta de documentação
- Uso de frameworks e bibliotecas open source
- APIs públicas de LLMs (pode usar chaves de API fornecidas para teste) se caso não preferir usar os simuladores.
- Uso de LLMs para auxilio na geração do código.

Bônus (pontos extras)

- Sistema de logs para monitorar inferências em produção
- Implementação de testes unitários para as classes refatoradas
- Configuração de um workflow de CI/CD para o Gitlab no projeto
- Implementação de um padrão de design adicional (Factory, Builder, Decorator, etc.)

Dicas

- Comece pelo design da arquitetura antes de implementar
- Foque na qualidade do código em vez de funcionalidades extras
- Para economizar tempo, você pode simular algumas interações com LLMs
- Priorize a implementação dos componentes core antes de adicionar recursos adicionais