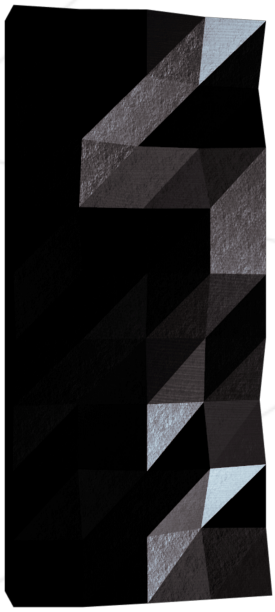

Imersão em GenAI para Desenvolvedores

Projeto 3: Introdução a *Prompt Engineering*

Preâmbulo: Neste projeto, você vai explorar plataformas de GenAI, aprender técnicas básicas de engenharia de prompt e comparar diferentes modelos linguísticos.

Versão: 1.0

Sumário

Capítulo I

Preâmbulo

> *No campo da aprendizagem de máquina, a expressão 'papagaio estocástico' serve como uma metáfora para ilustrar a ideia de que, embora os modelos linguísticos possam produzir linguagem coerente, eles carecem de uma compreensão do significado por trás da linguagem que geram.*

Este termo foi introduzido por Emily M. Bender et al. em 2021, no artigo de pesquisa em inteligência artifical intitulado:

"Sobre os Perigos dos Papagaios Estocásticos: Os Modelos Linguísticos Podem Ser Grandes Demais?"

A metáfora do papagaio estocástico destaca:

- Uma limitação fundamental dos modelos linguísticos atuais
- A capacidade de gerar texto convincente sem compreensão real
- A semelhança com um papagaio que repete frases sem entender seu significado

Implicações para engenharia de prompt:

1. **Precisão:** Necessidade de formular instruções altamente precisas
2. **Contextualização:** Importância de fornecer contexto adequado
3. **Reconhecimento de limitações:** Compreensão das capacidades e restrições do modelo

Ao entender que estamos essencialmente 'guiando' um sistema sem compreensão real, podemos:

- Desenvolver prompts mais eficazes
- Aproveitar ao máximo as capacidades do modelo
- Reconhecer e mitigar suas limitações inerentes

— *Baseado no trabalho de Bender et al., 2021*[1]

[1]Bender, E. M., Gebru, T., McMillan-Major, A., & Mitchell, M. (2021). On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency.

Capítulo II

Instruções gerais

- Esta página é sua única referência oficial. Não confie em informações não verificadas.
- Os exercícios estão organizados em ordem crescente de complexidade. É essencial dominar cada exercício antes de prosseguir para o próximo.
- Preste atenção às permissões de seus arquivos e pastas.
- Siga rigorosamente o procedimento de entrega para todos os exercícios.
- Seus exercícios serão avaliados por colegas da Imersão.
- Para exercícios em Shell, utilize /bin/zsh.
- Mantenha em sua pasta apenas os arquivos explicitamente solicitados nos enunciados.
- Em caso de dúvidas, consulte seus colegas à direita ou à esquerda.
- Utilize recursos como Google, manuais online e a Internet como referência.
- Leia os exemplos com atenção. Eles podem conter requisitos não explicitamente mencionados no enunciado.
- Para exercícios em Python:
    - Use a versão do Python especificada no exercício de configuração do ambiente.
    - Utilize os modelos e provedores sugeridos para garantir tempos de resposta adequados e consistência nos testes.
- Esteja atento a erros em todos os exercícios. Eles raramente são tolerados durante a avaliação.
- **Aviso sobre o uso de ferramentas de AI (como ChatGPT):**
    - O uso de ferramentas como o ChatGPT não deve ser encarado como um substituto para seu próprio esforço e entendimento.
    - O aprendizado efetivo ocorre quando você interage ativamente com o conteúdo: pesquisando, refletindo e aplicando o que aprendeu.
    - Nas avaliações, serão feitas perguntas para avaliar sua compreensão real sobre o assunto.

- E durante as avaliações, seus colegas também avaliarão seu nível de conhecimento.

Capítulo III

Ex00 – Grandes modelos linguísticos também são pequenos

	Exercício : 00
Realizar uma análise comparativa de três modelos linguísticos: Qwen, Llama e Gemini, testando suas capacidades e limitações em primeira mão para obter uma compreensão prática de cada modelo. Também vamos configurar ferramentas para utilizarmos nos exercícios seguintes.	
Pasta de entrega : $ex00/$	
Arquivos para entregar : `comparacao_modelos.txt`	

Conceitos fundamentais sobre *Large Language Models* (LLMs)

1. O que são LLMs?
   - LLMs são modelos linguísticos treinados em grandes quantidades de texto, capazes de entender, gerar e processar linguagem natural de maneira semelhante aos humanos.
   - Eles aprendem padrões e relações no texto, permitindo-lhes realizar uma ampla gama de tarefas, desde completar frases até responder perguntas e gerar textos coerentes.
   - Saiba mais: O que são grandes modelos linguísticos?
2. Parâmetros (*weights*)
   - Os parâmetros são os valores aprendidos que determinam o comportamento e as capacidades do modelo.

- O número de parâmetros, geralmente na ordem de bilhões (B) ou trilhões (T), é uma medida da capacidade e complexidade do modelo.
- Modelos maiores com mais parâmetros geralmente são capazes de desempenho melhor e mais nuances, mas também são mais desafiadores de treinar e servir.

3. Tokens e tokenização
   - Antes de ser processado por um *Large Language Model* (LLM), o texto é dividido em tokens, que podem ser palavras, subpalavras ou até mesmo caracteres individuais.
   - A tokenização é o processo de converter texto em uma sequência de tokens, que serve como entrada para o modelo.
   - A contagem de tokens é uma medida importante, pois a maioria dos LLMs tem um comprimento máximo de sequência (por exemplo, 8192 tokens) e os custos são geralmente calculados por token.
   - Saiba mais: O que são tokens e como contá-los?
4. Aplicações dos LLMs
   - Geração de texto (escrita criativa, redação de artigos, diálogo de chatbots)
   - Completar tarefas (responder perguntas, fornecer definições e explicações, dar instruções passo a passo)
   - Análise de texto (classificação de sentimentos, extração de entidades, sumarização)
   - E muito mais: novos casos de uso estão constantemente sendo explorados à medida que as capacidades dos LLMs avançam.

Configurações

Você configurará e usará três modelos principais de GenAI: Qwen (via Ollama), Llama (via Groq) e Gemini (via Google). Siga as instruções abaixo para configurar cada uma delas.

Ollama

- O Ollama é uma ferramenta que permite executar modelos linguísticos localmente em seu próprio hardware.
- Já está instalado neste computador. Para saber mais sobre as instruções de instalação, visite: `https://github.com/ollama/ollama`

- Vamos utilizar o modelo: qwen2:1.5b
- Vantagens: Gratuito, execução local, sem limites de uso além do seu próprio hardware.
- Configuração:

1. Verifique a instalação: `which ollama`
2. Faça o pull do modelo: `ollama pull qwen2:1.5b`. Verifique os modelos instalados com: `ollama list`
3. Use o terminal para interagir com o modelo: `ollama`
4. Use o comando `/set verbose` para visualizar estatísticas de processamento

Groq

- Groq oferece acesso a modelos linguísticos via API com um plano gratuito generoso.
- Acesse `https://console.groq.com/docs/quickstart` para criar uma conta e obter uma chave de API
- Modelo a ser utilizado: llama3-8b-8192
- Use o playground para testar o modelo

Google

- Google oferece acesso ao seu modelo Gemini via Google AI Studio e API, com um plano gratuito também generoso.
- Acesse `https://ai.google.dev/gemini-api/docs/ai-studio-quickstart` para criar uma conta (ou use sua conta Google existente) e obter uma chave de API
- Modelo a ser utilizado: gemini-1.5-flash
- Utilize o playground para conversar com o modelo

Lembre-se de guardar suas chaves de API em um lugar seguro e não compartilhá-las. Cada plataforma tem suas próprias políticas de uso e limites de taxa, então familiarize-se com elas antes de começar. O uso dentro dos limites gratuitos é suficiente para esta Imersão.

Instruções

Você irá fazer uma série de perguntas a cada modelo, tanto em português quanto em inglês, e analisar suas respostas. Isto permitirá que você observe as diferenças de performance entre os modelos e entre os idiomas.

1. Crie um arquivo de texto chamado "comparacao_modelos.txt".

2. Para cada pergunta abaixo, faça a consulta em cada modelo e anote a última resposta que você obteve no arquivo de texto usando o seguinte formato:

```
pergunta: [PERGUNTA]

respostas:
* qwen: [ÚLTIMA RESPOSTA DO QWEN]
* llama: [ÚLTIMA RESPOSTA DO LLAMA]
* gemini: [ÚLTIMA RESPOSTA DO GEMINI]

================================

pergunta: [PERGUNTA]
...
```

3. Perguntas a serem feitas para cada modelo:

   (a) Quais foram os países medalistas finais das Olimpíadas de 2024?

   (b) Quem é o patrono da cidade de São Bernardo do Campo e qual é a sua história?

   (c) If we lay 5 shirts out in the sun and it takes 4 hours to dry, how long would 20 shirts take to dry? Answer immediately giving the number and nothing else.

   (d) Q: The odd numbers in this group add up to an even number: 15, 32, 5, 13, 82, 7, 1. A:

4. Para cada pergunta, registre no arquivo:

   - A pergunta
   - A última resposta de cada modelo

5. Após registrar as respostas, reflita sobre as seguintes questões:

   - Qual modelo teve a melhor performance?
   - Quais são os pontos fortes e fracos de cada modelo?

- Quais diferenças notáveis você observou entre as respostas em português e inglês?

6. Salve o arquivo “comparacao_modelos.txt”

Lembre-se, o objetivo é entender os limites dos modelos. Respostas incorretas ou "Não sei" também são informativas sobre o conhecimento de cada modelo.

Velocidade vs performance: Por convenção sobre LLMs, velocidade refere-se à rapidez de inferência (geração de tokens por segundo (T/s)), enquanto performance refere-se à qualidade dos resultados gerados.

Capítulo IV

Ex01 – Seja claro, direto e detalhado

	Exercício : 01
Aprender a criar prompts eficazes para interagir com modelos linguísticos, focando em clareza, objetividade e riqueza de detalhes contextuais	
Pasta de entrega : $ex01/$	
Arquivos para entregar : `be_structured.py, job_description.txt`	

Ao interagir com um modelo, imagine-o como um funcionário bem qualificado, porém inexperiente (e com amnésia), que necessita de instruções claras. Semelhante a um novo colaborador, o modelo não possui conhecimento prévio sobre suas normas, estilos, diretrizes ou métodos de trabalho preferidos. Quanto mais detalhadamente você explicar o que deseja, melhor será a resposta do modelo.

Dê ao modelo informações contextuais: Assim como você poderia ter um desempenho melhor em uma tarefa se soubesse mais informações, o modelo terá um desempenho melhor se tiver mais informações contextuais.

Alguns exemplos de informações contextuais:

- Para que os resultados da tarefa serão usados
- Para qual público o resultado é destinado
- De qual fluxo de trabalho a tarefa faz parte e onde esta tarefa se encaixa nesse fluxo
- O objetivo final da tarefa, ou como é uma conclusão bem-sucedida da tarefa

Seja específico sobre o que você quer que o modelo faça: Por exemplo, se você deseja que o modelo gere apenas código e nada mais, seja explícito ao dizer isso.

Forneça instruções em etapas sequenciais: Utilize listas numeradas ou com marcadores para garantir que o modelo execute a tarefa exatamente como você deseja.

IV.1 Exemplo 1: Escrever email para uma campanha de marketing

Prompt vago:

```
Escreva um e-mail de marketing para nossos novas Imersão em GenAI para não técnicos.
```

Prompt detalhado:

```
Sua tarefa é criar um e-mail convidativo para a Imersão em GenAI, voltada para não técnicos.

Foco:
- Público: Profissionais de diversas áreas sem conhecimento técnico em IA.
- Destaques: Aplicações práticas, linguagem acessível, mentoria personalizada.
- Tom: Profissional, encorajador, acessível.
- CTA: Inscrição com bônus exclusivo.
- Assunto: <50 caracteres, com "GenAI" e "sem código".
- Personalização: {{NOME_DA_EMPRESA}} e {{NOME_DO_CONTATO}}.

Estrutura:
1. Assunto chamativo.
2. Corpo do e-mail (breve e impactante).
3. CTA claro e convidativo.
```

IV.2 Exemplo 2: à incidente de infraestrutura

Prompt vago:

```
Analise este relatório de interrupção dos serviços da 42 São Paulo e resuma os pontos principais.

{{DOCUMENT}}
```

Prompt detalhado:

```
Analise este relatório de interrupção de serviços da 42 São Paulo. Pule o preâmbulo. Mantenha sua resposta
      concisa e escreva apenas as informações estritamente necessárias. Liste apenas:
1) Causa
2) Duração
3) Serviços afetados
4) Número de usuários afetados
5) Perda estimada de receita.

Aqui está o relatório: {{DOCUMENT}}
```

IV.3 Exercício

Você está explorando técnicas de engenharia de prompt para melhorar a comunicação com modelos de GenAI. Você vai criar um programa Python que analisa descrições de emprego e extrai informações estruturadas utilizando zero-shot e few-shot prompting.

Instruções

1. Obtenção de descrições de emprego:
    - Buscar pelo menos uma descrição de emprego real no LinkedIn e copiar o texto para o arquivo "job_description.txt".

2. Formatação do prompt: Criar uma função para formatar o prompt baseado na descrição do emprego.

3. Consulta a múltiplos modelos: Implementar consultas nos três modelos disponíveis.

4. Análise comparativa: Apresentar os resultados de cada modelo no seu output.

Como o seu programa será chamado:

```
def main():
    with open("job_description.txt", "r") as file:
        job_description = file.read()

    formatted_prompt = format_prompt(job_description)
    results = query_all_models(formatted_prompt)

    for model, response in results.items():
        print(f"\nAnálise do {model}:")
        print(response)
        print("-" * 50)

if __name__ == "__main__":
    main()
```

Saída esperada:

```
-> python be_structured.py
Consultando Gemini...
Consultando Groq...
Consultando Ollama...

Análise do Gemini 1.5 Flash:
Name of role: Pleno Software Engineer (BackEnd)
Working hours: Flexible
Country: No information
Tech skills: Python (Django or FastAPI), Java (Spring), C#, .NET, REST, PostgreSQL, Oracle, SQL Server,
     OpenStack, AWS, GCP, Azure, Programação Reativa, TDD, MongoDB, Docker, Kubernetes, Jenkins, Github,
     Gitlab

--------------------------------------------------

Análise do Llama 3 8B:
Based on the provided job description, here is the extracted information in the requested categories:

**Name of role:** Pleno Software Engineer (BackEnd)

**Working hours:** Flexível (Flexible)

**Country:** Não informado (Not specified)

**Tech skills:**

* Python (Django or FastAPI)
* Java (Spring)
* C# (.NET)
* PostgreSQL, Oracle, SQL Server (database management systems)
* OpenStack, AWS, GCP, Azure (cloud computing platforms)
* Docker, Kubernetes (containers and orquestrators)
* Jenkins, Github, Gitlab (CI/CD tools)
* Programação Reativa (Reactive programming)
```

```
* TDD (Test-Driven Development)

Note that some skills are mentioned as desirable or additional, but I have only listed the core tech
     skills required for the role.
--------------------------------------------------

Análise do Qwen2 1.5B:
Name of role: Pleno Software Engineer (BackEnd)
Working hours: 8-5 (Monday to Friday)
Country: United States
Tech skills: Python, Java, C#, programming methodologies and frameworks
No information
--------------------------------------------------
```

Considere as particularidades comuns em descrições de emprego para melhorar a extração de informações.

Não deixe a sua chave de API no código submetido. Durante a avaliação, você precisa configurá-la adequadamente.

Capítulo V

Ex02 – Use separadores para estruturar seus prompts

	Exercício : 02
Aprender a utilizar separadores para estruturar prompts de forma mais precisa e organizada, melhorando a comunicação com modelos linguísticos	
Pasta de entrega : $ex02/$	
Arquivos para entregar : `prompt_generator.py`	

Quando seus prompts envolvem vários componentes como contexto, instruções e exemplos, o uso de separadores pode fazer toda a diferença. Separadores, como tags XML ou estruturas JSON, ajudam qualquer LLM a analisar seus prompts com mais precisão, resultando em saídas de maior qualidade.

Dependendo do treinamento do modelo, ele pode se comportar com mais performance para diferentes tipos de separadores. JSON é amplamente utilizado em APIs, facilitando a integração com outras ferramentas. Por outro lado, XML oferece uma estrutura mais verbosa, mas altamente legível, com suporte para dados mais complexos.

Neste exercício, focaremos no uso de tags XML devido à sua clareza e flexibilidade para estruturar prompts.

Por que usar tags XML?

- **Clareza:** Separe claramente diferentes partes do seu prompt e garanta que ele esteja bem estruturado.
- **Precisão:** Reduza erros causados pelo modelo interpretando incorretamente partes do seu prompt.
- **Flexibilidade:** Encontre, adicione, remova ou modifique facilmente partes do seu prompt sem reescrever tudo.

- **Pós-processamento:** Fazer com que o LLM use tags XML em sua saída facilita a extração de partes específicas de sua resposta por pós-processamento.

Prompts avançados geralmente consistem em diversas seções, como um conjunto de regras ou contexto adicional. Essas seções precisam ser claramente delimitadas para evitar confusão de conteúdo.

Por um tempo, separadores (*separators*) como `###` ou `"""` ou ‘‘‘ foram usados no conteúdo do prompt. No entanto, eles não funcionam bem com dados que não são fixos, que pode conter, por exemplo, sintaxe markdown cujo conteúdo pode entrar em conflito com o separador escolhido. Ainda que um LLM possa lidar com essa situação, um prompt maior se torna menos compreensível pessoas, o que também é um fator importante a ser considerado.

Boas práticas

- **Seja consistente:** Utilize os mesmos nomes de tags em todos os seus prompts e faça referência a esses nomes ao falar sobre o conteúdo (por exemplo, "Usando o contrato nas tags <contrato>...").

- **Tags dentro de tags:** Você deve aninhar tags <externo><interno></interno></externo> para criar hierarquia no seu prompt.

Diversas técnicas podem ser utilizadas em conjunto para aprimorar ainda mais seus prompts. Por exemplo, você pode combinar tags XML com few-shot (usando tags como <exemplos></exemplos>).

V.1 Exemplo 1: Gerando relatórios financeiros

Sem tags XML:

```
Você é um analista financeiro da AcmeCorp. Gere um relatório financeiro do segundo trimestre para nossos
    investidores. Inclua seções sobre Crescimento da Receita, Margens de Lucro e Fluxo de Caixa, como
    neste exemplo do ano passado: {{Q1_REPORT}}.

Use os pontos de dados desta planilha: {{SPREADSHEET_DATA}}.

O relatório deve ser extremamente conciso, direto ao ponto, profissional e em formato de lista. Ele deve
    destacar os pontos fortes e as áreas que precisam de melhorias.
```

Com tags XML:

```
Você é um analista financeiro da InovaIA, uma startup de IA em São Paulo. Gere um relatório financeiro do
    segundo trimestre para nossos investidores.

Nossos investidores valorizam a transparência e insights acionáveis.

Use estes dados para o seu relatório:
<dados>
{{SPREADSHEET_DATA}}
</dados>

<instruções>
Inclua seções: Crescimento da Receita, Margens de Lucro, Fluxo de Caixa.
```

```
Destaque os pontos fortes e as áreas que precisam de melhorias.
</instruções>

Faça seu tom conciso e profissional. Siga esta estrutura:
<exemplo_de_formatação>
{{Q1_REPORT}}
</exemplo_de_formatação>
```

V.2 Exercício

Você sabe de uma técnica para melhorar seus prompts: estruturação com tags XML. Esta abordagem permite uma comunicação mais precisa e organizada com os modelos, reduzindo mal-entendidos e aumentando a flexibilidade. Você vai criar um gerador de prompts que utilize tags XML para estruturar o conteúdo de forma eficaz.

Instruções

1. Implementar uma função create_prompt que gera um prompt estruturado com tags XML.

2. Utilizar a função para criar prompts para diferentes cenários.

3. Enviar o prompt gerado para o modelo Gemini 1.5 Flash.

4. Analisar e comparar as respostas obtidas.

Como o seu programa será chamado:

```
role = "especialista em filosofia e história da ciência"
task = "explicar o pensamento de Descartes e sua influência para iniciantes em filosofia"
topic = "René Descartes e o Método Cartesiano"
specific_question = "Quem foi René Descartes e qual é o significado da frase 'Penso, logo existo'?"

prompt = create_prompt(role, task, topic, specific_question)
response = send_to_gemini(prompt)
print("\nResposta do Gemini 1.5 Flash:")
print(response)
```

Saída esperada:

```
Resposta do Gemini 1.5 Flash:
## Quem foi René Descartes e qual é o significado da frase 'Penso, logo existo'?

### 1. Explicação básica do conceito:

René Descartes foi um filósofo e matemático francês do século XVII considerado o pai da filosofia moderna.
     Ele buscava um fundamento sólido para o conhecimento em uma época marcada por incertezas. Sua frase
    mais famosa, "Penso, logo existo", resume sua ideia central: a única certeza que temos é a de nossa
    própria existência, pois duvidar dela já comprova que pensamos, e portanto, existimos.
```

```
### 2. Analogia do cotidiano:

Imagine que você está sonhando. No sonho, tudo parece real, mas ao acordar você percebe que não era.
    Descartes se perguntou como ter certeza de que não estamos vivendo em um sonho constante. A única
    certeza que encontrou foi o próprio ato de duvidar, de pensar, pois isso indicava a existência de um
    "eu" que duvida.

### 3. Solução passo a passo da pergunta:

* **Quem foi René Descartes?** Um filósofo e matemático que buscava a verdade e a certeza em tempos de
    incerteza.
* **O que ele buscava?** Um fundamento sólido para o conhecimento, algo que fosse irrefutavelmente
    verdadeiro.
* **Como ele chegou à frase?** Duvidando de tudo, inclusive do mundo externo e do próprio corpo.
* **Qual a conclusão?** A única certeza é que, ao duvidar, ele estava pensando, e se ele pensa, ele existe
    .

### 4. Exemplo detalhado:

Imagine que você está diante de uma ilusão de ótica. Seus olhos veem uma coisa, mas você sabe que é uma
    ilusão, que a realidade é diferente. Descartes aplica essa ideia ao conhecimento em geral: como saber
     se o que percebemos é real ou ilusório? Para ele, a única certeza é o ato de duvidar e questionar,
    pois isso implica um sujeito pensante.

**Exemplo:**

1. **Dúvida:** Posso duvidar da existência do mundo exterior.
2. **Dúvida ainda maior:** Posso duvidar da existência do meu próprio corpo.
3. **Certeza:** Não posso duvidar que estou duvidando, pois duvidar é uma forma de pensar.
4. **Conclusão:** Se penso, logo existo ("Cogito, ergo sum").

### 5. Dica prática para iniciantes:

Lembre-se da frase "Penso, logo existo" quando se deparar com informações complexas ou questionamentos
    sobre a realidade. Ela nos lembra que a capacidade de pensar criticamente e questionar é fundamental
    para a busca da verdade.
```

Experimente com diferentes formatos de saída para cada técnica.

Não deixe a sua chave de API no código submetido. Durante a avaliação, você precisa configurá-la adequadamente.

Exemplos adicionais

Utilize os exemplos do arquivo fornecidos abaixo para utilizar outras entradas e verificar a saída do modelo.

```
role = "assistente especializado em ensinar programação Python para iniciantes"
task = "explicar conceitos básicos de Python e fornecer exemplos simples e práticos"
topic = "list comprehensions em Python"
specific_question = "O que é uma list comprehension e como posso usá-la para criar uma lista de números
    pares de 0 a 10?"
```

```
role = "especialista em métodos de educação inovadores em tecnologia"
task = "explicar o conceito e a abordagem única da École 42 para interessados em educação em tecnologia"
topic = "École 42 e seu método de ensino"
specific_question = "O que é a École 42 e como seu método de ensino difere das faculdades tradicionais de
    computação?"

role = "historiador da ciência da computação e teoria da informação"
task = "explicar a importância de Claude Shannon e suas contribuições para iniciantes em ciência da
    computação"
topic = "Claude Shannon e a Teoria da Informação"
specific_question = "Quem foi Claude Shannon e qual foi sua principal contribuição para a ciência da
    computação e comunicação?"
```

Capítulo VI

Entrega e Avaliação entre pares

VI.1 Processo de Entrega

- Submeta seu trabalho no repositório Git gerado na página principal do projeto.
- Certifique-se de que todos os arquivos necessários estejam incluídos e organizados conforme as instruções do projeto.
- Respeite o prazo de entrega estabelecido.

VI.2 Avaliação entre pares

- Seu projeto será avaliado por um dos seus colegas.
- A avaliação focará na qualidade do seu código e na aderência aos requisitos do projeto.
- Critérios de avaliação podem incluir:

    1. Funcionalidade: O código atende a todos os requisitos especificados?
    2. Legibilidade: O código é claro e bem estruturado?
    3. Eficiência: As soluções implementadas são otimizadas e seguem boas práticas?
    4. Organização: Os arquivos e estrutura do projeto estão bem organizados?

- Feedback detalhado é esperado, mas pode variar em extensão e detalhamento.

VI.3 Dicas para uma avaliação bem-sucedida

- Revise seu código antes da submissão final.

- Teste exaustivamente todas as funcionalidades implementadas.
- Se entender necessário para clareza, documente claramente qualquer decisão ou suposição feita.
- Esteja preparado para explicar suas escolhas de implementação.

A avaliação entre pares é uma oportunidade para aprendizado e crescimento pessoal e profissional. Esteja aberto ao feedback recebido e use-o para aprimorar suas habilidades.