Inteligência Artificial 2

Aula 28: Hackathon do Projeto Final (Integração e Interface)

Prof. Aléssio Miranda Júnior
alessio@cefetmg.br

CEFET-MG - Campus Timoteo
Dep. Engenharia de Computação

Inteligência Artificial 2

Sumário

Onde Paramos?

Ontem vocês construíram o **cérebro do produto**:

- Pipeline de ingestão: PDFs → chunks → VectorStore.
- Endpoint `POST /api/chat` respondendo com RAG no Postman.
- System Prompt definido com persona, tom e guardrails.

A Missão de Hoje

O cérebro funciona, mas é **invisível**.

Hoje vocês constroem o **rosto** — a interface do produto.

A Invisibilidade do Código Backend

- Durante os laboratórios anteriores, a sua equipe otimizou Embeddings, gerenciou janelas de contexto com RAG e validou regras estritas com LangChain.
- **A dura realidade comercial:** Para o usuário final, nada disso existe. Uma IA majestosa que roda apenas via cURL no terminal ou chamadas do Postman possui **valor comercial nulo** para clientes leigos.
- O "Fator Uau!" que atrai investimentos e retém clientes reside unicamente na **Experiência do Usuário (UX)** e na facilidade com que ele interage com as funcionalidades construídas pela engenharia.

Paradigmas de Interface de IA

Como as Inteligências Artificiais são apresentadas ao mercado?

1. **Agente Conversacional (Chatbot):** A interface clássica à la ChatGPT. Uma tela em branco centralizada num input de texto, onde o histórico dita o fluxo. (Ideal para consultas gerais).
2. **Copilotos (Copilots):** IAs incorporadas na lateral da ferramenta principal. Ex: Um editor de planilhas na esquerda, e o assistente sugerindo fórmulas na direita. (Alto acoplamento, maior valor utilitário).
3. **Geração Invisível (Generative UI):** O usuário não digita um *prompt*. Ele aperta um botão "Resumir este mês" em um Dashboard e a IA altera o componente visual (um gráfico) e o texto ao redor dinamicamente.

A Escolha da Ferramenta de Frontend

Para o Hackathon, o relógio é o maior inimigo. Qual ferramenta adotar?

A Via Rápida: Streamlit

- Escrito puramente em Python.
- Desenha botões, caixas de texto e exibe dados automaticamente sem saber HTML/CSS.
- Curva de aprendizado de 2 horas.
- **Contras:** Difícil customização extrema. Arquitetura "top-down" estranha para desenvolvedores web nativos.

A Via Escalável: React / Next.js

- O padrão-ouro do mercado. O HTML e o estado (Variáveis) reagem dinamicamente.
- Total liberdade de Design, permitindo produtos visualmente impecáveis.
- **Contras:** Curva de aprendizado íngreme. Exige servidor próprio para *hosting* (Node.js/Vercel).

A Mecânica do Streamlit (Script Re-run)

A maior confusão para quem codifica em *Streamlit* é o fluxo de execução.

- Quando um usuário clica num botão, a página **não** faz um POST tradicional.
- O servidor do Streamlit re-executa **todo o arquivo Python de cima a baixo**.
- Variáveis comuns declaradas no código (ex: `chat_history = []`) serão apagadas e recriadas vazias a cada nova mensagem enviada, destruindo a conversa na tela!

Streamlit: Gerenciando o Histórico (Session State)

Para que o chat não suma a cada clique, deve-se guardar o "Estado" de forma global à sessão do navegador.

```
import streamlit as st
import requests

# 1. Inicializa o banco de memoria do usuario atual
if ''messages'' not in st.session_state:
    st.session_state.messages = []

# 2. Exibe as mensagens anteriores na tela sempre que a pagina
   recarregar
for msg in st.session_state.messages:
    with st.chat_message(msg[''role'']):
        st.markdown(msg[''content''])

# 3. Captura nova mensagem e salva no estado
if prompt := st.chat_input(''Digite sua pergunta...''):
```

Componentes Web Inteligentes (Vercel AI SDK)

Se a equipe optou por **React/Next.js**, o mercado não constrói mais caixas de texto com código do zero. Usa-se o Vercel AI SDK.

```
import { useChat } from 'ai/react';

export default function Chat() {
  // A biblioteca gerencia as mensagens, e o estado de ''isLoading''
      sozinha
  const { messages, input, handleInputChange, handleSubmit,
     isLoading } = useChat({
    api: 'http://localhost:8000/api/chat' // Aponta para o Backend
       Python
  });

  return (
    <form onSubmit={handleSubmit}>
      {messages.map(m => <div key={m.id}>{m.role}: {m.content}</div
          >)}
```

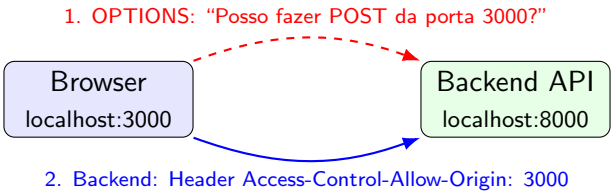
A Política de Mesma Origem (SOP)

Para times que utilizam frameworks JavaScript (React, Vue, Vanilla JS), a maior dor de cabeça do dia é ver a tela vermelha no console: **Blocked by CORS policy**.

- O Navegador Web possui uma lei de segurança estrita: O *Same-Origin Policy* (SOP).
- Se o seu Frontend foi carregado a partir de `http://localhost:3000`, e tenta enviar um POST invisível via JavaScript (`fetch`) para o Backend em `http://localhost:8000`, o Navegador bloqueia a leitura dos dados na volta.
- Por quê? Para impedir que um site hacker (`malicioso.com`) roube informações simulando chamadas no navegador do cliente, sem que o cliente saiba.

CORS e a Requisição OPTIONS (Preflight)

Como resolvemos isso? O seu **Backend** precisa avisar o navegador que confia naquela interface específica. Quando o Frontend tenta o `POST`, o navegador pausa e envia secretamente um pacote HTTP leve chamado `OPTIONS` (o "Preflight").



Se o Backend autorizar as regras pelo cabeçalho, o navegador libera o `POST` real com os dados.

Configurando o CORS (Backend FastAPI e Spring)

Você não escreve headers à mão. Usa *Middlewares* nas linguagens. **Em Python / FastAPI:**

```
from fastapi.middleware.cors import CORSMiddleware

app.add_middleware(
    CORSMiddleware,
    allow_origins=[''http://localhost:3000''], # Quem pode chamar
    allow_credentials=True,
    allow_methods=[''GET'', ''POST'', ''OPTIONS''], # Metodos
       autorizados
    allow_headers=[''*''], # Cabecalhos autorizados (Ex:
       Authorization)
)
```

Se esquecer esta configuração, seu Frontend moderno ficará eternamente cego, sem acessar as APIs de IA!

O Paradoxo do Tempo em IA

Bancos de dados clássicos (SQL) retornam informações em 20 milissegundos. Sistemas de IA modernos empilham gargalos:

- *RAG Dense Retrieval:* $\sim$ 100 a 400ms.
- *Geração (API Externa via Rede):* $\sim$ 500ms só para o primeiro byte.
- *Decodificação Autoregressiva:* $\sim$ 3 a 15 segundos para completar uma resposta longa e detalhada.

A Fuga Psicológica: Se a UI ficar travada por 15 segundos, 90% dos usuários leigos atualizarão a página (`F5`) ou fecharão a aba achando que "deu pau".

UX Trick 1: Indicadores Visuais (Skeletons/Spinners)

A regra número um de interfaces humanas (Nielsen Norman Group): **Mantenha a Visibilidade do Estado do Sistema.**

- Assim que o usuário clicar em "Enviar", limpe o campo de texto *imediatamente* e trave o botão (Desabilite).
- Insira uma bolinha "Pensando..." ou "Buscando nos documentos...".
- Essa pequena transição diz ao cérebro do usuário: *"O seu clique funcionou, a responsabilidade agora é da máquina. Aguarde."*

UX Trick 2: O Streaming de Tokens

A técnica que tornou o ChatGPT globalmente aceitável foi exibir a resposta "sendo digitada" em vez de esperar um blocão de texto gigante surgir do nada. O conceito subjacente é o

Server-Sent Events (SSE) ou **WebSockets**:

- Numa requisição HTTP normal, o servidor fecha a porta só após processar 100% dos dados, entregando tudo num único pacote.
- No SSE, o Backend sinaliza `Transfer-Encoding: chunked`. A porta fica entreaberta.
- A cada *token* gerado pelo LLM (ex: "Bom", depois "dia,"), o Backend injeta um mini-pacote na conexão TCP. O Frontend captura o evento e renderiza a palavra na tela **em tempo real**.

Backend Streaming na Prática (Yield)

Para fazer *Streaming* no Backend, a função não usa o clássico `return`. Ela usa geradores assíncronos (`yield`).

```
from fastapi.responses import StreamingResponse

async def token\_generator():
    # LLM gerando token a token...
    yield ‘‘Este ’’
    await asyncio.sleep(0.1)
    yield ‘‘é o ’’
    await asyncio.sleep(0.1)
    yield ‘‘futuro.’’

app.post(‘‘/stream’’)async def chat_stream():return
    StreamingResponse(token_generator(), media_type=‘‘text/event-stream’’)
```

Exibição de Fontes em RAG Corporativo

O RAG traz um desafio adicional de interface: **Auditoria (Citações)**.

- O LLM gerar a resposta correta não basta; o gerente da empresa precisa do "Recibo" de onde o modelo tirou essa resposta.
- O contrato da API (*Backend*) deve enviar, junto com o texto gerado, um *Array* de `metadados` informando "Página 4 do PDF Regras2024".
- O Frontend deve criar *tooltips* visuais ou "Cartões de Referência" clicáveis ao final da resposta. A credibilidade do sistema aos olhos do cliente decola quando ele clica e abre a página específica do manual.

A Comunicação Full-Duplex (WebSockets)

Se o projeto exige mais que apenas streaming de texto (ex: Agentes conversando em áudio bidirecional em tempo real):

- O *Server-Sent Events* (SSE) possui uma falha: é unidirecional (Apenas o Backend fala após abrir).
- **WebSockets (ws://):** Cria um túnel aberto e persistente. O usuário pode interromper a IA no meio da fala enviando um pacote *JSON* pelo mesmo túnel, mandando-a parar de gerar.
- É o protocolo padrão utilizado no modo de voz nativo do *ChatGPT App*.

Segurança no Frontend (XSS via LLM Output)

O texto retornado pelo LLM é perigoso se renderizado cegamente na interface.

- **Cross-Site Scripting (XSS):** Se o LLM alucinar e gerar um HTML como `<script>roubar_cookie()</script>`, e o seu *Frontend* React renderizar com `dangerouslySetInnerHTML`, o ataque será executado no navegador do usuário.
- **Solução (Sanitização):** O Frontend de IA moderno sempre passa a resposta do Backend através de bibliotecas como `DOMPurify` ou conversores restritos de Markdown (`react-markdown`) que "esterilizam" *tags* de código antes de pintá-las na tela.

Gestão de Estado Global em IA (Redux / Context API)

Para interfaces mais ricas que um simples Chat (Copilotos complexos):

- O estado da IA não reside apenas no balão de conversa. A IA pode decidir que o painel esquerdo deve fechar ou que a cor de fundo deve ficar escura.
- Se as respostas da API afetam toda a tela, o Frontend deve utilizar **Estado Global** (ex: *Context API* em React ou *Zustand*).
- O *payload* JSON que chega do servidor atualiza o estado global, e toda a UI "reage" simultaneamente, concretizando o paradigma *Generative UI*.

O Encontro das Águas (Dia 3)

Na Aula 28 (Terceiro dia de Hackathon), as rotas paralelas da equipe devem colidir:

1. **A Fusão:** O desenvolvedor de Backend expõe sua API na rede local da Faculdade ou via `NGROK` (túnel temporário). O desenvolvedor de Frontend troca os dados falsos (*Mocks*) pelas rotas HTTP reais da API do colega.
2. **O Teste de Carga de Fogo:** Tentem enviar prompts imensos. Vejam se o JSON quebra. Vejam se o CORS bloqueia. Vejam se há caracteres inválidos quebrando a renderização.
3. **Gestão da Crise:** Bugs não previstos saltarão na tela. A comunicação contínua (ex: "Backend, a chave 'user_id' não está no seu retorno") definirá o sucesso.

Tratamento Defensivo na Interface

O *Frontend* nunca pode confiar integralmente no *Backend*. Se o Backend morrer (Crash), a Interface deve tratar.

- Usem blocos `try/catch` (em JS) ou controle de exceção no `Streamlit`.
- Exibam "Falha no Serviço. Tente novamente mais tarde." em vez de exibir um erro cru de Python "JSONDecodeError" na cara do cliente.
- Isso prova senioridade da equipe.

Laboratório 28: A Criação do Rosto (Tangibilização)

Objetivos de Hoje:

1. Desenvolver as telas do sistema (*Chat, Dashboard, Área de Upload de Documentos*).
2. Integrar a `Interface` de fato com a `API Especialista` via Rede.
3. Lidar com estados de carregamento (*Loaders, CORS*).
4. Ao fim do dia, sua aplicação deve funcionar de "ponta a ponta" sem tocar num terminal, bastando que o cliente abra um navegador.

Dúvidas?

Email: alessio@cefetmg.br
Aula: Segunda e Quarta-Feira, 7h as 8:40h (Sala 39)
WhatsApp: Dúvidas rápidas

Lembrete

Confira sua Issue no GitLab!

Próxima aula: **Capstone Sprint 4**

100% da Aula 28 Concluída!