Capítulo 28 – Hackathon do Projeto Final (Integração e Interface)

Aléssio Miranda Jr.

Novembro - 2026/2

Capítulo em Revisão

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

O que você verá neste capítulo

O cérebro do seu projeto está funcionando perfeitamente no terminal, mas clientes não compram terminais pretos e requisições HTTP; eles compram soluções visuais que resolvem problemas. Neste capítulo do Hackathon, migraremos nossa atenção para a camada de visualização: o **Front-end**. Exploraremos as decisões de stack visual (desde o rápido Streamlit até a robustez do React), discutiremos a crítica configuração de CORS que trava integrações, e enfatizaremos as boas práticas de UX específicas para interfaces geradas por IA (estado de carregamento e *streaming* de resposta).

1 A Invisibilidade do Backend e o “Fator Uau”

Por mais brilhante que seja a engenharia de software construída nas aulas passadas – com bancos vetoriais finamente otimizados, cadeias complexas do LangChain e Agentes ReAct robustos que tomam decisões de milissegundos – ela permanece invisível e silenciosa. Para o usuário final comum, o gerente financeiro, o advogado ou o médico, APIs REST, parâmetros formatados em JSON e telas pretas de respostas do *Postman* não significam absolutamente nada.

O valor mercadológico da sua aplicação, e o chamado “Fator Uau!” esperado em bancas de Demoday, só acontecem quando a engenharia ganha uma **Interface**. A Aula 28 é inteiramente dedicada à tangibilização visual do produto. O objetivo é construir a porta de entrada amigável para que pes-

soas sem nenhum conhecimento de programação consigam interagir fluida e intuitivamente com a inteligência artificial desenvolvida pelo seu time.

2 Opções de Interface e Stack Tecnológico

No modelo de Hackathon, a criatividade e a escolha tecnológica são livres. A decisão de qual stack de Front-end utilizar deve sempre levar em consideração o conhecimento prévio dos membros da equipe e a terrível escassez de tempo (o Demoday não espera).

• Teoria: title=Avaliando as Escolhas de Front-end

- **Streamlit (O Favorito dos Cientistas de Dados):** É a rota mais rápida. O *Streamlit* é um framework que permite gerar interfaces web interativas diretamente a partir de scripts em Python. Ele possui componentes gráficos focados em GenAI (como o `st.chat_message`), ocultando completamente o *HTML/CSS*. Ideal para times que são nativos em Python e querem a interface rodando em 2 horas.
- **React / Next.js / Vue (A Rota Profissional Moderna):** A escolha recomendada para aplicações comerciais escaláveis. Permite criar componentes extremamente bonitos (com TailwindCSS) e gerenciar de forma profissional o estado global e o histórico longo de um chat. Leva mais tempo para estruturar o esqueleto e exige configuração manual do cliente HTTP (Axios/Fetch).
- **HTML, CSS e JavaScript Vanilla:** A rota clássica old-school. Arquivos estáticos conectados à sua API através da API nativa `fetch()`. Perfeito para quem tem agilidade extrema com design puro, porém, exige gerenciar o DOM (“adicionar a div de resposta na tela”) manualmente.

3 O Monstro Incompreendido: CORS

No exato momento em que um arquivo HTML (rodando em `localhost:3000`) tenta pedir dados para o seu Backend FastAPI/Spring (rodando em `localhost:8000`), o navegador moderno bloqueará a requisição exibindo uma tela vermelha de erro.

Isso se chama **CORS** (*Cross-Origin Resource Sharing*). O CORS é um mecanismo de segurança nativo dos navegadores (Chrome, Firefox) que impede que sites maliciosos façam requisições furtivas para outras APIs em segundo plano.

Tabela 1: Matriz de Decisão: Tecnologias de Front-end

Tecnologia	Curva de Aprendizado	Velocidade	Caso de Uso Ideal no Hackathon
Streamlit	Muito Baixa	Altíssima	Equipes compostas apenas por Cientistas de Dados (foco 100% em Python).
HTML/JS Puro	Baixa	Média	Equipes que querem controle total do design sem instalar Node.js.
React/Next.js	Alta	Baixa	Equipes que possuem desenvolvedores Web experientes e visam escalabilidade corporativa.

▶ Prática: title=Como Vencer o CORS no Hackathon

Para integrar seu Front e seu Back, a sua equipe de Backend **precisa explicitamente autorizar** a origem do Front-end.

- **No FastAPI (Python):** Você precisa importar o `CORSMiddleware` e configurá-lo no `app.add_middleware`, definindo `allow_origins=[“*”]` (para testes) ou apontando para a porta do seu React.
- **No Spring Boot (Java):** Você deve anotar o seu `@RestController` com `@CrossOrigin(origins = “*”)` ou configurar um filtro CORS global na classe de WebConfig.

Não perca horas tentando consertar o HTML: o erro de CORS é sempre resolvido liberando o cadeado no lado do Servidor.

4 Consumo de APIs de IA e Experiência do Usuário (UX)

Durante o consumo de uma API de inteligência artificial generativa, existe um ponto fulcral no desenvolvimento visual: o **Feedback Imediato**.

Diferente de sistemas de cadastro, que respondem em 10 milissegundos, sistemas de RAG podem demorar. Se o RAG precisar processar a query, buscar em 5 bancos vetoriais, formatar o contexto e aguardar os servidores lentos da OpenAI para gerar a resposta, a requisição pode demorar 10 ou 15 segundos. Se, durante esses intermináveis 15 segundos, a interface visual congelar e

não der nenhum retorno, o usuário instintivamente fechará a aba, concluindo que o seu sistema "travou".

Para evitar essa péssima impressão, invistam pesadamente em UX reativa:

- **Indicadores de Digitação (Spinners):** Ao apertar ENTER, desabilite o botão enviar, coloque um texto de "A IA está pensando..." ou ícones de carregamento (*loaders*) animados.
- **Streaming (Server-Sent Events):** Equipes avançadas podem configurar a API para retornar *chunks* da resposta à medida que o LLM vai gerando letra por letra (igualzinho ao modelo original do ChatGPT). Isso dá a impressão visual ao usuário de que a máquina está agindo rápido, anulando a percepção psicológica da demora global.

O final deste laboratório marca o glorioso momento onde as duas extremidades do sistema se encontram, o cérebro se liga ao rosto, e o seu produto finalmente acorda.

✔ Takeaways

- A interface é o **"Fator Uau"**: clientes não compram APIs — compram experiências visuais que resolvem problemas.
- **Streamlit** é a rota mais rápida (Python puro), **React/Next.js** é a rota profissional, e **HTML+JS vanilla** oferece controle total sem Node.js.
- **CORS** é o bloqueio #1 em integrações front ↔ back. É resolvido no lado do servidor (`CORSMiddleware` no FastAPI ou `@CrossOrigin` no Spring).
- **Feedback imediato** (spinners, "A IA está pensando...") é obrigatório — 15 segundos de tela em branco fazem o usuário fechar a aba.
- **Streaming via SSE** (Server-Sent Events) anula a percepção de demora ao exibir a resposta letra por letra, como o ChatGPT faz.

5 Conclusão

Com o cérebro (Backend) conectado ao rosto (Front-end), o produto **existe**. No próximo encontro, faremos a transição final: levar tudo para a **nuvem** (Cloud Deploy), para que qualquer pessoa no mundo possa acessar seu sistema via URL pública.

✎ Questões: Autoavaliação

1 **[Direta]**Compare Streamlit, React e HTML+JS vanilla como opções de front-end para um Hackathon. Em que cenário cada escolha é ótima?

2 **[Direta]**Explique o que é CORS, por que o navegador bloqueia requisições cross-origin, e como resolver o problema no FastAPI e no Spring Boot.

3 **[Direta]**Descreva duas técnicas de UX reativa (indicadores de digitação e streaming) e explique como cada uma melhora a percepção de velocidade do usuário.

4 **[Reflexiva]**Uma equipe gastou 4 horas tentando resolver um erro de CORS no HTML, sem sucesso. Seguindo a "Regra dos 20 Minutos", o que deveria ter acontecido? Qual é o custo real (em progresso do Hackathon) dessa obstinação?

5 **[Reflexiva]**Discuta a tensão entre "velocidade de entrega" (Streamlit) e "qualidade profissional" (React) em um Hackathon de 5 dias. Como uma equipe deve tomar essa decisão?

Lab. 28: Construindo a Experiência do Usuário (UX)

Capítulo em Revisão

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

⋆ Objetivo do Laboratório

O objetivo desta atividade é construir o Front-end do projeto e realizar a comunicação (*fetching*) com os *Endpoints* desenvolvidos no laboratório passado. O produto da *Startup* da equipe deve sair das telas de código e terminal e passar a operar visualmente em um Navegador Web.

6 Atividade Prática

1. Configuração do Servidor (CORS)

A primeira barreira ao separar o Front-end do Back-end no navegador é o *Cross-Origin Resource Sharing* (CORS). Certifique-se de que o seu servidor Python ou Java autorize conexões locais.

No FastAPI:

```
from fastapi.middleware.cors import CORSMiddleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=[‘‘*’’], # Atenção: Em produção limite isso para a URL do Front
    allow_credentials=True,
    allow_methods=[‘‘*’’],
    allow_headers=[‘‘*’’],
)
```

2. Escrevendo a Interface

Divida o trabalho na equipe. Um membro deve criar os componentes gráficos visuais (seja num arquivo `index.html` isolado, num projeto `React` ou numa aba `Streamlit`).

Concentre-se em construir:

- Um contêiner que simule uma tela de conversa.
- Um campo de entrada (*input text*) e botão de Enviar.

- Um indicador de “A IA está pensando...”.

3. O Fetch (Comunicação)

Faça a interligação lógica. O Front-end não raciocina, apenas espelha dados.

```
// Exemplo em JavaScript nativo atrelado a um botão ‘‘Enviar’’:
async function enviarMensagem() {
    const texto = document.getElementById(‘‘inputUsuario’’).value;
    mostrarNoChat(texto, ‘‘usuario’’);
    mostrarLoading();

    try {
        const resposta = await fetch(‘‘http://localhost:8000/api/chat’’, {
            method: ‘‘POST’’,
            headers: { ‘‘Content-Type’’: ‘‘application/json’’ },
            body: JSON.stringify({ mensagem: texto })
        });
        const dados = await resposta.json();
        removerLoading();
        mostrarNoChat(dados.resposta, ‘‘ia’’);
    } catch (erro) {
        alert(‘‘O servidor de IA está offline ou com problemas.’’);
    }
}
```

4. Testes de Usabilidade

Passe 15 minutos do final da aula tentando “quebrar” a própria interface. O que acontece se o usuário clicar no botão de enviar 10 vezes seguidas? A UI trata isso ou a IA responde 10 vezes? Corrijam esses pequenos “bugs” de usabilidade.

7 Critério de Avaliação

O laboratório 28 é validado com sucesso pela submissão do código visual e comprovação presencial ou via print da aplicação consumindo com sucesso o servidor Backend construído na aula 27, diretamente de uma página web.

✔ Takeaways

- O produto agora tem um **rosto**: front-end funcional consumindo o back-kend via `fetch()` — a integração Full-Stack está completa.
- O CORS foi resolvido no servidor (não no HTML) — `CORSMiddleware` (FastAPI) ou `@CrossOrigin` (Spring).
- O indicador "A IA está pensando..." melhora dramaticamente a UX — 15 segundos de tela em branco fazem o usuário fechar a aba.
- O teste de usabilidade (clicar 10x no botão) revelou bugs de UX que seriam fatais no Demoday.

◇ Informação

Gancho para Amanhã: O produto funciona **na sua máquina**. Mas "na minha máquina funciona" é inaceitável para uma startup. Amanhã vocês farão o **Deploy na Nuvem** — o produto estará acessível via URL pública para qualquer pessoa no mundo. É a reta final antes do Demoday!