Capítulo 27 – Hackathon do Projeto Final (Backend e RAG)

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

Com o planejamento arquitetural e o repositório configurados no Dia 1, é hora de sujar as mãos com código-fonte. O foco deste capítulo (e do respectivo encontro do Hackathon) é o desenvolvimento da camada invisível, porém vital, de qualquer sistema corporativo: o **Back-end**. As equipes devem se concentrar na engenharia de dados (ingestão e *chunking*), na configuração do banco de dados vetorial e na construção das rotas (APIs) que orquestrarão o modelo RAG. É aqui que o cérebro da sua aplicação ganha vida.

1 O Core da Inteligência: Saindo do Notebook

No segundo dia de Hackathon, o desafio imediato das equipes é transpor as provas de conceito (*PoCs*) soltas e os *scripts* de Jupyter Notebook para o miolo profissional do produto: o serviço de retaguarda (Backend).

Não importa o quão elegante e moderna seja a sua interface de usuário no futuro; se a inteligência que sustenta o produto "alucinar" constantemente, demorar um minuto para responder ou quebrar com erros de formatação JSON, o produto falhará no momento da apresentação. Portanto, todo o esforço desta etapa deve se concentrar em consolidar a infraestrutura (utilizando *FastAPI* no ecossistema Python ou *Spring Boot* no ecossistema Java).

• Teoria: title=A Arquitetura de Microserviço

O seu Backend não deve ser um monólito misturado com telas. Ele deve atuar como um **Microserviço Autônomo**. Sua aplicação Back-end exporá *Endpoints* HTTP (ex: `POST /api/chat`) que receberão mensagens do usuário em formato JSON, processarão a lógica de IA pesada, consultarão os bancos de dados, e retornarão a resposta processada limpa. Isso permite que amanhã você conecte uma interface Web, um aplicativo Android, ou um bot de WhatsApp na mesma API, sem reescrever a inteligência.

2 Desenvolvimento da Lógica Interna e RAG

As principais missões de engenharia para este estágio do projeto envolvem três frentes de batalha que podem (e devem) ser atacadas em paralelo pelos membros da equipe:

1. **Ingestão de Dados e Vector Store:** Se a aplicação da sua startup promete ser um "Assistente Especialista em CLT", é mandatório construir um *pipeline* de ingestão forte.
   - *Ação:* Desenvolver rotinas que leiam os PDFs/Textos originais, fatia-os inteligentemente preservando parágrafos inteiros (*Chunking* semântico), convertam esse texto em Embeddings (usando a API da OpenAI ou modelos HuggingFace locais) e gravem no banco (ex: ChromaDB ou Pinecone).
2. **Criação das Rotas (Endpoints REST):** O esqueleto da comunicação.
   - *Ação:* O responsável por integração deve erguer os controladores REST. No mínimo, vocês precisarão de uma rota `/chat` (para enviar perguntas e receber respostas) e possivelmente uma rota `/upload` (se o sistema permitir que o usuário suba um PDF próprio dinamicamente para análise).
3. **Montagem do Chain RAG / Agente:** O cérebro em si.
   - *Ação:* Isolar a lógica do LangChain (LCEL) ou Spring AI. O fluxo deve: capturar a *Query* do usuário → vetorizá-la → buscar os top-K documentos no banco → empacotar tudo no Prompt de Sistema → acionar o LLM → formatar a saída.

3 Cuidado com Dependências, Erros e a Regra dos 20 Minutos

O maior inimigo dos Hackathons não é a falta de lógica da equipe, mas as armadilhas de configuração.

Equipes utilizando Python costumam esbarrar frequentemente em conflitos de versão de bibliotecas (o LangChain é atualizado semanalmente, quebrando códigos antigos) ou erros bizarros de compilação de pacotes em C++ associados a bibliotecas de vetorização como o FAISS ou Chroma. Já as equipes utilizando o ecossistema Java enfrentam barreiras de injeção de dependências estritas, configuração extensa de *Beans* e formatações exatas no arquivo `application.yml`.

► Prática: title=A Regra do Desbloqueio (Regra dos 20 Minutos)

Durante a construção da empresa, o ego e a obstinação devem dar lugar à agilidade. Se um problema técnico, um *bug* obscuro ou uma configuração de banco de dados não for resolvido pela equipe em **20 minutos cronometrados** de tentativa e leitura de documentação: levante a mão imediatamente.

O papel do Professor/Tech Lead não é fazer o código por vocês, mas é destrancar "nós cegos" estruturais (como configurações de CORS, bloqueios de firewall, e erros de porta local) que travam o progresso produtivo da equipe por horas a fio.

Tabela 1: Checklist de Sanidade do Backend (RAG)

Componente	O que deve fazer?	Sintoma de Falha
Document Loader	Extrair texto legível de PDFs corporativos e sites.	O modelo RAG diz que não encontrou a informação, pois o PDF estava escaneado como imagem.
Text Splitter	Fatiar o texto em "Chunks" de ∼500 a 1000 tokens com *overlap*.	Respostas do LLM cortadas pela metade ou falta de contexto completo nas respostas.
Vector Store	Indexar os chunks e realizar busca rápida por similaridade de cosseno.	Erro de porta de conexão (ex: 8000) no ChromaDB ou demora superior a 20s para buscar.
API Endpoint	Receber o JSON do usuário via HTTP POST e devolver a string final.	Erro de CORS ou `422 Unprocessable Entity` no Postman.

A arquitetura do Backend deve ser testada massivamente pelo *Postman* ou *Swagger* e dada como consolidada ao final desta etapa. O Cérebro está pronto. No próximo encontro, daremos a ele um rosto.

✔ Takeaways

- O Dia 2 é sobre **infraestrutura invisível**: ingestão de dados, Vector Store e rotas REST — sem isso, não há produto.
- O Backend deve ser um **microserviço autônomo** com endpoints HTTP (ex: `POST /api/chat`) — isso permite conectar qualquer interface (web, mobile, WhatsApp) ao mesmo cérebro.
- O pipeline de ingestão (PDF → Chunking → Embedding → Vector Store) é a espinha dorsal do RAG; chunking ruim = respostas ruins.
- A **Regra dos 20 Minutos** evita perda de tempo: se um bug não é resolvido em 20 min, peça ajuda ao Tech Lead imediatamente.
- O checklist de sanidade (Document Loader, Text Splitter, Vector Store, API Endpoint) deve ser validado com Postman antes de avançar para o front-end.

4 Conclusão

Com o cérebro (Backend + RAG) validado via Postman, a equipe está pronta para o Dia 3: dar ao sistema um **rosto** — a interface de usuário que transforma chamadas HTTP em uma experiência visual interativa.

✎ Questões: Autoavaliação

1 **[Direta]**Explique por que o Backend deve ser projetado como um microserviço autônomo. Que vantagem isso traz para a escalabilidade futura do produto?

2 **[Direta]**Descreva o pipeline completo de ingestão de dados: quais componentes são necessários para ir de um PDF bruto até chunks indexados no Vector Store?

3 **[Direta]**O que é a “Regra dos 20 Minutos” e por que ela é crítica em Hackathons? Dê um exemplo de bug que justifica pedir ajuda imediatamente.

4 **[Reflexiva]**Um PDF escaneado (imagem, sem texto extraível) é ingerido pelo pipeline. O Document Loader extrai “lixo”. Proponha uma solução técnica para esse problema usando OCR e discuta os trade-offs de qualidade vs. tempo.

5 **[Reflexiva]**O chunking por contagem de tokens pode cortar uma tabela de dados pela metade, destruindo a semântica. Proponha estratégias alternativas de chunking para documentos que contêm tabelas e listas.

Lab. 27: Implementação do Core (IA e RAG)

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 é que o Backend se torne completamente operacional na camada de IA, recebendo os dados do contexto (seja por um banco local como Chroma ou serviços de nuvem), passando a requisição do usuário pelos *Prompts* definidos pela equipe e retornando respostas via JSON.

5 Atividade Prática

1. Codificação do Core

O time deve se organizar para que a lógica essencial seja submetida. Algumas das tarefas que devem constar no histórico de versão ao final do laboratório incluem:

1. **Carga Base:** *Scripts* utilitários inseridos que convertem a base de dados em embeddings para o repositório vetorial de escolha da equipe.
2. **O Prompt Matriz:** A criação das classes/funções encarregadas do *System Prompt*. É aqui que as regras de comportamento, tom de voz (Persona) e a blindagem contra injeções (*Guardrails*) precisam ser parametrizadas.
3. **Endpoints:** O esqueleto do Laboratório 26 deve ser preenchido, de modo que chamadas HTTP POST para `/chat` ou análogas respondam não mais *strings* soltas, mas sim objetos estruturados provindos do fluxo LangChain/Spring AI.

2. Integração e Revisão

O Hackathon simula o fluxo do mercado. Se o "Aluno A" escreveu a *Pipeline* de PDF e o "Aluno B" fez a Rota HTTP, seus códigos precisam se unir:

- Utilizem *Merge Requests (MR)* ou *Pull Requests (PR)*.
- Revisem o código do parceiro para garantir que ele atende o combinado antes de injetar na *branch main*.

3. Testes via Insomnia / Postman

Para garantir que a meta do dia foi cumprida, vocês devem conseguir enviar um JSON complexo contendo o histórico de mensagens simulado e validar que o servidor Python/Java está conseguindo realizar o RAG (Recuperação) interno e retornando o que é pedido.

6 Critério de Avaliação

O laboratório 27 não possui entrega unitária avulsa. A avaliação será computada pelo andamento do repositório no GitLab. O professor validará a densidade do código e a evolução técnica no final da aula de Hackathon, observando a consolidação das funcionalidades do Backend.

✔ Takeaways

- O **Backend está operacional**: ingestão de dados, VectorStore populado, endpoints respondendo perguntas via RAG.
- O **Prompt Matriz** (System Prompt) define persona, tom, guardrails e fontes — é o documento de engenharia que controla a IA.
- Os **Merge Requests** simulam o workflow corporativo real: código revisado antes de entrar na main.
- A validação via Postman/Insomnia comprova que o cérebro funciona independentemente de qualquer interface visual.

◇ Informação

Gancho para Amanhã: O cérebro está pronto e validado via Postman. Amanhã vocês construirão o **rosto** — a interface web que transforma chamadas HTTP em uma experiência visual interativa. O objetivo do Lab 28 é que o produto tenha uma tela funcional consumindo o backend.