Quando recebo um desafio técnico, meu primeiro impulso não é sair codando, mas entender profundamente o problema que estou tentando resolver. No projeto do Coco Bambu, por exemplo, meu primeiro passo foi interpretar a estrutura dos dados operacionais do restaurante — principalmente o endpoint getGuestChecks, que traz informações essenciais como comandas, itens vendidos, impostos e valores pagos.

A partir disso, começo a construir um modelo mental das entidades e seus relacionamentos. Vejo os dados como representações do mundo real e tento organizá-los da forma mais lógica e robusta possível. Neste caso, percebi rapidamente que se tratava de uma estrutura altamente relacional, então optei por um modelo de banco de dados normalizado. Tive uma preocupação especial com campos aninhados e opcionais, como discount, serviceCharge e menuItem, que poderiam gerar inconsistência e dificultar análises se tratados de maneira direta. Minha solução foi separar essas estruturas em tabelas auxiliares com chaves bem definidas, garantindo integridade e escalabilidade.

Pensando no crescimento da operação e na possibilidade de múltiplas lojas/franquias, planejei uma arquitetura de dados baseada em um Data Lake particionado por data, loja e tipo de dado (endpoint). Desenvolvi um script de ingestão (ingestao_apis.py) que salva as respostas das APIs em uma estrutura organizada e compactada, garantindo versionamento implícito e histórico para reprocessamentos futuros. Além disso, escrevi uma DAG no Airflow (dag_ingestao_receita.py) para permitir a automação desse processo, pensando em integração com pipelines reais.

Minha forma de trabalhar é sempre incremental e testável. Antes de escalar qualquer solução, gosto de validar com um conjunto pequeno de dados reais. Para isso, usei o Jupyter Notebook (CocoBambu.ipynb) como ambiente de experimentação: carreguei o JSON, transformei em DataFrames, criei um banco SQLite em memória e comecei a validar as queries que trariam valor para o negócio — como ticket médio, itens mais vendidos, impostos aplicados por comanda, entre outras.

Também tive uma preocupação muito clara com mudanças de schema na API. Sei que sistemas externos mudam e, se isso não for previsto, pode quebrar tudo em produção. Por isso, implementei fallback de campos com lógica condicional no código (guest.get("taxation") or guest.get("taxes")) e pensei em um mecanismo de versionamento de schema, permitindo manter a rastreabilidade dos dados mesmo com mudanças ao longo do tempo.

No fim das contas, o que guia meu raciocínio é a criação de soluções escaláveis, automatizadas e resilientes. Evito ao máximo “gambiarras” que funcionam hoje, mas quebram amanhã. Minha missão é transformar dados brutos em estruturas úteis, acessíveis e preparadas para o crescimento, sempre com foco em reutilização, governança e clareza. Gosto de pensar em como as coisas vão funcionar quando eu não estiver mais por perto, e é isso que define minha forma de pensar, desenvolver soluções e resolver problemas.