# Projeto tabela FIPE-consultas

**Alexsandro Pereira**

11/02/2026 — Entrega final

## Sumário



Eu sou o time inteiro e preciso agir como empresa: **planejar → documentar → prototipar → codar → testar → apresentar**.

**Escopo mínimo implementável**

- Interface de consulta (usuário final), estilo FIPE:
  - Dropdown 1: **Marca**
  - Dropdown 2: **Modelo**
  - Dropdown 3: **Ano-modelo / versão**
  - Botão **Consultar**
  - Resultado: **preço + mês de referência**
- Log simples de consulta (registrar que houve consulta).
- Regra de fallback: **se não existir cotação no mês escolhido, retornar o último mês disponível**.

**Fora do escopo de código (apenas projetado/documentado):** Admin / Gerente / Coordenador / Pesquisador / Lojista / Batch mensal (fechamento).

# 1 1) Planning da semana

## 1.1 1.1 Board

Figura abaixo: visão do Jira (*SEM3 board*) com tarefas de planejamento concluídas e rastreabilidade do trabalho.

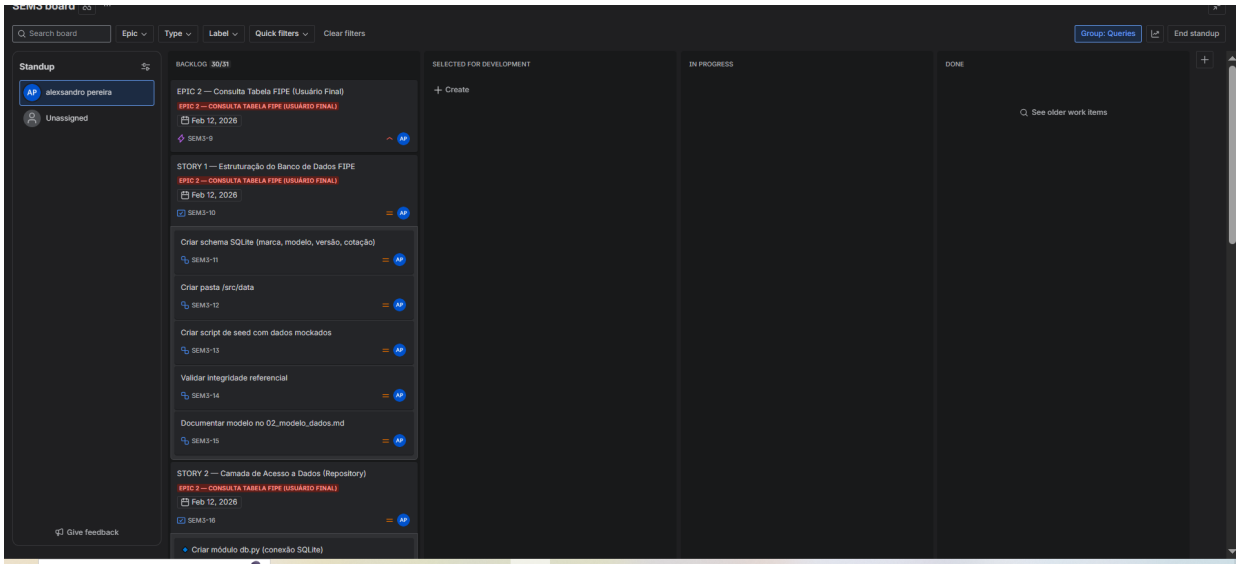


Figura 1: Jira SEM3 — organização da planning.

## 1.2 1.2 Board da semana

A semana é planejada para maximizar **clareza de arquitetura** + **entrega mínima funcional** dentro do tempo.

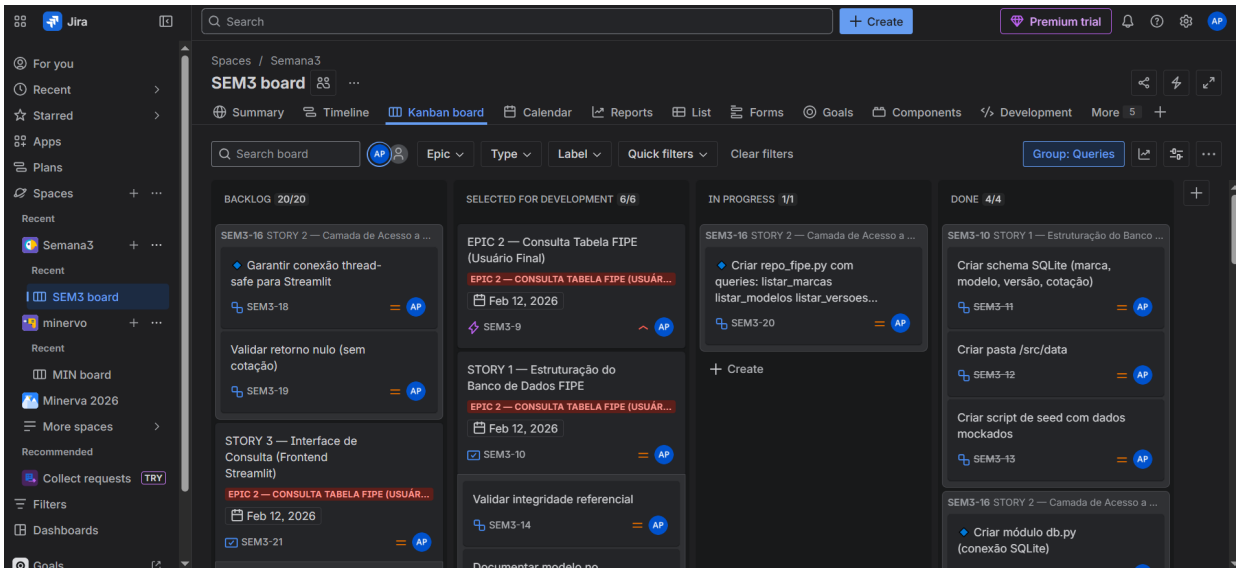


Figura 2: Plano semanal (macro)

## 1.3 6. Visão do Sistema Completo (Projeto Global)

Embora apenas o módulo de **consulta pública** tenha sido implementado, o sistema foi projetado considerando o fluxo completo de geração da Tabela FIPE, incluindo papéis operacionais e processamento mensal (batch). O objetivo é separar claramente **entidades do domínio**, **papéis (roles)**, **processos** e **responsabilidades**.

### 1.3.1 6.3 Entidades Projetadas

As entidades abaixo fazem parte do desenho global, mas não foram implementadas por restrição de escopo:

- `usuario` (com `role` para admin/gerente/coordenador/pesquisador)
- `regiao`, `loja`
- `coleta_preco` (registro bruto)
- `batch_execucao` (processamento mensal)

### 1.3.2 6.1 Papéis do Sistema (Role-Based)

Os papéis são **atributos comportamentais do usuário** (roles), e **não entidades independentes**. Assim, o sistema utiliza uma única entidade `usuario` e diferencia ações por `role`.

- **Usuário Final**: consulta valores consolidados.
- **Usuário Administrador (`role=admin`)**: gerencia usuários e permissões.
- **Usuário Gerente de Catálogo**: cadastra marca/modelo/versão.
- **Usuário Coordenador Regional**: organiza regiões, lojas e atribuições.
- **Usuário Pesquisador**: registra preços coletados em lojas.

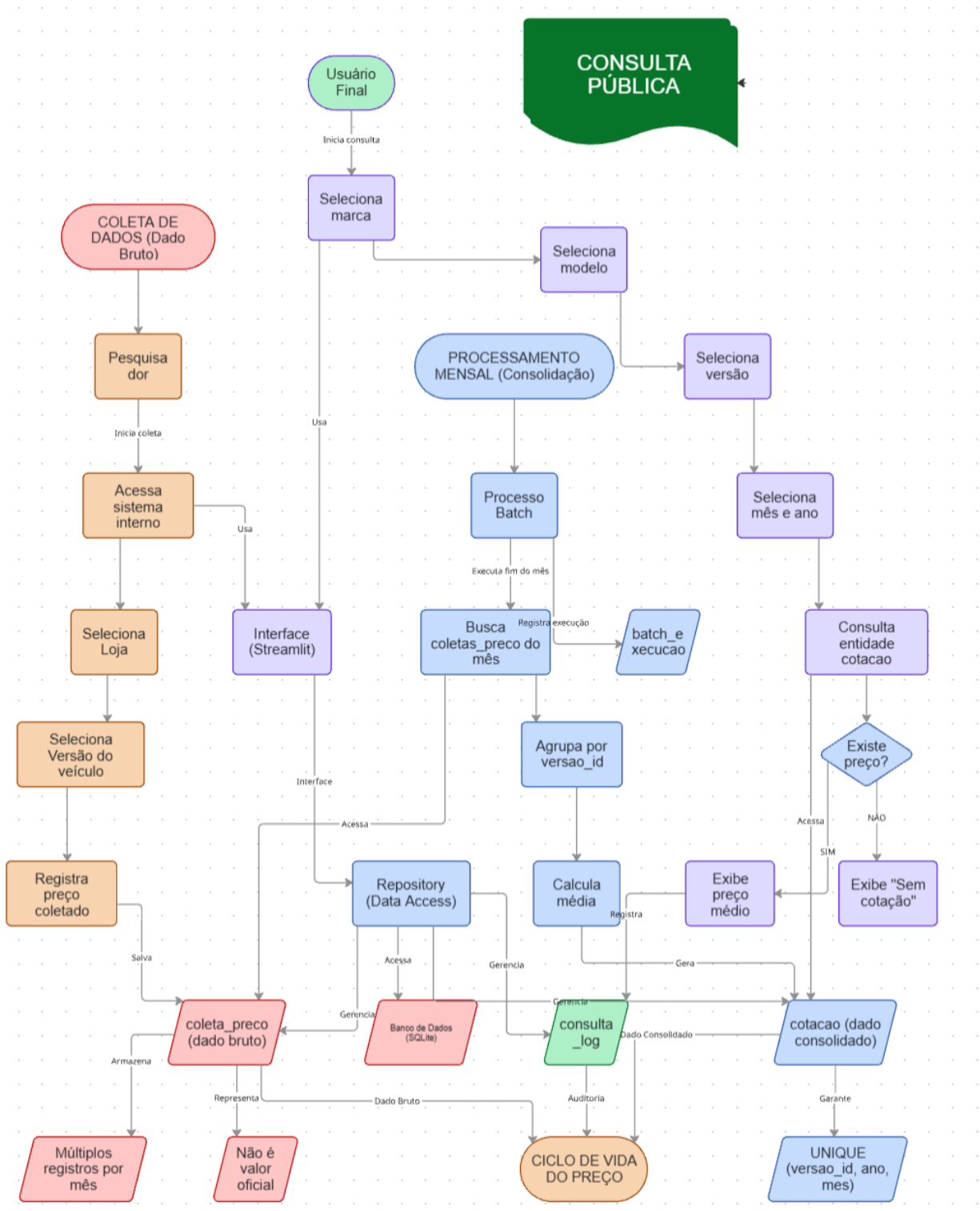


Figura 3: Enter Caption

# 2 Descrição do Fluxograma

## 2.1 Consulta Pública

O sistema separa claramente o **Usuário Final (consulta pública)** dos fluxos internos de coleta e processamento. A interface pública permite exclusivamente a leitura de dados

consolidados, sem acesso a dados brutos ou funcionalidades administrativas.

## 2.2 2. Ciclo de Coleta (Dados Brutos)

O **Pesquisador** alimenta o sistema com preços coletados em campo (veículo por veículo). Esses registros são armazenados como *dados brutos* na entidade `coleta_preco` e não representam valores oficiais.

Podem existir múltiplos registros para a mesma versão dentro de um mesmo mês, refletindo a variação real de mercado.

## 2.3 3. Governança e Validação

O sistema prevê papéis distintos com responsabilidades bem definidas:

- **Administrador**: Gerencia usuários e configurações globais.
- **Gerente**: Mantém o cadastro mestre (Marca, Modelo e Versão).
- **Coordenador**: Organiza regiões e supervisiona a coleta.
- **Pesquisador**: Executa a coleta de preços nas lojas.

Essa estrutura garante que os dados consolidados derivem de um domínio organizado e controlado.

## 2.4 4. Cálculo do Preço (Batch Mensal)

Ao final de cada mês, o sistema executa um **processo batch de consolidação**.

Esse processo:

1. Busca todos os registros brutos do período.
2. Agrupa por `versao_id`.
3. Calcula a média mensal.
4. Persiste o resultado na tabela `cotacao`.

A tabela consolidada possui a restrição:

```
UNIQUE (versao_id, ano, mes)
```

Garantindo apenas uma cotação oficial por versão em cada período.

## 2.5 5. Consulta de Cotação

Na área pública, o usuário realiza a seguinte sequência:

1. Seleciona marca
2. Seleciona modelo
3. Seleciona versão
4. Seleciona mês e ano

O sistema consulta exclusivamente a tabela `cotacao` (dados consolidados).

- Se existir registro: exibe o preço médio.
- Caso contrário: exibe a mensagem “Sem cotação”.

## 2.6 6. Registro e Análise de Uso

Cada consulta bem-sucedida gera um registro na tabela `consulta_log`, armazenando:

- Data e hora
- Marca
- Modelo
- Versão
- Período consultado

Esse mecanismo permite análises futuras de uso e comportamento, fechando o ciclo de vida do dado dentro do sistema.

## 2.7 7. Modelo Conceitual Completo (ER)

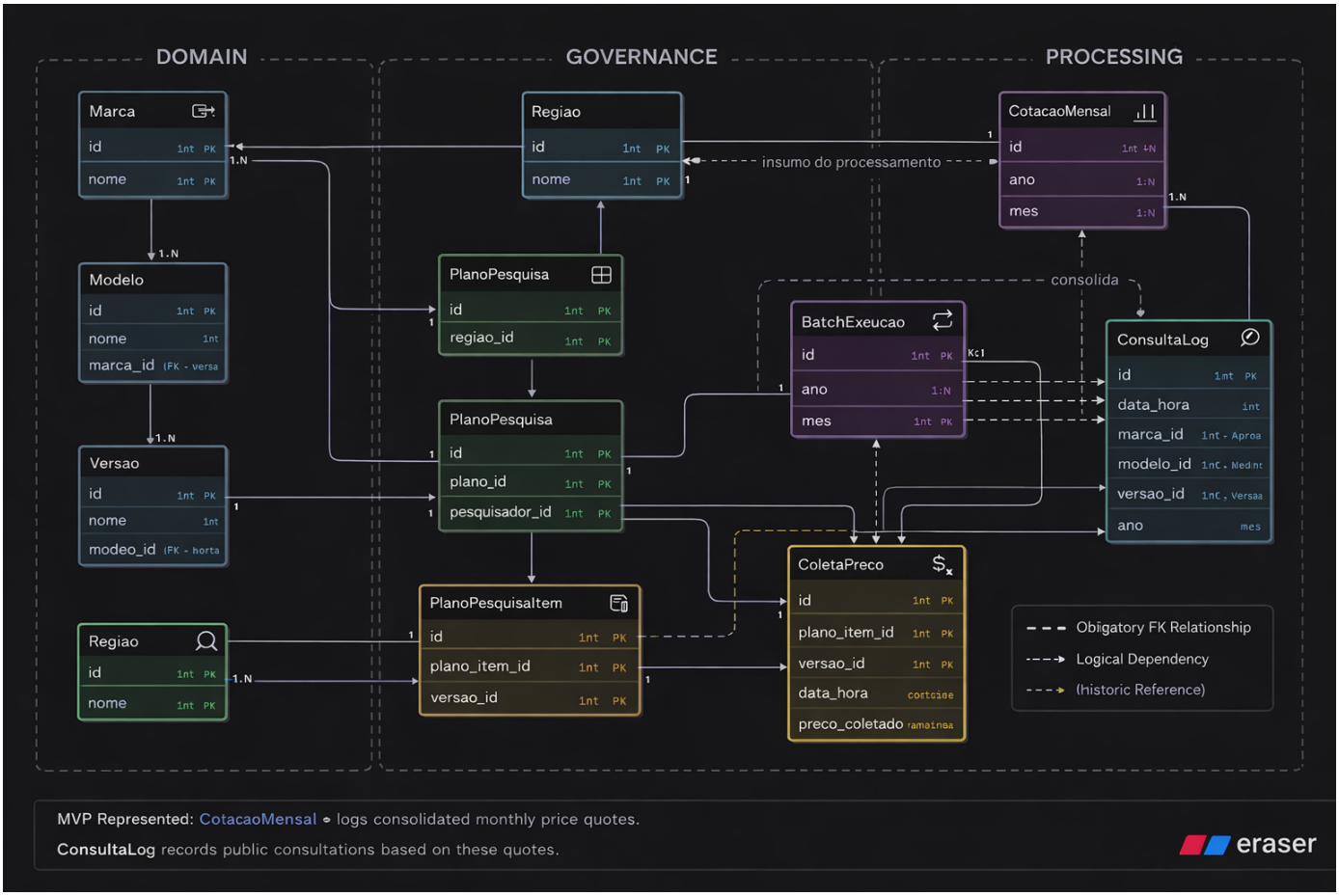


Figura 4: ERD completo (projetado) com foco no **MVP** e regra de fallback.

## 2.8 Descrição do Diagrama ER Completo

O diagrama apresenta a modelagem conceitual completa do sistema FIPE-like, organizado em quatro domínios principais: **Master Data**, **Governança**, **Processamento Mensal** e **Consulta Pública**.

**1. Master Data:** Representa a hierarquia estrutural dos veículos, composta por *Marca*, *Modelo* e *Versao*, em relações 1:N sucessivas. Essa camada define a identidade estável do domínio.

**2. Governança e Planejamento:** A entidade *Regiao* estrutura a atuação territorial. Coordenadores, Pesquisadores e Lojas estão vinculados a regiões específicas. O *PlanoPesquisa* organiza ciclos semanais de coleta, detalhados por *PlanoPesquisaItem*, que define quais lojas devem ser visitadas por cada pesquisador.

**3. Coleta e Processamento (Pipeline Analítico):** Os registros brutos são armazenados em *ColetaPreco*. Mensalmente, um processo representado por *BatchExecucao* consolida esses dados na entidade *CotacaoMensal*, garantindo unicidade por meio da restrição `UNIQUE(versao, ano, mes)`.

**4. Consulta Pública e Auditoria:** A entidade *ConsultaLog* registra consultas realizadas na interface pública, referenciando a cotação consolidada utilizada na exibição.

As entidades destacadas visualmente correspondem ao escopo MVP implementado nesta etapa do projeto.

## 2.9 8.1 Arquitetura do Software (Camadas e Componentes)

A arquitetura adotada é um **monolito organizado em camadas**, adequada ao escopo e ao tempo de implementação, preservando separação de responsabilidades.

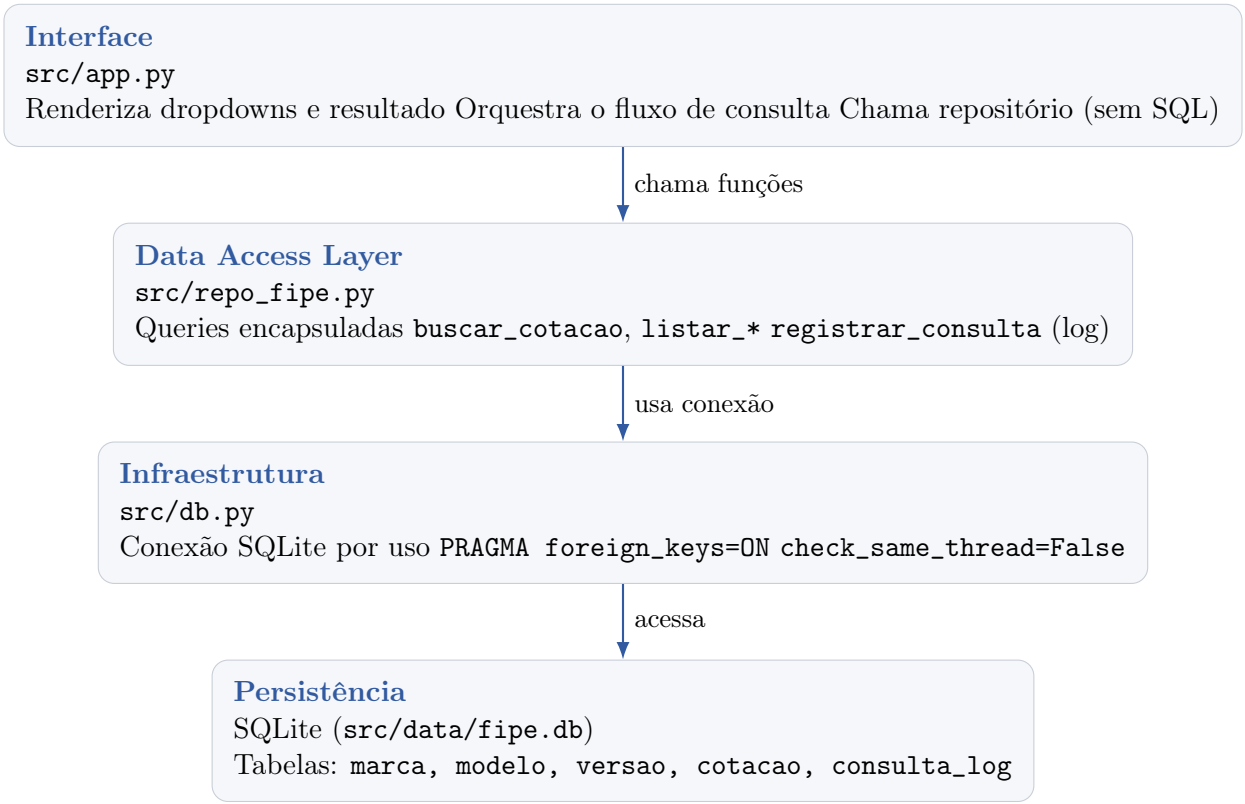


Figura 5: Arquitetura em camadas do módulo implementado (consulta pública).

## 2.10 1.3 Entregas e DoD (Definition of Done)

**Entrega 2:**

- Documento com: Planning + Modelo de Dados + Modelo de Componentes (TDD resumido).

**DoD do MVP :**

- Tela Streamlit com 3 dropdowns em cascata + botão Consultar.
- Retorna preço + mês de referência.
- Aplica fallback do último mês disponível quando faltar cotação no mês selecionado.
- Registra log simples da consulta.
- Código organizado em camadas (UI / Service / Repository).

# 3 Consulta de Cotação

## 3.1 Decisões explícitas (Mini ADRs)

### ADR-001 — Persistência

**Decisão:** usar **SQLite** com **seed local**.
**Motivo:** velocidade de implementação + simplicidade + rastreabilidade do dado.

**Consequência:** dados reais de operação não existem; simulação via seed.

## ADR-002 — Arquitetura

**Decisão:** monólito em camadas (**UI → Service → Repository**).
**Motivo:** fácil de avaliar e evoluir (troca regra sem mexer na UI; troca BD sem quebrar regra).
**Consequência:** sem microserviços; foco na clareza.

## ADR-003 — Regra de consulta com fallback

**Decisão:** se mês selecionado não tiver cotação, retornar **último mês disponível** para a mesma versão/ano.
**Motivo:** comportamento realista e robusto (dados podem estar incompletos).
**Consequência:** precisa de query por "máximo mês disponível".

## 3.2 5.3 Sequência do fluxo (consulta)

1. Usuário escolhe Marca → Modelo → Versão/Ano e um mês.
2. UI chama `QueryService.consultar()`.
3. Service tenta `Repo.get_cotacao()`.
4. Se não existir: Service chama `Repo.get_ultima_cotacao()` e retorna o último mês disponível.
5. Service registra `ConsultaLog`.
6. UI exibe preço + mês retornado.

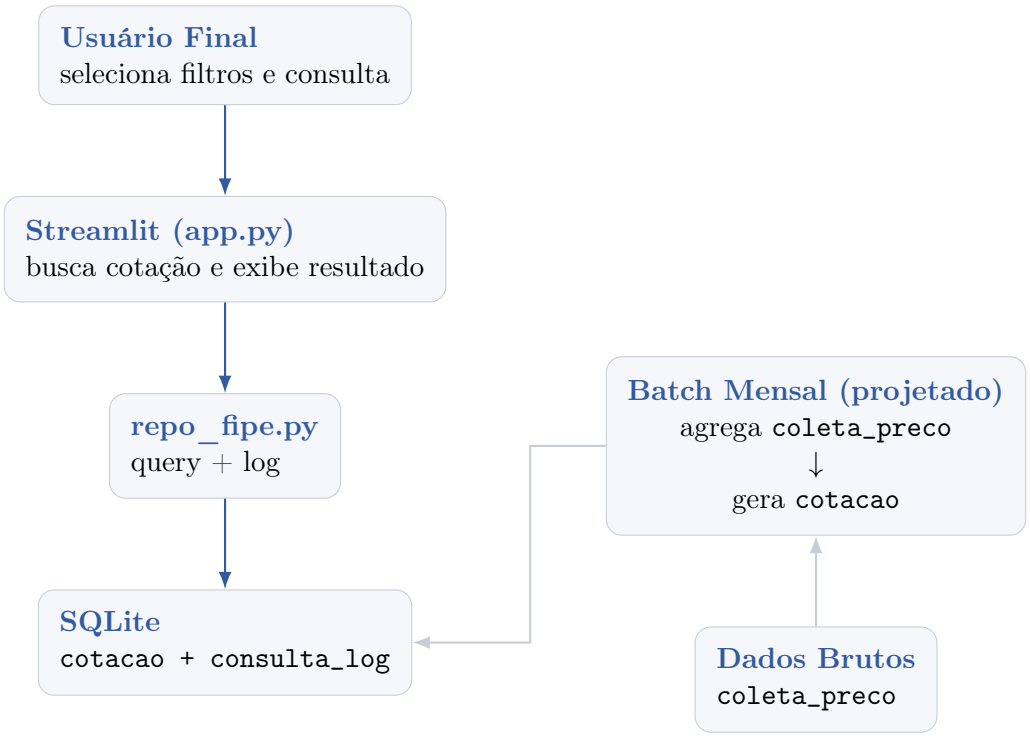


Figura 6: Fluxo síncrono de consulta (implementado) e processamento batch mensal (projetado).

# 4 Rastreabilidade mínima (Requisito → Entidade → Componente)

| Requisito | Entidades (dados) | Componentes (código) |
|---|---|---|
| Consulta FIPE (MVP) | Marca, Modelo, VersaoAno, CotacaoMensal | UI(Streamlit), QueryService, SQLiteRepo |
| Fallbackúltimo mês | CotacaoMensal | QueryService (regra) + SQLiteRepo (query ordenada por mês) |
| Log simples de consulta | ConsultaLog | QueryService (orquestra) + SQLiteRepo (insert log) |

Tabela 1: Rastreabilidade do MVP (evidência de pensamento de liderança técnica).

# 5 Modelo de Dados(ERD )

**MVP Entidades do MVP:** Marca, Modelo, VersaoAno, CotacaoMensal, ConsultaLog.

==========

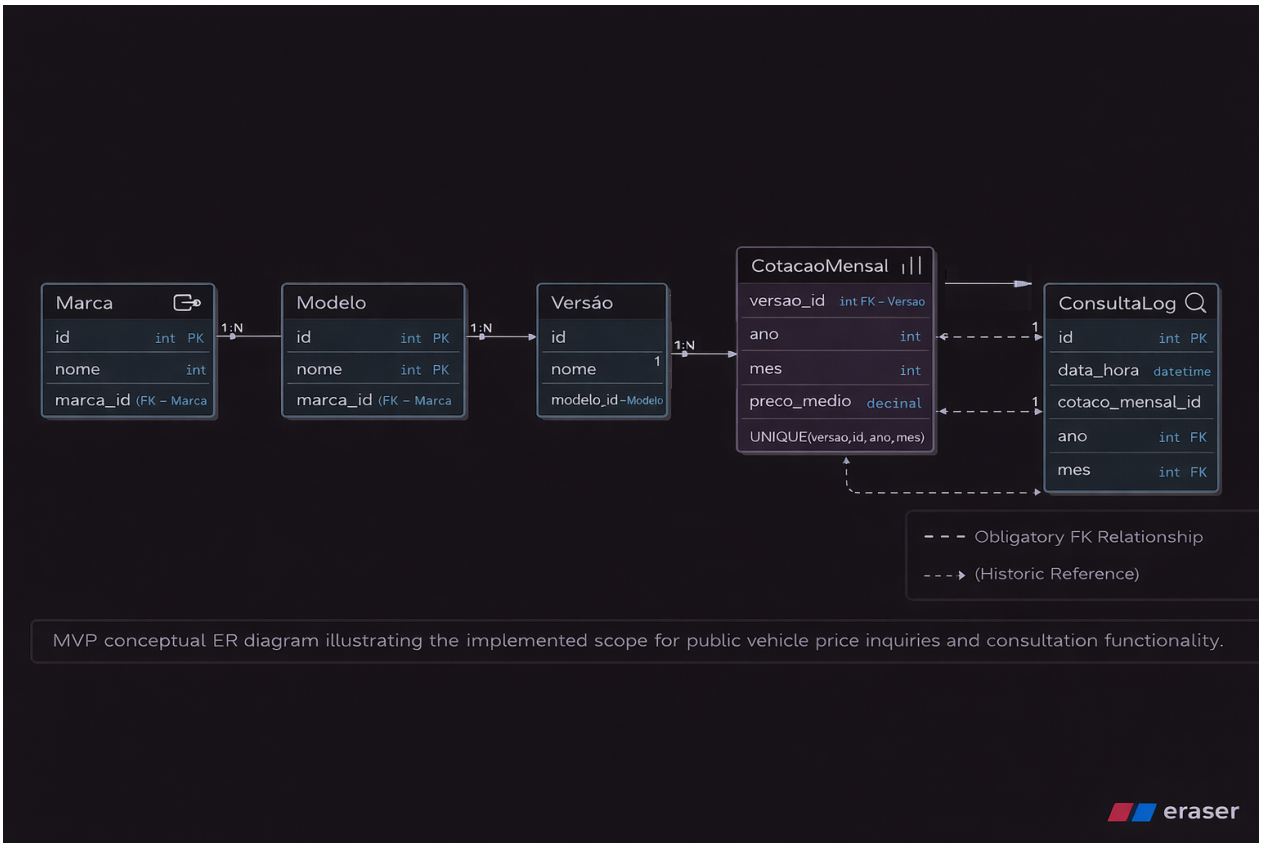


Figura 7: Modelo de Dados(ERD)

# 6 Modelo de Componentes (TDD resumido)

## 6.1 Camadas e responsabilidades

- **UI (Streamlit)**: captura entradas e renderiza saída. Sem regra de negócio.
- **Service (Domínio)**: valida seleção, aplica fallback do mês, orquestra log.
- **Repository (SQLite)**: queries e persistência (cotação e log).

## 6.2 5.2 Contratos (interfaces) — versão mínima

- **QueryService**
  - `list_marcas()`
  - `list_modelos(marca_id)`
  - `list_versoes(modelo_id)`
  - `consultar(versao_id, mes_solicitado) → (preco, mes_retornado)`
- **SQLiteRepo**
  - `get_cotacao(versao_id, mes)`
  - `get_ultima_cotacao(versao_id, mes_limite)`
  - `insert_log(...)`

==========

# 7 Critérios de aceite do MVP (3 bullets)

- Dropdowns em cascata (marca → modelo → versão/ano) com estado consistente.
- Consulta retorna **preço + mês de referência**.
- Na ausência do mês solicitado, retorna **último mês disponível** e registra log.

# 8 Plano de teste mínimo

## Testes essenciais (manual + 1 automatizado)

- **Happy path**: existe cotação no mês escolhido → retorna preço correto e log gravado.
- **Fallback**: mês solicitado sem cotação → retorna último mês disponível + log com mes_retornado.
- **Dados incompletos**: marca sem modelos (ou modelo sem versões) → UI não quebra, mostra mensagem clara.
- **Automatizado (unit)**: teste do Service para fallback (mock do repo).

# 9 Prototipo

## 9.1 Arvore do codigo

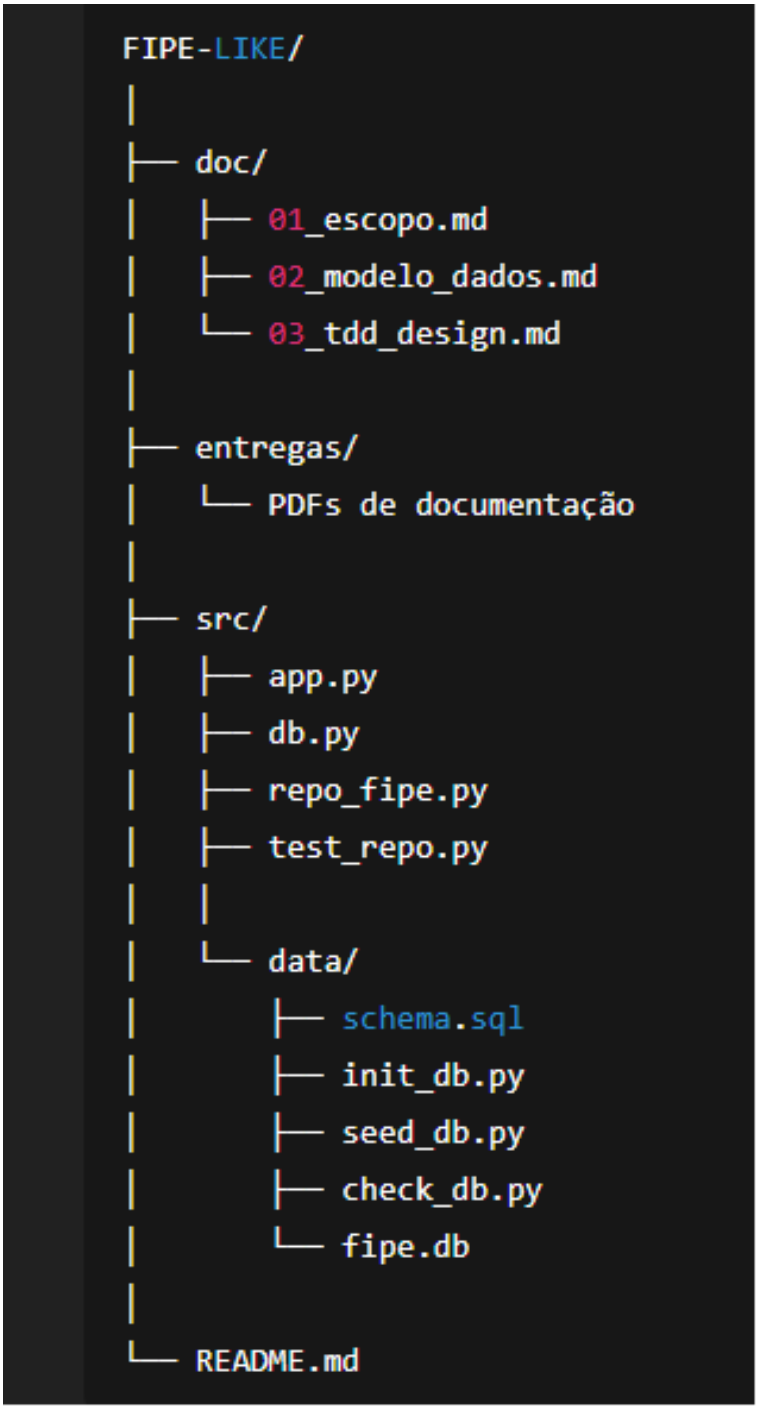


Figura 8: Esqueleto consulta

---

*Entrega 2 : clareza + organização + MVP consultável e robusto.*

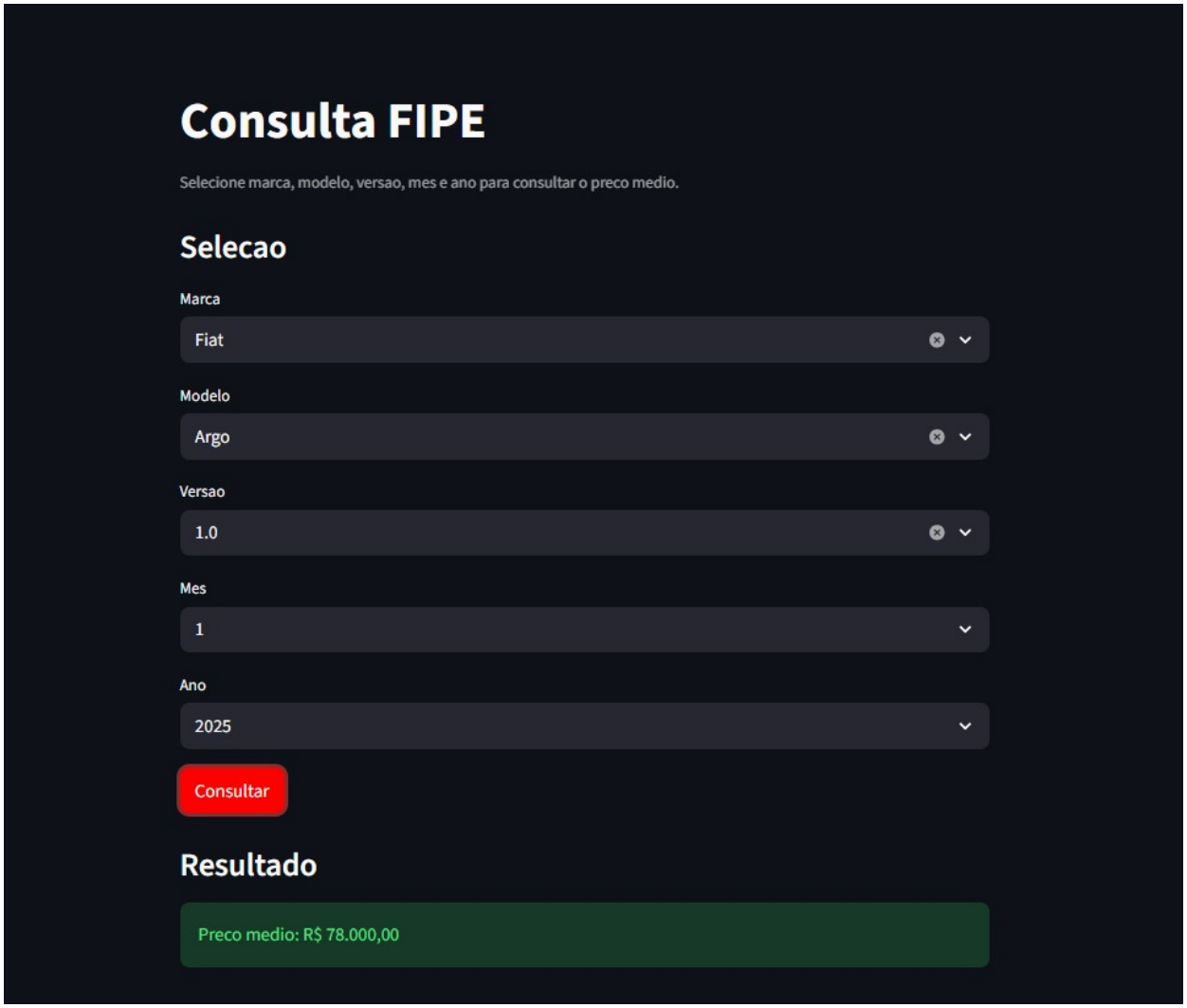


Figura 9: tela de consult

# 10 Validação e Testes

A estratégia de testes adotada no projeto FIPE-like, organizando as atividades em duas categorias principais: **Testes Automáticos** (estruturados e repetíveis) e **Testes Manuais** (validação de uso e comportamentos de interface). Além disso, apresenta a integração contínua e automação de qualidade associada ao repositório.

## 10.1 Testes Automáticos

Os testes automáticos foram desenvolvidos utilizando scripts Python que verificam o funcionamento de componentes críticos da aplicação. Eles são projetados para serem executados de forma repetível, garantindo que mudanças futuras não introduzam regressões.

### 10.1.1 Estrutura de Testes com `test_repo.py`

O arquivo `src/test_repo.py` contém uma série de validações que cobrem as funções da camada de repositório, assegurando que os métodos retornam dados corretos em diferentes cenários.

Os testes implementados incluem:

- **listar_marcas()**: Verifica se a lista de marcas retornada corresponde aos dados de seed.
- **listar_modelos(marca_id)**: Confirma que apenas os modelos vinculados à marca especificada são retornados.
- **listar_versoes(modelo_id)**: Valida o retorno correto de versões para um modelo dado.
- **buscar_cotacao(versao_id, mes, ano)**: Confirma dois cenários:
  1. Retorno de valor monetário quando há cotação.
  2. Retorno de `None` quando não há cotação no período.
- **Registro de Consulta (`consulta_log`)**: Valida se:
  - O log é incremetado em consultas bem-sucedidas.
  - O log não é gravado quando não há cotação.

Os testes garantem que as funções de acesso à base de dados:

- Não lançam exceções inesperadas;
- Tratam corretamente as conexões (abrem e fecham via `get_connection()`);
- Não realizam SQL diretamente no frontend.

### 10.1.2 Execução Automatizada

A base de testes foi projetada para ser executada com um simples:

```
python src/test_repo.py
```

Este comando valida o conjunto de métodos do repositório, retornando saídas esperadas de acordo com os dados de teste populados via script de seed.

### 10.1.3 Cobertura de Integridade Referencial

Inclui validação automática de integridade referencial após a configuração do banco:

- `PRAGMA foreign_keys = ON` habilitado globalmente;
- Inserções inválidas são bloqueadas com `IntegrityError`;
- Exclusões condicionais são corretamente restringidas.

—

## 10.2 Testes Manuais

Os Testes Manuais complementam os testes automáticos por cobrir interações de interface, fluxos de usuário e experiência visual. Eles foram realizados através da execução da aplicação Streamlit em ambiente de desenvolvimento.

### 10.2.1 Fluxo de Consulta Pública

Os seguintes cenários foram verificados manualmente:

1. **Fluxo Completo Feliz:** Marca, modelo e versão foram selecionados corretamente, com retorno de cotação válida.
2. **Consulta sem Cotação:** Cenário onde não existe cotação para o período gerou a mensagem apropriada ao usuário.
3. **Encadeamento de Dropdowns:** Os dropdowns de marca, modelo e versão atualizam sequencialmente de forma correta.
4. **Validação de Seleções Incompletas:** Tentativas de consulta sem seleção completa são bloqueadas ou não executadas.
5. **Exibição de Resultados:** Preço médio devidamente formatado em moeda (R$), ou mensagem "Sem cotação para o período".

### 10.2.2 Regra de Negócio Log

A associação entre exibição de preço e registro de consulta foi verificada conforme esperado:

- Consultas válidas geram entradas em `consulta_log`;
- Falhas ou inexistência de cotação não geram logs;
- A interface não trava caso o registro do log falhe, respeitando resiliência de UX.

—

## 10.3 Integração Contínua (CI/CD) — GitHub Actions

Para assegurar que os testes automáticos sejam executados a cada alteração do código, foi proposta a configuração de um pipeline de integração contínua com GitHub Actions.

### 10.3.1 Objetivos do Pipeline

O pipeline deve:

- Instalar dependências (Python, SQLite);
- Executar os testes automatizados;

- Validar integridade referencial;
- Reportar falhas antes de qualquer merge para as branches principais.

### 10.3.2 Workflow YAML Proposto

A seguir está um exemplo simplificado do workflow sugerido:

```
name: Python CI

on:
  push:
    branches: [ staging, main ]
  pull_request:
    branches: [ staging, main ]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up Python
        uses: actions/setup-python@v3
        with:
          python-version: '3.x'
      - name: Install Dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
      - name: Run Tests
        run: |
          python src/test_repo.py
```

Este pipeline garante que qualquer alteração no repositório seja validada automaticamente conforme as regras de qualidade e correção implementadas.

—

## 10.4 Conclusão dos Testes

A estratégia híbrida de testes (automáticos + manuais) combinada com um pipeline automatizado de CI/CD assegura:

- Confiabilidade funcional de componentes críticos;
- Estabilidade estrutural da base de dados;
- Resiliência da interface de consulta;
- Preparação para evolução contínua da base de código.

O conjunto de testes oferece um nível de qualidade compatível com padrões de engenharia de software profissional e acadêmico.