ChatGPT

Simulador de Compra de Apartamento na Planta

Escopo e premissas do app

Este projeto entrega um web app “bem direto ao ponto”, pensado para ser fácil de manter e de evoluir por pessoas em início de carreira: **apenas HTML + CSS + JavaScript Vanilla** com **ES Modules** (sem frameworks e sem build tools). Para ES Modules funcionarem corretamente no navegador, o carregamento do JavaScript é feito com `type="module"` e `import/export` entre arquivos. [1]

A persistência é feita **exclusivamente via** `localStorage` (chave fixa `APTO_SIM_DB_V1`). O `localStorage` armazena **somente strings**, então o banco é serializado com `JSON.stringify` e reidratado com `JSON.parse`. [2]

O simulador modela três níveis principais:

- **Projetos**: cadastro do empreendimento (nome, cidade, UF, incorporadora/“developer”).
- **Simulações**: cadastro por projeto (nome, datas relevantes, preço-base).
- **Editor de simulação**: itens de fluxo de caixa (mensal / balão / único), correção por índice (opcional, ex.: INCC), financiamento (opcional: SAC ou PRICE), recalcular.
- **Resultados**: totais por fase, linha do tempo mensal, flags de risco, export/import JSON.

A correção por índice é tratada como **capitalização composta** mensal (produto de fatores mensais), alinhada ao uso de juros compostos em séries temporais financeiras. [3]
O INCC (da FGV IBRE [4]) é referenciado como exemplo de índice de custos de construção e é comumente acompanhado mensalmente (o app deixa genérico: qualquer série mensal em %). [5]

Modelo de dados e persistência em `localStorage`

O “banco” (`APTO_SIM_DB_V1`) é um JSON com **versão**, timestamps e coleções lineares (arrays) para evitar atualizações profundas e complexas.

Pontos de design:

- **Versão (**`version: 1`**)** para permitir migração futura.
- **IDs string** simples (`crypto.randomUUID()` quando disponível; fallback com timestamp).
- **Tudo em um único objeto** para facilitar export/import e reduzir chances de inconsistência.

Por que `localStorage`: por ser simples e disponível no navegador, mas exige cuidado com disponibilidade (modo privado / quota / bloqueios) e com a serialização de objetos. [6]

Motor de cálculos: linha do tempo, correção e financiamento

Linha do tempo mensal e correção por índice

A linha do tempo é construída mês a mês (`YYYY-MM`) desde o mês do contrato até:

- o último mês com item de fluxo de caixa, ou
- o final do financiamento (se habilitado),

o que for maior.

Quando a correção é habilitada, o app calcula um **fator acumulado** mensal:

- mês inicial do contrato: fator = 1
- meses seguintes: `fator[mês] = fator[mês_anterior] * (1 + taxa_do_mês)`

Isso é uma aplicação direta de composição por períodos (juros compostos/capitalização), generalizando para taxas variáveis. [3]

A série "INCC" entra como exemplo por ser um índice de custos da construção civil e por ter divulgação mensal. [5]

SAC e PRICE

O app oferece financiamento com:

- **SAC**: amortização constante (`PV / n`), juros sobre saldo, prestação decrescente. [7]
- **PRICE (Sistema Francês)**: prestação constante, juros decrescentes e amortização crescente; prestação calculada por coeficiente/annuity. [8]

A fórmula de prestação do PRICE é implementada via coeficiente K:
$K = \frac{i(1+i)^n}{(1+i)^n - 1}$ e $PMT = PV \cdot K$. [9]

Conversão de taxa anual para mensal

O app assume que a taxa informada no editor é **ao ano (% a.a.)** e converte para taxa mensal efetiva por:
`i_m = (1 + i_a)^(1/12) - 1`
que é um uso direto da lógica de capitalização composta por períodos. [3]

Alertas de risco e motivação das regras

O módulo de riscos é propositalmente pequeno: gera flags por regras booleanas simples (sem "IA", sem probabilidades), para apoiar checklist prático.

Regras implementadas:

- **Quadro-resumo ausente**: a Lei 4.591/1964 (texto compilado no Portal do Planalto [10]) determina que contratos de incorporação iniciem com **quadro-resumo** contendo, entre outros, preço total, corretagem, índices de correção e sistema de amortização. [11]

- **Corretagem não confirmada**: o quadro-resumo inclui informações de corretagem e beneficiário; então a ausência/indefinição é um risco de comunicação/planejamento. [12]
- **ITBI não provisionado**: ITBI é imposto municipal na transmissão onerosa inter vivos e costuma ser devido no contexto de registro/transferência (o app trata como provisionamento financeiro). [13]
- **Tolerância acima de 180 dias**: o texto legal (Lei 13.786/2018 no Planalto [10]) explicita o marco de **até 180 dias corridos** para cláusula de tolerância destacada. [14]

Integração opcional: BCB SGS (sem backend)

Como extra opcional, há função para buscar séries no **SGS** do Banco Central do Brasil [15] via `fetch` direto do navegador.

A documentação do recurso JSON no portal de dados do BCB descreve o padrão de URL, parâmetros (`codigo_serie`, `dataInicial`, `dataFinal`) e restrições: intervalo máximo de **10 anos** entre datas e endpoint de "últimos N" (com limite informado no recurso). [16]

O app deixa isso "atrás de um `<details>`" no editor e não depende disso para funcionar.

Árvore de arquivos e código completo

Estrutura:

- `index.html`
- `styles.css`
- `js/`
- `app.js`
- `storage.js`
- `calc.js`
- `render.js`
- `risks.js`
- `faq.js`

index.html

```
<!doctype html>
<html lang="pt-BR">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Simulador Compra na Planta • Apto</title>
    <link rel="stylesheet" href="./styles.css" />
  </head>

  <body>
    <header class="app-header">
      <div class="container header-row">
        <div class="brand">
          <div class="brand-title">Apto • Simulador "na planta"</div>
```

```
          <div class="brand-sub" id="app-subtitle">localStorage •
APTO_SIM_DB_V1</div>
        </div>

        <nav class="nav">
          <a class="nav-link" href="#projects" data-nav>Projetos</a>
          <a class="nav-link" href="#sims" data-nav>Simulações</a>
          <a class="nav-link" href="#editor" data-nav>Editor</a>
          <a class="nav-link" href="#results" data-nav>Resultados</a>
          <a class="nav-link" href="#faq" data-nav>FAQ</a>
        </nav>
      </div>
    </header>

    <main class="container app-main">
      <section class="statusbar">
        <div class="status-pill">
          <span class="status-label">Projeto:</span>
          <span id="status-project">—</span>
        </div>
        <div class="status-pill">
          <span class="status-label">Simulação:</span>
          <span id="status-sim">—</span>
        </div>
        <div class="status-spacer"></div>
        <button class="btn btn-ghost" id="btn-export" type="button"
title="Baixar export JSON">
          Exportar JSON
        </button>
        <label class="btn btn-ghost file-btn"
title="Importar JSON (substitui o banco)">
          Importar JSON
          <input id="file-import" type="file" accept="application/json" />
        </label>
        <button class="btn btn-danger" id="btn-reset" type="button"
title="Voltar para os dados de exemplo">
          Resetar exemplo
        </button>
      </section>

      <section id="view-projects" class="view">
        <h2>Projetos</h2>
        <p class="muted">
          Cadastre empreendimentos (nome, cidade/UF e incorporadora). Depois
crie simulações por projeto.
        </p>

        <div class="grid-2">
          <div class="card">
            <h3>Lista</h3>
            <div id="projects-list"></div>
```

```
          </div>

          <div class="card">
            <h3 id="project-form-title">Novo projeto</h3>
            <form id="project-form" class="form">
              <input type="hidden" id="project-id" />

              <label>
                Nome do projeto
                <input id="project-name" type="text" required
placeholder="Ex.: Residencial Vista Park" />
              </label>

              <div class="grid-2">
                <label>
                  Cidade
                  <input id="project-city" type="text" required
placeholder="Ex.: São Paulo" />
                </label>

                <label>
                  UF
                  <input id="project-uf" type="text" required maxlength="2"
placeholder="SP" />
                </label>
              </div>

              <label>
                Incorporadora / Developer
                <input id="project-developer" type="text" required
placeholder="Ex.: Construtora X" />
              </label>

              <div class="row">
                <button class="btn" type="submit">Salvar</button>
                <button class="btn btn-ghost" id="project-cancel"
type="button">Limpar</button>
              </div>
            </form>
          </div>
        </div>
      </section>

      <section id="view-sims" class="view hidden">
        <h2>Simulações</h2>
        <p class="muted">
          Cada simulação pertence a um projeto. Aqui você define datas e
preço-base; os detalhes ficam no editor.
        </p>

        <div id="sims-empty" class="notice hidden">
```

```
          Selecione um projeto em <b>Projetos</b> para ver/criar simulações.
        </div>

        <div id="sims-content" class="grid-2 hidden">
          <div class="card">
            <h3>Lista</h3>
            <div id="sims-list"></div>
          </div>

          <div class="card">
            <h3 id="sim-form-title">Nova simulação</h3>

            <form id="sim-form" class="form">
              <input type="hidden" id="sim-id" />

              <label>
                Nome da simulação
                <input id="sim-name" type="text" required placeholder="Ex.:
Cenário base" />
              </label>

              <div class="grid-2">
                <label>
                  Data do contrato
                  <input id="sim-contractDate" type="date" required />
                </label>

                <label>
                  Data prevista de entrega
                  <input id="sim-deliveryDate" type="date" required />
                </label>
              </div>

              <div class="grid-2">
                <label>
                  Tolerância (dias)
                  <input id="sim-toleranceDays" type="number" min="0"
step="1" value="180" />
                </label>

                <label>
                  Preço-base (R$)
                  <input id="sim-basePrice" type="number" min="0"
step="0.01" required placeholder="500000" />
                </label>
              </div>

              <div class="row">
                <button class="btn" type="submit">Salvar</button>
                <button class="btn btn-ghost" id="sim-cancel"
type="button">Limpar</button>
```

```
              </div>

              <div class="hint">
                Dica: depois de salvar, abra no <b>Editor</b> para criar
fluxo de caixa, correção e financiamento.
              </div>
            </form>
          </div>
        </div>
      </section>

      <section id="view-editor" class="view hidden">
        <h2>Editor da simulação</h2>

        <div id="editor-empty" class="notice hidden">
          Selecione uma simulação em <b>Simulações</b> para editar.
        </div>

        <div id="editor-content" class="hidden">
          <div class="card">
            <h3>Checklist e premissas</h3>
            <div class="grid-2">
              <label class="check">
                <input id="flag-quadro" type="checkbox" />
                Tenho quadro-resumo / informações essenciais confirmadas
              </label>

              <label class="check">
                <input id="flag-corretagem" type="checkbox" />
                Corretagem confirmada (valor/beneficiário/condições)
              </label>

              <label class="check">
                <input id="flag-itbi" type="checkbox" />
                ITBI provisionado no planejamento
              </label>
            </div>

            <div class="row row-spaced">
              <button class="btn" id="btn-save-flags" type="button">Salvar
checklist</button>
            </div>

            <div class="hint">
              O app usa essas flags para gerar alertas simples (não é
recomendação jurídica/financeira).
            </div>
          </div>

          <div class="grid-2">
            <div class="card">
```

```
              <h3>Itens de fluxo de caixa</h3>
              <div id="cashflow-table"></div>

              <h4 id="cashflow-form-title">Novo item</h4>
              <form id="cashflow-form" class="form">
                <input type="hidden" id="cashflow-id" />

                <label>
                  Tipo
                  <select id="cashflow-kind">
                    <option value="monthly">Mensal</option>
                    <option value="balloon">Balão</option>
                    <option value="once">Único</option>
                  </select>
                </label>

                <label>
                  Descrição
                  <input id="cashflow-label" type="text" required
placeholder="Ex.: Parcela obra / Entrada / ITBI" />
                </label>

                <label>
                  Valor base (R$)
                  <input id="cashflow-amount" type="number" min="0"
step="0.01" required />
                </label>

                <div class="grid-2">
                  <label>
                    Início (para mensal)
                    <input id="cashflow-startDate" type="date" />
                  </label>
                  <label>
                    Fim (para mensal)
                    <input id="cashflow-endDate" type="date" />
                  </label>
                </div>

                <label>
                  Vencimento (para balão/único)
                  <input id="cashflow-dueDate" type="date" />
                </label>

                <div class="grid-2">
                  <label class="check">
                    <input id="cashflow-corr" type="checkbox" checked />
                    Aplicar correção por índice (se habilitada)
                  </label>

                  <label class="check">
```

```
                    <input id="cashflow-principal" type="checkbox" checked />
                    Abate saldo do imóvel (principal)
                  </label>
                </div>

                <div class="row">
                  <button class="btn" type="submit">Salvar item</button>
                  <button class="btn btn-ghost" id="cashflow-cancel"
type="button">Limpar</button>
                </div>

                <div class="hint">
                  “Mensal” gera lançamentos em todos os meses entre início e
fim. “Balão” e “Único” caem em 1 mês.
                </div>
              </form>
            </div>

            <div class="card">
              <h3>Correção por índice</h3>

              <label class="check">
                <input id="corr-enabled" type="checkbox" />
                Habilitar correção mensal por série (ex.: INCC)
              </label>

              <div class="grid-2">
                <label>
                  Série
                  <select id="corr-series"></select>
                </label>

                <label>
                  Observação
                  <input id="corr-note" type="text"
placeholder="Ex.: INCC-M, acumulado por mês" />
                </label>
              </div>

              <details class="details">
                <summary>Editar série (manual)</summary>

                <div class="hint">
                  Cole um JSON simples no formato: <code>{"YYYY-MM": 0.21,
"YYYY-MM": 0.30}</code> (valores em %).
                </div>

                <label>
                  JSON de variações mensais (%)
                  <textarea id="corr-json" rows="7" spellcheck="false"></
textarea>
```

```
                </label>

                <div class="row">
                  <button class="btn" id="btn-corr-apply"
type="button">Aplicar no cadastro da série</button>
                </div>
              </details>

              <details class="details">
                <summary>Opcional: importar do BCB (SGS) via fetch</summary>

                <div class="hint">
                  Usa o endpoint público do SGS em JSON. O próprio BCB
documenta parâmetros e limites (como período
                  máximo entre datas). Se não quiser usar, ignore esta seção.
                </div>

                <div class="grid-3">
                  <label>
                    Código da série (SGS)
                    <input id="bcb-seriesCode" type="number" min="1"
step="1" placeholder="Ex.: 433" />
                  </label>

                  <label>
                    Data inicial
                    <input id="bcb-start" type="date" />
                  </label>

                  <label>
                    Data final
                    <input id="bcb-end" type="date" />
                  </label>
                </div>

                <div class="row">
                  <button class="btn btn-ghost" id="btn-bcb-fetch"
type="button">
                    Buscar e salvar como nova série
                  </button>
                </div>

                <div class="hint">
                  Observação: o SGS retorna datas no formato dd/MM/aaaa e
valores como string.
                </div>
              </details>

              <div class="row row-spaced">
                <button class="btn" id="btn-save-corr" type="button">Salvar
configurações de correção</button>
```

```
              </div>
            </div>
          </div>

          <div class="card">
            <h3>Plano de financiamento</h3>

            <label class="check">
              <input id="fin-enabled" type="checkbox" />
              Habilitar financiamento do saldo (SAC ou PRICE)
            </label>

            <div class="grid-4">
              <label>
                Sistema
                <select id="fin-type">
                  <option value="SAC">SAC</option>
                  <option value="PRICE">PRICE</option>
                </select>
              </label>

              <label>
                Prazo (meses)
                <input id="fin-months" type="number" min="1" step="1"
value="240" />
              </label>

              <label>
                Juros (% a.a.)
                <input id="fin-annualRate" type="number" min="0" step="0.01"
value="10.00" />
              </label>

              <label>
                Início do financiamento
                <input id="fin-startDate" type="date" />
              </label>
            </div>

            <div class="hint" id="fin-rate-hint">—</div>

            <div class="row row-spaced">
              <button class="btn" id="btn-save-fin" type="button">Salvar
financiamento</button>
              <button class="btn" id="btn-recalc" type="button">Recalcular</
button>
            </div>

            <div class="hint">
              “Recalcular” reconstrói a linha do tempo e atualiza os
resultados. Você pode abrir “Resultados” depois.
```

```
            </div>
          </div>
        </div>
      </section>

      <section id="view-results" class="view hidden">
        <h2>Resultados</h2>

        <div id="results-empty" class="notice hidden">
          Selecione uma simulação em <b>Simulações</b> e clique em
<b>Recalcular</b> no editor.
        </div>

        <div id="results-content" class="hidden">
          <div class="grid-2">
            <div class="card">
              <h3>Totais</h3>
              <div id="results-totals"></div>
            </div>

            <div class="card">
              <h3>Alertas de risco</h3>
              <div id="results-risks"></div>
            </div>
          </div>

          <div class="card">
            <h3>Linha do tempo mensal</h3>
            <div class="hint">
              Valores em R$; índice em % (mês). O simulador é aproximado e
serve para comparações/planejamento.
            </div>
            <div class="table-wrap" id="results-timeline"></div>
          </div>

          <div class="card">
            <h3>Exportar / Importar</h3>
            <div class="grid-2">
              <div>
                <div class="row">
                  <button class="btn btn-ghost" id="btn-export-inline"
type="button">Gerar JSON abaixo</button>
                  <button class="btn" id="btn-copy-json"
type="button">Copiar</button>
                </div>
                <textarea id="export-json" rows="10" spellcheck="false"
placeholder="Clique em “Gerar JSON abaixo”..."></textarea>
              </div>

              <div>
                <div class="hint">Cole um JSON exportado e importe (isso
```

```
substitui todo o banco local).</div>
                <div class="row">
                  <button class="btn btn-danger" id="btn-import-inline"
type="button">Importar (substituir)</button>
                </div>
                <textarea id="import-json" rows="10" spellcheck="false"
placeholder='Cole aqui um JSON do APTO_SIM_DB_V1...'></textarea>
              </div>
            </div>
          </div>
        </div>
      </section>

      <section id="view-faq" class="view hidden">
        <h2>FAQ</h2>
        <div id="faq-content"></div>
      </section>

      <div id="toast" class="toast hidden" role="status" aria-
live="polite"></div>
    </main>

    <script type="module" src="./js/app.js"></script>
  </body>
</html>
```

styles.css

```
:root {
  --bg: #0b0e14;
  --panel: #121826;
  --panel2: #0f1420;
  --text: #e8eefc;
  --muted: #aab6d6;
  --border: rgba(255, 255, 255, 0.12);
  --shadow: 0 10px 30px rgba(0, 0, 0, 0.35);

  --good: #2dd4bf;
  --warn: #fbbf24;
  --bad: #fb7185;

  --radius: 14px;
  --gap: 14px;
  --mono: ui-monospace, SFMono-Regular, Menlo, Monaco, Consolas, "Liberation
Mono", "Courier New", monospace;
}

* { box-sizing: border-box; }

html, body {
```

```
  margin: 0;
  padding: 0;
  background: radial-gradient(1200px 600px at 15% 0%, rgba(45, 212, 191,
0.12), transparent 60%),
              radial-gradient(1000px 600px at 90% 10%, rgba(251, 191, 36,
0.10), transparent 55%),
              var(--bg);
  color: var(--text);
  font-family: ui-sans-serif, system-ui, -apple-system, Segoe UI, Roboto,
Ubuntu, Cantarell, Noto Sans, Helvetica, Arial;
}

a { color: inherit; text-decoration: none; }
a:hover { text-decoration: underline; }

.container {
  width: min(1100px, calc(100vw - 24px));
  margin: 0 auto;
}

.app-header {
  position: sticky;
  top: 0;
  z-index: 10;
  background: rgba(11, 14, 20, 0.72);
  backdrop-filter: blur(10px);
  border-bottom: 1px solid var(--border);
}

.header-row {
  display: flex;
  align-items: center;
  justify-content: space-between;
  padding: 14px 0;
  gap: var(--gap);
}

.brand-title {
  font-weight: 700;
  letter-spacing: 0.2px;
}
.brand-sub {
  font-size: 12px;
  color: var(--muted);
}

.nav {
  display: flex;
  gap: 8px;
  flex-wrap: wrap;
  justify-content: flex-end;
```

```
}

.nav-link {
  padding: 8px 10px;
  border: 1px solid var(--border);
  border-radius: 999px;
  background: rgba(255, 255, 255, 0.03);
}
.nav-link.active {
  border-color: rgba(45, 212, 191, 0.55);
  box-shadow: 0 0 0 2px rgba(45, 212, 191, 0.12) inset;
}

.app-main { padding: 18px 0 40px; }

h2 { margin: 10px 0 10px; }
h3 { margin: 0 0 10px; }
h4 { margin: 18px 0 10px; }

.muted { color: var(--muted); margin-top: 0; }

.statusbar {
  display: flex;
  align-items: center;
  gap: 10px;
  padding: 10px;
  border: 1px solid var(--border);
  border-radius: var(--radius);
  background: rgba(18, 24, 38, 0.55);
  box-shadow: var(--shadow);
  margin-bottom: 16px;
}
.status-pill {
  display: inline-flex;
  gap: 8px;
  align-items: center;
  padding: 6px 10px;
  border: 1px solid var(--border);
  border-radius: 999px;
  background: rgba(255, 255, 255, 0.03);
  font-size: 13px;
}
.status-label { color: var(--muted); }
.status-spacer { flex: 1; }

.grid-2 {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: var(--gap);
}
.grid-3 {
```

```
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  gap: var(--gap);
}
.grid-4 {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr 1fr;
  gap: var(--gap);
}

@media (max-width: 920px) {
  .grid-2, .grid-3, .grid-4 { grid-template-columns: 1fr; }
}

.card {
  border: 1px solid var(--border);
  border-radius: var(--radius);
  background: linear-gradient(180deg, rgba(18, 24, 38, 0.68), rgba(15, 20,
32, 0.65));
  box-shadow: var(--shadow);
  padding: 14px;
}

.form { display: grid; gap: 10px; }
label { display: grid; gap: 6px; font-size: 13px; color: var(--muted); }
input, select, textarea {
  padding: 10px 10px;
  border-radius: 12px;
  border: 1px solid var(--border);
  background: rgba(0, 0, 0, 0.20);
  color: var(--text);
  outline: none;
}
textarea { font-family: var(--mono); resize: vertical; }

.row {
  display: flex;
  gap: 10px;
  align-items: center;
  flex-wrap: wrap;
}
.row-spaced { margin-top: 10px; }

.btn {
  appearance: none;
  border: 1px solid rgba(45, 212, 191, 0.45);
  background: rgba(45, 212, 191, 0.14);
  color: var(--text);
  padding: 10px 12px;
  border-radius: 12px;
  cursor: pointer;
```

```
  font-weight: 600;
}
.btn:hover { background: rgba(45, 212, 191, 0.20); }
.btn:active { transform: translateY(1px); }

.btn-ghost {
  border-color: var(--border);
  background: rgba(255, 255, 255, 0.04);
}
.btn-ghost:hover { background: rgba(255, 255, 255, 0.08); }

.btn-danger {
  border-color: rgba(251, 113, 133, 0.50);
  background: rgba(251, 113, 133, 0.14);
}
.btn-danger:hover { background: rgba(251, 113, 133, 0.20); }

.file-btn input[type="file"] { display: none; }

.notice {
  padding: 12px;
  border: 1px dashed var(--border);
  border-radius: var(--radius);
  background: rgba(255, 255, 255, 0.04);
}

.hint {
  font-size: 12px;
  color: var(--muted);
  line-height: 1.35;
}

.check {
  display: flex;
  gap: 10px;
  align-items: center;
  color: var(--text);
  font-size: 13px;
}
.check input { width: 18px; height: 18px; }

.hidden { display: none !important; }

.table-wrap { overflow: auto; border-radius: 12px; border: 1px solid var(--
border); }
table {
  width: 100%;
  border-collapse: collapse;
  min-width: 980px;
  background: rgba(0,0,0,0.18);
}
```

```
th, td {
  padding: 10px 10px;
  border-bottom: 1px solid var(--border);
  vertical-align: top;
  font-size: 12px;
}
th {
  position: sticky;
  top: 0;
  background: rgba(18, 24, 38, 0.95);
  text-align: left;
  color: var(--muted);
  font-weight: 700;
}
td.mono { font-family: var(--mono); }

.badge {
  display: inline-flex;
  align-items: center;
  padding: 4px 8px;
  border-radius: 999px;
  border: 1px solid var(--border);
  font-size: 12px;
  background: rgba(255,255,255,0.04);
  color: var(--muted);
  gap: 8px;
}
.badge.good { border-color: rgba(45, 212, 191, 0.45); color: var(--good); }
.badge.warn { border-color: rgba(251, 191, 36, 0.45); color: var(--warn); }
.badge.bad { border-color: rgba(251, 113, 133, 0.50); color: var(--bad); }

.details summary { cursor: pointer; color: var(--text); margin: 8px 0; }
.toast {
  position: fixed;
  left: 50%;
  bottom: 18px;
  transform: translateX(-50%);
  padding: 10px 12px;
  border-radius: 999px;
  border: 1px solid var(--border);
  background: rgba(18, 24, 38, 0.92);
  box-shadow: var(--shadow);
  font-size: 13px;
}
```

js/faq.js

```
export const FAQ_HTML = `
  <div class="card">
    <h3>O que este simulador é (e o que não é)</h3>
```

```
    <p class="muted">
      Este app é um <b>simulador simples</b> para organizar fluxo de caixa,
correção por índice e
      financiamento em cenários de compra de imóvel na planta. Ele não
substitui planilhas avançadas,
      nem análise jurídica/financeira profissional.
    </p>
  </div>

  <div class="card">
    <h3>Como a correção por índice funciona aqui</h3>
    <p class="muted">
      Você cadastra variações mensais em % por mês (YYYY-MM). O app calcula
um fator acumulado e
      multiplica o preço-base e (opcionalmente) itens marcados com “Aplicar
correção”.
      Se um mês não tiver taxa cadastrada, o app considera 0% naquele mês (e
segue).
    </p>
    <p class="muted">
      Para INCC, use uma série mensal e mantenha o cadastro atualizado
conforme sua fonte preferida.
    </p>
  </div>

  <div class="card">
    <h3>Fluxo de caixa: tipos</h3>
    <p class="muted">
      <b>Mensal</b>: repete todo mês do período (início → fim).<br/>
      <b>Balão</b>: um pagamento grande em um mês específico (normalmente
“intermediária”).<br/>
      <b>Único</b>: custos pontuais (ex.: ITBI, cartório, mudança).
    </p>
  </div>

  <div class="card">
    <h3>Financiamento (SAC e PRICE) – simplificação</h3>
    <p class="muted">
      O financiamento parte do saldo estimado no mês de início:
      <b>valor do contrato (corrigido)</b> menos <b>pagamentos que abatem
principal</b> antes do início.
      Depois gera a tabela mensal por SAC ou PRICE.
    </p>
    <p class="muted">
      Observação: bancos podem usar regras adicionais (seguros, CET, taxas,
datas de corte),
      então use este resultado como referência de cenário.
    </p>
  </div>

  <div class="card">
```

```
    <h3>Como rodar</h3>
    <p class="muted">
      Este projeto usa ES Modules. Rode com um servidor local (ex.: VSCode
Live Server).
      Abrir o HTML direto via <code>file://</code> pode falhar em alguns
navegadores.
    </p>
  </div>
`;
```

js/storage.js

```
const STORAGE_KEY = "APTO_SIM_DB_V1";

function nowIso() {
  return new Date().toISOString();
}

function safeJsonParse(text) {
  try {
    return JSON.parse(text);
  } catch {
    return null;
  }
}

function genId(prefix) {
  // Preferir UUID; fallback simples.
  if (typeof crypto !== "undefined" && crypto.randomUUID) {
    return `${prefix}_${crypto.randomUUID()}`;
  }
  return `${prefix}_${Date.now()}_${Math.random().toString(16).slice(2)}`;
}

function isLocalStorageAvailable() {
  try {
    const k = "__apto_test__";
    window.localStorage.setItem(k, "1");
    window.localStorage.removeItem(k);
    return true;
  } catch {
    return false;
  }
}

function seedDb() {
  const createdAt = nowIso();

  const project1 = {
    id: genId("p"),
```

```
  name: "Residencial Vista Park",
  city: "São Paulo",
  uf: "SP",
  developer: "Construtora Exemplo S.A.",
  createdAt,
  updatedAt: createdAt,
};

// Série exemplo de INCC (valores fictícios só para demonstrar).
const idx1 = {
  id: genId("idx"),
  name: "INCC (exemplo)",
  note: "Valores fictícios para demonstração",
  source: "manual",
  monthlyPct: {
    "2026-02": 0.20,
    "2026-03": 0.25,
    "2026-04": 0.30,
    "2026-05": 0.28,
    "2026-06": 0.26,
    "2026-07": 0.22,
    "2026-08": 0.24,
    "2026-09": 0.27,
    "2026-10": 0.23,
    "2026-11": 0.21,
    "2026-12": 0.19,
  },
  createdAt,
  updatedAt: createdAt,
};

const sim1 = {
  id: genId("s"),
  projectId: project1.id,
  name: "Cenário base",
  contractDate: "2026-02-01",
  deliveryDate: "2028-06-30",
  toleranceDays: 180,
  basePrice: 500000,

  flags: {
    hasQuadroResumo: false,
    brokerFeeConfirmed: false,
    itbiProvisioned: false,
  },

  correction: {
    enabled: true,
    seriesId: idx1.id,
    note: "INCC (exemplo)",
  },
```

```
financing: {
  enabled: true,
  type: "SAC",
  months: 240,
  annualRatePct: 10.0,
  startDate: "2028-07-01",
},

cashflow: [
  {
    id: genId("cf"),
    kind: "once",
    label: "Sinal / Entrada",
    amount: 50000,
    dueDate: "2026-02-10",
    startDate: null,
    endDate: null,
    applyCorrection: false,
    affectsPrincipal: true,
  },
  {
    id: genId("cf"),
    kind: "monthly",
    label: "Parcelas obra",
    amount: 3000,
    startDate: "2026-03-01",
    endDate: "2028-05-01",
    dueDate: null,
    applyCorrection: true,
    affectsPrincipal: true,
  },
  {
    id: genId("cf"),
    kind: "balloon",
    label: "Intermediária",
    amount: 20000,
    dueDate: "2027-12-05",
    startDate: null,
    endDate: null,
    applyCorrection: true,
    affectsPrincipal: true,
  },
  {
    id: genId("cf"),
    kind: "once",
    label: "ITBI (provisão)",
    amount: 15000,
    dueDate: "2028-07-10",
    startDate: null,
    endDate: null,
```

```
        applyCorrection: false,
        affectsPrincipal: false,
      },
    ],

    createdAt,
    updatedAt: createdAt,
  };

  return {
    version: 1,
    createdAt,
    updatedAt: createdAt,
    projects: [project1],
    simulations: [sim1],
    indexSeries: [idx1],
  };
}

function readDbRaw() {
  if (!isLocalStorageAvailable()) {
    return null;
  }
  const text = window.localStorage.getItem(STORAGE_KEY);
  if (!text) return null;
  return safeJsonParse(text);
}

function writeDbRaw(db) {
  if (!isLocalStorageAvailable()) {
    throw new Error("localStorage indisponível neste navegador/sessão.");
  }
  window.localStorage.setItem(STORAGE_KEY, JSON.stringify(db));
}

function normalizeDb(db) {
  // Normalização mínima para evitar quebra em versões futuras.
  if (!db || typeof db !== "object") return null;
  if (db.version !== 1) return null;
  db.projects ||= [];
  db.simulations ||= [];
  db.indexSeries ||= [];
  db.createdAt ||= nowIso();
  db.updatedAt ||= nowIso();
  return db;
}

export function ensureDb() {
  const existing = normalizeDb(readDbRaw());
  if (existing) return existing;
```

```
  const fresh = seedDb();
  try {
    writeDbRaw(fresh);
  } catch {
    // Se não conseguir escrever, ainda retorna seed para rodar (sem
persistência).
  }
  return fresh;
}

export function getDb() {
  const db = normalizeDb(readDbRaw());
  if (db) return db;

  // fallback: garantir seed no fluxo normal
  return ensureDb();
}

export function replaceDb(db) {
  const normalized = normalizeDb(db);
  if (!normalized) throw new Error("JSON inválido ou versão incompatível.");
  normalized.updatedAt = nowIso();
  writeDbRaw(normalized);
  return normalized;
}

export function resetDbToSeed() {
  const fresh = seedDb();
  writeDbRaw(fresh);
  return fresh;
}

export function exportDbJson() {
  const db = getDb();
  return JSON.stringify(db, null, 2);
}

export function importDbJson(jsonText) {
  const parsed = safeJsonParse(jsonText);
  if (!parsed) throw new Error("JSON inválido.");
  return replaceDb(parsed);
}

/* Projects CRUD */

export function listProjects(db) {
  return [...db.projects].sort((a, b) => a.name.localeCompare(b.name));
}

export function getProject(db, projectId) {
  return db.projects.find((p) => p.id === projectId) || null;
```

```
}

export function saveProject(db, payload) {
  const t = nowIso();
  const uf = String(payload.uf || "").trim().toUpperCase();

  if (payload.id) {
    const idx = db.projects.findIndex((p) => p.id === payload.id);
    if (idx === -1) throw new Error("Projeto não encontrado.");

    db.projects[idx] = {
      ...db.projects[idx],
      name: String(payload.name || "").trim(),
      city: String(payload.city || "").trim(),
      uf,
      developer: String(payload.developer || "").trim(),
      updatedAt: t,
    };
  } else {
    db.projects.push({
      id: genId("p"),
      name: String(payload.name || "").trim(),
      city: String(payload.city || "").trim(),
      uf,
      developer: String(payload.developer || "").trim(),
      createdAt: t,
      updatedAt: t,
    });
  }

  db.updatedAt = t;
  writeDbRaw(db);
  return db;
}

export function deleteProject(db, projectId) {
  db.projects = db.projects.filter((p) => p.id !== projectId);
  db.simulations = db.simulations.filter((s) => s.projectId !== projectId);
  db.updatedAt = nowIso();
  writeDbRaw(db);
  return db;
}

/* Simulations CRUD */

export function listSimulationsByProject(db, projectId) {
  return db.simulations
    .filter((s) => s.projectId === projectId)
    .sort((a, b) => a.name.localeCompare(b.name));
}
```

```
export function getSimulation(db, simId) {
  return db.simulations.find((s) => s.id === simId) || null;
}

export function saveSimulation(db, projectId, payload) {
  const t = nowIso();

  const basePrice = Number(payload.basePrice || 0);
  const toleranceDays = Number(payload.toleranceDays || 0);

  if (payload.id) {
    const idx = db.simulations.findIndex((s) => s.id === payload.id);
    if (idx === -1) throw new Error("Simulação não encontrada.");

    db.simulations[idx] = {
      ...db.simulations[idx],
      name: String(payload.name || "").trim(),
      contractDate: payload.contractDate || db.simulations[idx].contractDate,
      deliveryDate: payload.deliveryDate || db.simulations[idx].deliveryDate,
      toleranceDays: Number.isFinite(toleranceDays) ? toleranceDays : 0,
      basePrice: Number.isFinite(basePrice) ? basePrice : 0,
      updatedAt: t,
    };
  } else {
    db.simulations.push({
      id: genId("s"),
      projectId,
      name: String(payload.name || "").trim(),
      contractDate: payload.contractDate,
      deliveryDate: payload.deliveryDate,
      toleranceDays: Number.isFinite(toleranceDays) ? toleranceDays : 0,
      basePrice: Number.isFinite(basePrice) ? basePrice : 0,

      flags: {
        hasQuadroResumo: false,
        brokerFeeConfirmed: false,
        itbiProvisioned: false,
      },

      correction: {
        enabled: false,
        seriesId: db.indexSeries[0]?.id || null,
        note: "",
      },

      financing: {
        enabled: false,
        type: "SAC",
        months: 240,
        annualRatePct: 10.0,
        startDate: payload.deliveryDate,
```

```
      },

      cashflow: [],
      createdAt: t,
      updatedAt: t,
    });
  }

  db.updatedAt = t;
  writeDbRaw(db);
  return db;
}

export function deleteSimulation(db, simId) {
  db.simulations = db.simulations.filter((s) => s.id !== simId);
  db.updatedAt = nowIso();
  writeDbRaw(db);
  return db;
}

/* Cashflow CRUD (dentro da simulação) */

export function saveCashflowItem(db, simId, payload) {
  const t = nowIso();
  const simIdx = db.simulations.findIndex((s) => s.id === simId);
  if (simIdx === -1) throw new Error("Simulação não encontrada.");

  const sim = db.simulations[simIdx];
  sim.cashflow ||= [];

  const item = {
    id: payload.id || genId("cf"),
    kind: payload.kind,
    label: String(payload.label || "").trim(),
    amount: Number(payload.amount || 0),
    startDate: payload.startDate || null,
    endDate: payload.endDate || null,
    dueDate: payload.dueDate || null,
    applyCorrection: Boolean(payload.applyCorrection),
    affectsPrincipal: Boolean(payload.affectsPrincipal),
  };

  const existingIdx = sim.cashflow.findIndex((x) => x.id === item.id);
  if (existingIdx >= 0) {
    sim.cashflow[existingIdx] = { ...sim.cashflow[existingIdx], ...item };
  } else {
    sim.cashflow.push(item);
  }

  sim.updatedAt = t;
  db.updatedAt = t;
```

```
  writeDbRaw(db);
  return db;
}

export function deleteCashflowItem(db, simId, cashflowId) {
  const t = nowIso();
  const sim = getSimulation(db, simId);
  if (!sim) throw new Error("Simulação não encontrada.");

  sim.cashflow = (sim.cashflow || []).filter((x) => x.id !== cashflowId);
  sim.updatedAt = t;
  db.updatedAt = t;
  writeDbRaw(db);
  return db;
}

/* Correção / Séries */

export function listIndexSeries(db) {
  return [...db.indexSeries].sort((a, b) => a.name.localeCompare(b.name));
}

export function getIndexSeries(db, seriesId) {
  return db.indexSeries.find((x) => x.id === seriesId) || null;
}

export function upsertIndexSeries(db, payload) {
  const t = nowIso();
  const item = {
    id: payload.id || genId("idx"),
    name: String(payload.name || "").trim() || "Série sem nome",
    note: String(payload.note || "").trim(),
    source: payload.source || "manual",
    monthlyPct: payload.monthlyPct || {},
    createdAt: payload.createdAt || t,
    updatedAt: t,
  };

  const idx = db.indexSeries.findIndex((x) => x.id === item.id);
  if (idx >= 0) db.indexSeries[idx] = { ...db.indexSeries[idx], ...item };
  else db.indexSeries.push(item);

  db.updatedAt = t;
  writeDbRaw(db);
  return db;
}

/* Flags da simulação */

export function saveSimFlags(db, simId, flagsPatch) {
  const t = nowIso();
```

```
  const sim = getSimulation(db, simId);
  if (!sim) throw new Error("Simulação não encontrada.");

  sim.flags = { ...(sim.flags || {}), ...(flagsPatch || {}) };
  sim.updatedAt = t;
  db.updatedAt = t;
  writeDbRaw(db);
  return db;
}

export function saveSimCorrection(db, simId, patch) {
  const t = nowIso();
  const sim = getSimulation(db, simId);
  if (!sim) throw new Error("Simulação não encontrada.");

  sim.correction = { ...(sim.correction || {}), ...(patch || {}) };
  sim.updatedAt = t;
  db.updatedAt = t;
  writeDbRaw(db);
  return db;
}

export function saveSimFinancing(db, simId, patch) {
  const t = nowIso();
  const sim = getSimulation(db, simId);
  if (!sim) throw new Error("Simulação não encontrada.");

  sim.financing = { ...(sim.financing || {}), ...(patch || {}) };
  sim.updatedAt = t;
  db.updatedAt = t;
  writeDbRaw(db);
  return db;
}

export { STORAGE_KEY };
```

js/calc.js

```
function pad2(n) {
  return String(n).padStart(2, "0");
}

export function toMonthKey(dateIsoOrMonthKey) {
  if (!dateIsoOrMonthKey) return null;
  const s = String(dateIsoOrMonthKey);
  if (/^\d{4}-\d{2}$/.test(s)) return s;
  if (/^\d{4}-\d{2}-\d{2}$/.test(s)) return s.slice(0, 7);
  return null;
}
```

```
export function monthKeyToLabel(monthKey) {
  const [y, m] = monthKey.split("-").map(Number);
  return `${pad2(m)}/${y}`;
}

export function addMonthsToMonthKey(monthKey, delta) {
  const [y0, m0] = monthKey.split("-").map(Number);
  const total = (y0 * 12 + (m0 - 1)) + delta;
  const y = Math.floor(total / 12);
  const m = (total % 12) + 1;
  return `${y}-${pad2(m)}`;
}

export function monthKeyCompare(a, b) {
  return a.localeCompare(b);
}

export function monthKeysBetween(startKey, endKey) {
  if (!startKey || !endKey) return [];
  if (startKey > endKey) return [];
  const out = [];
  let cur = startKey;
  while (cur <= endKey) {
    out.push(cur);
    cur = addMonthsToMonthKey(cur, 1);
  }
  return out;
}

function maxMonthKey(a, b) {
  if (!a) return b;
  if (!b) return a;
  return a > b ? a : b;
}

/* ============================
   Correção por índice (mensal)
   ============================ */

export function buildCumulativeFactors(startKey, endKey, monthlyPct,
enabled) {
  const factors = {};
  const keys = monthKeysBetween(startKey, endKey);

  if (!enabled) {
    for (const k of keys) factors[k] = 1;
    return factors;
  }

  if (keys.length === 0) return factors;
```

```
  factors[keys[0]] = 1;

  for (let i = 1; i < keys.length; i++) {
    const k = keys[i];
    const prev = keys[i - 1];
    const pct = Number(monthlyPct?.[k] ?? 0);
    const r = (Number.isFinite(pct) ? pct : 0) / 100;
    factors[k] = factors[prev] * (1 + r);
  }

  return factors;
}

/* ============================
   Fluxo de caixa → mês
   ============================ */

export function cashflowItemsInMonth(sim, monthKey) {
  const items = sim.cashflow || [];
  const out = [];

  for (const it of items) {
    const kind = it.kind;

    if (kind === "monthly") {
      const a = toMonthKey(it.startDate);
      const b = toMonthKey(it.endDate);
      if (a && b && monthKey >= a && monthKey <= b) out.push(it);
      continue;
    }

    if (kind === "balloon" || kind === "once") {
      const d = toMonthKey(it.dueDate);
      if (d === monthKey) out.push(it);
      continue;
    }
  }

  return out;
}

export function getLastCashflowMonth(sim) {
  let last = null;

  for (const it of sim.cashflow || []) {
    if (it.kind === "monthly") {
      last = maxMonthKey(last, toMonthKey(it.endDate));
    } else {
      last = maxMonthKey(last, toMonthKey(it.dueDate));
    }
  }
```

```
  return last;
}

/* ============================
   Financiamento (SAC / PRICE)
   ============================ */

export function annualToMonthlyRate(annualPct) {
  const a = (Number(annualPct || 0) / 100);
  if (!Number.isFinite(a) || a <= 0) return 0;
  return Math.pow(1 + a, 1 / 12) - 1;
}

export function scheduleSAC(pv, monthlyRate, n, startMonthKey) {
  const out = [];
  const PV = Math.max(0, Number(pv || 0));
  const i = Math.max(0, Number(monthlyRate || 0));
  const N = Math.max(1, Math.floor(Number(n || 1)));

  const amort = PV / N;
  let bal = PV;

  for (let k = 1; k <= N; k++) {
    const interest = bal * i;
    const installment = amort + interest;
    bal = Math.max(0, bal - amort);

    out.push({
      k,
      monthKey: addMonthsToMonthKey(startMonthKey, k - 1),
      installment,
      interest,
      amort,
      balance: bal,
    });
  }

  return out;
}

export function schedulePRICE(pv, monthlyRate, n, startMonthKey) {
  const out = [];
  const PV = Math.max(0, Number(pv || 0));
  const i = Math.max(0, Number(monthlyRate || 0));
  const N = Math.max(1, Math.floor(Number(n || 1)));

  let installment = 0;
  if (i === 0) {
    installment = PV / N;
  } else {
```

```
    const pow = Math.pow(1 + i, N);
    const k = (i * pow) / (pow - 1);
    installment = PV * k;
  }

  let bal = PV;

  for (let t = 1; t <= N; t++) {
    const interest = bal * i;
    const amort = installment - interest;
    bal = Math.max(0, bal - amort);

    out.push({
      k: t,
      monthKey: addMonthsToMonthKey(startMonthKey, t - 1),
      installment,
      interest,
      amort,
      balance: bal,
    });
  }

  return out;
}

/* ============================
   Resultado final (timeline)
   ============================ */

export function computeSimulationResults(sim, indexSeries) {
  const contractMonth = toMonthKey(sim.contractDate);
  if (!contractMonth) throw new Error("Data do contrato inválida.");

  const corrEnabled = Boolean(sim.correction?.enabled);
  const monthlyPct = indexSeries?.monthlyPct || {};

  const lastCashflow = getLastCashflowMonth(sim);
  const finStart = sim.financing?.enabled ?
toMonthKey(sim.financing?.startDate) : null;

  let endMonth = maxMonthKey(contractMonth, lastCashflow);
  if (sim.financing?.enabled && finStart) {
    const finEnd = addMonthsToMonthKey(finStart, Math.max(1,
Number(sim.financing.months || 1)) - 1);
    endMonth = maxMonthKey(endMonth, finEnd);
  }

  // Garantir pelo menos até mês da entrega
  endMonth = maxMonthKey(endMonth, toMonthKey(sim.deliveryDate));

  const factors = buildCumulativeFactors(contractMonth, endMonth,
```

```
monthlyPct, corrEnabled);

  const months = monthKeysBetween(contractMonth, endMonth);
  const timeline = [];

  // 1) Construir timeline com fluxo de caixa (sem financiamento).
  for (const m of months) {
    const idxPct = Number(monthlyPct?.[m] ?? 0);
    const factor = Number(factors?.[m] ?? 1);

    const contractValue = Number(sim.basePrice || 0);
    const contractCorrected = corrEnabled ? (contractValue * factor) :
contractValue;

    const monthItems = cashflowItemsInMonth(sim, m).map((it) => {
      const base = Number(it.amount || 0);
      const amount = (corrEnabled && it.applyCorrection) ? (base * factor) :
base;

      return {
        id: it.id,
        kind: it.kind,
        label: it.label,
        baseAmount: base,
        amount,
        applyCorrection: Boolean(it.applyCorrection),
        affectsPrincipal: Boolean(it.affectsPrincipal),
      };
    });

    const cashflowTotal = monthItems.reduce((acc, x) => acc + x.amount, 0);
    const cashflowPrincipal = monthItems.filter((x) =>
x.affectsPrincipal).reduce((acc, x) => acc + x.amount, 0);
    const cashflowExtra = monthItems.filter((x) => !
x.affectsPrincipal).reduce((acc, x) => acc + x.amount, 0);

    timeline.push({
      monthKey: m,
      monthLabel: monthKeyToLabel(m),

      indexPct: Number.isFinite(idxPct) ? idxPct : 0,
      factor,

      contractCorrected,

      cashflowItems: monthItems,
      cashflowTotal,
      cashflowPrincipal,
      cashflowExtra,

      finInstallment: 0,
```

```
      finInterest: 0,
      finAmort: 0,
      finBalance: null,
    });
  }

  // 2) Financiamento: calcular PV no mês de início, com base no saldo
corrigido menos principal pago antes.
  let financing = null;

  if (sim.financing?.enabled && finStart) {
    const annualPct = Number(sim.financing.annualRatePct || 0);
    const monthlyRate = annualToMonthlyRate(annualPct);
    const n = Math.max(1, Math.floor(Number(sim.financing.months || 1)));

    const idxStart = timeline.findIndex((x) => x.monthKey === finStart);
    if (idxStart >= 0) {
      const prePaidPrincipal = timeline
        .slice(0, idxStart)
        .reduce((acc, x) => acc + x.cashflowPrincipal, 0);

      const contractAtStart = timeline[idxStart].contractCorrected;
      const pv = Math.max(0, contractAtStart - prePaidPrincipal);

      const schedule =
        (sim.financing.type === "PRICE")
          ? schedulePRICE(pv, monthlyRate, n, finStart)
          : scheduleSAC(pv, monthlyRate, n, finStart);

      // Merge
      for (const s of schedule) {
        const row = timeline.find((x) => x.monthKey === s.monthKey);
        if (!row) continue;

        row.finInstallment = s.installment;
        row.finInterest = s.interest;
        row.finAmort = s.amort;
        row.finBalance = s.balance;
      }

      const totalInstallments = schedule.reduce((acc, x) => acc +
x.installment, 0);
      const totalInterest = schedule.reduce((acc, x) => acc + x.interest, 0);
      const totalAmort = schedule.reduce((acc, x) => acc + x.amort, 0);

      financing = {
        enabled: true,
        type: sim.financing.type,
        startMonthKey: finStart,
        months: n,
        annualRatePct: annualPct,
```

```
        monthlyRate,
        pv,
        prePaidPrincipal,
        contractAtStart,
        totalInstallments,
        totalInterest,
        totalAmort,
      };
    }
  }

  // 3) Totais (por "fase" simples)
  const totalCashflow = timeline.reduce((acc, x) => acc + x.cashflowTotal,
0);
  const totalCashflowPrincipal = timeline.reduce((acc, x) => acc +
x.cashflowPrincipal, 0);
  const totalCashflowExtra = timeline.reduce((acc, x) => acc +
x.cashflowExtra, 0);

  const totalFinInstallments = timeline.reduce((acc, x) => acc +
x.finInstallment, 0);
  const totalFinInterest = timeline.reduce((acc, x) => acc + x.finInterest,
0);

  // "Pago total" = cashflow (inclui extras) + parcelas do financiamento
  const totalPaid = totalCashflow + totalFinInstallments;

  const totals = {
    basePrice: Number(sim.basePrice || 0),
    totalCashflow,
    totalCashflowPrincipal,
    totalCashflowExtra,
    totalFinInstallments,
    totalFinInterest,
    totalPaid,
  };

  return { timeline, totals, financing, meta: { contractMonth, endMonth,
corrEnabled } };
}

/* ============================
   Opcional: fetch SGS (BCB)
   ============================ */

// Esta função é opcional/isolada. Ela NÃO é necessária para o app funcionar.
// Retorna uma lista: [{ data: "dd/MM/aaaa", valor: "0.21" }, ...]
export async function fetchBcbSgsSeries(seriesCode, startDate, endDate) {
  const code = String(seriesCode || "").trim();
  if (!code) throw new Error("Informe um código de série (SGS).");
```

```
  // BCB/SGS JSON:
  // https://api.bcb.gov.br/dados/serie/bcdata.sgs.{codigo_serie}/dados?
formato=json&dataInicial=dd/MM/aaaa&dataFinal=dd/MM/aaaa
  const toBr = (iso) => {
    if (!iso) return null;
    const [y, m, d] = iso.split("-");
    return `${d}/${m}/${y}`;
  };

  const di = toBr(startDate);
  const df = toBr(endDate);

  let url = `https://api.bcb.gov.br/dados/serie/bcdata.sgs.${code}/dados?
formato=json`;
  if (di) url += `&dataInicial=${encodeURIComponent(di)}`;
  if (df) url += `&dataFinal=${encodeURIComponent(df)}`;

  const res = await fetch(url, { method: "GET" });
  if (!res.ok) {
    const text = await res.text();
    throw new Error(`BCB/SGS falhou (${res.status}). ${text.slice(0, 200)}`);
  }

  const data = await res.json();
  if (!Array.isArray(data)) throw new Error("Resposta inesperada do SGS.");
  return data;
}

// Converte retorno SGS (datas dd/MM/aaaa com valor string) em mapa mensal
YYYY-MM -> percent
export function sgsJsonToMonthlyPctMap(sgsArray) {
  const out = {};
  for (const row of sgsArray || []) {
    const d = String(row.data || "");
    const v = Number(String(row.valor || "").replace(",", "."));
    if (!d || !Number.isFinite(v)) continue;

    // d = dd/MM/aaaa
    const [dd, mm, yyyy] = d.split("/");
    if (!yyyy || !mm) continue;

    const key = `${yyyy}-${mm}`;
    // Se for série diária, haverá repetição do mesmo mês. Aqui pegamos a
última ocorrência.
    out[key] = v;
  }
  return out;
}
```

js/risks.js

```
import { toMonthKey } from "./calc.js";

function containsItbi(sim) {
  const items = sim.cashflow || [];
  for (const it of items) {
    const label = String(it.label || "").toLowerCase();
    if (label.includes("itbi")) return true;
  }
  return false;
}

export function buildRiskFlags(project, sim) {
  const flags = [];

  const hasQuadro = Boolean(sim.flags?.hasQuadroResumo);
  const corrOk = Boolean(sim.flags?.brokerFeeConfirmed);
  const itbiOk = Boolean(sim.flags?.itbiProvisioned) || containsItbi(sim);
  const tol = Number(sim.toleranceDays || 0);

  if (!hasQuadro) {
    flags.push({
      id: "missing_quadro",
      severity: "bad",
      title: "Quadro-resumo não confirmado",
      detail: "Risco de faltar informação essencial (preço, índices,
corretagem, sistema de amortização etc.).",
    });
  }

  if (!corrOk) {
    flags.push({
      id: "broker_fee",
      severity: "warn",
      title: "Corretagem não confirmada",
      detail: "Confirme valor, condições e beneficiário; evite surpresa no
caixa.",
    });
  }

  if (!itbiOk) {
    flags.push({
      id: "itbi",
      severity: "warn",
      title: "ITBI não provisionado",
      detail: "Inclua ITBI e custos de registro/transferência no
planejamento.",
    });
  }
```

```
  if (tol > 180) {
    flags.push({
      id: "tolerance_days",
      severity: "bad",
      title: "Tolerância acima de 180 dias",
      detail: `Você definiu $
{tol} dias. Regra simples do simulador: acima de 180 acende alerta.`,
    });
  }

  // Heurística extra simples: correção habilitada sem série selecionada.
  if (sim.correction?.enabled && !sim.correction?.seriesId) {
    flags.push({
      id: "corr_missing_series",
      severity: "warn",
      title: "Correção habilitada sem série",
      detail: "Selecione ou cadastre uma série mensal para a correção
funcionar.",
    });
  }

  // Financiamento habilitado sem startDate
  if (sim.financing?.enabled && !toMonthKey(sim.financing?.startDate)) {
    flags.push({
      id: "fin_missing_start",
      severity: "warn",
      title: "Financiamento sem data de início",
      detail:
"Defina a data de início do financiamento (mês) para simular as parcelas.",
    });
  }

  return flags;
}
```

js/render.js

```
import { monthKeyToLabel } from "./calc.js";

function esc(s) {
  return String(s ?? "")
    .replaceAll("&", "&amp;")
    .replaceAll("<", "&lt;")
    .replaceAll(">", "&gt;")
    .replaceAll('"', "&quot;");
}

export function fmtBRL(n) {
  const x = Number(n || 0);
  return x.toLocaleString("pt-BR", { style: "currency", currency: "BRL" });
```

```
}

export function fmtPct(n) {
  const x = Number(n || 0);
  return `${x.toLocaleString("pt-BR", { minimumFractionDigits: 2,
maximumFractionDigits: 2 })}%`;
}

export function setActiveNav(hash) {
  document.querySelectorAll("[data-nav]").forEach((a) => {
    const href = a.getAttribute("href") || "";
    a.classList.toggle("active", href === hash);
  });
}

export function showView(viewId) {
  document.querySelectorAll(".view").forEach((v) =>
v.classList.add("hidden"));
  const el = document.getElementById(viewId);
  if (el) el.classList.remove("hidden");
}

export function renderStatus(project, sim) {
  document.getElementById("status-project").textContent = project ?
project.name : "—";
  document.getElementById("status-sim").textContent = sim ? sim.name : "—";
}

export function renderProjectsList(container, projects, simsCountByProject,
selectedProjectId) {
  if (!projects.length) {
    container.innerHTML = `<div class="notice">Nenhum projeto. Crie um no
formulário ao lado.</div>`;
    return;
  }

  container.innerHTML = projects
    .map((p) => {
      const count = simsCountByProject[p.id] || 0;
      const selected = p.id === selectedProjectId;

      return `
        <div class="row" style="justify-content: space-between; border-
bottom: 1px solid rgba(255,255,255,0.10); padding: 10px 0;">
          <div>
            <div><b>${esc(p.name)}</b></div>
            <div class="hint">${esc(p.city)} / ${esc(p.uf)} • $
{esc(p.developer)} • ${count} simulações</div>
          </div>

          <div class="row">
```

```
            <button class="btn btn-ghost" data-action="project_select" data-
id="${esc(p.id)}">
              ${selected ? "Selecionado" : "Selecionar"}
            </button>
            <button class="btn btn-ghost" data-action="project_edit" data-
id="${esc(p.id)}">Editar</button>
            <button class="btn btn-danger" data-action="project_delete" data-
id="${esc(p.id)}">Excluir</button>
          </div>
        </div>
      `;
    })
    .join("");
}

export function renderSimsList(container, sims, selectedSimId) {
  if (!sims.length) {
    container.innerHTML = `<div class="notice">Nenhuma simulação neste
projeto. Crie uma no formulário ao lado.</div>`;
    return;
  }

  container.innerHTML = sims
    .map((s) => {
      const selected = s.id === selectedSimId;
      const contract = s.contractDate ? new
Date(s.contractDate).toLocaleDateString("pt-BR") : "—";
      const delivery = s.deliveryDate ? new
Date(s.deliveryDate).toLocaleDateString("pt-BR") : "—";

      return `
        <div class="row" style="justify-content: space-between; border-
bottom: 1px solid rgba(255,255,255,0.10); padding: 10px 0;">
          <div>
            <div><b>${esc(s.name)}</b></div>
            <div class="hint">
              Contrato: ${esc(contract)} • Entrega: ${esc(delivery)} •
Tolerância: ${Number(s.toleranceDays || 0)}d
              <br/>
              Preço-base: ${fmtBRL(s.basePrice)}
            </div>
          </div>

          <div class="row">
            <button class="btn btn-ghost" data-action="sim_select" data-id="$
{esc(s.id)}">
              ${selected ? "Selecionado" : "Selecionar"}
            </button>
            <button class="btn btn-ghost" data-action="sim_edit" data-id="$
{esc(s.id)}">Editar</button>
            <button class="btn btn-danger" data-action="sim_delete" data-
```

```
id="${esc(s.id)}">Excluir</button>
          </div>
        </div>
      `;
    })
    .join("");
}

export function renderCashflowTable(container, sim) {
  const items = sim.cashflow || [];
  if (!items.length) {
    container.innerHTML = `<div class="notice">Nenhum item. Crie o primeiro
abaixo.</div>`;
    return;
  }

  const rows = items
    .slice()
    .sort((a, b) => {
      const ma = a.kind === "monthly" ? (a.startDate || "") : (a.dueDate ||
"");
      const mb = b.kind === "monthly" ? (b.startDate || "") : (b.dueDate ||
"");
      return ma.localeCompare(mb);
    })
    .map((it) => {
      const when = it.kind === "monthly"
        ? `${esc(new Date(it.startDate).toLocaleDateString("pt-BR"))} → $
{esc(new Date(it.endDate).toLocaleDateString("pt-BR"))}`
        : `${esc(new Date(it.dueDate).toLocaleDateString("pt-BR"))}`;

      return `
        <tr>
          <td class="mono">${esc(it.kind)}</td>
          <td>${esc(it.label)}</td>
          <td class="mono">${when}</td>
          <td class="mono">${fmtBRL(it.amount)}</td>
          <td class="mono">${it.applyCorrection ? "sim" : "não"}</td>
          <td class="mono">${it.affectsPprincipal ? "sim" : "não"}</td>
          <td>
            <div class="row">
              <button class="btn btn-ghost" data-action="cashflow_edit" data-
id="${esc(it.id)}">Editar</button>
              <button class="btn btn-danger" data-action="cashflow_delete"
data-id="${esc(it.id)}">Excluir</button>
            </div>
          </td>
        </tr>
      `;
    })
    .join("");
```

```
  container.innerHTML = `
    <div class="table-wrap">
      <table>
        <thead>
          <tr>
            <th>Tipo</th>
            <th>Descrição</th>
            <th>Quando</th>
            <th>Valor base</th>
            <th>Corrige?</th>
            <th>Abate principal?</th>
            <th>Ações</th>
          </tr>
        </thead>
        <tbody>${rows}</tbody>
      </table>
    </div>
  `;
}

export function renderIndexSeriesOptions(selectEl, seriesList, selectedId) {
  if (!seriesList.length) {
    selectEl.innerHTML = `<option value="">(Sem séries)</option>`;
    return;
  }

  selectEl.innerHTML = seriesList
    .map((s) => `<option value="${esc(s.id)}" ${s.id === selectedId ?
"selected" : ""}>${esc(s.name)}</option>`)
    .join("");
}

export function renderTotals(container, results) {
  const t = results.totals;
  const fin = results.financing;

  container.innerHTML = `
    <div class="row">
      <span class="badge">Preço-base: <b>${fmtBRL(t.basePrice)}</b></span>
      <span class="badge">Total cashflow: <b>${fmtBRL(t.totalCashflow)}</b></
span>
      <span class="badge">Cashflow (principal): <b>$
{fmtBRL(t.totalCashflowPrincipal)}</b></span>
      <span class="badge">Cashflow (extras): <b>$
{fmtBRL(t.totalCashflowExtra)}</b></span>
    </div>

    <div class="row" style="margin-top: 10px;">
      <span class="badge">Total financiamento (parcelas): <b>$
{fmtBRL(t.totalFinInstallments)}</b></span>
```

```
      <span class="badge">Total juros (fin): <b>${fmtBRL(t.totalFinInterest)}
</b></span>
      <span class="badge good">Pago total (cashflow + fin): <b>$
{fmtBRL(t.totalPaid)}</b></span>
    </div>

    ${
      fin
        ? `
          <div style="margin-top: 12px;">
            <div class="hint"><b>Financiamento</b></div>
            <div class="row">
              <span class="badge">Sistema: <b>${esc(fin.type)}</b></span>
              <span class="badge">Início: <b>$
{monthKeyToLabel(fin.startMonthKey)}</b></span>
              <span class="badge">Prazo: <b>${Number(fin.months)}m</b></span>
              <span class="badge">Taxa a.a.: <b>${fmtPct(fin.annualRatePct)}
</b></span>
              <span class="badge">Taxa a.m. (eq.): <b>$
{fmtPct(fin.monthlyRate * 100)}</b></span>
            </div>
            <div class="row" style="margin-top: 6px;">
              <span class="badge">Contrato corrigido no início: <b>$
{fmtBRL(fin.contractAtStart)}</b></span>
              <span class="badge">Principal pago antes: <b>$
{fmtBRL(fin.prePaidPrincipal)}</b></span>
              <span class="badge warn">Saldo p/ financiar (PV): <b>$
{fmtBRL(fin.pv)}</b></span>
            </div>
          </div>
        `
        : `<div class="hint" style="margin-top: 12px;">Financiamento
desabilitado.</div>`
    }
  `;
}

export function renderRisks(container, riskFlags) {
  if (!riskFlags.length) {
    container.innerHTML = `<span class="badge good">Nenhum alerta (pelas
regras atuais).</span>`;
    return;
  }

  container.innerHTML = riskFlags
    .map((r) => {
      const cls = r.severity === "bad" ? "bad" : (r.severity === "warn" ?
"warn" : "good");
      return `
        <div style="margin-bottom: 10px;">
          <div class="badge ${cls}"><b>${esc(r.title)}</b></div>
```

```
          <div class="hint">${esc(r.detail)}</div>
        </div>
      `;
    })
    .join("");
}

export function renderTimeline(container, results) {
  const rows = results.timeline
    .map((m) => {
      const items = (m.cashflowItems || [])
        .map((x) => `${esc(x.label)} (${fmtBRL(x.amount)})`)
        .join("<br/>");

      return `
        <tr>
          <td class="mono">${esc(m.monthKey)}</td>
          <td class="mono">${esc(m.monthLabel)}</td>
          <td class="mono">${fmtPct(m.indexPct)}</td>
          <td class="mono">${m.factor.toFixed(6)}</td>
          <td class="mono">${fmtBRL(m.contractCorrected)}</td>
          <td>${items || "<span class='hint'>—</span>"}</td>
          <td class="mono">${fmtBRL(m.cashflowTotal)}</td>
          <td class="mono">${fmtBRL(m.cashflowPrincipal)}</td>
          <td class="mono">${fmtBRL(m.cashflowExtra)}</td>
          <td class="mono">${fmtBRL(m.finInstallment)}</td>
          <td class="mono">${fmtBRL(m.finInterest)}</td>
          <td class="mono">${fmtBRL(m.finAmort)}</td>
          <td class="mono">${m.finBalance === null ? "—" :
fmtBRL(m.finBalance)}</td>
        </tr>
      `;
    })
    .join("");

  container.innerHTML = `
    <div class="table-wrap">
      <table>
        <thead>
          <tr>
            <th>Mês</th>
            <th>Label</th>
            <th>Índice (%)</th>
            <th>Fator acum.</th>
            <th>Contrato (corr.)</th>
            <th>Itens</th>
            <th>Cashflow</th>
            <th>CF (principal)</th>
            <th>CF (extras)</th>
            <th>Prestação fin</th>
            <th>Juros</th>
```

```
            <th>Amort</th>
            <th>Saldo</th>
          </tr>
        </thead>
        <tbody>${rows}</tbody>
      </table>
    </div>
  `;
}

export function toast(msg) {
  const el = document.getElementById("toast");
  if (!el) return;
  el.textContent = msg;
  el.classList.remove("hidden");

  window.clearTimeout(toast._t);
  toast._t = window.setTimeout(() => el.classList.add("hidden"), 2200);
}
```

js/app.js

```
import {
  ensureDb,
  getDb,
  resetDbToSeed,
  exportDbJson,
  importDbJson,

  listProjects,
  getProject,
  saveProject,
  deleteProject,

  listSimulationsByProject,
  getSimulation,
  saveSimulation,
  deleteSimulation,

  listIndexSeries,
  getIndexSeries,
  upsertIndexSeries,

  saveCashflowItem,
  deleteCashflowItem,

  saveSimFlags,
  saveSimCorrection,
  saveSimFinancing,
} from "./storage.js";
```

```
import {
  computeSimulationResults,
  annualToMonthlyRate,
  fetchBcbSgsSeries,
  sgsJsonToMonthlyPctMap,
} from "./calc.js";

import { buildRiskFlags } from "./risks.js";
import {
  setActiveNav,
  showView,
  renderStatus,
  renderProjectsList,
  renderSimsList,
  renderCashflowTable,
  renderIndexSeriesOptions,
  renderTotals,
  renderRisks,
  renderTimeline,
  toast,
  fmtPct,
} from "./render.js";

import { FAQ_HTML } from "./faq.js";

/* ============================
   Estado básico
   ============================ */

const state = {
  db: ensureDb(),
  selectedProjectId: null,
  selectedSimId: null,
  lastResultsBySimId: {}, // cache em memória
};

function refreshDb() {
  state.db = getDb();
}

function getSelectedProject() {
  if (!state.selectedProjectId) return null;
  return getProject(state.db, state.selectedProjectId);
}

function getSelectedSim() {
  if (!state.selectedSimId) return null;
  return getSimulation(state.db, state.selectedSimId);
}
```

```
function computeSimsCountByProject(db) {
  const map = {};
  for (const s of db.simulations || []) {
    map[s.projectId] = (map[s.projectId] || 0) + 1;
  }
  return map;
}

/* ============================
   Render principal por rota
   ============================ */

function route() {
  refreshDb();

  const hash = (location.hash || "#projects").split("?")[0];
  setActiveNav(hash);

  const project = getSelectedProject();
  const sim = getSelectedSim();
  renderStatus(project, sim);

  if (hash === "#projects") {
    showView("view-projects");
    renderProjects();
    return;
  }

  if (hash === "#sims") {
    showView("view-sims");
    renderSims();
    return;
  }

  if (hash === "#editor") {
    showView("view-editor");
    renderEditor();
    return;
  }

  if (hash === "#results") {
    showView("view-results");
    renderResults();
    return;
  }

  if (hash === "#faq") {
    showView("view-faq");
    document.getElementById("faq-content").innerHTML = FAQ_HTML;
    return;
  }
```

```
  location.hash = "#projects";
}

function renderProjects() {
  const container = document.getElementById("projects-list");
  const projects = listProjects(state.db);
  const simsCount = computeSimsCountByProject(state.db);

  renderProjectsList(container, projects, simsCount,
state.selectedProjectId);

  // Ajustar título do form conforme seleção de edição
  // (os valores do form são preenchidos ao clicar em "Editar")
}

function renderSims() {
  const empty = document.getElementById("sims-empty");
  const content = document.getElementById("sims-content");

  const project = getSelectedProject();
  if (!project) {
    empty.classList.remove("hidden");
    content.classList.add("hidden");
    return;
  }

  empty.classList.add("hidden");
  content.classList.remove("hidden");

  const sims = listSimulationsByProject(state.db, project.id);
  renderSimsList(document.getElementById("sims-list"), sims,
state.selectedSimId);
}

function renderEditor() {
  const empty = document.getElementById("editor-empty");
  const content = document.getElementById("editor-content");

  const sim = getSelectedSim();
  if (!sim) {
    empty.classList.remove("hidden");
    content.classList.add("hidden");
    return;
  }

  empty.classList.add("hidden");
  content.classList.remove("hidden");

  // Flags
  document.getElementById("flag-quadro").checked =
```

```
Boolean(sim.flags?.hasQuadroResumo);
  document.getElementById("flag-corretagem").checked =
Boolean(sim.flags?.brokerFeeConfirmed);
  document.getElementById("flag-itbi").checked =
Boolean(sim.flags?.itbiProvisioned);

  // Cashflow
  renderCashflowTable(document.getElementById("cashflow-table"), sim);
  syncCashflowKindVisibility();

  // Correção
  const seriesList = listIndexSeries(state.db);
  renderIndexSeriesOptions(
    document.getElementById("corr-series"),
    seriesList,
    sim.correction?.seriesId || (seriesList[0]?.id || "")
  );

  document.getElementById("corr-enabled").checked =
Boolean(sim.correction?.enabled);
  document.getElementById("corr-note").value = sim.correction?.note || "";
  document.getElementById("corr-json").value = "";

  // Financiamento
  document.getElementById("fin-enabled").checked =
Boolean(sim.financing?.enabled);
  document.getElementById("fin-type").value = sim.financing?.type || "SAC";
  document.getElementById("fin-months").value = String(sim.financing?.months
|| 240);
  document.getElementById("fin-annualRate").value =
String(sim.financing?.annualRatePct || 10.0);
  document.getElementById("fin-startDate").value = sim.financing?.startDate
|| sim.deliveryDate || "";

  updateFinRateHint();
}

function renderResults() {
  const empty = document.getElementById("results-empty");
  const content = document.getElementById("results-content");

  const project = getSelectedProject();
  const sim = getSelectedSim();
  if (!project || !sim) {
    empty.classList.remove("hidden");
    content.classList.add("hidden");
    return;
  }

  // Precisamos de resultados calculados (via Recalcular) – mas se não tiver
cache, calcula na hora.
```

```
  const results = state.lastResultsBySimId[sim.id] || calcNow(sim);

  empty.classList.add("hidden");
  content.classList.remove("hidden");

  renderTotals(document.getElementById("results-totals"), results);

  const risks = buildRiskFlags(project, sim);
  renderRisks(document.getElementById("results-risks"), risks);

  renderTimeline(document.getElementById("results-timeline"), results);
}

function calcNow(sim) {
  const seriesId = sim.correction?.seriesId;
  const series = seriesId ? getIndexSeries(state.db, seriesId) : null;
  const res = computeSimulationResults(sim, series);
  state.lastResultsBySimId[sim.id] = res;
  return res;
}

/* ============================
   Helpers de formulário
   ============================ */

function clearProjectForm() {
  document.getElementById("project-form-title").textContent = "Novo projeto";
  document.getElementById("project-id").value = "";
  document.getElementById("project-name").value = "";
  document.getElementById("project-city").value = "";
  document.getElementById("project-uf").value = "";
  document.getElementById("project-developer").value = "";
}

function clearSimForm() {
  document.getElementById("sim-form-title").textContent = "Nova simulação";
  document.getElementById("sim-id").value = "";
  document.getElementById("sim-name").value = "";
  document.getElementById("sim-contractDate").value = "";
  document.getElementById("sim-deliveryDate").value = "";
  document.getElementById("sim-toleranceDays").value = "180";
  document.getElementById("sim-basePrice").value = "";
}

function clearCashflowForm() {
  document.getElementById("cashflow-form-title").textContent = "Novo item";
  document.getElementById("cashflow-id").value = "";
  document.getElementById("cashflow-kind").value = "monthly";
  document.getElementById("cashflow-label").value = "";
  document.getElementById("cashflow-amount").value = "";
  document.getElementById("cashflow-startDate").value = "";
```

```
  document.getElementById("cashflow-endDate").value = "";
  document.getElementById("cashflow-dueDate").value = "";
  document.getElementById("cashflow-corr").checked = true;
  document.getElementById("cashflow-principal").checked = true;
  syncCashflowKindVisibility();
}

function syncCashflowKindVisibility() {
  const kind = document.getElementById("cashflow-kind").value;
  const start = document.getElementById("cashflow-startDate");
  const end = document.getElementById("cashflow-endDate");
  const due = document.getElementById("cashflow-dueDate");

  const isMonthly = kind === "monthly";
  start.disabled = !isMonthly;
  end.disabled = !isMonthly;
  due.disabled = isMonthly;
}

function updateFinRateHint() {
  const annual = Number(document.getElementById("fin-annualRate").value ||
0);
  const im = annualToMonthlyRate(annual);
  document.getElementById("fin-rate-hint").textContent =
    `Taxa mensal equivalente (aprox.): ${fmtPct(im * 100)} a.m.`;
}

/* ============================
   Eventos: navegação e ações
   ============================ */

window.addEventListener("hashchange", route);

document.addEventListener("click", async (ev) => {
  const btn = ev.target.closest("[data-action]");
  if (!btn) return;

  const action = btn.getAttribute("data-action");
  const id = btn.getAttribute("data-id");

  try {
    if (action === "project_select") {
      state.selectedProjectId = id;
      state.selectedSimId = null;
      toast("Projeto selecionado.");
      if (location.hash !== "#sims") location.hash = "#sims";
      else route();
      return;
    }

    if (action === "project_edit") {
```

```
      const p = getProject(state.db, id);
      if (!p) return;

      document.getElementById("project-form-title").textContent = "Editar
projeto";
      document.getElementById("project-id").value = p.id;
      document.getElementById("project-name").value = p.name;
      document.getElementById("project-city").value = p.city;
      document.getElementById("project-uf").value = p.uf;
      document.getElementById("project-developer").value = p.developer;
      toast("Editando projeto.");
      return;
    }

    if (action === "project_delete") {
      if (!confirm("Excluir este projeto e TODAS as simulações dele?"))
return;

      const db = getDb();
      deleteProject(db, id);

      // Ajustar seleção
      if (state.selectedProjectId === id) {
        state.selectedProjectId = null;
        state.selectedSimId = null;
      }

      toast("Projeto excluído.");
      route();
      return;
    }

    if (action === "sim_select") {
      state.selectedSimId = id;
      toast("Simulação selecionada.");
      if (location.hash !== "#editor") location.hash = "#editor";
      else route();
      return;
    }

    if (action === "sim_edit") {
      const s = getSimulation(state.db, id);
      if (!s) return;

      document.getElementById("sim-form-title").textContent = "Editar
simulação";
      document.getElementById("sim-id").value = s.id;
      document.getElementById("sim-name").value = s.name;
      document.getElementById("sim-contractDate").value = s.contractDate;
      document.getElementById("sim-deliveryDate").value = s.deliveryDate;
      document.getElementById("sim-toleranceDays").value =
```

```
String(s.toleranceDays || 0);
      document.getElementById("sim-basePrice").value = String(s.basePrice ||
0);
      toast("Editando simulação.");
      return;
    }

    if (action === "sim_delete") {
      if (!confirm("Excluir esta simulação?")) return;

      const db = getDb();
      deleteSimulation(db, id);

      if (state.selectedSimId === id) {
        state.selectedSimId = null;
      }
      toast("Simulação excluída.");
      route();
      return;
    }

    if (action === "cashflow_edit") {
      const sim = getSelectedSim();
      if (!sim) return;

      const it = (sim.cashflow || []).find((x) => x.id === id);
      if (!it) return;

      document.getElementById("cashflow-form-title").textContent = "Editar
item";
      document.getElementById("cashflow-id").value = it.id;
      document.getElementById("cashflow-kind").value = it.kind;
      document.getElementById("cashflow-label").value = it.label;
      document.getElementById("cashflow-amount").value = String(it.amount);
      document.getElementById("cashflow-startDate").value = it.startDate ||
"";
      document.getElementById("cashflow-endDate").value = it.endDate || "";
      document.getElementById("cashflow-dueDate").value = it.dueDate || "";
      document.getElementById("cashflow-corr").checked =
Boolean(it.applyCorrection);
      document.getElementById("cashflow-principal").checked =
Boolean(it.affectsPrincipal);

      syncCashflowKindVisibility();
      toast("Editando item.");
      return;
    }

    if (action === "cashflow_delete") {
      if (!confirm("Excluir este item?")) return;
```

```
      const sim = getSelectedSim();
      if (!sim) return;

      const db = getDb();
      deleteCashflowItem(db, sim.id, id);
      toast("Item excluído.");
      route();
      return;
    }
  } catch (err) {
    alert(err.message || String(err));
  }
});

/* ============================
   Form submits
   ============================ */

document.getElementById("project-form").addEventListener("submit", (ev) => {
  ev.preventDefault();

  try {
    const db = getDb();
    const payload = {
      id: document.getElementById("project-id").value || null,
      name: document.getElementById("project-name").value,
      city: document.getElementById("project-city").value,
      uf: document.getElementById("project-uf").value,
      developer: document.getElementById("project-developer").value,
    };

    saveProject(db, payload);
    clearProjectForm();
    toast("Projeto salvo.");
    route();
  } catch (err) {
    alert(err.message || String(err));
  }
});

document.getElementById("sim-form").addEventListener("submit", (ev) => {
  ev.preventDefault();

  const project = getSelectedProject();
  if (!project) {
    alert("Selecione um projeto primeiro.");
    return;
  }

  try {
    const db = getDb();
```

```
    const payload = {
      id: document.getElementById("sim-id").value || null,
      name: document.getElementById("sim-name").value,
      contractDate: document.getElementById("sim-contractDate").value,
      deliveryDate: document.getElementById("sim-deliveryDate").value,
      toleranceDays: document.getElementById("sim-toleranceDays").value,
      basePrice: document.getElementById("sim-basePrice").value,
    };

    saveSimulation(db, project.id, payload);
    clearSimForm();
    toast("Simulação salva.");
    route();
  } catch (err) {
    alert(err.message || String(err));
  }
});

document.getElementById("cashflow-form").addEventListener("submit", (ev) => {
  ev.preventDefault();

  const sim = getSelectedSim();
  if (!sim) {
    alert("Selecione uma simulação primeiro.");
    return;
  }

  try {
    const kind = document.getElementById("cashflow-kind").value;

    const payload = {
      id: document.getElementById("cashflow-id").value || null,
      kind,
      label: document.getElementById("cashflow-label").value,
      amount: document.getElementById("cashflow-amount").value,
      startDate: document.getElementById("cashflow-startDate").value || null,
      endDate: document.getElementById("cashflow-endDate").value || null,
      dueDate: document.getElementById("cashflow-dueDate").value || null,
      applyCorrection: document.getElementById("cashflow-corr").checked,
      affectsPrincipal: document.getElementById("cashflow-
principal").checked,
    };

    // validação simples por tipo
    if (kind === "monthly") {
      if (!payload.startDate || !payload.endDate) throw new Error("Mensal
precisa de início e fim.");
    } else {
      if (!payload.dueDate) throw new Error("Balão/Único precisa de
vencimento.");
      payload.startDate = null;
```

```
      payload.endDate = null;
    }

    const db = getDb();
    saveCashflowItem(db, sim.id, payload);

    clearCashflowForm();
    toast("Item salvo.");
    route();
  } catch (err) {
    alert(err.message || String(err));
  }
});

/* ============================
   Botões utilitários
   ============================ */

document.getElementById("project-cancel").addEventListener("click",
clearProjectForm);
document.getElementById("sim-cancel").addEventListener("click",
clearSimForm);

document.getElementById("cashflow-cancel").addEventListener("click",
clearCashflowForm);
document.getElementById("cashflow-kind").addEventListener("change",
syncCashflowKindVisibility);

document.getElementById("fin-annualRate").addEventListener("input",
updateFinRateHint);

/* Flags */
document.getElementById("btn-save-flags").addEventListener("click", () => {
  const sim = getSelectedSim();
  if (!sim) return;

  try {
    const db = getDb();
    saveSimFlags(db, sim.id, {
      hasQuadroResumo: document.getElementById("flag-quadro").checked,
      brokerFeeConfirmed: document.getElementById("flag-corretagem").checked,
      itbiProvisioned: document.getElementById("flag-itbi").checked,
    });

    toast("Checklist salvo.");
    route();
  } catch (err) {
    alert(err.message || String(err));
  }
});
```

```
/* Correção */
document.getElementById("btn-save-corr").addEventListener("click", () => {
  const sim = getSelectedSim();
  if (!sim) return;

  try {
    const db = getDb();
    saveSimCorrection(db, sim.id, {
      enabled: document.getElementById("corr-enabled").checked,
      seriesId: document.getElementById("corr-series").value || null,
      note: document.getElementById("corr-note").value || "",
    });

    toast("Correção salva.");
    route();
  } catch (err) {
    alert(err.message || String(err));
  }
});

document.getElementById("btn-corr-apply").addEventListener("click", () => {
  const sim = getSelectedSim();
  if (!sim) return;

  try {
    const selectedSeriesId = document.getElementById("corr-series").value;
    if (!selectedSeriesId) throw new Error("Selecione uma série primeiro.");

    const text = document.getElementById("corr-json").value.trim();
    if (!text) throw new Error("Cole um JSON no campo.");

    const parsed = JSON.parse(text);
    if (!parsed || typeof parsed !== "object") throw new Error("JSON
inválido.");

    // Validação mínima: chaves YYYY-MM e valores numéricos.
    const monthlyPct = {};
    for (const [k, v] of Object.entries(parsed)) {
      if (!/^\d{4}-\d{2}$/.test(k)) continue;
      const num = Number(v);
      if (!Number.isFinite(num)) continue;
      monthlyPct[k] = num;
    }

    const db = getDb();
    const series = getIndexSeries(db, selectedSeriesId);
    if (!series) throw new Error("Série não encontrada.");

    upsertIndexSeries(db, {
      ...series,
      note: document.getElementById("corr-note").value || series.note,
```

```
      monthlyPct: { ...(series.monthlyPct || {}), ...monthlyPct },
      source: "manual",
    });

    toast("Série atualizada.");
    document.getElementById("corr-json").value = "";
    route();
  } catch (err) {
    alert(err.message || String(err));
  }
});

/* Financiamento */
document.getElementById("btn-save-fin").addEventListener("click", () => {
  const sim = getSelectedSim();
  if (!sim) return;

  try {
    const db = getDb();
    saveSimFinancing(db, sim.id, {
      enabled: document.getElementById("fin-enabled").checked,
      type: document.getElementById("fin-type").value,
      months: Number(document.getElementById("fin-months").value || 1),
      annualRatePct: Number(document.getElementById("fin-annualRate").value
|| 0),
      startDate: document.getElementById("fin-startDate").value ||
sim.deliveryDate,
    });

    toast("Financiamento salvo.");
    route();
  } catch (err) {
    alert(err.message || String(err));
  }
});

/* Recalcular */
document.getElementById("btn-recalc").addEventListener("click", () => {
  const sim = getSelectedSim();
  if (!sim) return;

  try {
    const res = calcNow(sim);
    toast("Recalculado.");
    location.hash = "#results";
  } catch (err) {
    alert(err.message || String(err));
  }
});

/* BCB fetch (opcional) */
```

```
document.getElementById("btn-bcb-fetch").addEventListener("click", async ()
=> {
  const sim = getSelectedSim();
  if (!sim) return;

  try {
    const code = document.getElementById("bcb-seriesCode").value;
    const start = document.getElementById("bcb-start").value || null;
    const end = document.getElementById("bcb-end").value || null;

    const arr = await fetchBcbSgsSeries(code, start, end);
    const monthlyPct = sgsJsonToMonthlyPctMap(arr);

    const db = getDb();
    const created = {
      id: null,
      name: `BCB/SGS ${code}`,
      note: `Importado do SGS • ${start || "—"} → ${end || "—"}`,
      source: "bcb_sgs",
      monthlyPct,
    };

    upsertIndexSeries(db, created);
    toast("Série do BCB salva.");

    // Atualizar dropdown e selecionar a nova série
    refreshDb();
    const all = listIndexSeries(state.db);
    const justCreated = all.find((x) => x.name === `BCB/SGS ${code}`);
    if (justCreated) {
      document.getElementById("corr-series").value = justCreated.id;
      document.getElementById("corr-note").value = justCreated.note || "";
    }
  } catch (err) {
    alert(err.message || String(err));
  }
});

/* Export/Import (topbar) */
document.getElementById("btn-export").addEventListener("click", () => {
  try {
    const json = exportDbJson();
    downloadJson(json, "apto_sim_db_v1.json");
    toast("Export gerado.");
  } catch (err) {
    alert(err.message || String(err));
  }
});

document.getElementById("btn-reset").addEventListener("click", () => {
  if (!confirm("Resetar para o exemplo? Isso apaga os dados atuais."))
```

```
return;
  try {
    resetDbToSeed();
    state.selectedProjectId = null;
    state.selectedSimId = null;
    state.lastResultsBySimId = {};
    toast("Resetado.");
    location.hash = "#projects";
    route();
  } catch (err) {
    alert(err.message || String(err));
  }
});

document.getElementById("file-import").addEventListener("change", async (ev)
=> {
  const file = ev.target.files?.[0];
  if (!file) return;

  try {
    const text = await file.text();
    importDbJson(text);

    state.selectedProjectId = null;
    state.selectedSimId = null;
    state.lastResultsBySimId = {};
    toast("Importado com sucesso.");

    location.hash = "#projects";
    route();
  } catch (err) {
    alert(err.message || String(err));
  } finally {
    ev.target.value = "";
  }
});

/* Export/Import (Resultados) */
document.getElementById("btn-export-inline").addEventListener("click", () =>
{
  document.getElementById("export-json").value = exportDbJson();
  toast("JSON gerado.");
});

document.getElementById("btn-copy-json").addEventListener("click", async ()
=> {
  const text = document.getElementById("export-json").value.trim();
  if (!text) {
    alert("Gere o JSON primeiro.");
    return;
  }
```

```
  await navigator.clipboard.writeText(text);
  toast("Copiado.");
});

document.getElementById("btn-import-inline").addEventListener("click", () =>
{
  const text = document.getElementById("import-json").value.trim();
  if (!text) {
    alert("Cole um JSON para importar.");
    return;
  }

  if (!confirm("Importar e substituir TODO o banco local?")) return;

  try {
    importDbJson(text);
    state.selectedProjectId = null;
    state.selectedSimId = null;
    state.lastResultsBySimId = {};
    toast("Importado com sucesso.");
    location.hash = "#projects";
    route();
  } catch (err) {
    alert(err.message || String(err));
  }
});

function downloadJson(text, filename) {
  const blob = new Blob([text], { type: "application/json;charset=utf-8" });
  const url = URL.createObjectURL(blob);

  const a = document.createElement("a");
  a.href = url;
  a.download = filename;
  document.body.appendChild(a);
  a.click();
  a.remove();

  URL.revokeObjectURL(url);
}

/* ============================
   Boot
   ============================ */

(function boot() {
  // Se o seed tem um projeto e simulação, seleciona automaticamente para
facilitar primeira execução.
  const db = getDb();
  if (db.projects?.length) state.selectedProjectId = db.projects[0].id;
  if (db.simulations?.length) state.selectedSimId = db.simulations[0].id;
```

```
  // Pré-carrega FAQ
  document.getElementById("faq-content").innerHTML = FAQ_HTML;

  // Forms iniciais (datas default)
  if (!document.getElementById("sim-contractDate").value) {
    // sugestão: hoje
    const today = new Date();
    const iso = today.toISOString().slice(0, 10);
    document.getElementById("sim-contractDate").value = iso;
    document.getElementById("sim-deliveryDate").value = iso;
  }

  clearProjectForm();
  clearSimForm();
  clearCashflowForm();

  route();
})();
```

js/risks.js e js/render.js já incluídos acima

(Arquivos completos já apresentados nas seções correspondentes.)

Como rodar no VSCode Live Server

1. Crie uma pasta (ex.: `apto-sim`) e coloque os arquivos exatamente na árvore mostrada.
2. Abra a pasta no VSCode.
3. Instale a extensão **Live Server** (se ainda não tiver).
4. Clique com botão direito em `index.html` → **Open with Live Server**.
5. O app abre no navegador e grava no `localStorage` na chave `APTO_SIM_DB_V1`.
   Lembrete: `localStorage` exige serialização e pode ter indisponibilidade em alguns contextos (como navegação privada). [6]

---

[1] https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/import
https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Statements/import

[2] https://developer.mozilla.org/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API
https://developer.mozilla.org/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API

[3] [10] https://repositorio.unb.br/bitstream/10482/43830/1/2022_LucianoOliveiradeMoraes.pdf
https://repositorio.unb.br/bitstream/10482/43830/1/2022_LucianoOliveiradeMoraes.pdf

[4] [14] https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13786.htm
https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13786.htm

[5] https://portalibre.fgv.br/incc
https://portalibre.fgv.br/incc

[6] [15] https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API
https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API

7 8 https://lgpd.tcm.sp.gov.br/Pagina/66309
https://lgpd.tcm.sp.gov.br/Pagina/66309

9 https://www.uel.br/projetos/matessencial/basico/financeira/amortiza.html
https://www.uel.br/projetos/matessencial/basico/financeira/amortiza.html

11 12 https://www.planalto.gov.br/ccivil_03/LEIS/L4591compilado.htm
https://www.planalto.gov.br/ccivil_03/LEIS/L4591compilado.htm

13 https://prefeitura.pbh.gov.br/fazenda/tributos/ITBI
https://prefeitura.pbh.gov.br/fazenda/tributos/ITBI

16 https://dadosabertos.bcb.gov.br/dataset/27803-sgs/resource/226d3f10-7d05-445d-a2a6-df62b7d48fd5
https://dadosabertos.bcb.gov.br/dataset/27803-sgs/resource/226d3f10-7d05-445d-a2a6-df62b7d48fd5