ChatGPT

Relatório de pesquisa e plano de implementação para “Simulador de Compra de Apartamento na Planta”

Resumo executivo

Conectores usados (consultados primeiro): github (repositório **hbtmarc/apto**).
No ambiente atual, o conector do GitHub confirmou a existência do repositório (branch padrão `main`, repositório público e muito pequeno), mas **não retornou conteúdo útil indexado para pesquisa de código** (buscas por termos-chave retornaram vazio e tentativas de acessar arquivos comuns não tiveram sucesso). Por isso, o plano e os prompts abaixo foram estruturados a partir de boas práticas para apps “no-framework” e de fontes primárias/autoridades (documentação oficial, legislação e referências acadêmicas), com ênfase em portais brasileiros e em português quando disponível.

O desenho proposto entrega um app **100% estático** (HTML + CSS + JS ES Modules), executável via **VSCode Live Server**, com roteamento por **show/hide de** `<section>` e persistência em `localStorage` sob a chave `APTO_SIM_DB_V1`. O app implementa CRUD de **Projetos** e **Simulações**, editor de simulação com CRUD de **itens de fluxo de caixa** (mensal/balão/único), correção opcional por **índice mensal composto** e financiamento opcional por **SAC** ou **PRICE** com geração de cronograma. O módulo de riscos gera flags simples, incluindo **quadro-resumo ausente** e **tolerância > 180 dias**, ancoradas na legislação federal. [1]

O grande foco para reduzir erros “junior”: (1) **modelo de dados explícito** com versão e validação, (2) **funções puras** para cálculo (fáceis de testar no console), (3) **tratamento robusto de** `localStorage.setItem()` (pode lançar exceções em modo privado/quotas), (4) **evitar armadilhas de timezone com** `Date.parse()` usando chaves `YYYY-MM` e manipulação por strings (o próprio MDN descreve diferenças de fuso ao interpretar strings ISO “date-only”). [2]

Necessidades de informação e varredura diversificada de fontes

O que você precisa aprender/confirmar para implementar bem (info needs)

1. **Regras mínimas de ES Modules no navegador** (como incluir `type="module"`, import/export e exigência de servidor local/MIME). [3]
2. **Semântica e falhas comuns de** `localStorage` (strings-only; exceções de quota/privacidade; fallback). [4]
3. **Base legal para alerts de risco**: quadro-resumo obrigatório (Lei 4.591/1964, art. 35-A) e tolerância de 180 dias (Lei 13.786/2018, art. 43-A). [1]
4. **Definições e fórmulas de financiamento SAC/PRICE** (amortização, juros, prestação; forma de cálculo). [5]
5. **Índice INCC (referência de domínio)**: natureza mensal e conceito (sem “hardcode” — apenas como exemplo de série). [6]
6. **Detalhes oficiais do endpoint BCB SGS** (parâmetros, limites 10 anos e limite de “últimos N”). [7]

O que foi encontrado nas fontes (discovery scan)

GitHub (hbtmarc/apto), consultado primeiro: o conector identificou o repositório, mas **não houve retorno de conteúdo pesquisável** para extrair padrões, componentes ou requisitos adicionais. Resultado prático: a arquitetura não pôde "herdar" convenções do repo; seguimos com especificação do pedido e fontes externas primárias.

MDN (ES Modules e Web Storage): - `import/export` só funcionam em módulos, e no HTML o script principal deve ser carregado com `type="module"`. [3]
- Para rodar exemplos de módulos localmente, o MDN recomenda **usar servidor web local**; também destaca a necessidade de **MIME type JavaScript** adequado. [8]
- Web Storage: o armazenamento é chave/valor e **chaves e valores são strings**; `setItem()` pode lançar exceção em cenários como storage cheio e modo privado em certos navegadores. [4]

BCB SGS (documentação oficial): o Portal de Dados Abertos do BCB descreve o endpoint JSON do SGS (`bcdata.sgs.{codigo_serie}`), parâmetros `dataInicial`/`dataFinal` (dd/MM/aaaa), restrição de **janela máxima de 10 anos** e endpoint de "últimos N" com limite informado no recurso. [7]

FGV INCC (domínio/índice de correção): o INCC acompanha evolução de preços de insumos e mão de obra da construção, possui periodicidade **mensal** e teve atualização metodológica/ponderação em 2023 (importante para entender que séries podem mudar, então o app deve ser genérico). [6]

Planalto (legislação, fonte primária): - Lei 4.591/1964 (texto compilado) — art. 35-A exige que contratos de incorporação sejam iniciados por **quadro-resumo** contendo, entre outros, preço total, entrada, corretagem (com beneficiário), forma de pagamento, índices de correção, juros e sistema de amortização. [9]
- Lei 13.786/2018 — art. 43-A disciplina a **tolerância de até 180 dias corridos**, desde que pactuada de forma clara e destacada; acima disso surgem consequências ao incorporador. [10]

SAC/PRICE (referências didáticas/autoridade): - O Tribunal de Contas do Município de São Paulo apresenta definição de prestação/juros/amortização/saldo e simulação comparativa SAC vs PRICE, incluindo que SAC tem amortização constante e prestação decrescente, e PRICE tem prestação constante com amortização crescente. [11]
- A UEL descreve o SAC com tabela e a lógica de amortização constante; e traz a fórmula do coeficiente do sistema Price (Sistema Francês) para prestação constante. [12]

Arquitetura proposta e modelo de dados

Estratégia geral de UI e roteamento

- **Single-page app "sem framework"**: `index.html` com 4–5 `<section>` (Projetos, Simulações, Editor, Resultados, FAQ).
- Roteamento por hash (`#projects`, `#sims`, `#editor`, `#results`, `#faq`), implementado com:
- `window.addEventListener("hashchange", ...)`
- `render.js` expondo `showView(viewId)` e `setActiveNav(hash)`.

Isso mantém o fluxo linear e evita estados complexos de roteador (princípio "junior-friendly").

Modelo de dados (“APTO_SIM_DB_V1”)

Recomendação: um objeto único com versão e coleções normalizadas por ID. Exemplo de shape (sem código, como especificação):

- `db.version = 1`
- `projects[]`: `{ id, name, city, uf, developer, createdAt, updatedAt }`
- `simulations[]`: `{ id, projectId, name, contractDate, deliveryDate, toleranceDays, basePrice, flags, correction, financing, cashflow[], createdAt, updatedAt }`
- `indexSeries[]`: `{ id, name, note, source, monthlyPct: { "YYYY-MM": numberPct }, createdAt, updatedAt }`

Por que `YYYY-MM` **como chave mensal:** evita bugs de fuso ao interpretar `YYYY-MM-DD` via parsing de string ISO, que pode ser assumida como UTC e deslocar datas dependendo do ambiente. [13]

Cálculo e integridade

- **Correção por índice**: fator acumulado por mês `F[m] = F[m-1] * (1 + pct[m]/100)`. Se não houver pct no mês, assumir `0`.
- **Financiamento SALDO**: PV (saldo financiado) calculado no mês de início do financiamento:
- `PV = max(0, preço_corrigido_no_mês - soma_pagamentos_principal_até_mês_anterior)`
- SAC/PRICE: gerar arrays mensais com `{ monthKey, installment, interest, amort, balance }` usando as definições das referências. [5]
- **Riscos**:
- “Quadro-resumo ausente” ancorado no art. 35-A da Lei 4.591/1964. [9]
- “Tolerância > 180 dias” ancorado no art. 43-A da Lei 13.786/2018. [10]

Diagramas (Mermaid)

```
graph TD
  app[js/app.js] --> storage[js/storage.js]
  app --> calc[js/calc.js]
  app --> render[js/render.js]
  app --> risks[js/risks.js]
  app --> faq[js/faq.js]

  render --> calc
  calc --> storage
```

```
flowchart TD
  A[Selecionar Simulação] --> B[Carregar DB + séries]
  B --> C[Determinar range mensal YYYY-MM]
  C --> D[Construir fatores acumulados de índice]
  D --> E[Expandir items cashflow por mês]
  E --> F[Somar totais: principal vs extras]
  F --> G{Financiamento habilitado?}
  G -- não --> H[Montar timeline final]
```

```
  G -- sim --> I[Calcular PV no mês de início]
  I --> J[Gerar cronograma SAC/PRICE]
  J --> K[Mesclar cronograma na timeline]
  K --> H
  H --> L[Totais por fase + flags de risco]
  L --> M[Renderizar Resultados]
```

```
erDiagram
  PROJECT ||--o{ SIMULATION : contains
  SIMULATION ||--o{ CASHFLOW_ITEM : has
  INDEX_SERIES ||--o{ SIMULATION : "referenced by (seriesId)"

  PROJECT {
    string id
    string name
    string city
    string uf
    string developer
  }

  SIMULATION {
    string id
    string projectId
    string name
    string contractDate
    string deliveryDate
    int toleranceDays
    number basePrice
    object flags
    object correction
    object financing
  }

  CASHFLOW_ITEM {
    string id
    string kind
    string label
    number amount
    string startDate
    string endDate
    string dueDate
    boolean applyCorrection
    boolean affectsPrincipal
  }

  INDEX_SERIES {
    string id
    string name
```

```
    object monthlyPct
  }
```

Plano de desenvolvimento com marcos e prompts para Codex 5.3

Estratégia para minimizar erros com Codex

- Gerar primeiro **contratos de API** (exports e shape de dados) e só depois UI.
- Forçar o Codex a criar funções **pequenas e nomeadas**, com comentários curtos e validações.
- Inserir **auto-testes simples** (executáveis no console) para o módulo `calc.js` e para o adaptador de `storage.js`.
- Evitar "parsing de datas" (Date.parse) no cálculo; preferir `YYYY-MM` e operações por string. [13]

Tabela de marcos → entregáveis → prompts

Marco	Entregáveis	Prompt "ready-to-paste"
Base do projeto	Estrutura de pastas + placeholders	PROMPT_BASE
UI estática	`index.html` com sections e ids	PROMPT_INDEX_HTML
Estilos	`styles.css` simples e legível	PROMPT_STYLES
FAQ	`js/faq.js` exportando HTML	PROMPT_FAQ
Persistência + CRUD	`js/storage.js` com seed, validação, fallback	PROMPT_STORAGE
Cálculos	`js/calc.js` (timeline, correção, SAC/PRICE, opcional BCB)	PROMPT_CALC
Riscos	`js/risks.js` regras objetivas	PROMPT_RISKS
Render	`js/render.js` (render lists/forms/tables)	PROMPT_RENDER
Orquestração	`js/app.js` (roteamento, eventos, recalc, export/import)	PROMPT_APP
QA/Autotestes	pequenos testes "sem libs"	PROMPT_TESTS

PROMPT_BASE (estrutura e princípios)

```
Você é o Codex 5.3 no VSCode. Objetivo: criar a estrutura do projeto web
"Off-plan apartment purchase simulator" SEM frameworks e SEM build tools.

Entradas:
- Root folder vazia.
- Restrições: apenas HTML + CSS + Vanilla JS (ES Modules). Sem libs. Código
linear, legível, sem overengineering.

Saídas esperadas:
- Criar a árvore:
```

```
  - index.html
  - styles.css
  - js/app.js
  - js/storage.js
  - js/calc.js
  - js/render.js
  - js/risks.js
  - js/faq.js
- Não implemente lógica ainda; apenas crie arquivos com comentário de
cabeçalho contendo: propósito do arquivo, exports esperados e dependências.

Critérios:
- Em todos os JS: usar `export`/`import`.
- Manter TODOs claros e curtos.
Testes:
- Abrir index.html com Live Server: não deve haver erro 404 de arquivos.
```

PROMPT_INDEX_HTML (sections, ids e roteamento show/hide)

```
Implemente index.html (SEM JS embutido), com layout simples e ids estáveis
para o app.

Entradas:
- Arquivo alvo: /index.html
- Rotas por hash: #projects, #sims, #editor, #results, #faq
- Deve carregar /js/app.js com <script type="module">.

Saídas esperadas:
- index.html contendo:
  1) Header com nav (links com href="#...") e indicação do projeto/simulação
selecionados.
  2) <main> com 5 <section>:
     - view-projects: lista + form de projeto
     - view-sims: lista + form de simulação (por projeto selecionado)
     - view-editor: editor da simulação (cashflow CRUD + correção +
financiamento + Recalcular)
     - view-results: totais + flags + tabela timeline + export/import JSON
     - view-faq: container para FAQ
  3) Cada área deve ter <div> placeholders com ids (ex.: projects-list, sims-
list, cashflow-table, results-timeline, etc).
  4) Inputs do editor:
     - cashflow kind: monthly/balloon/once
     - datas (type=date) e valores numéricos
     - checkboxes: applyCorrection, affectsPrincipal
     - correção: enabled + select series
     - financiamento: enabled + select SAC/PRICE + months + annualRate +
startDate
  5) Botões: salvar/limpar, “Recalcular”, “Exportar JSON”, “Importar JSON”.

Restrições:
```

```
- Sem frameworks, sem inline JS.
- Ids devem ser consistentes e “autoexplicativos”.

Teste:
- Abrir a página: estrutura aparece, sem depender de JS para existir.
```

PROMPT_STYLES (CSS minimalista e acessível)

```
Implemente /styles.css com estilo minimalista (dark ou light), fácil de ler,
responsivo e sem dependências.

Entradas:
- Arquivo alvo: /styles.css
- Deve estilizar: header/nav, cards, forms, tabelas, botões, badges de risco
(ok/warn/bad), mensagens vazias, toast.

Saídas esperadas:
- CSS com:
  - tokens CSS (:root) para cores, espaçamentos e radius
  - classes utilitárias: .hidden, .row, .grid-2
(responsivo), .card, .btn, .btn-danger, .badge (ok/warn/bad)
  - tabela com overflow horizontal para timeline longa

Restrições:
- Sem frameworks (Bootstrap etc).
- Evitar “CSS complexo”; preferir regras curtas.
Teste:
- Redimensionar viewport: grids viram 1 coluna.
```

PROMPT_FAQ (conteúdo estático)

```
Implemente /js/faq.js exportando conteúdo HTML estático como string.

Entradas:
- Arquivo alvo: /js/faq.js

Saídas esperadas:
- Export: `export const FAQ_HTML = "...";`
- Conteúdo FAQ em pt-BR explicando:
  - o que o simulador faz e limitações
  - como funciona a correção por índice (mensal, acumulada)
  - diferença entre SAC e PRICE (bem resumido)
  - como rodar via Live Server (mencionar ES Modules)

Restrições:
- Sem dependências.
Teste:
- app.js conseguirá inserir isso em #view-faq.
```

PROMPT_STORAGE (localStorage + fallback + CRUD + seed)

```
Implemente /js/storage.js como “camada de persistência” com CRUD e seed.

Entradas:
- Chave do storage: APTO_SIM_DB_V1
- localStorage pode falhar (quota, modo privado). Se falhar, usar fallback em
memória.
- Modelo de dados: db.version=1 e coleções projects[], simulations[],
indexSeries[].

Saídas esperadas:
1) Exports obrigatórios:
- `export const STORAGE_KEY = "APTO_SIM_DB_V1";`
- `export function ensureDb(): Db`
- `export function getDb(): Db`
- `export function replaceDb(db: Db): Db` (valida versão e shape mínimo)
- `export function resetDbToSeed(): Db`
- `export function exportDbJson(): string`
- `export function importDbJson(text: string): Db`

2) CRUD (exports):
- Projects: list/get/save/delete
- Simulations: listByProject/get/save/delete
- Cashflow (dentro de simulation): save/delete
- IndexSeries: list/get/upsert
- Flags e configs da simulação: saveFlags/saveCorrection/saveFinancing

3) Seed data pequeno:
- 1 projeto
- 1 simulação com:
  - basePrice, contractDate, deliveryDate, toleranceDays
  - 3–4 itens de cashflow (entrada, mensal obra, balão, ITBI provisão)
  - 1 série de índice “INCC (exemplo)” com alguns meses YYYY-MM -> pct
  - financiamento habilitado (ex.: SAC 240m, 10% a.a., startDate no mês pós
entrega)

Restrições:
- Código linear, funções pequenas, comentários curtos.
- Nunca jogar exceção não tratada por setItem/getItem; encapsular try/catch e
marcar “memoryMode”.

Testes (sem libs):
- Expor `__storageSelfTest()` (export) que:
  - cria seed
  - salva/recupera
  - faz um CRUD simples
  - retorna {ok:true} ou lança Error com mensagem clara.
```

PROMPT_CALC (timeline, correção composta, SAC/PRICE, opcional BCB)

```
Implemente /js/calc.js com funções puras para cálculos.

Fontes/definições:
- SAC/PRICE: usar conceitos de prestação=amortização+juros e saldo devedor
decrescente; SAC com amortização constante; PRICE com prestação constante.
(vide referências do TCMSP e UEL)
- Correção por índice: fator acumulado mensal por multiplicação (1 + pct/
100).
- Evitar Date.parse e bugs de timezone; trabalhar com monthKey "YYYY-MM"
derivado da string do input date.

Entradas:
- simulação (db.simulations[i])
- série de índice indexSeries (monthlyPct: {"YYYY-MM": pct})

Saídas esperadas (exports):
- Utilitários: `toMonthKey(dateIsoOrMonthKey)`, `addMonths(monthKey,n)`,
`monthKeysBetween(a,b)`, `monthKeyToLabel(monthKey)`
- Correção: `buildCumulativeFactors(startKey, endKey, monthlyPct, enabled)`
- Cashflow: `cashflowItemsInMonth(sim, monthKey)` e
`getLastCashflowMonth(sim)`
- Taxas: `annualToMonthlyRate(annualPct)` (documentar fórmula e suposição:
taxa efetiva anual -> efetiva mensal)
- Financiamento:
  - `scheduleSAC(pv, monthlyRate, n, startMonthKey)`
  - `schedulePRICE(pv, monthlyRate, n, startMonthKey)`
  retornando array com: {k, monthKey, installment, interest, amort, balance}
- Motor principal: `computeSimulationResults(sim, indexSeries)` retornando:
  - timeline[] com colunas para tabela
  - totals (incl. totalCashflow, totalFinInstallments, totalPaid)
  - financing meta (pv, monthlyRate, etc.)

Opcional (bem separado no fim do arquivo):
- `fetchBcbSgsSeries(seriesCode, startDateIso, endDateIso)` usando fetch e
endpoint oficial do SGS.
- `sgsJsonToMonthlyPctMap(sgsArray)` convertendo retorno para {"YYYY-MM":
pct}.

Testes:
- Export `runCalcSelfTests()` que valida:
  - monthKeysBetween funciona
  - fator acumulado funciona com meses faltantes=0
  - scheduleSAC e schedulePRICE fecham saldo ~0 no fim (tolerância de
centavos)
  - computeSimulationResults retorna timeline com length > 0 e totals
coerentes
```

PROMPT_RISKS (regras legais e checklist)

```
Implemente /js/risks.js para gerar flags simples de risco.

Entradas:
- project, sim
- sim.flags:
  - hasQuadroResumo (bool)
  - brokerFeeConfirmed (bool)
  - itbiProvisioned (bool)
- sim.toleranceDays (number)

Saídas esperadas:
- Export: `buildRiskFlags(project, sim): RiskFlag[]`
  onde RiskFlag = {id, severity: "ok"|"warn"|"bad", title, detail}

Regras:
- missing_quadro: se !hasQuadroResumo => severity "bad"
  (fundamento: quadro-resumo exigido pela Lei 4.591 art. 35-A)
- broker_fee_not_confirmed: se !brokerFeeConfirmed => "warn"
  (no quadro-resumo consta corretagem/beneficiário)
- missing_itbi: se !itbiProvisioned e não existir item cashflow com label
contendo "itbi" => "warn"
- tolerance_gt_180: se toleranceDays > 180 => "bad"
  (fundamento: Lei 13.786 art. 43-A fala de 180 dias corridos)

Restrições:
- Sem “IA”, apenas regras booleanas.
- Mensagens curtas e claras em pt-BR.

Teste:
- Export `riskSelfTest()` com 3 cenários e contagem esperada de flags.
```

PROMPT_RENDER (renderização e formatação)

```
Implemente /js/render.js para renderizar listas, forms e tabelas no DOM.

Entradas:
- Containers por id do index.html (ex.: projects-list, sims-list, cashflow-
table, results-totals, results-timeline, results-risks)
- Dados vindos de storage e calc

Saídas esperadas (exports):
- UI helpers: `showView(viewId)`, `setActiveNav(hash)`,
`renderStatus(project, sim)`, `toast(msg)`
- Render CRUD:
  - `renderProjectsList(container, projects, simsCountByProject,
selectedProjectId)`
  - `renderSimsList(container, sims, selectedSimId)`
```

```
  - `renderCashflowTable(container, sim)`
- Render resultados:
  - `renderTotals(container, results)`
  - `renderRisks(container, riskFlags)`
  - `renderTimeline(container, results)`
- Formatadores:
  - `fmtBRL(number)` e `fmtPct(number)`

Restrições:
- Sem libs; usar innerHTML com escaping básico para textos e data-action/
data-id para delegação de eventos.
- Tabelas grandes: usar wrapper com overflow.

Testes:
- Criar `renderSelfTestSmoke()` que monta dados fake mínimos e renderiza em
containers (sem crash).
```

PROMPT_APP (estado, roteamento, eventos, recalc, export/import)

```
Implemente /js/app.js como orquestrador do app.

Entradas:
- Dependências:
  - storage.js (CRUD + export/import)
  - calc.js (computeSimulationResults + opcional BCB helper)
  - risks.js (buildRiskFlags)
  - render.js (render e helpers)
  - faq.js (FAQ_HTML)
- Rotas hash: #projects, #sims, #editor, #results, #faq

Saídas esperadas:
- Estado em memória:
  - selectedProjectId, selectedSimId
  - cache resultsBySimId
- Handlers:
  - hashchange -> route()
  - submit forms (project-form, sim-form, cashflow-form)
  - click delegation em botões data-action para select/edit/delete
  - botões: Recalcular, Exportar JSON (download), Importar JSON (file input),
Reset seed
- Lógica:
  - Route decide qual section mostrar e chama render correspondente.
  - Editor: atualiza campos (flags, correção, financiamento) e salva no
storage.
  - Recalcular: roda computeSimulationResults e navega para #results.
  - Results: mostra totals por fase + timeline + flags.
  - FAQ: injeta FAQ_HTML.
- Incluir notas no código sobre:
  - usar monthKey YYYY-MM (evitar Date.parse)
  - tratar erros do storage com mensagens ao usuário
```

```
Restrições:
- Código linear, sem classes/arquitetura complexa.
- Sem dependências externas; ES Modules.

Testes:
- Em dev, habilitar `?selftest=1`:
  - roda storage.__storageSelfTest(), calc.runCalcSelfTests(),
risks.riskSelfTest()
  - loga resultados no console e mostra toast.
```

PROMPT_TESTS (checklist automatizável + cenários de borda)

```
Sem criar novos arquivos além dos exigidos, adicione “self-tests” exportados
em storage.js, calc.js e risks.js e um gatilho em app.js para rodar via URL.

Entradas:
- Arquivos: js/storage.js, js/calc.js, js/risks.js, js/app.js

Saídas esperadas:
- storage.js: export __storageSelfTest()
- calc.js: export runCalcSelfTests()
- risks.js: export riskSelfTest()
- app.js: se location.search contém selftest=1, executar os 3 testes no boot
e:
  - console.log resultados
  - mostrar toast “Self-tests OK” ou alert com erro

Restrições:
- Sem libs, sem runner.
- Testes devem ser determinísticos e rápidos (<50ms).

Validações:
- SAC e PRICE: saldo ao final deve ser <= R$0,05 (por arredondamento).
- Correção: fator acumulado correto para 2–3 meses.
```

Checklist de QA e testes de borda

Tabela de QA (testes locais e resultado esperado)

Caso	Passos	Resultado esperado
Primeiro carregamento	Abrir com Live Server	Seed carrega; existe 1 projeto e 1 simulação; sem erros no console
CRUD Projeto	Criar/editar/excluir projeto	Lista atualiza; excluir remove simulações do projeto
CRUD Simulação	Com projeto selecionado, criar/editar/excluir simulação	Só aparece simulações do projeto; seleção funciona

Caso	Passos	Resultado esperado
Cashflow mensal	Criar item monthly de 2026-03 a 2026-06	Timeline contém 4 meses com esse item
Cashflow balão/ único	Criar item balloon em um mês	Timeline contém item apenas nesse mês
Correção ON/OFF	Toggle correção e recalcular	Com correção ON, valores “corrigidos” mudam conforme fator; OFF = fator 1
Índice com meses faltando	Remover pct de um mês intermediário	Mês faltante assume 0% e não quebra cálculo (fator fica igual ao mês anterior)
Financiamento SAC	Habilitar SAC, prazo curto (ex.: 12), recalcular	Cronograma tem 12 linhas; saldo final ~0; prestações decrescentes [14]
Financiamento PRICE	Habilitar PRICE, prazo curto, recalcular	Prestação aproximadamente constante; juros decrescentes [15]
Risco tolerância	Set toleranceDays = 181	Flag “bad” aparece (tolerância > 180) [10]
Risco quadro-resumo	Desmarcar hasQuadroResumo	Flag “bad” aparece (quadro-resumo ausente) [16]
Export JSON	Exportar e baixar	Arquivo JSON contém version=1 e coleções; pode ser reimportado
Import JSON inválido	Colar JSON quebrado e importar	Mostra erro amigável; DB não corrompe
localStorage indisponível	Simular bloqueio (modo privado/limite) e salvar	App continua com fallback em memória; mostra aviso/console; não quebra [17]
Linha do tempo grande	Prazo 360 meses + cashflow longo	UI não trava; tabela tem overflow horizontal

Falhas prováveis, mitigações e itens não especificados

Tabela de modos de falha → mitigação

Falha provável	Sintoma	Mitigação recomendada
`localStorage.setItem()` lança exceção	Perda de persistência; crash ao salvar	Encapsular em try/catch e ativar “memoryMode”; mensagens claras [17]
`localStorage` só armazena strings	DB volta quebrado (object vira “[object Object]”)	Sempre `JSON.stringify` no write e `JSON.parse` no read; validar shape mínimo [18]
Parsing de datas com timezone	Meses “escapam” (dia anterior) dependendo do fuso	Não usar `Date.parse` para regra de mês; usar `YYYY-MM` derivado de string; evitar ISO date-only parsing [13]

Falha provável	Sintoma	Mitigação recomendada
Arredondamento de float (centavos)	Saldo final ≠ 0,01–0,10	Função `round2`; tolerância nos testes; apresentar valores formatados
Meses faltantes no índice	“NaN” em fator e explosão na tabela	Tratar pct ausente como 0; validar número finito
Timeline muito grande	Lentidão ao renderizar tabela	Limite soft (ex.: 1200 meses) + aviso; renderização simples; overflow e sem DOM pesado
Input inválido	Datas vazias, fim antes do início, negativos	Validação no submit; mensagens curtas; impedir salvar
Import JSON com versão diferente	Quebra de schema	`replaceDb` valida `version==1`; rejeitar com erro; (futuro: migration)

Assunções e pontos não especificados (explicitamente)

1. **Definição de “totais por fase”** não foi especificada. Assunção recomendada:
2. “Pré-entrega”: do mês do contrato até o mês da entrega (inclusive)
3. “Pós-entrega”: meses após entrega
4. Totais também segmentados por: principal vs extras vs financiamento (parcelas e juros).
5. **Regra exata para PV do financiamento** não foi especificada. Assunção: PV é o saldo do preço corrigido no mês de início menos pagamentos marcados como `affectsPrincipal` até o mês anterior.
6. **Conversão de taxa anual para mensal**: assunção de taxa efetiva anual → efetiva mensal; documentar claramente no UI/FAQ.
7. **Correção por índice**: assunção de aplicação “composta” mensal e aplicada ao preço e a itens com `applyCorrection=true` (itens sem flag não corrigem).
8. **ITBI**: não foi especificado se deve ser “apenas alerta” ou cálculo automático. Assunção: alerta se não houver flag e não houver item “ITBI” no cashflow; o valor é inserido manualmente. (Conceito de ITBI como tributo municipal pode ser explicado via páginas de prefeituras.) [19]
9. **Modo BCB SGS**: é opcional e deve ficar “isolado” (não mudar funcionamento do app sem uso explícito do usuário). A documentação oficial do BCB impõe limites (ex.: 10 anos). [7]

Checklist de execução no VSCode e seed data

Checklist para rodar com VSCode + Live Server (sem build tools)

1. Abrir a pasta do projeto no VSCode.
2. Instalar extensão **Live Server**.
3. Rodar “Open with Live Server” no `index.html`.
4. Confirmar que o script está como **ES Module** (`<script type="module" src="./js/app.js">`). [3]
5. Se tentar abrir via `file://`, é comum dar problema com módulos; o MDN recomenda **servidor local** para executar exemplos de módulos. [8]
6. Testar `?selftest=1` (quando implementado) para validar motor de cálculo + storage rapidamente.

Seed data (snippet de referência para o `storage.js`)

Use algo pequeno, com datas próximas e 1 série de índice "INCC (exemplo)" (valores fictícios). Exemplo **apenas como dado**, não como código:

```
{
  "version": 1,
  "projects": [
    { "id": "p1", "name": "Residencial Vista Park", "city": "São Paulo",
"uf": "SP", "developer": "Construtora Exemplo S.A." }
  ],
  "indexSeries": [
    {
      "id": "idx_incc_demo",
      "name": "INCC (exemplo)",
      "note": "Valores fictícios para demonstrar correção mensal",
      "source": "manual",
      "monthlyPct": { "2026-02": 0.20, "2026-03": 0.25, "2026-04": 0.30 }
    }
  ],
  "simulations": [
    {
      "id": "s1",
      "projectId": "p1",
      "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": "idx_incc_demo" },
      "financing": { "enabled": true, "type": "SAC", "months": 240,
"annualRatePct": 10.0, "startDate": "2028-07-01" },
      "cashflow": [
        { "id": "cf1", "kind": "once", "label": "Entrada", "amount": 50000,
"dueDate": "2026-02-10", "applyCorrection": false, "affectsPrincipal":
true },
        { "id": "cf2", "kind": "monthly", "label": "Parcelas obra",
"amount": 3000, "startDate": "2026-03-01", "endDate": "2028-05-01",
"applyCorrection": true, "affectsPrincipal": true },
        { "id": "cf3", "kind": "balloon", "label": "Intermediária",
"amount": 20000, "dueDate": "2027-12-05", "applyCorrection": true,
"affectsPrincipal": true },
        { "id": "cf4", "kind": "once", "label": "ITBI (provisão)", "amount":
15000, "dueDate": "2028-07-10", "applyCorrection": false,
"affectsPrincipal": false }
      ]
    }
```

```
  ]
}
```

Nota final sobre o helper opcional do BCB SGS

Ao implementar o helper opcional, use exatamente os parâmetros e limites documentados no recurso oficial (endpoint `api.bcb.gov.br`, formato JSON, `dataInicial/dataFinal` e diferença máxima de 10 anos; endpoint de "últimos N" com limite). [7]

---

[1] [9] [16] L4591compilado
https://www.planalto.gov.br/ccivil_03/LEIS/L4591compilado.htm?utm_source=chatgpt.com

[2] [17] Storage.setItem() - APIs da Web | MDN
https://developer.mozilla.org/pt-BR/docs/Web/API/Storage/setItem?utm_source=chatgpt.com

[3] [8] Módulos JavaScript - JavaScript | MDN
https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Modules

[4] [18] Usando a API Web Storage - APIs da Web | MDN
https://developer.mozilla.org/pt-BR/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API?utm_source=chatgpt.com

[5] [11] [14] [15] Sistemas de Amortização de Financiamentos: SAC, PRICE - Tribunal de Contas do Município de São Paulo
https://lgpd.tcm.sp.gov.br/Pagina/66309?utm_source=chatgpt.com

[6] INCC | IBRE
https://portalibre.fgv.br/incc?utm_source=chatgpt.com

[7] Fator diário Taxa Extramercado - json_serie-sgs-27803 - Portal de Dados Abertos do Banco Central do Brasil
https://dadosabertos.bcb.gov.br/it/dataset/27803-sgs/resource/226d3f10-7d05-445d-a2a6-df62b7d48fd5

[10] L13786
https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13786.htm?utm_source=chatgpt.com

[12] Matemática Essencial :: Matemática Financeira :: Sistemas de Amortização
https://www.uel.br/projetos/matessencial/basico/financeira/amortiza.html?utm_source=chatgpt.com

[13] Date.parse() - JavaScript | MDN
https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Date/parse?utm_source=chatgpt.com

[19] ITBI | Prefeitura de Belo Horizonte
https://prefeitura.pbh.gov.br/fazenda/tributos/ITBI?utm_source=chatgpt.com