- Dindin — Gestão de Gastos: Product Requirements Document (PRD)
  - 1. Goals and Background Context
    - 1.1 Goals
    - 1.2 Background Context
    - 1.3 Change Log
  - 2. Requirements
    - 2.1 Functional Requirements
    - 2.2 Non-Functional Requirements
  - 3. User Interface Design Goals
    - 3.1 Overall UX Vision
    - 3.2 Key Interaction Paradigms
    - 3.3 Core Screens and Views
    - 3.4 Accessibility
    - 3.5 Branding
    - 3.6 Target Device and Platforms
  - 4. Technical Assumptions
    - 4.1 Repository Structure: Monorepo
    - 4.2 Service Architecture
    - 4.3 Testing Requirements
    - 4.4 Additional Technical Assumptions
  - 5. Epic List
    - Fora de Escopo
  - 6. Epic Details
    - Epic 1: Qualidade & Fundação Técnica
      - Story 1.1: Configurar infraestrutura de testes
      - Story 1.2: Testes para useBalanceCalculator
      - Story 1.3: Testes para useSharePayments
      - Story 1.4: Testes para lógica de fatura de cartão
      - Story 1.5: Atualizar configurações técnicas
    - Epic 2: Orçamentos Mensais
      - Story 2.1: Criar tabela e hook de orçamentos
      - Story 2.2: Tela de gerenciamento de orçamentos
      - Story 2.3: Cálculo de progresso gasto vs. orçamento
      - Story 2.4: Widget de orçamentos no Dashboard
    - Epic 3: Relatórios & Exportação
      - Story 3.1: Criar tela de relatórios com filtros
      - Story 3.2: Gráficos analíticos no relatório

        - Story 3.3: Exportação em CSV
        - Story 3.4: Exportação em PDF
    - Epic 4: Notificações Push
        - Story 4.1: Infraestrutura de notificações push (Service Worker + Supabase)
        - Story 4.2: Notificações de vencimento de contas
        - Story 4.3: Notificações de orçamento atingido
        - Story 4.4: Notificações de atividade em grupos
        - Story 4.5: Tela de configurações de notificações
    - Epic 5: Importação Multiplataforma
        - Story 5.1: Unificar componente de upload de arquivo (mobile + desktop)
        - Story 5.2: Melhorar importação de fatura PDF
        - Story 5.3: Melhorar importação de transações CSV
        - Story 5.4: Histórico de importações
- 7. Checklist Results Report
    - Executive Summary
    - Category Analysis
    - Final Decision
- 8. Next Steps
    - UX Expert Prompt
    - Architect Prompt

Dindin — Gestão de Gastos: Product Requirements Document (PRD)

1. Goals and Background Context

1.1 Goals

- Centralizar o controle financeiro pessoal e compartilhado em uma única plataforma acessível via mobile e desktop
- Automatizar o rastreamento de despesas, receitas e investimentos, reduzindo o esforço manual de gestão financeira

- Facilitar a divisão de gastos entre membros de grupos e perfis familiares (incluindo perfis "fantasma" sem conta)
- Oferecer visibilidade clara do fluxo financeiro mensal: quanto entra, quanto sai, o que está pendente de pagamento/recebimento
- Gerenciar faturas de cartão de crédito com controle de fechamento, vencimento e parcelamento
- Acompanhar contas recorrentes e assinaturas, garantindo que nada seja esquecido
- Estabelecer e monitorar metas de economia, promovendo disciplina financeira
- Entregar uma experiência mobile-first (PWA) com suporte offline, haptic feedback e gestos de swipe

1.2 Background Context

O **Dindin** é um aplicativo de gestão de gastos pessoais e compartilhados construído como Progressive Web App (PWA) com React 19 e Supabase. Nasceu da necessidade de ter uma ferramenta simples, porém completa, para controlar finanças do dia a dia — desde gastos individuais até despesas compartilhadas com família e amigos.

Atualmente, o app já conta com 13 módulos funcionais (dashboard, transações, cartões, grupos, contas recorrentes, assinaturas, investimentos, calendário, perfis, categorias, conta de usuário, autenticação e landing page). A base de código está madura, com contextos globais, hooks customizados, real-time subscriptions e policies de RLS no Supabase. Este PRD visa documentar formalmente os requisitos do produto, identificar lacunas e planejar os próximos passos de evolução.

1.3 Change Log

Date	Version	Description	Author
2026-02-20	1.0	Criação inicial do PRD	Morgan (PM AI)

2. Requirements

2.1 Functional Requirements

- **FR1:** O sistema deve permitir cadastro e autenticação de usuários via email/senha com suporte a biometria (Face ID/Touch ID)
- **FR2:** O sistema deve permitir criar, editar, excluir e listar transações (despesas, receitas, investimentos e contas)
- **FR3:** O sistema deve suportar divisão de gastos entre múltiplos perfis, com rastreamento de valores pendentes e pagos por participante
- **FR4:** O sistema deve permitir criação de perfis "fantasma" (sem autenticação) para representar familiares ou terceiros em divisões
- **FR5:** O sistema deve gerenciar cartões de crédito e débito com controle de fatura mensal, dia de fechamento e vencimento
- **FR6:** O sistema deve associar transações a faturas de cartão com base na data de competência vs data da compra
- **FR7:** O sistema deve suportar transações recorrentes (séries) com opções de exclusão individual ou em lote
- **FR8:** O sistema deve suportar parcelamento de compras com rastreamento automático das parcelas
- **FR9:** O sistema deve permitir criação e gerenciamento de grupos para despesas compartilhadas, com sistema de convites
- **FR10:** O sistema deve oferecer dashboard com visão consolidada: saldo, receitas, despesas, a receber, a pagar e burn rate mensal
- **FR11:** O sistema deve exibir gráficos analíticos (pizza por categoria, linha cumulativa de gastos)
- **FR12:** O sistema deve permitir gerenciar categorias customizáveis com ícone e cor
- **FR13:** O sistema deve permitir criar e acompanhar metas de economia com valor alvo e progresso
- **FR14:** O sistema deve oferecer visão de calendário das transações
- **FR15:** O sistema deve suportar importação de faturas via PDF e transações via CSV, tanto em desktop quanto mobile
- **FR16:** O sistema deve fornecer hub de assinaturas recorrentes
- **FR17:** O sistema deve rastrear investimentos
- **FR18:** O sistema deve suportar notificações em tempo real via Supabase Realtime
- **FR19:** O sistema deve suportar notificações push para alertas de vencimento e atualizações
- **FR20:** O sistema deve permitir exportação de relatórios financeiros (PDF/CSV)
- **FR21:** O sistema deve oferecer orçamentos mensais por categoria com acompanhamento de limites

2.2 Non-Functional Requirements

- **NFR1:** O app deve funcionar como PWA com suporte offline via Service Worker
- **NFR2:** A interface deve ser mobile-first e responsiva, funcionando em smartphones e desktops
- **NFR3:** O tempo de carregamento inicial deve ser inferior a 3 segundos em conexão 4G
- **NFR4:** Todas as operações de dados devem ser protegidas por Row Level Security (RLS) no Supabase
- **NFR5:** O app deve fornecer haptic feedback em interações mobile para melhor UX
- **NFR6:** O app deve suportar gestos de swipe para navegação e ações rápidas em mobile
- **NFR7:** O sistema deve manter paridade funcional entre versões mobile e desktop
- **NFR8:** Os dados financeiros devem ser processados com precisão de 2 casas decimais

---

3. User Interface Design Goals

3.1 Overall UX Vision

A experiência do Dindin deve ser **simples, rápida e confiável** — um app que o usuário abre diariamente sem fricção. A interface prioriza **ações rápidas** (registrar gasto em poucos toques) sobre dashboards complexos. O design segue o paradigma mobile-first com foco em **uma mão só**: ações principais acessíveis na zona do polegar.

3.2 Key Interaction Paradigms

- **Quick-add pattern:** Registro de transação com mínimo de campos obrigatórios (valor, descrição, categoria)
- **Swipe actions:** Gestos de deslizar para editar, excluir ou marcar como pago
- **Pull-to-refresh:** Atualização de dados com gesto nativo
- **Bottom navigation:** Navegação principal na parte inferior da tela
- **Modal sheets:** Formulários e detalhes em modal deslizante (bottom sheet pattern)
- **Haptic feedback:** Confirmação tátil em ações importantes

3.3 Core Screens and Views

1. **Login/Signup** — Autenticação com suporte a biometria
2. **Dashboard** — Visão consolidada: saldo, receitas vs despesas, gráficos, burn rate
3. **Transações (Lista)** — Listagem filtrada por mês, busca, categorias
4. **Adicionar/Editar Transação** — Formulário com divisão de gastos, cartão, categoria
5. **Cartões** — Lista de cartões com fatura mensal detalhada
6. **Fatura do Cartão** — Transações agrupadas por ciclo de faturamento
7. **Grupos** — Gestão de despesas compartilhadas com membros e convites
8. **Perfis** — Perfis de usuário e perfis fantasma
9. **Contas Recorrentes (Bills)** — Gestão de contas fixas com status de pagamento
10. **Assinaturas** — Hub de assinaturas recorrentes
11. **Calendário** — Visão temporal de transações
12. **Metas** — Metas de economia com progresso visual
13. **Investimentos** — Rastreamento de investimentos
14. **Orçamentos** *(novo)* — Limites mensais por categoria com indicadores visuais
15. **Relatórios** *(novo)* — Relatórios exportáveis com filtros de período e categoria
16. **Conta/Configurações** — Configurações do usuário

3.4 Accessibility

Nenhum padrão formal (WCAG) definido neste momento. Acessibilidade básica (contraste legível, labels em inputs, navegação por teclado) mantida como boa prática.

3.5 Branding

- **Nome do produto:** Dindin
- **Identidade visual:** Sem guia de estilo formal. Ícones via Lucide React. Esquema de cores definido via CSS modules/inline styles.

3.6 Target Device and Platforms

Web Responsive (PWA) — Mobile-first, funcionando em todos os navegadores modernos. Instalável como app via PWA em iOS e Android. Desktop como experiência

secundária mas funcional.

---

4. Technical Assumptions

4.1 Repository Structure: Monorepo

Repositório único contendo frontend (React/Vite) e configurações de banco (Supabase). Estrutura já consolidada em um único repo Git.

4.2 Service Architecture

- **Frontend:** React 19.2.0 + Vite 7.2.4 (SPA com client-side routing via React Router DOM 7.12)
- **Backend/BaaS:** Supabase (PostgreSQL + Auth + Realtime + Storage + RLS)
- **Padrão:** Serverless — sem backend próprio, toda lógica de negócio via RLS policies, funções RPC do Supabase e lógica no client
- **Estado global:** React Context API (AuthContext, TransactionsContext, CardsContext, GoalsContext, DashboardContext)
- **Hooks customizados:** 12+ hooks (useTransactions, useCards, useGroups, useProfiles, useCategories, useBalanceCalculator, etc.)
- **Estilização:** CSS Modules / Inline styles
- **Ícones:** Lucide React
- **Libs auxiliares:** date-fns, papaparse (CSV), pdfjs-dist (PDF), uuid, classnames, react-swipeable-list
- **Deploy:** Vercel
- **Linguagem:** JavaScript (JSX) — sem TypeScript neste momento

4.3 Testing Requirements

- **Estado atual:** Sem testes automatizados configurados
- **Linting:** ESLint configurado (`npm run lint`)
- **Plano:** Implementar testes unitários com Vitest para hooks financeiros críticos (useBalanceCalculator, useSharePayments, lógica de fatura de cartão)

4.4 Additional Technical Assumptions

- **PWA:** Service Worker já implementado para suporte offline e instalação
- **Autenticação:** Supabase Auth (email/senha + biometria via atributos de formulário)
- **Segurança de dados:** Row Level Security (RLS) no PostgreSQL via Supabase
- **Real-time:** Supabase Realtime subscriptions para atualizações ao vivo
- **Sem migração de stack planejada:** Continuidade com React + Vite + Supabase
- **Migração TypeScript:** Planejada como PRD separado futuro

---

5. Epic List

Epic	Title	Goal
1	Qualidade & Fundação Técnica	Estabelecer testes automatizados para hooks financeiros críticos e atualizar configurações técnicas
2	Orçamentos Mensais	Permitir definição de limites mensais por categoria com acompanhamento em tempo real
3	Relatórios & Exportação	Oferecer relatórios financeiros filtrados com exportação em PDF e CSV
4	Notificações Push	Implementar push notifications para vencimentos, orçamentos e atividades em grupo
5	Importação Multiplataforma	Garantir importação de faturas e transações funcional em mobile e desktop

Fora de Escopo

- Tema claro/escuro (futuro)
- Migração para TypeScript (PRD separado)
- App nativo (iOS/Android)
- OCR de imagem de fatura

---

6. Epic Details

Epic 1: Qualidade & Fundação Técnica

Objetivo: Estabelecer a base de qualidade do projeto com testes automatizados nos hooks financeiros críticos, atualizar configurações técnicas desatualizadas e preparar o terreno para as novas features com segurança.

Story 1.1: Configurar infraestrutura de testes

> Como desenvolvedor, eu quero ter um test runner configurado no projeto, para que eu possa escrever e executar testes automatizados.

Acceptance Criteria:

1. Vitest configurado como test runner no projeto com script `npm test` funcional
2. Configuração de coverage report habilitada
3. Arquivo de exemplo de teste criado e passando
4. `package.json` atualizado com dependências de teste (vitest, @testing-library/react)
5. Pasta de testes segue convenção co-located (`*.test.js` junto ao arquivo testado)

Story 1.2: Testes para useBalanceCalculator

> Como desenvolvedor, eu quero testes unitários para o hook useBalanceCalculator, para que cálculos de saldo, "a receber" e "a pagar" estejam validados.

Acceptance Criteria:

1. Testes cobrem cálculo de saldo líquido (receitas - despesas)
2. Testes cobrem cálculo de valores "a receber" (shares pendentes de outros)
3. Testes cobrem cálculo de valores "a pagar" (shares pendentes do usuário)
4. Testes cobrem cenários com lista vazia de transações
5. Testes cobrem cenários com múltiplas moedas/precisão decimal (2 casas)
6. Coverage mínimo de 90% no hook

Story 1.3: Testes para useSharePayments

> Como desenvolvedor, eu quero testes unitários para o hook useSharePayments, para que a lógica de divisão de gastos entre perfis esteja validada.

Acceptance Criteria:

1. Testes cobrem divisão igualitária entre N perfis
2. Testes cobrem divisão com valores customizados por perfil
3. Testes cobrem cenário de share com perfil fantasma (ghost profile)
4. Testes cobrem transição de status pending → paid
5. Testes cobrem cálculo de dívidas entre perfis (quem deve quanto a quem)
6. Coverage mínimo de 90% no hook

Story 1.4: Testes para lógica de fatura de cartão

> Como desenvolvedor, eu quero testes unitários para a lógica de cálculo de fatura de cartão, para que o agrupamento por ciclo de fechamento esteja correto.

Acceptance Criteria:

1. Testes cobrem associação de transação ao mês correto da fatura baseado em closing_day
2. Testes cobrem transações na fronteira do dia de fechamento (compra no dia do fechamento vs dia seguinte)
3. Testes cobrem cálculo de total da fatura por período
4. Testes cobrem cartões com diferentes dias de fechamento
5. Coverage mínimo de 90% na lógica testada

Story 1.5: Atualizar configurações técnicas

> Como desenvolvedor, eu quero que o arquivo de technical-preferences reflita a stack real do projeto, para que agentes AI e documentação estejam alinhados com a realidade.

Acceptance Criteria:

1. `.aios-core/data/technical-preferences.md` atualizado com stack real: React 19, Vite, Supabase, CSS Modules
2. Preset ativo alterado de `nextjs-react` para configuração customizada ou novo preset `react-vite-supabase`
3. Seção de User Preferences preenchida com as preferências reais do projeto
4. Deploy target documentado como Vercel

Epic 2: Orçamentos Mensais

Objetivo: Permitir que o usuário defina limites de gasto mensais por categoria, acompanhe em tempo real quanto já consumiu de cada orçamento e receba indicadores visuais claros quando estiver se aproximando ou ultrapassando os limites — promovendo disciplina financeira no dia a dia.

Story 2.1: Criar tabela e hook de orçamentos

> Como desenvolvedor, eu quero uma estrutura de dados e hook para orçamentos mensais, para que o sistema consiga armazenar e consultar limites por categoria/mês.

Acceptance Criteria:

1. Tabela `budgets` criada no Supabase com campos: id (UUID), user_id, category_id, amount (limite), month (YYYY-MM), created_at
2. RLS policy configurada: usuário só acessa seus próprios orçamentos
3. Constraint unique em (user_id, category_id, month) para evitar duplicatas
4. Hook `useBudgets` implementado com operações CRUD (create, read, update, delete)
5. Hook retorna orçamentos do mês selecionado com dados da categoria associada
6. Testes unitários para o hook cobrindo operações CRUD e edge cases

Story 2.2: Tela de gerenciamento de orçamentos

> Como usuário, eu quero uma tela para criar e gerenciar meus orçamentos mensais, para que eu possa definir limites de gasto por categoria.

Acceptance Criteria:

1. Nova rota `/budgets` acessível via navegação principal
2. Lista de orçamentos do mês atual com: nome da categoria, ícone, cor, valor limite
3. Botão para adicionar novo orçamento com seleção de categoria e valor limite
4. Categorias já com orçamento não aparecem na lista de seleção (evitar duplicatas)
5. Ação de editar valor limite de um orçamento existente
6. Ação de excluir orçamento com confirmação
7. Navegação entre meses (anterior/próximo) para ver orçamentos de outros períodos
8. Layout responsivo: funcional em mobile e desktop
9. Haptic feedback em ações de criar/editar/excluir no mobile

Story 2.3: Cálculo de progresso gasto vs. orçamento

> Como usuário, eu quero ver quanto já gastei de cada orçamento no mês, para que eu saiba em tempo real quanto ainda posso gastar por categoria.

Acceptance Criteria:

1. Hook `useBudgets` enriquecido com campo `spent` calculado a partir das transações do mês/categoria
2. Cálculo considera apenas transações do tipo 'expense' do mês correspondente
3. Cálculo considera transações de todos os perfis do usuário (incluindo ghost profiles)
4. Barra de progresso visual para cada orçamento: verde (0-75%), amarelo (75-90%), vermelho (90%+)
5. Exibição de valores: "R$X,XX de R$ Y,YY" e percentual utilizado
6. Indicador visual claro quando orçamento ultrapassado (> 100%)
7. Testes unitários para lógica de cálculo de progresso

Story 2.4: Widget de orçamentos no Dashboard

> Como usuário, eu quero ver um resumo dos meus orçamentos no dashboard, para que eu tenha visibilidade rápida sem entrar na tela de orçamentos.

Acceptance Criteria:

1. Card/widget no dashboard mostrando os 3 orçamentos mais próximos do limite (maior % utilizado)
2. Cada item mostra: categoria (ícone + nome), barra de progresso compacta, percentual
3. Se nenhum orçamento cadastrado, exibe CTA "Criar orçamento" linkando para `/budgets`
4. Se todos os orçamentos estão saudáveis (< 75%), exibe mensagem positiva
5. Link "Ver todos" direciona para a tela completa de orçamentos
6. Widget responsivo e integrado ao layout existente do dashboard

---

Epic 3: Relatórios & Exportação

Objetivo: Oferecer ao usuário relatórios financeiros claros e exportáveis, permitindo análise detalhada por período e categoria, com exportação em PDF e CSV — facilitando controle fiscal, compartilhamento com cônjuge/parceiro e planejamento financeiro de longo prazo.

Story 3.1: Criar tela de relatórios com filtros

> Como usuário, eu quero uma tela de relatórios com filtros de período e categoria, para que eu possa analisar meus gastos de forma detalhada e personalizada.

Acceptance Criteria:

1. Nova rota `/reports` acessível via navegação principal
2. Filtro de período: mês atual (padrão), mês específico, intervalo customizado (data início/fim)
3. Filtro por categoria: todas (padrão) ou seleção múltipla de categorias
4. Filtro por tipo: despesas, receitas, todos
5. Resumo numérico: total de receitas, total de despesas, saldo do período
6. Lista de transações agrupadas por categoria com subtotal por grupo
7. Layout responsivo: funcional em mobile e desktop
8. Estado de filtros persistido durante a sessão (não reseta ao navegar e voltar)

Story 3.2: Gráficos analíticos no relatório

> Como usuário, eu quero visualizar gráficos no relatório, para que eu tenha uma visão clara da distribuição e evolução dos meus gastos.

Acceptance Criteria:

1. Gráfico de pizza: distribuição de gastos por categoria no período filtrado
2. Gráfico de linha cumulativa: evolução de gastos ao longo do período
3. Gráficos reutilizam os componentes existentes (CumulativeLineChart, CategoryPieChart)
4. Gráficos atualizam dinamicamente ao alterar filtros
5. Exibição de top 5 categorias com maior gasto e percentual do total
6. Em mobile, gráficos ocupam largura total com scroll horizontal se necessário

Story 3.3: Exportação em CSV

> Como usuário, eu quero exportar meus relatórios em formato CSV, para que eu possa manipular os dados em planilhas (Excel, Google Sheets).

Acceptance Criteria:

1. Botão "Exportar CSV" visível na tela de relatórios
2. Exportação respeita os filtros ativos (período, categoria, tipo)

3. CSV inclui colunas: Data, Descrição, Categoria, Tipo, Valor, Cartão, Status
4. Valores formatados com separador decimal correto (vírgula para pt-BR)
5. Nome do arquivo segue padrão: `dindin-relatorio-{data-inicio}-{data-fim}.csv`
6. Utiliza biblioteca PapaParse já existente no projeto para geração do CSV
7. Funciona tanto em mobile (share/download) quanto desktop (download direto)

Story 3.4: Exportação em PDF

> Como usuário, eu quero exportar meus relatórios em formato PDF, para que eu tenha um documento formal para guardar ou compartilhar.

Acceptance Criteria:

1. Botão "Exportar PDF" visível na tela de relatórios
2. PDF inclui: cabeçalho com nome do app e período, resumo numérico, tabela de transações agrupada por categoria
3. PDF inclui gráfico de pizza renderizado como imagem estática
4. Exportação respeita os filtros ativos
5. Layout do PDF otimizado para impressão (A4, margens adequadas)
6. Nome do arquivo segue padrão: `dindin-relatorio-{data-inicio}-{data-fim}.pdf`
7. Funciona tanto em mobile quanto desktop
8. Utilizar abordagem leve (ex: html2canvas + jsPDF ou react-pdf) sem dependência pesada

---

Epic 4: Notificações Push

Objetivo: Implementar notificações push via PWA para manter o usuário informado sobre vencimentos de contas, limites de orçamento atingidos e atividades em grupos compartilhados — reduzindo esquecimentos e promovendo engajamento diário com o app.

Story 4.1: Infraestrutura de notificações push (Service Worker + Supabase)

> Como desenvolvedor, eu quero a infraestrutura de notificações push configurada, para que o sistema consiga enviar e receber push notifications via PWA.

Acceptance Criteria:

1. Service Worker existente estendido com handler de push events (`self.addEventListener('push', ...)`)
2. Lógica de solicitação de permissão (`Notification.requestPermission()`) implementada com UX não-intrusiva (solicitar após primeira ação significativa, não no primeiro acesso)
3. Tabela `push_subscriptions` criada no Supabase com campos: id, user_id, subscription (JSON com endpoint + keys), created_at, updated_at
4. RLS policy: usuário só acessa suas próprias subscriptions
5. Endpoint ou Supabase Edge Function para receber e armazenar subscription do navegador
6. Função utilitária para enviar push notification via Web Push API (usando Edge Function do Supabase)
7. Tratamento de cenário onde usuário nega permissão (não solicitar novamente, exibir instrução em configurações)

Story 4.2: Notificações de vencimento de contas

> Como usuário, eu quero receber notificações antes do vencimento das minhas contas, para que eu não esqueça de pagar e evite multas/juros.

Acceptance Criteria:

1. Supabase Edge Function (ou pg_cron) que roda diariamente verificando contas com vencimento nos próximos 3 dias
2. Notificação enviada com: título "Conta próxima do vencimento", corpo com nome da conta e valor, data de vencimento
3. Notificação enviada apenas para contas com status 'pending' (não pagas)
4. Ao clicar na notificação, usuário é direcionado para a tela de bills (`/bills`)
5. Configuração no perfil do usuário para ativar/desativar este tipo de notificação
6. Não enviar notificações duplicadas para a mesma conta no mesmo dia

Story 4.3: Notificações de orçamento atingido

> Como usuário, eu quero ser notificado quando meu gasto atingir 90% de um orçamento, para que eu possa frear os gastos antes de ultrapassar o limite.

Acceptance Criteria:

1. Lógica que verifica, ao registrar nova transação, se algum orçamento da categoria atingiu 90% ou mais
2. Notificação enviada com: título "Orçamento quase no limite", corpo com nome da categoria, valor gasto vs. limite e percentual
3. Notificação enviada apenas uma vez por orçamento/mês ao cruzar o threshold de 90%
4. Tabela ou flag para rastrear notificações já enviadas (evitar duplicatas)
5. Ao clicar na notificação, usuário é direcionado para `/budgets`
6. Configuração no perfil do usuário para ativar/desativar este tipo de notificação
7. Depende do Epic 2 (orçamentos) estar implementado

Story 4.4: Notificações de atividade em grupos

> Como usuário, eu quero ser notificado quando alguém adicionar uma despesa no meu grupo, para que eu fique atualizado sobre gastos compartilhados.

Acceptance Criteria:

1. Ao criar transação em grupo, notificação enviada para todos os membros do grupo (exceto quem criou)
2. Notificação com: título "Nova despesa no grupo {nome}", corpo com descrição, valor e quem pagou
3. Ao clicar na notificação, usuário é direcionado para o grupo específico (`/groups/:id`)
4. Configuração no perfil do usuário para ativar/desativar notificações de grupo
5. Não enviar notificações para perfis fantasma (ghost profiles não têm push subscription)

Story 4.5: Tela de configurações de notificações

> Como usuário, eu quero gerenciar quais notificações recebo, para que eu tenha controle sobre o que me é notificado.

Acceptance Criteria:

1. Seção "Notificações" na tela de conta (`/account`)
2. Toggles individuais para cada tipo: vencimento de contas, orçamento atingido, atividade em grupos
3. Estado dos toggles persistido no Supabase (tabela `notification_preferences` ou campo JSON no perfil)

4. Toggle master "Ativar/desativar todas as notificações"
5. Se permissão de push negada no navegador, exibir instrução de como reativar nas configurações do dispositivo
6. Haptic feedback nos toggles em mobile

Epic 5: Importação Multiplataforma

Objetivo: Garantir que a importação de faturas (PDF) e transações (CSV) funcione de forma consistente e confiável tanto em mobile quanto desktop, resolvendo diferenças de File API entre plataformas e melhorando a experiência de importação com feedback visual e validação.

Story 5.1: Unificar componente de upload de arquivo (mobile + desktop)

> Como usuário, eu quero selecionar arquivos para importação tanto no celular quanto no computador, para que a experiência de upload funcione sem fricção em qualquer dispositivo.

Acceptance Criteria:

1. Componente reutilizável `FileUploader` que abstrai diferenças de File API entre plataformas
2. Em desktop: suporte a drag & drop + clique para selecionar arquivo
3. Em mobile: clique para abrir seletor de arquivo nativo (câmera/arquivos/fotos conforme tipo)
4. Aceita filtro de tipos: `.pdf`, `.csv` (configurável por uso)
5. Validação de tamanho máximo de arquivo (ex: 10MB) com mensagem clara de erro
6. Preview do nome do arquivo selecionado antes de processar
7. Haptic feedback ao selecionar arquivo no mobile
8. Testes unitários para validações do componente

Story 5.2: Melhorar importação de fatura PDF

> Como usuário, eu quero importar faturas de cartão de crédito em PDF, para que minhas transações sejam cadastradas automaticamente sem digitação manual.

Acceptance Criteria:

1. Utiliza o componente `FileUploader` da Story 5.1
2. Parsing de PDF via pdfjs-dist (já no projeto) com extração de linhas de texto
3. Tela de preview mostrando transações detectadas antes de confirmar importação
4. Cada transação detectada mostra: data, descrição, valor — com opção de editar ou excluir individualmente
5. Seleção de cartão destino para associar as transações importadas
6. Detecção automática do mês da fatura a partir do conteúdo do PDF
7. Indicador de progresso durante o parsing (loading state)
8. Tratamento de erro claro quando PDF não é reconhecido como fatura (formato não suportado)
9. Funciona em mobile (iOS Safari, Chrome Android) e desktop (Chrome, Firefox, Safari)

Story 5.3: Melhorar importação de transações CSV

> Como usuário, eu quero importar transações via arquivo CSV, para que eu possa migrar dados de outros apps ou planilhas facilmente.

Acceptance Criteria:

1. Utiliza o componente `FileUploader` da Story 5.1
2. Parsing via PapaParse (já no projeto) com detecção automática de separador (vírgula, ponto e vírgula, tab)
3. Tela de mapeamento de colunas: usuário associa colunas do CSV aos campos (data, descrição, valor, categoria)
4. Preview das primeiras 5 linhas para validação visual antes de importar
5. Validação de dados: valores numéricos, datas válidas, campos obrigatórios preenchidos
6. Relatório pós-importação: X transações importadas, Y ignoradas (com motivo)
7. Opção de desfazer importação em lote (excluir todas as transações importadas)
8. Funciona em mobile e desktop
9. Suporte a encoding UTF-8 e ISO-8859-1 (comum em exports de bancos brasileiros)

Story 5.4: Histórico de importações

> Como usuário, eu quero ver o histórico das minhas importações, para que eu possa rastrear o que foi importado e desfazer se necessário.

Acceptance Criteria:

1. Tabela `import_logs` no Supabase com: id, user_id, type (pdf/csv), filename, transaction_count, created_at, status
2. RLS policy: usuário só acessa seus próprios logs
3. Seção "Histórico de importações" acessível via tela de conta (`/account`) ou via tela de transações
4. Lista de importações com: data, tipo (PDF/CSV), nome do arquivo, quantidade de transações
5. Ação de desfazer importação: exclui todas as transações associadas (com confirmação)
6. Tag ou campo `import_id` nas transações importadas para rastreabilidade

---

7. Checklist Results Report

Executive Summary

- **Completude geral do PRD:** ~78%
- **Escopo MVP:** Adequado (Just Right)
- **Prontidão para fase de arquitetura:** Nearly Ready
- **Gaps mais críticos:** Ausência de Project Brief formal, métricas de sucesso não quantificadas, falta de personas de usuário detalhadas

Category Analysis

Category	Status	Critical Issues
1. Problem Definition & Context	PARTIAL	Problema articulado mas sem quantificação de impacto. Sem pesquisa formal.
2. MVP Scope Definition	PASS	Escopo claro com 5 epics. Itens fora de escopo documentados.
3. User Experience Requirements	PASS	Flows cobertos. Plataformas definidas. Paradigmas claros.
4. Functional Requirements	PASS	21 FRs numerados e rastreáveis.

Category	Status	Critical Issues
5. Non-Functional Requirements	PARTIAL	8 NFRs. Faltam: escalabilidade, throughput, disponibilidade.
6. Epic & Story Structure	PASS	5 epics sequenciais, 22 stories com ACs testáveis.
7. Technical Guidance	PARTIAL	Stack documentada. Faltam: riscos técnicos formais.
8. Cross-Functional Requirements	PARTIAL	Entidades implícitas. Faltam: retenção, monitoramento.
9. Clarity & Communication	PASS	Linguagem clara e consistente.

Final Decision

NEARLY READY FOR ARCHITECT — O PRD cobre funcionalidades, UX e estrutura de epics com qualidade. Os gaps identificados são melhorias de qualidade, não bloqueios para o Architect iniciar.

---

8. Next Steps

UX Expert Prompt

> @ux-design-expert — Revise o PRD em `docs/prd.md` para o projeto Dindin (Gestão de Gastos). Foque nas 2 novas features (Orçamentos Mensais e Relatórios) e na melhoria do fluxo de Importação. O app é um PWA mobile-first com React. Defina wireframes conceituais para as telas novas e valide os paradigmas de interação propostos (barras de progresso de orçamento, filtros de relatório, mapeamento de colunas CSV). Considere as 16 core screens listadas na seção UI/UX Goals.

Architect Prompt

> @architect — Crie a arquitetura técnica baseada no PRD em `docs/prd.md` para o projeto Dindin. Stack atual: React 19 + Vite + Supabase (PostgreSQL + Auth + Realtime + Edge Functions). Foque em: (1) estrutura de tabelas novas (budgets, push_subscriptions, import_logs, notification_preferences), (2) Edge Functions para push notifications e cron de vencimentos, (3) estratégia de geração de PDF no client, (4) componente FileUploader multiplataforma. Deploy em Vercel. Sem migração para TypeScript neste momento.