Development Roadmap

Timeline Overview

Total Duration: 10 weeks (extended from original 8 weeks) **Target Launch:** Week 10
Rationale: The addition of projects, wallets, and category-first navigation adds complexity that requires additional development time while maintaining quality and thorough testing.

Week 1: Project Setup & Foundation

Goals

- Establish development environment and repository
- Finalize technical decisions
- Set up core infrastructure
- Create detailed technical specifications

Tasks

Repository & Documentation (Days 1-2)

- [ ] Create GitHub repository with proper structure
- [ ] Initialize README with project overview
- [ ] Set up project board (GitHub Projects or similar)
- [ ] Document coding standards and conventions
- [ ] Create contribution guidelines
- [ ] Set up issue templates (bug, feature, question)

Development Environment (Days 2-3)

- [ ] Initialize Vite + React + TypeScript project
- [ ] Configure Tailwind CSS
- [ ] Set up ESLint + Prettier
- [ ] Configure Husky for pre-commit hooks
- [ ] Set up VS Code workspace settings
- [ ] Install and configure development dependencies

Backend Setup (Days 3-4)

- [ ] Initialize Node.js + Express + TypeScript project
- [ ] Set up Supabase project (database, auth)
- [ ] Configure Prisma with PostgreSQL connection
- [ ] Create initial database schema design document
- [ ] Set up environment variables structure
- [ ] Configure CORS and security middleware

CI/CD Pipeline (Days 4-5)

- [ ] Set up GitHub Actions for automated testing
- [ ] Configure Vercel project for frontend
- [ ] Configure Render/Railway project for backend
- [ ] Set up staging and production environments
- [ ] Configure automatic deployments on branch updates
- [ ] Set up Sentry for error tracking

Final Setup (Days 5-7)

- [ ] Create detailed data model diagrams
- [ ] Write technical specification for Projects & Wallets
- [ ] Plan component hierarchy and architecture
- [ ] Set up Storybook (optional for component development)
- [ ] Conduct technical review meeting
- [ ] Begin Week 2 preparation

Deliverables

- ✅ Fully configured development environments
- ✅ GitHub repository with CI/CD
- ✅ Supabase project with initial setup
- ✅ Technical specifications document
- ✅ Data model diagrams

Week 2-3: UI/UX Design & Component Library

Goals

- Create comprehensive design system
- Build Figma prototypes for all screens
- Develop reusable component library
- Validate user flows with clickable prototypes

Week 2 Tasks

Design System (Days 1-3)

- [ ] Define color palette in Figma
- [ ] Create typography scale and styles
- [ ] Design icon set (or configure Lucide)
- [ ] Define spacing system (4px base)
- [ ] Create button variants and states
- [ ] Design input field styles
- [ ] Create card component designs
- [ ] Define animation guidelines

Key Screen Mockups (Days 3-5)

- [ ] Welcome/Sign-in screens
- [ ] Onboarding flow (all screens with new currency/wallet setup)
- [ ] Dashboard with project header and wallet cards
- [ ] Categories grid view
- [ ] Category detail view
- [ ] Transaction entry modal (sequential flow)
- [ ] Project switcher dropdown
- [ ] Wallet management screens

Interactive Prototype (Days 5-7)

- [ ] Connect all screens in Figma
- [ ] Add transitions and interactions
- [ ] Create clickable prototype for user flows
- [ ] Document interaction patterns
- [ ] Conduct internal design review
- [ ] Gather feedback and iterate

Week 3 Tasks

Component Library Development (Days 1-4)

- [ ] Set up component structure in React
- [ ] Build Button component (all variants)
- [ ] Build Input component (text, number, date)
- [ ] Build Card component (base + variants)
- [ ] Build Modal/Sheet component
- [ ] Build Dropdown/Select component
- [ ] Build Progress Bar component
- [ ] Build Toast notification system
- [ ] Build FAB component with radial menu
- [ ] Build Loading states (skeleton, spinner)

Layout Components (Days 4-5)

- [ ] Header component with project switcher
- [ ] Navigation components (FAB, radial menu)
- [ ] Screen container with responsive behavior
- [ ] Empty state component
- [ ] Error boundary component

Testing & Documentation (Days 5-7)

- [ ] Write component tests (Jest + RTL)
- [ ] Document components with examples
- [ ] Test responsive behavior (320px - 1440px+)
- [ ] Accessibility audit (WCAG 2.1 AA)
- [ ] Create component demo pages
- [ ] Conduct component library review

Deliverables

- ✅ Complete Figma design system
- ✅ Clickable prototypes for all flows
- ✅ Reusable component library (15-20 components)
- ✅ Component documentation
- ✅ Responsive layouts tested

Week 4: Database, Authentication & Core Data Models

Goals

- Implement complete database schema
- Set up authentication with OAuth
- Create API foundation
- Build data access layer

Tasks

Database Schema Implementation (Days 1-2)

- [ ] Create Prisma schema for all models:
    - User model
    - Project model
    - Wallet model
    - Category model
    - Transaction model
    - Budget model
    - Debt model
- [ ] Generate and run initial migration
- [ ] Set up database indexes
- [ ] Create seed script for standard categories
- [ ] Test schema with sample data

Row Level Security (Days 2-3)

- [ ] Implement RLS policies for user isolation
- [ ] Create policies for project ownership
- [ ] Test RLS with different user scenarios
- [ ] Document security policies

Authentication (Days 3-4)

- [ ] Configure Supabase Auth with OAuth providers
    - Google OAuth
    - Apple Sign-In
    - Microsoft OAuth
- [ ] Implement email/password authentication
- [ ] Set up JWT token handling
- [ ] Create auth middleware for Express

- [ ] Build session management
- [ ] Test authentication flows

API Foundation (Days 4-6)

- [ ] Set up Express routes structure
- [ ] Implement authentication endpoints
- [ ] Create error handling middleware
- [ ] Set up request validation (Zod)
- [ ] Configure CORS properly
- [ ] Implement rate limiting
- [ ] Create API documentation structure

Core API Endpoints (Days 6-7)

- [ ] User endpoints (GET /api/auth/me)
- [ ] Project CRUD endpoints
- [ ] Wallet CRUD endpoints
- [ ] Category CRUD endpoints
- [ ] Test endpoints with Postman/Insomnia
- [ ] Write API integration tests

Deliverables

- ✅ Complete database schema with migrations
- ✅ Working authentication with OAuth
- ✅ Core API endpoints (User, Project, Wallet, Category)
- ✅ API documentation (README or Swagger)
- ✅ Postman collection for testing

Week 5: Frontend State Management & Project/Wallet Features

Goals

- Implement Redux store architecture
- Build project management features
- Build wallet management features
- Connect frontend to backend APIs

Tasks

State Management Setup (Days 1-2)

- [ ] Configure Redux Toolkit
- [ ] Set up Redux Persist
- [ ] Create store structure (auth, projects, wallets, categories, transactions, sync, ui)
- [ ] Implement authentication slice
- [ ] Create API service layer (axios/fetch)
- [ ] Set up React Query (optional)

Authentication Frontend (Days 2-3)

- [ ] Build sign-in/sign-up screens
- [ ] Implement OAuth flows (Google, Apple, Microsoft)
- [ ] Create protected route wrapper
- [ ] Build authentication state management
- [ ] Handle token refresh
- [ ] Test authentication on frontend

Project Management (Days 3-4)

- [ ] Build project slice in Redux
- [ ] Implement project switcher dropdown
- [ ] Create project management screen
- [ ] Build create/edit/delete project flows
- [ ] Implement "set as primary" logic
- [ ] Connect to backend APIs
- [ ] Handle project switching with state updates

Wallet Management (Days 4-6)

- [ ] Build wallet slice in Redux
- [ ] Create wallet cards on dashboard
- [ ] Implement wallet creation flow
- [ ] Build wallet detail view
- [ ] Create wallet edit/adjustment features
- [ ] Implement wallet transfer preparation (UI only, full logic in Week 6)
- [ ] Connect to backend APIs

Testing & Refinement (Days 6-7)

- [ ] Test project switching thoroughly
- [ ] Test wallet CRUD operations
- [ ] Verify state persistence
- [ ] Check error handling
- [ ] Responsive testing on multiple devices
- [ ] Fix bugs and polish UI

Deliverables

- ✅ Redux store with core slices
- ✅ Working project management system
- ✅ Wallet management features
- ✅ Frontend connected to backend for projects/wallets
- ✅ State persistence working

Week 6: Transaction System & Category Features

Goals

- Implement complete transaction system
- Build category navigation and management
- Create transaction entry flow
- Implement recurring transaction logic

Tasks

Transaction API (Days 1-2)

- [ ] Create transaction CRUD endpoints
- [ ] Implement transaction validation
- [ ] Build bulk sync endpoint
- [ ] Create transaction filtering/querying
- [ ] Handle recurring transaction generation (server-side logic)
- [ ] Test transaction APIs thoroughly

Transaction Frontend - Redux & Logic (Days 2-3)

- [ ] Build transaction slice in Redux
- [ ] Implement wallet balance calculation logic
- [ ] Create Free to Spend calculation engine
- [ ] Build transaction filtering logic
- [ ] Set up transaction sync queue

Transaction Entry Flow (Days 3-5)

- [ ] Build sequential transaction modal
- [ ] Implement income flow (amount → wallet → category → recurring → notes → review)
- [ ] Implement expense flow (amount → wallet → category → recurring → notes → review)
- [ ] Implement transfer flow (amount → from wallet → to wallet → fee → notes → review)
- [ ] Add validation at each step
- [ ] Implement back/skip navigation
- [ ] Connect to transaction APIs
- [ ] Test all transaction types

Category Features (Days 5-6)

- [ ] Build categories grid view

- [ ] Create category detail view
- [ ] Implement category budget setting
- [ ] Build custom category creation
- [ ] Implement category hide/show
- [ ] Connect categories to transactions
- [ ] Test category navigation

Recurring Transactions (Days 6-7)

- [ ] Implement recurring pattern creation
- [ ] Build recurring badge/indicator
- [ ] Create edit recurring pattern flow
- [ ] Implement pause/delete recurring
- [ ] Test recurring generation logic
- [ ] Handle edge cases (end dates, confirmations)

Deliverables

- ✅ Complete transaction system (income, expense, transfer)
- ✅ Category navigation and management
- ✅ Recurring transaction features
- ✅ Transaction APIs fully functional
- ✅ Wallet balances updating correctly

Week 7: Budgets, Debt Tracking & Dashboard Polish

Goals

- Implement budget system
- Build debt tracking features
- Polish dashboard with all data
- Integrate all features

Tasks

Budget System (Days 1-2)

- [ ] Create budget API endpoints
- [ ] Build budget slice in Redux
- [ ] Implement category budget setting UI
- [ ] Create budget progress bars
- [ ] Build budget alert logic
- [ ] Test budget calculations
- [ ] Handle budget period (daily/weekly/monthly)

Debt Tracking (Days 2-3)

- [ ] Create debt API endpoints
- [ ] Build debt slice in Redux
- [ ] Implement debt creation flow
- [ ] Build debt list view
- [ ] Create debt detail/progress view
- [ ] Implement debt repayment linking
- [ ] Test debt calculations

Dashboard Integration (Days 3-5)

- [ ] Connect Free to Spend calculation with real data
- [ ] Display wallet cards with live balances
- [ ] Show budget progress
- [ ] Display upcoming bills
- [ ] Show top spending categories
- [ ] Add recent transactions section (optional)
- [ ] Implement dashboard loading states
- [ ] Add empty states for new users

Reports View (Days 5-6)

- [ ] Build reports screen layout
- [ ] Implement time-based transaction list
- [ ] Create date range filtering
- [ ] Build category/wallet filters
- [ ] Add summary cards (income, expense, net)
- [ ] Implement insights section
- [ ] Create CSV export functionality

Navigation & Polish (Days 6-7)

- [ ] Implement swipe gestures between screens
- [ ] Finalize FAB and radial menu
- [ ] Test navigation flows thoroughly
- [ ] Polish animations and transitions
- [ ] Implement haptic feedback (mobile)
- [ ] Test on multiple devices
- [ ] Fix UI bugs and inconsistencies

Deliverables

- ✅ Budget system with alerts
- ✅ Debt tracking features
- ✅ Fully functional dashboard
- ✅ Reports view with filtering
- ✅ Complete navigation system

Week 8: Offline Functionality & Sync

Goals

- Implement complete offline-first architecture
- Build background sync system
- Create conflict resolution
- Ensure data integrity

Tasks

IndexedDB Setup (Days 1-2)

- [ ] Configure Dexie.js
- [ ] Create local database schema matching backend
- [ ] Implement data access layer for IndexedDB
- [ ] Set up transaction queue table
- [ ] Test local storage operations

Offline Write Operations (Days 2-3)

- [ ] Modify transaction creation to write locally first
- [ ] Implement optimistic UI updates
- [ ] Add pending sync indicators
- [ ] Handle offline wallet balance updates
- [ ] Store offline project/wallet/category changes
- [ ] Test offline transaction creation

Service Worker & Background Sync (Days 3-5)

- [ ] Set up Workbox for Service Worker
- [ ] Configure asset caching strategy
- [ ] Implement background sync API
- [ ] Create sync queue processor
- [ ] Build retry logic with exponential backoff
- [ ] Handle network status detection
- [ ] Test background sync

Sync Logic & Conflict Resolution (Days 5-6)

- [ ] Implement timestamp-based conflict resolution
- [ ] Build sync status tracking
- [ ] Create manual "Sync Now" feature

- [ ] Handle partial sync failures
- [ ] Implement sync progress indicators
- [ ] Test conflict scenarios

PWA Configuration (Days 6-7)

- [ ] Create Web App Manifest
- [ ] Configure service worker registration
- [ ] Set up install prompts
- [ ] Test PWA installation (Android, iOS, Desktop)
- [ ] Verify offline functionality completely
- [ ] Test sync after extended offline periods
- [ ] Handle edge cases (app open during sync, etc.)

Deliverables

- ✅ Fully functional offline mode
- ✅ Background sync system
- ✅ Conflict resolution working
- ✅ PWA installable on all platforms
- ✅ 100% core features work offline

Week 9: Onboarding, Tutorial & Notifications

Goals

- Build complete onboarding flow
- Implement contextual tutorials
- Set up notification system
- Add smart nudges and insights

Tasks

Onboarding Flow (Days 1-3)

- [ ] Build welcome screen
- [ ] Create sign-up/sign-in screen with OAuth
- [ ] Implement currency selection
- [ ] Build wallet setup screen
- [ ] Create income entry screen
- [ ] Build recurring bills setup
- [ ] Implement tracking period selection
- [ ] Create success/completion screen
- [ ] Test entire onboarding flow
- [ ] Ensure default project creation

Contextual Tutorials (Days 3-4)

- [ ] Build tutorial overlay system
- [ ] Implement dashboard tutorial (Free to Spend + FAB)
- [ ] Create categories view tutorial
- [ ] Build wallet tap tutorial
- [ ] Create reports view tutorial
- [ ] Implement project switcher tutorial
- [ ] Add "Skip all tutorials" option
- [ ] Create tutorial settings (reset, toggle)
- [ ] Test tutorial flow thoroughly

Extended Onboarding (Days 4-5)

- [ ] Implement Week 1 delayed setup prompts
- [ ] Build category cleanup prompt
- [ ] Create category budget setup flow
- [ ] Implement debt addition prompt

- [ ] Build notification preferences setup
- [ ] Test delayed onboarding timing

Notification System (Days 5-6)

- [ ] Configure Firebase Cloud Messaging (or alternative)
- [ ] Build notification permission request
- [ ] Implement daily spending reminder
- [ ] Create weekly summary notification
- [ ] Build budget alert notifications (80%, 100%, 120%)
- [ ] Implement quiet hours logic
- [ ] Test notification scheduling
- [ ] Handle notification preferences

Insights & Smart Nudges (Days 6-7)

- [ ] Build spending pattern analysis
- [ ] Create insight generation logic
- [ ] Implement positive reinforcement messages
- [ ] Build time-based insights (weekends, evenings)
- [ ] Create notification timing logic (morning/evening)
- [ ] Test insight accuracy
- [ ] Ensure max 2 notifications per day

Deliverables

- ✅ Complete onboarding experience
- ✅ Contextual tutorial system
- ✅ Notification system with smart timing
- ✅ Insights and nudges working
- ✅ Extended setup prompts

Week 10: Testing, Bug Fixes & Launch Preparation

Goals

- Comprehensive testing across all features
- Fix critical and high-priority bugs
- Performance optimization
- Prepare for production deployment

Tasks

Testing Phase (Days 1-3)

- [ ] **Unit Testing**
    - Test all Redux reducers and actions
    - Test calculation functions (Free to Spend, wallet balances)
    - Test utility functions
    - Achieve >70% code coverage
- [ ] **Integration Testing**
    - Test API endpoints end-to-end
    - Test authentication flows
    - Test offline sync process
    - Test project/wallet CRUD with real data
- [ ] **E2E Testing**
    - Test complete user journey (signup → transaction → budget)
    - Test offline → online sync flow
    - Test project switching scenarios
    - Test transaction entry (all types)
- [ ] **Cross-Device Testing**
    - Test on iOS Safari (iPhone, iPad)
    - Test on Android Chrome (various devices)
    - Test on Desktop (Chrome, Firefox, Edge)
    - Test PWA installation on all platforms
- [ ] **Accessibility Testing**
    - Screen reader testing (NVDA, JAWS, VoiceOver)
    - Keyboard navigation testing
    - Color contrast verification
    - Touch target size validation

Bug Fixing (Days 3-5)

- [ ] Prioritize bugs (Critical > High > Medium > Low)

- [ ] Fix critical bugs (data loss, crashes, auth failures)
- [ ] Fix high-priority bugs (sync failures, calculation errors)
- [ ] Fix medium-priority bugs (UI issues, edge cases)
- [ ] Document known low-priority bugs for post-launch
- [ ] Regression testing after fixes
- [ ] Code review for bug fixes

Performance Optimization (Days 5-6)

- [ ] Analyze bundle size and optimize
- [ ] Implement code splitting for routes
- [ ] Optimize images and assets
- [ ] Reduce API response times
- [ ] Optimize database queries (add indexes if needed)
- [ ] Test dashboard load time (<2s target)
- [ ] Test transaction entry time (<30s target)
- [ ] Verify sync performance (<5s for 100 transactions)
- [ ] Run Lighthouse audits and improve scores

Documentation & Deployment (Days 6-7)

- [ ] Write user documentation/FAQ
- [ ] Document known issues and workarounds
- [ ] Create release notes
- [ ] Prepare marketing materials (screenshots, video)
- [ ] Set up production environment variables
- [ ] Configure production database (Supabase)
- [ ] Deploy backend to production (Render/Railway)
- [ ] Deploy frontend to production (Vercel)
- [ ] Set up custom domain (optional)
- [ ] Configure production monitoring (Sentry)
- [ ] Test production deployment thoroughly
- [ ] Create rollback plan

Soft Launch (Day 7)

- [ ] Deploy to production
- [ ] Invite beta users (10-20 people)
- [ ] Monitor for critical issues
- [ ] Gather initial feedback
- [ ] Quick hotfixes if needed
- [ ] Prepare for public launch

Deliverables

- ✅ Comprehensive test suite
- ✅ All critical bugs fixed
- ✅ Performance optimized
- ✅ Production deployment complete
- ✅ Beta users testing
- ✅ Ready for public launch

Post-Launch (Week 11+)

Immediate Post-Launch (Week 11-12)

- Monitor error rates and performance metrics
- Gather user feedback actively
- Fix critical bugs within 24 hours
- Implement quick wins based on feedback
- Conduct user interviews (5-10 users)
- Analyze usage patterns and metrics

Short-Term Roadmap (Week 13-16)

- Implement most-requested features from feedback
- Improve onboarding based on drop-off data
- Enhance performance in slow areas
- Add missing UI polish and animations
- Improve notification timing and content
- Expand tutorial coverage if needed

Feature Additions (Month 2-3)

- Implement Phase 2 features from Section 5:
    - Predictive suggestions
    - Voice/chat input
    - Bank statement import
    - Enhanced analytics
- Consider premium features for monetization
- Build referral/sharing system
- Add more currency support

Risk Mitigation Strategies

Technical Risks

Risk: Offline sync complexity causes data conflicts

- **Mitigation:** Implement timestamp-based resolution early (Week 8)
- **Mitigation:** Extensive testing with various offline scenarios
- **Mitigation:** User-visible sync status and manual sync option
- **Fallback:** Server-side conflict resolution logs for manual review

Risk: Performance issues with large transaction datasets

- **Mitigation:** Implement pagination early (Week 6)
- **Mitigation:** Use virtual scrolling for long lists
- **Mitigation:** Database indexes on frequently queried fields
- **Fallback:** Lazy loading and progressive enhancement

Risk: OAuth integration issues with providers

- **Mitigation:** Use Supabase Auth which handles OAuth complexity
- **Mitigation:** Test OAuth flows early (Week 4)
- **Mitigation:** Provide email/password backup option
- **Fallback:** Focus on one provider (Google) if others fail

Timeline Risks

Risk: Scope creep delays MVP launch

- **Mitigation:** Strict feature prioritization (P0 only for MVP)
- **Mitigation:** Weekly progress reviews
- **Mitigation:** Move P1 features to post-launch if needed
- **Fallback:** Launch with core features only, iterate rapidly

Risk: Unexpected technical challenges in Week 8 (Offline)

- **Mitigation:** Research offline patterns extensively in Week 1
- **Mitigation:** Start basic offline functionality in Week 5
- **Mitigation:** Allocate buffer time in Week 9-10
- **Fallback:** Launch with basic offline (no sync), add later

Risk: Testing reveals critical issues in Week 10

- **Mitigation:** Continuous testing throughout development
- **Mitigation:** Unit tests written alongside features
- **Mitigation:** Weekly integration testing
- **Fallback:** Delay launch by 1-2 weeks if necessary

Resource Risks

Risk: Solo developer burnout

- **Mitigation:** Realistic timeline (10 weeks, not rushed)
- **Mitigation:** Clear priorities and scope limits
- **Mitigation:** Regular breaks and sustainable pace
- **Fallback:** Extend timeline or reduce scope

Risk: Learning curve for new technologies

- **Mitigation:** Week 1 includes research and setup
- **Mitigation:** Use well-documented tools (Supabase, Prisma)
- **Mitigation:** Seek help from communities when stuck
- **Fallback:** Swap tools if truly blocking progress

Success Checkpoints

End of Week 2

- ✅ Complete design system and prototypes
- ✅ Component library with 15+ components
- ✅ User flows validated

End of Week 4

- ✅ Database schema complete
- ✅ Authentication working
- ✅ Core APIs functional

End of Week 6

- ✅ Transaction system working end-to-end
- ✅ Projects and wallets manageable
- ✅ Categories functional

End of Week 8

- ✅ Offline mode fully functional
- ✅ Sync working reliably
- ✅ PWA installable

End of Week 10

- ✅ All P0 features complete
- ✅ Testing passed
- ✅ Production deployed
- ✅ Beta users onboarded

Tools & Workflow

Project Management:

- GitHub Projects or Notion for task tracking
- Daily task list with priorities
- Weekly goals and reviews

Version Control:

- Feature branches for each major feature
- Regular commits with conventional commit messages
- Pull requests even for solo work (documentation)

Communication:

- Weekly progress updates (blog post, Twitter, etc.)
- Document decisions and rationale
- Share learnings with community

Launch Checklist

Pre-Launch:

- [ ] All P0 features implemented and tested
- [ ] Performance meets targets (<2s dashboard, >99% sync)
- [ ] No critical bugs
- [ ] Documentation complete (user guide, FAQ)
- [ ] Privacy policy and terms of service ready
- [ ] Analytics and monitoring set up
- [ ] Error tracking configured (Sentry)
- [ ] Production environment stable
- [ ] Backup and recovery tested

Launch Day:

- [ ] Deploy to production
- [ ] Verify all features working in production
- [ ] Monitor error rates and performance
- [ ] Send launch announcement
- [ ] Post on social media

- [ ] Submit to Product Hunt (optional)
- [ ] Monitor user sign-ups
- [ ] Respond to early feedback quickly

Post-Launch:

- [ ] Daily monitoring for first week
- [ ] Hotfix critical issues within 24 hours
- [ ] Gather user feedback actively
- [ ] Conduct user interviews
- [ ] Analyze metrics against targets
- [ ] Plan iteration based on data

Flexibility & Adaptation

This roadmap is a guide, not a contract. Be prepared to:

- Adjust timeline based on actual progress
- Prioritize differently based on discoveries
- Cut scope if necessary to meet quality bar
- Add buffer time where needed
- Celebrate progress and iterate

Remember: Shipping a solid MVP is better than a delayed perfect product. Focus on core value, launch, and iterate based on real user feedback.