FinCoach – Design Documentation

Ansar Shaikh

Case Study

Smart Financial Coach Application

Palo Alto Network

Project Overview

FinCoach is a personal finance management platform that empowers users to make better financial decisions. It leverages AI-powered categorization, intuitive design, and real-time analytics to help users:

- Automate transaction categorization with intelligent pattern recognition.
- Access real-time insights into spending habits.
- Set and track financial goals with progress monitoring.
- Enjoy a seamless, mobile-first user experience.
- Ensure robust data privacy and security.

Design Choices

User Interface (UI) Design

- Mobile-First: Optimized for smartphones with responsive layouts.
- Clean & Minimalist: Focus on visual clarity and data visualization.
- Accessibility: High contrast colors, large touch targets (44px+).
- Progressive Disclosure: Gradually reveals information to prevent overload.
- Color-Coded Categories: Each category is visually distinct.
- Card-Based Layout: Financial data displayed in modular cards.
- Interactive Charts: Hoverable and clickable insights.
- Real-Time Feedback: Immediate UI responses to user actions.

User Experience (UX) Design

- Navigation: Clear sections (Dashboard, Analytics, Goals, Settings).
- Onboarding: Simplified login/signup flow.
- Data Import: Supports CSV and PDF statement parsing.
- Review: Users can adjust AI categorizations.
- Analytics: Interactive dashboard with spending insights.
- Goal Tracking: Easy setup and monitoring of savings/financial goals.
- Key UX Enhancements: One-click actions, contextual help, error prevention, and undo functionality.

Technical Stack

Frontend

- Framework: React 19.1.1
- Language: TypeScript 5.8.3
- Build Tool: Vite 7.1.2
- Styling: CSS3 with Grid & Flexbox
- Data Visualization: Chart.js 4.5.0 + react-chartjs-2 5.3.0
- State Management: React Hooks (useState, useEffect, useMemo) + Custom Hooks

Backend (BaaS)

- Platform: Firebase 12.2.1
- Database: Firestore (NoSQL, real-time)

- Authentication: Firebase Auth (email/password, secure tokens)
- Hosting: Firebase Hosting (CDN + HTTPS)

Data Processing

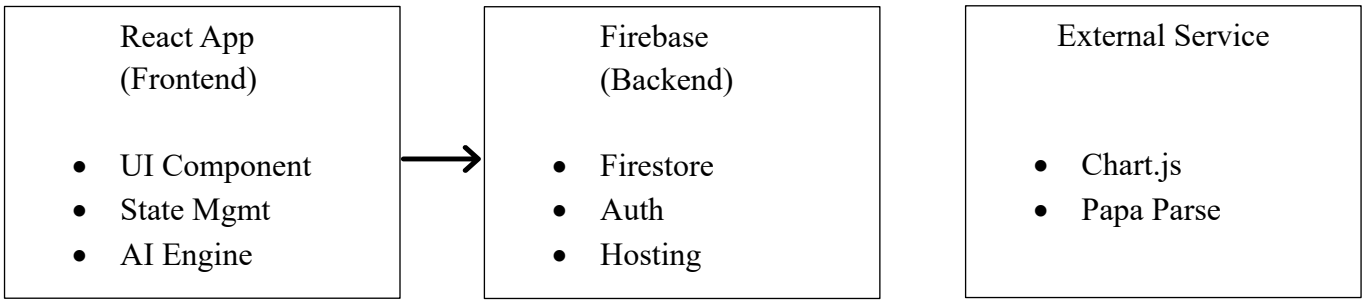
- CSV Parsing: Papa Parse 5.5.3
- AI Categorization: Client-side rule-based engine

Development Tools

- Code Quality: ESLint 9.33.0 + TypeScript ESLint rules
- Build & Deploy: Vite + Firebase CLI

Architecture

System Architecture



System Data Flow

1. User Input → React Components
2. State Updates → React Hooks
3. Data Processing → AI Engine (client-side)

4. Persistence → Firestore
5. Real-Time Sync → Firebase Listeners
6. UI Updates → React Re-renders

Component Architecture

```
App
├─── Pages
│    ├─── Login
│    ├─── Dashboard
│    ├─── Import
│    └─── Transactions
├─── Components
│    ├─── KPI
│    ├─── Chart
│    └─── TransactionItem
├─── Hooks
│    ├─── useAuth
│    ├─── useTransactions
│    ├─── useUserData
│    └─── useImportHistory
└─── Lib
   ├─── ai.ts (Categorization Engine)
   └─── importCsv.ts
```

- Onboarding & Authentication
- Data Import & Parsing
- Categorization Engine
- Analytics Dashboard
- Goal Tracking & Insights

AI Implementation

Categorization Engine

- Approach: Rule-based pattern matching (no external API, works offline).
- Database: 1000+ merchant patterns mapped to 14 categories.
- Fallback: Transactions default to “Other” if unmatched.

```
const categories = [
'Coffee', 'Food Delivery', 'Groceries', 'Subscription', 'Gas',
'Transportation', 'Shopping', 'Electronics', 'Clothing',
  'Entertainment', 'Dining', 'Utilities', 'Fees', 'Other'
];
const categorizeMerchant = (merchant: string): string => {
const lower = merchant.toLowerCase();
  // Gas stations
  if (lower.includes('shell') || lower.includes('sunoco') ||
      lower.includes('gas') || lower.includes('station')) {
    return 'Gas';
  }
  // Dining
  if (lower.includes('mcdonalds') || lower.includes('chick-fil-a') ||
      lower.includes('restaurant') || lower.includes('dining')) {
    return 'Dining';
```

- Remove store numbers, phone numbers, and state abbreviations.
- Normalize whitespace for consistent processing.

Statement Parser

- Parses TXT/CSV statements.

```
MM/DD  MERCHANT NAME    AMOUNT
06/23  PTC EZPASS CSC WEB IVR 877-736-6727 PA   11.60
```

1. Line Parsing: Split by lines and filter empty

2. Date Extraction: Parse MM/DD format, assume current year
3. Amount Extraction: Parse numeric values, filter negative amounts
- Merchant Extraction: Extract text between date and amount
5. Categorization: Apply AI categorization engine
6. Deduplication: Remove duplicate transactions
7. CSV Generation: Output structured CSV format

Database Design

User Collection:

```
users/{userId} {
  uid: string
  email: string
  displayName: string
  createdAt: Timestamp
  lastLoginAt: Timestamp
}
```

Transactions Collection:

```
users/{userId}/transactions/{transactionId} {
  date: string (YYYY-MM-DD)
  amount: number
  merchant: string
  category: string
  source: string
  memo: string
  createdAt: Timestamp
}
```

Import History Collection:

```
users/{userId}/imports/{importId} {
  fileName: string
  fileSize: number
  transactionCount: number
  importDate: Timestamp
}
```

Data Relationships

- One-to-Many: User → Transactions
- One-to-Many: User → Import History
- No Cross-User Data: Each user's data is isolated

Indexing Strategy

Firestore Indexes:

- date (descending) - For transaction sorting
- category - For category-based queries
- merchant - For search functionality

Security Considerations

Authentication & Authorization

Firebase Auth Integration:

- Email/password authentication
- Secure session management
- Automatic token refresh
- Logout functionality

Data Access Control:

- User-specific data isolation
- Firestore security rules
- No cross-user data access

Data Privacy

Client-Side Processing:

- All AI categorization happens in browser
- No data sent to external AI services
- User data stays within Firebase

Data Encryption:

- Firebase handles encryption at rest
- HTTPS for all communications
- Secure authentication tokens

Performance Optimizations

Frontend Optimizations

React Optimizations:

I. useMemo for expensive calculations
II. useCallback for event handlers
III. Component memoization where appropriate
IV. Lazy loading for large datasets

Bundle Optimization:

- Vite's built-in code splitting
- Tree shaking for unused code
- Optimized asset loading

Data Processing Optimizations

Client-Side Processing:

- Efficient pattern matching algorithms
- Cached categorization results
- Batch processing for large datasets
- Debounced search functionality

Future Enhancements

Short-term (3-6 months)

Enhanced AI Features:

- Machine learning model integration
- Improved categorization accuracy
- Learning from user corrections
- Merchant name normalization

User Experience:

- Dark mode support
- Advanced filtering options
- Export functionality (PDF reports)
- Data backup and restore

Analytics Improvements:

- More chart types (line charts, bar charts)
- Custom date ranges

- Category comparison tools
- Spending trend analysis

Medium-term (6-12 months)

Mobile Applications:

- Native iOS app (React Native)
- Native Android app (React Native)
- Offline functionality
- Push notifications

Advanced Features:

- Budget planning and tracking
- Bill reminders and alerts
- Investment tracking
- Receipt scanning (OCR)

Integration Capabilities:

- Bank API integrations
- Credit card API connections
- Third-party financial services
- Data import from other platforms

Long-term (12+ months)

AI & Machine Learning:

- Custom AI models for categorization
- Predictive analytics
- Anomaly detection
- Personalized insights

Enterprise Features:

- Multi-user accounts
- Team financial management
- Advanced reporting
- API for third-party integrations

Advanced Analytics:

- Machine learning insights
- Predictive spending models
- Financial health scoring
- Goal optimization recommendations

Deployment

Current Deployment

Platform: Firebase Hosting

Domain: Custom domain configuration

SSL: Automatic HTTPS

CDN: Global content delivery

Deployment Process

1. Build: `npm run build`
2. Deploy: `firebase deploy`
3. Verification: Automated testing
4. Monitoring: Firebase Analytics

Environment Configuration

Development:

- Local development server
- Firebase emulators
- Hot reload enabled

Production:

- Optimized build
- Firebase production project
- Error monitoring
- Performance tracking

Conclusion

FinCoach represents a modern approach to personal finance management, combining intuitive design with powerful AI-driven categorization. The technical stack chosen prioritizes performance, security, and user experience while maintaining scalability for future enhancements.The rule-based AI approach provides immediate value without external dependencies, while the Firebase backend ensures reliable, real-time data synchronization. The mobile-first design ensures accessibility across all devices, making financial management truly portable.Future development will focus on enhancing the AI capabilities, expanding mobile functionality, and adding advanced analytics features to provide even deeper insights into users' financial patterns and opportunities for improvement.