Finex v1 - Comprehensive Project Explanation

Project Overview

Finex is a sophisticated full-stack web application designed to revolutionize personal finance management through intelligent automation and data-driven insights. At its core, the application enables users to digitize their financial records by simply uploading receipt images, which are then processed using advanced Optical Character Recognition (OCR) technology combined with artificial intelligence. The system automatically extracts itemized transaction details, categorizes expenses, and provides comprehensive analytics to help users understand and optimize their spending habits.

The application addresses a common pain point for individuals who struggle to track their expenses manually. Traditional expense tracking requires tedious manual data entry, which often leads to incomplete records and missed insights. Finex eliminates this friction by automating the entire process from receipt capture to financial analysis, making personal finance management accessible and efficient for everyone.

Core Features and Capabilities

User Authentication and Security

The authentication system in Finex employs a multi-layered security approach combining Supabase authentication services with custom OTP (One-Time Password) verification. When a new user signs up, the system generates a secure six-digit OTP code that is sent to their email address using the Resend email service. This code remains valid for ten minutes and allows up to three verification attempts before expiring for security purposes. The OTP records are stored in MongoDB with automatic expiration through TTL (Time-To-Live) indexes, ensuring that sensitive verification data is automatically purged after use.

Once verified, users receive a JWT (JSON Web Token) from Supabase, which is stored in the browser's local storage and included in all subsequent API requests. The backend validates these tokens on every request through a dedicated authentication middleware, ensuring that users can only access their own data. The system supports secure session management with proper logout functionality that clears all stored credentials and redirects users to the login page.

Receipt Upload and OCR Processing

The receipt processing pipeline represents one of the most technically sophisticated aspects of Finex. Users can upload receipt images through an intuitive drag-and-drop interface or traditional file browser selection. The system supports batch uploads, allowing multiple receipts to be processed simultaneously for efficiency.

When a receipt image is uploaded, it first undergoes validation to ensure it meets the accepted file type and size requirements. The image is then saved to the server's storage directory and passed to the OCR processing engine. Finex utilizes PaddleOCR, an advanced open-source OCR library that excels at extracting text from complex document layouts such as receipt images. PaddleOCR's vision-language model is specifically optimized for receipt processing, capable of handling various layouts, fonts, and image qualities commonly found in retail receipts.

The extracted raw text is then parsed through a multi-stage processing pipeline. The primary parser uses PaddleOCR's layout detection capabilities to identify distinct sections of the receipt including headers, item listings, and totals. A fallback regex-based parser ensures reliable extraction even when layout detection is suboptimal. The parsing algorithms identify and extract key information including the store name, transaction date, individual items with their prices and quantities, subtotals, tax amounts, and grand totals.

After text extraction and parsing, each item undergoes intelligent categorization using a sophisticated fuzzy matching algorithm. This system compares item names against a comprehensive database of over one thousand keywords organized across ten expense categories. The matching algorithm employs a three-tier priority system: exact substring matching yields the highest confidence scores, partial matching handles incomplete item names, and fuzzy similarity matching using Python's difflib SequenceMatcher catches variations and typographical errors. Items that cannot be confidently matched to any category default to "Other" for manual categorization.

OCR Verification Workflow

Understanding that automated OCR is not always perfect, Finex implements a user-friendly verification workflow. After OCR processing completes, users are presented with a modal dialog displaying all extracted receipt information in a structured, readable format. This interface shows the store name, transaction date, total amount, tax, and a detailed list of all detected items with their individual prices and assigned categories.

Users have the option to verify the extracted data as-is and save it directly to the database, or they can enter edit mode to correct any OCR errors before saving. The edit mode allows modification of all fields including store name, date, amounts, and individual item details. This hybrid approach combines the efficiency of automation with the accuracy of human oversight, ensuring that saved receipt data is always correct.

Budget Planning and Management

The budget planning feature enables users to create comprehensive monthly budgets with category-specific allocations. Users set their monthly income and then distribute funds across expense categories such as Food and Dining, Groceries, Transportation, Entertainment, Utilities, and others. The system supports eleven predefined categories that align with the automatic categorization system used for receipts.

The budget interface provides real-time visualization of spending against allocated amounts for each category. Progress bars and percentage indicators show users how much of their budget remains in each category, with color-coded warnings when spending approaches or exceeds limits. This immediate feedback helps users make informed spending decisions throughout the month.

Budget data is stored in MongoDB with a one-to-one relationship to user accounts, ensuring each user has a single active budget that can be updated as financial situations change. The budget system integrates seamlessly with the analytics engine to provide meaningful comparisons between planned and actual spending.

Bill Reminders and Subscription Tracking

Finex helps users stay on top of recurring financial obligations through dedicated bill reminder and subscription tracking features. Users can create bill entries specifying the biller name, amount due, due date, and category. Bills can be marked as recurring with customizable frequencies including weekly, monthly, and yearly options. The system tracks bill payment status and allows users to set reminder notifications days before due dates.

The subscription management feature provides a centralized view of all ongoing service subscriptions. Users can track streaming services, software subscriptions, gym memberships, and other recurring charges. Each subscription entry includes the service name, cost, billing frequency, next billing date, and payment method. Users can mark subscriptions as active or paused, helping them identify potential savings by canceling unused services.

Comprehensive Analytics Dashboard

The analytics feature transforms raw transaction data into actionable financial insights through a rich dashboard interface. The system calculates and presents multiple analytical views:

Monthly Comparison Analytics show spending trends by comparing current month totals against the previous month, calculating percentage changes, and indicating whether spending is trending upward, downward, or remaining stable. This helps users quickly assess whether their spending habits are improving.

Category Breakdown Charts present spending distribution across all expense categories using interactive pie charts and detailed tables. Users can see at a glance which categories consume the largest portions of their budget, with percentage calculations and item counts for each category.

Six-Month Spending Trends provide historical context through area charts that visualize spending patterns over the past half-year. This long-term view helps users identify seasonal spending variations and track progress toward financial goals.

Daily Spending Patterns within the current month are displayed through line charts, helping users understand which days tend to involve higher spending. This granular view can reveal habits such as increased weekend spending or impulse purchases.

Top Merchants Analysis ranks stores and vendors by total spending and visit frequency, helping users understand their shopping patterns and identify opportunities to consolidate purchases or find better deals.

Financial Projections use historical data to estimate month-end spending totals, warning users if they are on track to exceed their budget. The system calculates average daily spending and extrapolates to provide early alerts about potential budget overruns.

AI-Powered Financial Suggestions

Finex incorporates artificial intelligence to provide personalized financial recommendations. The AI suggestions system operates in two modes to ensure reliability and flexibility.

The primary mode uses Google Gemini Pro on the backend, where the system aggregates comprehensive financial data including all-time spending, recent trends, category breakdowns, pending bills, and budget allocations. This data is structured into a detailed prompt that asks the AI to generate three to ten actionable suggestions covering potential savings opportunities, spending pattern concerns, and budget optimization strategies.

A fallback mode operates on the frontend using the Groq API with Llama 3.3 70B when backend AI processing is unavailable. This mode uses cached data from React Query and localStorage to generate suggestions locally, ensuring users always have access to AI insights even during backend issues.

Each suggestion includes an impact rating (high, medium, or low), potential savings calculations where applicable, and specific actionable advice. Users can dismiss suggestions they've already acted upon, and the system tracks dismissals to avoid repetitive recommendations.

User Profile and Settings

The profile management system allows users to customize their Finex experience. Users can update personal information including display name, phone number, and email preferences. The settings page provides options for notification preferences, allowing users to enable or disable email notifications, push notifications, bill reminders, and budget alerts.

The application supports both light and dark themes through a dedicated theme context provider, allowing users to switch between modes based on their preference or system settings. Currency and language preferences can also be configured to localize the financial display format.

Technology Stack

Frontend Technologies

The frontend application is built using React 18 with TypeScript, providing a type-safe development experience and improved code quality through static type checking. Vite serves as the build tool and development server, offering extremely fast hot module replacement and optimized production builds.

User interface styling is implemented using Tailwind CSS, a utility-first CSS framework that enables rapid UI development through composable utility classes. The design language follows iOS-inspired aesthetics with gradient backgrounds, subtle shadows, and smooth animations throughout the interface.

State management combines React Query (TanStack Query) for server state with React Context API for client-side state. React Query handles all API data fetching with automatic caching, background refetching, and stale-while-revalidate strategies that keep the UI responsive while maintaining data freshness.

Form handling utilizes React Hook Form for efficient, performant form state management with minimal re-renders. Input validation is handled through Zod schema validation, providing type-safe validation rules that integrate seamlessly with TypeScript.

Data visualization is implemented using Recharts, a composable charting library built on React components and D3. The analytics dashboard uses various chart types including area charts, bar charts, line charts, and pie charts to present financial data in intuitive visual formats.

Routing is managed through React Router version 6, providing declarative routing with nested routes and protected route components that redirect unauthenticated users to the login page.

Backend Technologies

The backend is built on FastAPI, a modern Python web framework known for its high performance, automatic API documentation, and native support for asynchronous operations. FastAPI's dependency injection system is used extensively for authentication, database connections, and service layer access.

The application runs on Uvicorn, an ASGI server that provides excellent performance for asynchronous Python web applications. The server supports concurrent request handling essential for OCR processing operations that can take several seconds per image.

MongoDB Atlas serves as the primary database, providing a flexible document-based data model ideal for storing variable receipt data with different item structures. The Motor library provides asynchronous database access, allowing the backend to handle multiple concurrent requests efficiently without blocking on database operations.

PaddleOCR handles text extraction from receipt images. This open-source OCR library was chosen for its accuracy with document images and its ability to run locally without external API dependencies. PaddleOCR's models are loaded lazily to minimize startup time and memory usage when OCR is not being actively used.

Google Gemini AI enhances the parsing and categorization capabilities when available. The Gemini model can provide improved extraction accuracy for complex receipts and powers the AI suggestions feature that generates personalized financial recommendations.

Pydantic provides data validation and serialization throughout the backend. All API request and response bodies are defined as Pydantic models, ensuring type safety and automatic validation. These models also generate the OpenAPI documentation that FastAPI exposes automatically.

External Services

Supabase provides the authentication infrastructure, handling secure password hashing, JWT token generation, and user account management. This choice allows Finex to leverage enterprise-grade authentication without implementing complex security code from scratch.

Resend handles transactional email delivery for OTP verification codes. This reliable email service ensures that verification emails are delivered promptly to users during the signup process.

MongoDB Atlas provides the cloud-hosted database infrastructure with automatic backups, scaling capabilities, and geographic distribution options. The free tier offers 512MB of storage, sufficient for individual users tracking personal finances.

Database Architecture

Collections and Schema Design

The MongoDB database consists of seven primary collections, each optimized for their specific access patterns through carefully designed indexes.

The **receipts** collection stores all uploaded receipt data including store information, transaction dates, amounts, itemized line items with categories, and references to stored images. Each receipt is associated with a userId for multi-tenant data isolation. Indexes on userId, date, and category support efficient filtering and aggregation queries.

The **budgets** collection maintains one document per user containing their monthly budget amount and category allocations. The userId index with unique constraint ensures data integrity and fast lookups.

The **bills** collection tracks recurring and one-time bill payments with amounts, due dates, and payment status. Compound indexes on userId, dueDate, and isPaid support the various query patterns needed for bill reminders and analytics.

The **subscriptions** collection manages ongoing service subscriptions with billing cycle information and next billing dates. Similar indexing strategies support subscription management queries.

The **otps** collection temporarily stores verification codes during the signup process. A TTL index on the expires_at field automatically removes expired codes, maintaining security and database hygiene.

The **profiles** collection stores extended user information beyond what Supabase authentication provides, including notification preferences, currency settings, and contact information.

Application Architecture

Layered Architecture Pattern

The backend follows a clean layered architecture with clear separation of concerns. The API layer, contained in router modules, handles HTTP request parsing, authentication verification, and response formatting. Business logic resides in the services layer, where complex operations like OCR processing, analytics calculations, and budget comparisons are implemented. The data access layer uses Motor for async MongoDB operations, abstracting database specifics from business logic.

Frontend Component Architecture

The React frontend organizes components into logical groupings. Page components in the pages directory represent full routes and compose smaller components. Reusable UI elements live in the components directory, further organized by feature area. Context providers manage authentication state and theme preferences. Custom hooks encapsulate complex logic for reuse across components. Service modules centralize API communication patterns.

Data Flow Patterns

Data flows through the application following predictable patterns. User interactions trigger React event handlers that call API services. Services use Axios to make HTTP requests to the FastAPI backend. The backend validates requests through Pydantic models, processes them through service layers, and returns structured responses. React Query caches responses and manages loading and error states, updating the UI automatically when data changes.

Security Implementation

Authentication Flow

The multi-step authentication process begins with OTP verification before account creation. When a user enters their email for signup, the backend generates a cryptographically random six-digit code and stores it in MongoDB with a ten-minute expiration. The code is emailed through Resend, and only after successful verification does the system create the Supabase user account and return an access token.

All protected API endpoints require valid JWT tokens in the Authorization header. A middleware function extracts and validates tokens against Supabase, extracting the user ID for data isolation in subsequent operations.

Data Protection

User data is strictly isolated through userId filtering on all database queries. Users can only access their own receipts, budgets, bills, and other personal data. The frontend stores authentication tokens in localStorage and includes them automatically in API requests through an Axios interceptor.

CORS configuration restricts API access to the expected frontend origin, preventing cross-origin attacks. Environment variables store all sensitive credentials including API keys and database connection strings, keeping them out of version control.

Deployment and Infrastructure

Development Environment

The development setup runs both frontend and backend locally with hot reloading enabled. The backend starts with Uvicorn on port 8000, and the frontend development server runs on port 5173. Environment files configure local and development credentials separately from production values.

Production Considerations

The frontend can be deployed to Vercel or similar static hosting platforms, with the build process handled by Vite's optimized production bundler. The backend can run on any Python-capable hosting platform with Uvicorn serving the FastAPI application. MongoDB Atlas provides the production database with automatic scaling and backups.

File Storage Architecture

Uploaded receipt images are stored in a local file system structure under the backend's storage directory. Three subdirectories organize files: uploads for original receipt images, thumbnails for generated preview images, and processed for any intermediate processing files. Each uploaded file receives a UUID-based filename to prevent conflicts and ensure uniqueness. File URLs are stored in receipt documents as relative paths, allowing flexible serving strategies in different deployment environments.

Summary

Finex represents a comprehensive solution for personal finance management that combines modern web technologies with artificial intelligence to automate expense tracking and provide actionable financial insights. The application demonstrates best practices in full-stack development including type-safe frontend development with React and TypeScript, high-performance backend development with FastAPI, flexible data modeling with MongoDB, and intelligent automation through OCR and AI integration. The system's architecture prioritizes security, performance, and user experience, making complex financial management accessible through an intuitive interface that minimizes manual data entry while maximizing the value of financial information.