Financial Management System

Complete Backend Architecture Documentation

Project Overview

This is a comprehensive financial management system built with Node.js, Express.js, and MongoDB. The system provides complete CRUD operations for managing users, categories, items, and transactions with advanced features including AI-powered analysis, real-time analytics, and multiple export formats.

Technology Stack

Component	Technology	Version/Details
Backend Framework	Node.js + Express.js	v5.1.0
Database	MongoDB Atlas	Cloud-hosted
ODM	Mongoose	v8.14.0
Authentication	JWT + bcrypt	Secure token-based
File Upload	Multer	v2.0.0
AI Integration	Groq API	Text & Voice Analysis
WebSocket	ws	Real-time updates
PDF Generation	PDFKit	Export functionality
Validation	Joi	Input validation
Security	CORS + Rate Limiting	API protection

Database Schema & Collections

1. Users Collection

Field	Type	Description	Constraints
_id	ObjectId	Primary Key	Auto-generated
fullname	String	Full name	Trimmed
firstName	String	First name	Trimmed
lastName	String	Last name	Trimmed
email	String	Email address	Unique, lowercase
password	String	Hashed password	bcrypt encrypted
phone	String	Phone number	Optional
confEmail	Boolean	Email confirmed	Default: false
isBlocked	Boolean	Account blocked	Default: false

role	String	User role	customer/admin
OTP	String	Verification code	4-6 digits
preferences	Object	User settings	Currency, language, etc.
createdAt	Date	Creation timestamp	Auto-generated

2. Categories Collection

Field	Type	Description	Constraints
_id	ObjectId	Primary Key	Auto-generated
user	ObjectId	User reference	Required, indexed
name	String	Category name	Required, trimmed
items	Array[ObjectId]	Item references	Default: []
color	String	Display color	Default: #ffffff
icon	String	Icon name	Default: category
budget	Number	Budget limit	Default: 0
isDefault	Boolean	System category	Default: false
createdAt	Date	Creation timestamp	Auto-generated
updatedAt	Date	Update timestamp	Auto-generated

3. Items Collection

Field	Type	Description	Constraints
_id	ObjectId	Primary Key	Auto-generated
user	ObjectId	User reference	Required, indexed
name	String	Item name	Required, trimmed
price	Number	Item price	Default: 0
description	String	Item description	Optional, trimmed
isActive	Boolean	Item status	Default: true
createdAt	Date	Creation timestamp	Auto-generated
updatedAt	Date	Update timestamp	Auto-generated

4. Transactions Collection

Field	Type	Description	Constraints
_id	ObjectId	Primary Key	Auto-generated
user	ObjectId	User reference	Required, indexed
category	ObjectId	Category reference	Optional, indexed
price	Number	Transaction amount	Default: 0
text	String	Description	Optional, trimmed
OCR_path	String	Image file path	Optional
voice_path	String	Audio file path	Optional
type	String	Transaction type	expense/income

notes	String	Additional notes	Optional, trimmed
isDeleted	Boolean	Soft delete flag	Default: false
createdAt	Date	Creation timestamp	Auto-generated
updatedAt	Date	Update timestamp	Auto-generated

Database Relationships

The system implements a well-structured relational model using MongoDB's document-based approach:

Relationship Details:

From	To	Type	Description
User	Categories	One-to-Many	Each user can have multiple categories
User	Items	One-to-Many	Each user can have multiple items
User	Transactions	One-to-Many	Each user can have multiple transactions
Category	Items	Many-to-Many	Categories contain array of item references
Category	Transactions	One-to-Many	Each transaction belongs to one category
User	User	Self-Reference	Password change tracking via passwordChangedAt

API Architecture

API Modules Overview:

Module	Base Route	Purpose	Database Models Used
Authentication	/auth	User management & security	User
Categories	/category	Category CRUD operations	Category, Item, Transactions
Items	/items	Item CRUD operations	Item, Category
Transactions	/transactions	Transaction management	Transactions, Category, Item
Analytics	/analytics	Data analysis & reporting	Transactions, Category
Export	/export	Data export (PDF/CSV/JSON)	Transactions
AI	/ai	AI-powered analysis	External APIs (Groq)
WebSocket	ws://localhost:3002	Real-time updates	Analysis results

Detailed API Endpoints

Authentication Endpoints:

Method	Endpoint	Description	Auth Required
POST	/auth/signup	Create new user account	No
POST	/auth/signin	User login	No
POST	/auth/configurationOTP	Verify OTP	No
POST	/auth/changePassword	Change password	Yes
POST	/auth/forgetPassword	Request password reset	No
POST	/auth/setNewPassword	Reset password	No
GET	/auth/profile	Get user profile	Yes
PUT	/auth/profile	Update user profile	Yes

DELETE	/auth/account	Delete user account	Yes

Transaction Endpoints:

Method	Endpoint	Description	Features
POST	/transactions/createWithText	Create text transaction	Text analysis
POST	/transactions/createWithVoice	Create voice transaction	Voice-to-text + analysis
POST	/transactions/createWithOCR	Create image transaction	OCR + analysis
GET	/transactions/my	Get user transactions	Pagination, filtering
GET	/transactions/:id	Get single transaction	User-specific
PUT	/transactions/:id	Update transaction	User-specific
DELETE	/transactions/:id	Delete transaction	User-specific
GET	/transactions/category/:id	Get by category	Category filtering
GET	/transactions/date-range	Get by date range	Date filtering

Security & Middleware

The system implements comprehensive security measures to protect user data and API endpoints:

Security Layer	Implementation	Purpose
Authentication	JWT Tokens	Stateless user authentication
Password Security	bcrypt Hashing	Secure password storage
Route Protection	protectedRoutes Middleware	Endpoint access control
Input Validation	Joi Schemas	Data validation & sanitization
Rate Limiting	Express Rate Limit	API abuse prevention
CORS	Cross-Origin Resource Sharing	Browser security
Input Sanitization	Custom Middleware	NoSQL injection prevention
File Upload Security	Multer with Validation	Safe file handling
Error Handling	Global Error Middleware	Secure error responses

Project Structure

The project follows a modular architecture with clear separation of concerns:

Directory	Purpose	Key Files
src/DB/	Database layer	models/, dbConnection.js
src/module/	Business logic modules	auth/, transactions/, categories/, etc.
src/middleware/	Request processing	auth.js, validate.js, sanitize.js
src/utils/	Utility functions	AppError.js, catchError.js, logger.js
Analysis/	Python analysis service	main.py, voice_main.py
wsClient/	WebSocket client	client.js
Root/	Configuration & entry	index.js, package.json, .env

Current Database Status

Live database statistics from MongoDB Atlas cluster:

Collection	Document Count	Status	Description
users	5	Active	Registered users with confirmed accounts
categories	4	Active	User-created categories with items
items	6	Active	Items distributed across categories
transactions	0	Ready	Ready for transaction creation

System Status & Conclusion

The Financial Management System is fully operational with all components properly integrated: ■ Database: MongoDB Atlas connection established with all collections created and populated ■ Backend: Complete Node.js/Express API with 40+ endpoints ■ Security: JWT authentication, input validation, and rate limiting implemented ■ Features: CRUD operations, file uploads, AI analysis, real-time updates, and data export ■ Architecture: Modular design with clear separation of concerns ■ Documentation: Comprehensive API documentation and code comments The system is production-ready and can handle user registration, transaction management, analytics, and reporting with advanced features like voice-to-text and OCR processing.

Generated on December 26, 2025 at 09:18 PM