### FinSight: Development Journey & Technical Log

This document outlines the complete development process for the FinSight platform, from initial concept and planning to the final integrated application. It serves as a log of our technical decisions, the challenges we faced, and the solutions we implemented along the way.

### The Initial Vision: What We Wanted to Build

The initial idea for FinSight was to create a single, unified fintech platform to serve three distinct user groups: individual customers seeking financial guidance, investment firms managing complex portfolios, and banks streamlining their loan processes. The core of the platform was always envisioned to be AI-powered, allowing users to perform advanced tasks like chatting with a financial co-pilot, receiving personalized investment tips, and analyzing dense financial documents automatically.

Our goal was to build a complete, working platform where:

* **Customers** could track their financial goals and get actionable investment advice.
* **Investment firms** could manage their portfolios and leverage AI for market insights.
* **Banks** could accelerate loan application processing with AI-driven analysis.

### Phase 0: Planning and Tech Stack Selection

Before writing a single line of code, we realized we needed a solid architectural plan. Our initial discussions focused on choosing the right tools for a project with such diverse needs.

* **Frontend Strategy:** We chose **Next.js 15 with the App Router**. This gave us a powerful foundation with server-side rendering for performance and a modern, component-based architecture. For the UI itself, we selected **shadcn/ui** and **Tailwind CSS** to build a professional, responsive interface quickly without starting from scratch.
* **Backend Strategy:** We decided on a split-backend approach. The primary web application backend would be built directly within **Next.js**, handling user authentication, session management, and standard data operations. For the future, computationally-intensive AI tasks, we planned for a separate **Python/FastAPI** microservice. This keeps the application modular and scalable.
* **Database Architecture:** This was a major point of discussion. We quickly concluded that a single database type wasn't optimal. We settled on a **hybrid model**:
  + **MySQL:** Chosen for all our structured, relational data. Its ACID compliance is perfect for handling critical financial records, user accounts, and organizational structures. We initially considered PostgreSQL but opted for MySQL due to broader familiarity and hosting simplicity.
  + **MongoDB:** Selected for its flexibility in handling semi-structured and unstructured data. This was the obvious choice for storing things like AI conversation histories, dynamic customer profiles, and the complex JSON outputs from document analysis.

### Phase 1: Frontend-First Website Build

We made a strategic decision to start with the frontend. It became clear that we can't design effective AI features without first having a concrete user interface and workflow to place them in.

**What we built:**

* A complete, professional landing page to communicate the platform's value.
* A full authentication system with login and signup pages, including logic for role selection.
* Three distinct, tailored dashboards for the Investment, Bank, and Customer personas.
* All major application pages: Goals Management, Document Center, a multi-tabbed Settings page, and the shell for the AI Workbench.

**Problems we faced:**

* Getting used to the new syntax and routing conventions in Next.js 15, especially with the App Router, caused some initial slowdowns.
* We spent a significant amount of time iterating on the layout and component structure to ensure the user flow felt intuitive.
* Implementing the role-based navigation and ensuring the correct dashboard and menu items appeared for each user was more complex than anticipated.

### Phase 2: Database Design & Implementation

With the frontend shell in place, we designed and implemented the database schema. This phase was more challenging than we initially thought.

**Final Database Structure:**

* **MySQL (12 tables):**
  + Core: organizations, users, team\_memberships
  + Investment: portfolios, portfolio\_holdings
  + Bank: loan\_applications, loan\_approval\_rules, transactions
  + Customer: financial\_goals, investment\_products
  + Shared: documents, knowledge\_base\_documents
* **MongoDB (4 collections):**
  + ai\_sessions (chat conversations)
  + customer\_profiles (onboarding data)
  + investment\_tips (AI-generated tips)
  + document\_analysis\_results (AI analysis outputs)

**Problems we faced:**

* A key debate was our primary key strategy. We ultimately decided on a hybrid approach: using UUIDs (as CHAR(36)) for core tables that needed to be referenced from MongoDB, and simple INT AUTO\_INCREMENT keys for tables with purely internal relationships.
* Setting up reliable connection pooling between the Next.js server and both databases took some trial and error to get right.
* We had to be very disciplined in how we managed references between the two databases to avoid data integrity issues.

### Phase 3: Backend API Development (37 Endpoints)

This was the most intensive phase, where we built the logic connecting the UI to the database. We broke it down into smaller, manageable chunks.

* **Authentication (4 endpoints):** Building a custom JWT-based system was a challenge, especially handling token expiration, refresh logic, and secure password hashing with bcrypt. The different logic paths for individual customer vs. organizational admin signup were complex.
* **Onboarding (7 endpoints):** The new file upload handling in Next.js 13+ (without libraries like multer) required some learning. Setting up the MongoDB schemas and connecting them for the customer profiles also took time.
* **Core Features (15 endpoints):** We ran into several tricky SQL issues, especially with parameter binding for LIMIT and OFFSET clauses in our queries. Implementing robust, role-based access control for every endpoint was tedious but necessary.
* **AI Integration (9 endpoints):** Since the actual AI service isn't built yet, we implemented these endpoints with **mock fallback responses**. This allows the entire frontend to be fully functional and integrated. Designing the session management for conversations in MongoDB was a key task here.

### Phase 4: Frontend-Backend Integration

This was the final hurdle where everything came together, and it exposed the most unexpected problems.

**Major Problems We Faced:**

* **Authentication Integration Hell:** Tying the frontend forms to the real APIs and managing the JWT token across page refreshes was difficult. The AuthContext we built needed several revisions to handle all edge cases, like role-based redirects after login.
* **Next.js 15 Compatibility Issues:** The App Router's new syntax for route parameters and async server components caused a lot of TypeScript errors that took time to debug and resolve.
* **SQL Parameter Binding Disasters:** We spent hours debugging a critical error: "Incorrect arguments to mysqld\_stmt\_execute". We eventually discovered that our MySQL prepared statement library required all parameters, even numbers for LIMIT and OFFSET, to be passed as strings.
* **File Upload Complexities:** Handling FormData correctly in API routes, validating file types, and ensuring secure storage and access required careful implementation.
* **State Management Chaos:** The sheer number of API calls meant we had to be rigorous about managing loading and error states for every user interaction to prevent a poor user experience.

### Current Status & Next Steps

**What's Working:**

* A complete, end-to-end authentication system with JWT.
* All 37 API endpoints are fully functional and have been tested.
* The entire frontend is integrated with the backend, displaying live data instead of mockups.
* A full file upload and management system for documents.
* A robust, role-based, multi-tenant architecture is in place.

What's Not Done (The AI Service):

The final piece of the puzzle is to build the Python/FastAPI service. The current Next.js backend is designed to seamlessly switch from using its internal mock responses to calling the real AI service once it's available. All the data structures and UI components are ready for it.

Technical:

We acknowledge that in the push to get the system functional, we've accumulated some technical debt. This includes several TypeScript any types that need proper definition, some ESLint warnings to clean up, and error handling that could be made more granular.

How We Overcame Problems:

The key to our progress was a phased approach. By tackling authentication first, then data display, and finally user interactions, we built on a solid foundation. Debugging the SQL issues required patience and deep-diving into documentation. For Next.js, we learned to rely heavily on the official documentation for version 15, as older tutorials were often misleading. Having a complete Postman collection for our APIs was invaluable during the integration phase.

The journey was longer than expected due to these unforeseen challenges, but the final result is a stable, scalable, and feature-complete platform ready for its AI core.