Low-Level Design: AI Financial Analyst v1.0

Document Version: 1.0
Date: May 24, 2025
Author: [Your AI Architect Persona]
Status: Draft for Development Team

1. Introduction

- **Purpose:** This Low-Level Design (LLD) document provides software developers with the detailed technical specifications required to build Version 1.0 of the AI Financial Analyst application. It translates the functional and non-functional requirements outlined in the AI Financial Analyst BRD v1.0 (link to BRD) into a concrete architectural and component-level design.
- **Architectural Goals:**
  - **Serverless First:** Prioritize managed, serverless components to maximize scalability, resilience, and operational efficiency, and to minimize infrastructure management.
  - **Microservices:** Decompose the system into small, independent, and loosely coupled services, each aligned with a specific business capability (Bounded Context from DDD).
  - **Domain-Driven Design (DDD):** Apply DDD principles to model each microservice around a clear domain and ubiquitous language.
  - **Responsiveness & Resilience:** Design for low latency and high availability.
  - **Scalability:** Ensure the system can handle growth in users and data (targeting NFR-SCA-001).
  - **Evolvability:** Structure for ease of maintenance and future feature additions (addressing "minimal re-engineering costs").
- **Traceability:** Each design component will reference the BRD requirements it addresses (e.g., [FR-XXX-001], [NFR-XXX-001]).

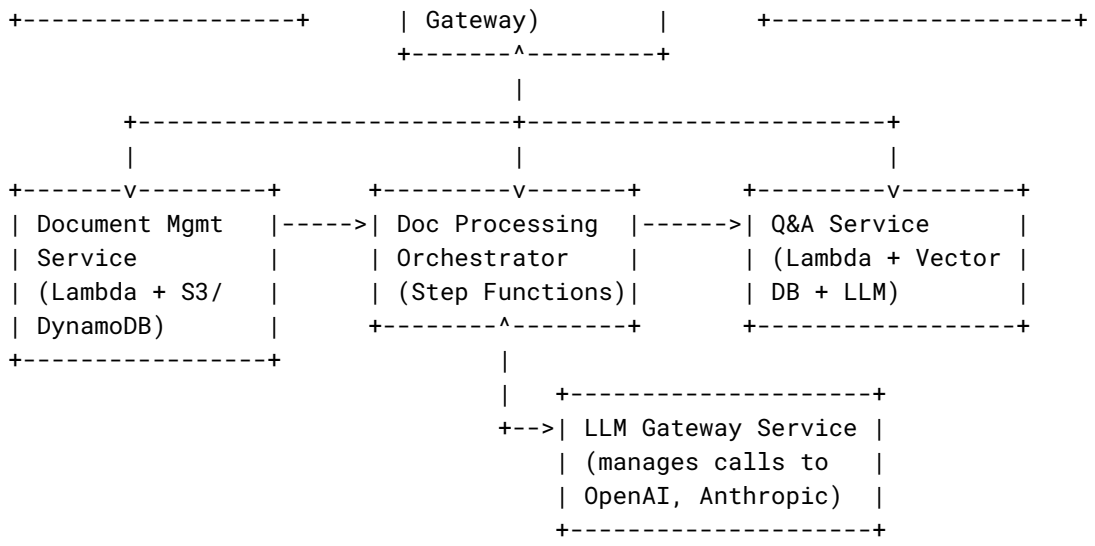
---

2. System Architecture Overview

The AI Financial Analyst will be a cloud-native application built on a microservices architecture, primarily leveraging serverless components.

- **High-Level Flow (Simplified):**
  1. User interacts with a **Frontend Application** (responsive web).
  2. Frontend communicates with a **Backend API Gateway**.
  3. API Gateway routes requests to various **Microservices**.
  4. Microservices are largely **AWS Lambda functions** (or equivalent), interacting with managed data stores (DynamoDB, S3, Vector DB, ElastiCache) and other services (SQS, SNS, Step Functions, LLM APIs).
- **Conceptual Diagram:**

```
+-------------------+      +-----------------+      +---------------------+
| Frontend (Mobile/ |----->| API Gateway     |----->| User Service        |
| Desktop Web App)  |      | (e.g., AWS API  |      | (Lambda + DynamoDB) |
```



Event Bus (e.g., SNS/SQS) facilitates async communication between services.
Other services: Summarization, Key Figure Extraction, Export (likely Lambdas invoked by Orchestrator or directly).

(A more detailed diagram using a diagramming tool would be provided in a full project setting, showing all services and primary data flows.)

- **Technology Choices (Illustrative - AWS Stack):**
  - **Compute:** AWS Lambda
  - **API Gateway:** Amazon API Gateway (RESTful APIs)
  - **Data Storage:**
    - User & Metadata: Amazon DynamoDB
    - Raw Documents: Amazon S3
    - Vector Embeddings: Amazon OpenSearch Service (with k-NN) or Amazon Aurora PostgreSQL (with pgvector extension), or a specialized Vector DB like Pinecone/Weaviate.
    - Cache (for public document analysis): Amazon ElastiCache (Redis/Memcached) or DynamoDB DAX. [FR-SYS-005]
  - **Messaging:** Amazon SQS (queues), Amazon SNS (topics) for asynchronous processing.
  - **Orchestration:** AWS Step Functions for document processing workflows.
  - **Authentication:** Amazon Cognito [NFR-SEC-001]
  - **Logging/Monitoring:** Amazon CloudWatch [NFR-REL-001]
  - **LLM Access:** A dedicated gateway service/Lambda layer to interact with LLM providers (OpenAI, Anthropic, Ollama if self-hosted). This allows for easier model switching and centralized API key management.

---

3. Microservice Design Details

The following microservices will be developed for MVP. Each service operates within its own bounded context.

3.1. User Service

- **Purpose:** Manages user identity, authentication, and basic profile information.
- **Bounded Context:** Identity & Access Management.
- **Domain Model:**
  - **Aggregate:** User (UserID, Email, PasswordHash, Status, CreatedAt, UpdatedAt)
- **Serverless Components:**
  - AWS Lambda functions for business logic.
  - Amazon Cognito for user pool management and authentication.
  - DynamoDB table for user profiles (if attributes beyond Cognito are needed).
- **APIs (via API Gateway):**
  - POST /users/register (Email, Password) -> 201 Created or 400 Bad Request [FR-USR-001]
  - POST /users/login (Email, Password) -> { accessToken, idToken, refreshToken } [FR-USR-002]
  - POST /users/forgot-password (Email) [FR-USR-003]
  - POST /users/reset-password (Token, NewPassword) [FR-USR-003]
  - GET /users/me (Authenticated) -> User profile info [FR-USR-004] (basic info)
- **Data Storage:** Amazon Cognito User Pools. Additional user attributes in DynamoDB if necessary.
- **BRD Traceability:** FR-USR-001, FR-USR-002, FR-USR-003, FR-USR-004 (partially for dashboard data source), NFR-SEC-001.
- **Future Enhancements:**
  - // TODO: Implement social logins (OAuth2/OIDC)
  - // TODO: Add role-based access control (RBAC) for future tiered features
  - // TODO: User preferences management

3.2. Document Management Service

- **Purpose:** Handles document uploads, storage of raw documents, metadata management, and user-document associations.
- **Bounded Context:** Document Lifecycle & Metadata.
- **Domain Model:**
  - **Aggregate:** Document (DocumentID, UserID, FileName, S3Key, UploadTimestamp, Status [Uploading, Processed, Error], ContentType, Size, SHA256Hash [for public doc identification/caching])
  - **Value Object:** DocumentStatus
- **Serverless Components:**
  - AWS Lambda functions for CRUD operations and upload logic.
  - Amazon S3 for storing raw PDF documents.
  - Amazon DynamoDB for document metadata.
- **APIs (via API Gateway):**
  - POST /documents/upload-url (Authenticated, FileName, ContentType) -> { documentId, uploadUrl (pre-signed S3 URL) } [FR-DOC-001]
  - PATCH /documents/{documentId}/status (Internal or authenticated for specific updates, e.g., client confirms upload complete)

    - GET /documents (Authenticated) -> List of user's documents with metadata [FR-USR-004]
    - GET /documents/{documentId}/metadata (Authenticated) -> Document metadata [FR-USR-004]
    - DELETE /documents/{documentId} (Authenticated) [FR-DOC-004]
- **Events Published (SNS/SQS):**
    - DocumentUploadedEvent (DocumentID, UserID, S3Key, FileName, SHA256Hash) - published after successful upload and metadata creation. [Triggers FR-DOC-002, FR-DOC-003]
- **BRD Traceability:** FR-DOC-001, FR-DOC-003 (initial status), FR-DOC-004, FR-USR-004 (source for document list).
- **Future Enhancements:**
    - // TODO: Support for more document types beyond PDF (e.g., DOCX, TXT)
    - // TODO: Document versioning for user-uploaded documents
    - // TODO: Sharing capabilities for documents between users

3.3. Document Processing Orchestrator Service

- **Purpose:** Manages the end-to-end workflow for analyzing a document once uploaded. Checks cache before initiating full processing.
- **Bounded Context:** Document Analysis Orchestration.
- **Serverless Components:**
    - AWS Step Functions state machine to define and execute the workflow.
    - AWS Lambda functions for individual steps within the workflow (e.g., cache check, invoking analysis services).
- **Workflow Steps (triggered by DocumentUploadedEvent):**
    1. **Receive DocumentUploadedEvent**. [FR-DOC-002, FR-DOC-003]
    2. **Cache Check Lambda:**
        - Input: Document SHA256Hash (for public docs like 10-K/Q), DocumentType.
        - Logic: Check ElastiCache/DynamoDB for existing processed results (summary, key figures, vector ID). [FR-SYS-005, FR-SYS-005.1]
        - Output: Cached results if found and valid, or signal to proceed with processing.
        - // TODO: Implement cache invalidation logic for FR-SYS-005.2 (e.g., based on new filing dates)
    3. **If Cache Miss or Stale:**
        - **Text Extraction Lambda:** (If not already done or if specialized processing needed)
            - Input: S3Key. Output: ExtractedTextS3Key.
            - Uses a PDF parsing library (e.g., PyMuPDF, Tika).
        - **Parallel Execution Branch (AWS Step Functions Parallel State):**
            - **Summarization Task:** (Invokes Summarization Service Lambda) [FR-AI-001]
            - **Key Figure Extraction Task:** (Invokes Key Figure Extraction Service Lambda) [FR-AI-002]

        - **Embedding Generation Task:** (Invokes Q&A Service's embedding Lambda) [FR-AI-003]
    4. **Aggregate Results Lambda:** Collects outputs from parallel tasks.
    5. **Store Analysis Results Lambda:**
        - Stores summary, key figures, and vector ID (from Q&A service) in DynamoDB (associated with DocumentID).
        - If it was a public document, populate the cache (ElastiCache/DynamoDB). [FR-SYS-005]
    6. **Update Document Status Lambda:** Update Document metadata in Document Management Service's DynamoDB to 'Processed' or 'Error'. [FR-DOC-003]
    7. **Publish DocumentProcessedEvent** (DocumentID, UserID, Status, links to results).
- **BRD Traceability:** FR-DOC-002, FR-DOC-003, FR-AI-001, FR-AI-002, FR-AI-003 (orchestration part), FR-SYS-005, FR-SYS-005.1, NFR-PER-001.
- **Future Enhancements:**
    - // TODO: Add steps for risk/sentiment analysis as a new parallel task
    - // TODO: Implement more sophisticated error handling and retry mechanisms within Step Functions
    - // TODO: Dynamic workflow based on document type or user preferences

3.4. LLM Gateway Service

- **Purpose:** A centralized internal service/layer to manage interactions with various LLM providers. Not directly user-facing but used by other backend services.
- **Bounded Context:** External LLM Interaction.
- **Serverless Components:** AWS Lambda (can be a Lambda Layer for easy inclusion by other functions).
- **Functionality:**
    - Handles API key management securely (e.g., AWS Secrets Manager).
    - Abstracts specific LLM provider SDKs (OpenAI, Anthropic).
    - Implements retry logic, basic prompt engineering/formatting if common.
    - Manages token limits and potentially basic cost tracking per call.
- **BRD Traceability:** Implicitly supports FR-AI-001, FR-AI-002, FR-AI-003. NFR-EFF-001 (by centralizing and potentially optimizing calls).
- **Future Enhancements:**
    - // TODO: Add support for more LLM providers or fine-tuned models
    - // TODO: Implement LLM response validation and sanitization
    - // TODO: A/B testing different LLMs/prompts

3.5. Summarization Service

- **Purpose:** Generates concise summaries from extracted document text.
- **Bounded Context:** Content Summarization.
- **Serverless Components:** AWS Lambda.
- **API (Internal - invoked by Document Processing Orchestrator):**
    - POST /internal/summarize (ExtractedTextS3Key or text payload) -> { summaryText }

- **Logic:**
    1. Retrieve extracted text.
    2. Chunk text if necessary to fit LLM context window. // TODO: Implement robust text chunking strategy
    3. Construct prompt for summarization (e.g., "Summarize the key financial performance, strategic initiatives, and outlook from the following text...").
    4. Call LLM via LLM Gateway Service.
- **BRD Traceability:** FR-AI-001, NFR-ACC-001 (accuracy of summaries).
- **Future Enhancements:**
    - // TODO: Allow user to specify summary length or focus areas
    - // TODO: Explore different summarization models/techniques for different document sections

3.6. Key Figure Extraction Service

- **Purpose:** Extracts a pre-defined set of key financial figures from document text.
- **Bounded Context:** Financial Data Point Extraction.
- **Serverless Components:** AWS Lambda.
- **API (Internal - invoked by Document Processing Orchestrator):**
    - POST /internal/extract-figures (ExtractedTextS3Key or text payload) -> { extractedFigures: [{ name: "Revenue", value: "100M", sourcePage: 5 }, ...] }
- **Logic:**
    1. Retrieve extracted text.
    2. Use LLM with specific prompts for each key figure or a combined prompt. (e.g., "Extract Revenue, Net Income, EPS... from the text. Provide value and page number if available.")
    3. // TODO: Implement validation rules for extracted figures (e.g., numeric checks, range checks)
    4. // TODO: Enhance source pinpointing (FR-AI-004)
- **Pre-defined Figures (MVP):** Revenue, Net Income, EPS, Operating Cash Flow, Total Debt, Gross Margin %. [FR-AI-002]
- **BRD Traceability:** FR-AI-002, FR-AI-004 (basic source), NFR-ACC-001 (accuracy of figures).
- **Future Enhancements:**
    - // TODO: Support for user-configurable data extraction (BRD Future Consideration)
    - // TODO: Use table parsing techniques in conjunction with LLMs for higher accuracy from financial statements
    - // TODO: Normalize currencies and units

3.7. Q&A Service (Embedding & Query)

- **Purpose:** Enables interactive Q&A on documents by creating vector embeddings and querying them.
- **Bounded Context:** Document Semantic Search & Comprehension.
- **Serverless Components:**
    - AWS Lambda for embedding generation (invoked by Orchestrator).

    - AWS Lambda for handling user queries (invoked by API Gateway).
    - Vector Database (e.g., OpenSearch with k-NN, Pinecone).
- **Internal API for Embedding Generation:**
    - POST /internal/generate-embeddings (ExtractedTextS3Key, DocumentID) -> { vectorDbIndexId }
    - Logic: Chunk text, generate embeddings using an embedding model (via LLM Gateway or dedicated model endpoint), store in Vector DB with DocumentID and chunk metadata. [Part of FR-AI-003]
- **Public API for Queries (via API Gateway):**
    - POST /documents/{documentId}/query (Authenticated, { question: "User's question" }) -> { answer: "AI-generated answer", sources: [{ page: X, snippet: "..."}] } [FR-AI-003, FR-AI-004]
    - Logic:
        1. Generate embedding for the user's question.
        2. Query Vector DB for relevant text chunks from the specified DocumentID.
        3. Pass question and retrieved chunks to an LLM (via LLM Gateway) with a prompt like "Based on the following context, answer the question: ... Question: ... Context: ...".
        4. Format answer and include source references.
- **BRD Traceability:** FR-AI-003, FR-AI-004.
- **Future Enhancements:**
    - // TODO: Implement conversational memory for follow-up questions
    - // TODO: Allow querying across multiple documents for a user
    - // TODO: Fine-tune embedding models for financial domain

3.8. Export Service

- **Purpose:** Generates export files (TXT, CSV) of summaries and key figures.
- **Bounded Context:** Data Dissemination.
- **Serverless Components:** AWS Lambda.
- **API (via API Gateway):**
    - GET /documents/{documentId}/export (Authenticated, { format: "csv" | "txt" }) -> File download or link to S3 pre-signed URL for the generated file. [FR-EXP-001, FR-EXP-002]
- **Logic:**
    1. Retrieve processed summary and key figures for the DocumentID (from DynamoDB where Orchestrator stored them).
    2. Format data into specified format (TXT or CSV).
    3. Return data directly or save to a temporary S3 location and return a pre-signed URL.
- **BRD Traceability:** FR-EXP-001, FR-EXP-002.
- **Future Enhancements:**
    - // TODO: Support for Excel (XLSX) and JSON export formats
    - // TODO: Allow users to select specific data points for export

---

4. Cross-Cutting Concerns

- **Authentication & Authorization:**
  - Amazon Cognito will manage user pools, authentication (JWTs). [NFR-SEC-001]
  - API Gateway will use Cognito authorizers to protect endpoints.
  - Service-to-service authorization will use IAM roles.
- **Logging & Monitoring:**
  - AWS CloudWatch for logs from Lambda, API Gateway, Step Functions. [NFR-REL-001]
  - CloudWatch Alarms for critical errors or performance degradation.
  - Centralized logging solution (e.g., OpenSearch for log analytics) for easier debugging.
  - // TODO: Implement distributed tracing (e.g., AWS X-Ray) across microservices
- **Error Handling & Resilience:**
  - Each Lambda function to implement try-catch blocks and return standardized error responses.
  - API Gateway to map backend errors to appropriate HTTP status codes.
  - SQS Dead-Letter Queues (DLQs) for failed asynchronous messages.
  - Retries with exponential backoff for calls to LLMs and other external services (built into LLM Gateway and SDKs).
  - // TODO: Implement circuit breaker patterns for critical inter-service calls if needed (e.g., using AWS Lambda Powertools)
- **Deployment Strategy (CI/CD):**
  - Infrastructure as Code (IaC): AWS CloudFormation or Terraform.
  - CI/CD Pipelines: AWS CodePipeline / CodeBuild / CodeDeploy (or Jenkins/GitLab CI).
  - Separate environments (dev, staging, prod).
  - Automated unit, integration, and end-to-end tests.
- **Data Management & Caching Strategy:** [FR-SYS-005, NFR-EFF-001]
  - **Public Document Analysis Cache:** (For 10-K, 10-Q etc.)
    - Key: SHA256Hash of the original document + AnalysisVersion (e.g., v1 for initial figure set).
    - Store: Summary, extracted figures, vectorDB reference/ID.
    - Technology: Amazon ElastiCache (Redis) for fast lookups, or a dedicated DynamoDB table with GSI on SHA256Hash.
    - Population: By Document Processing Orchestrator after successful analysis of a public doc.
    - Invalidation: [FR-SYS-005.2] // TODO: Develop a mechanism to detect new versions/amendments of public filings (e.g., periodic check of SEC EDGAR, or allow admin to trigger invalidation). For MVP, cache might have a TTL or be manually clearable.
  - **User-Specific Data:** Stored in DynamoDB and S3, scoped by UserID. Not globally cached.

---

5. User Interface (UI) Design Considerations

- **Framework:** Modern JavaScript framework (React, Vue, Angular). [NFR-USA-001]
- **Responsiveness:** Must be fully responsive for mobile, tablet, and desktop. Mobile-first design approach recommended. [NFR-USA-001]
- **API Interaction:**
  - All backend communication via the central API Gateway.
  - Use JWTs obtained from User Service (Cognito) for authenticated calls.
  - Handle asynchronous operations gracefully (e.g., document processing status updates via polling or WebSockets).
  - // TODO: Consider WebSockets for real-time document processing status updates instead of polling.
- **Key UI Flows:**
  - User registration/login.
  - Document upload interface.
  - Dashboard displaying list of documents and their statuses. [FR-USR-004]
  - View for displaying document summary, extracted figures. [FR-AI-001, FR-AI-002]
  - Interactive Q&A input and display area. [FR-AI-003]
  - Export options. [FR-EXP-001]

---

6. Data Models (High-Level)

- **User Table (DynamoDB - if extending Cognito):**
  - userId (PK - Cognito Sub)
  - email (GSI SK)
  - createdAt, updatedAt
  - // TODO: Add fields for user preferences, subscription tier etc.
- **DocumentMetadata Table (DynamoDB):**
  - documentId (PK)
  - userId (GSI PK - for user-specific queries)
  - fileName, s3Key, contentType, size, uploadTimestamp
  - status (e.g., 'UPLOADED', 'PROCESSING', 'PROCESSED', 'ERROR') (GSI SK with userId for status filtering)
  - sha256Hash (GSI PK - for public doc identification and cache lookup, GSI SK status)
  - analysisResults (Map: { summaryS3Key, extractedFigures: [list], vectorDbIndexId })
  - // TODO: Add versioning info for documents
- **PublicAnalysisCache Table (DynamoDB or ElastiCache structure):**
  - documentSha256Hash (PK)
  - analysisVersion (SK - to allow different versions of analysis for the same doc)
  - summaryText or summaryS3Key
  - extractedFigures (JSON string or map)
  - vectorDbIndexId
  - lastAccessed, createdAt, cacheExpiryTimestamp

    - [FR-SYS-005]
- **Vector DB Structure (Conceptual):**
    - Collection per document type or a global collection with metadata filtering.
    - Vectors with associated documentId, chunkId, original text snippet, page number. [FR-AI-004]

---

7. Scalability, Performance, and Reliability Considerations

- **Scalability:** [NFR-SCA-001]
    - Serverless (Lambda, S3, DynamoDB, API Gateway, Step Functions) components scale automatically with load.
    - Microservices allow independent scaling of components.
    - Asynchronous processing via SQS/SNS handles bursts in uploads.
- **Performance:** [NFR-PER-001]
    - Caching of public document analyses critical for perceived performance. [FR-SYS-005]
    - Optimized LLM prompts and choice of models.
    - Efficient data retrieval from DynamoDB using appropriate indexes.
    - Vector DB choice and indexing strategy for fast semantic search.
    - Frontend optimization (CDN for static assets, code splitting).
- **Reliability:** [NFR-REL-001]
    - Managed services offer high availability.
    - Step Functions for robust, stateful orchestration of document processing.
    - DLQs for message durability.
    - Redundancy in S3 and DynamoDB (multi-AZ by default).

---

8. Future Evolution Path

This serverless, microservice architecture is designed for evolvability:

- **Configurable Data Extraction:** Key Figure Extraction Service can be enhanced to accept user-defined schemas. The LLM prompts and post-processing logic would need to be dynamic.
- **Historical Data Aggregation:** A new "Trend Analysis Service" could consume DocumentProcessedEvents, aggregate data over time for a company (identified by ticker/CIK, which would need to be added to Document metadata), and store time-series data.
- **Comparative Analysis:** Could be built on top of the aggregated historical data, allowing queries across multiple company datasets.
- **New Document Types:** Document Management Service and Processing Orchestrator can be extended. New parsing Lambdas and potentially specialized analysis services might be added.
- **Alerting:** A new "Alerting Service" could subscribe to DocumentProcessedEvents or periodically scan for conditions defined by users.

By keeping services small and focused, new features can often be added as new services or by extending existing ones without major impact on unrelated parts of the system.

This LLD provides a comprehensive blueprint. The next steps involve detailed API contract definitions (e.g., OpenAPI specs), rigorous testing strategies, and iterative development sprints.