Architecture Document

1. Introduction

1. **Purpose**
   The purpose of this document is to outline the architectural design and specifications for the Bank Management System. The system aims to provide an efficient platform for managing customer accounts, facilitating transactions, and ensuring secure access to account-related services.
2. **Scope**
   This document covers the architectural representation, goals, constraints, and deployment view of the Bank Management System. It is designed to serve as a reference for both developers and stakeholders to understand the structure and flow of the system.
3. **Definitions, Acronyms, and Abbreviations**
   - **DBMS**: Database Management System
   - **CRUD**: Create, Read, Update, Delete
   - **UI**: User Interface
   - **API**: Application Programming Interface
4. **References**
   - MySQL Documentation
   - React Documentation
   - RESTful API Standards

2. Architectural Representation

The architecture is a three-tier system comprising the presentation layer, business logic layer, and data layer. The front-end is developed using React, while the backend is powered by a MySQL database, enabling efficient CRUD operations.

3. Architectural Goals and Constraints

1. **Goals**
   - To provide a robust and scalable system capable of handling large volumes of transactions.
   - To ensure data security and integrity for all account-related operations.
   - To offer an intuitive user interface for easy navigation and operation.
2. **Constraints**
   - The system should adhere to ACID (Atomicity, Consistency, Isolation, Durability) properties for transaction management.
   - Integration with MySQL may limit certain NoSQL-based operations.
   - Real-time processing of fund transfers between accounts.

4. Use-Case View

1. **Architecturally Significant Use Cases**
   - **User Registration and Authentication**
     The system should handle secure registration of customers, with role-based access and encrypted password storage.
   - **Account Management**
     Customers should be able to open and close accounts, select account types (savings or current), and update contact information.
   - **Fund Management**
     The system should support deposits, withdrawals, and fund transfers between accounts.
   - **Transaction History and Statements**
     Customers should be able to view account statements and transaction history, with filters for date and type of transaction.

5. Logical View

1. **Architecture Overview – Package and Subsystem Layering**
   The system is divided into three main layers:
   - **Presentation Layer**: React frontend that handles user interactions.
   - **Business Logic Layer**: RESTful APIs that manage account operations and transactions.
   - **Data Layer**: MySQL database that stores details related to customers, accounts, and transactions.

6. Process View

1. **Processes**
   The system includes the following processes:
   - **User Registration and Login**
   - **Account Creation and Closure**
   - **Deposit, Withdrawal, and Fund Transfer**
   - **Viewing Account Statements**
2. **Process to Design Elements**
   - Registration/Login: Handles customer registration and encrypted password authentication.
   - Transaction Operations: Manages all CRUD operations related to accounts and transactions.
   - Account Statements: Generates and filters transaction history.
3. **Process Model to Design**
   The design model will follow a microservices-based approach for handling distinct operations (e.g., account management, transaction management), enhancing modularity and scalability.
4. **Model Dependencies**
   - The front-end depends on the RESTful API for data operations.
   - API endpoints are linked to the database schema for CRUD operations.
5. **Processes to the Implementation**
   The processes will be implemented using MySQL procedures for complex queries and React for dynamic front-end rendering.

7. Deployment View

1. **External Desktop PC**
   - For customer access through a web browser.
2. **Desktop PC**
   - Admin access for managing branches and customer accounts.
3. **Registration Server**
   - For handling user registration and login.
4. **Course Catalog**
   - Not applicable.
5. **Billing System**
   - Integrated for managing fee-related operations (if applicable).

8. Performance

The system should be able to handle multiple concurrent users and support real-time transaction processing with low latency.

9. Quality

The application must maintain high standards of data integrity, security, and ease of use. Comprehensive testing and validation should be performed for each module to ensure compliance with these standards.