BANKING SYSTEM PROJECT - DAY 2: SYSTEM DESIGN

==============================================

ARCHITECTURE OVERVIEW

---------------------

The Indian Banking System follows a layered architecture pattern with clear separation of concerns. The system is designed to be scalable, maintainable, and secure, with proper abstraction layers between the presentation, business logic, and data access components.

SYSTEM ARCHITECTURE

-------------------

1. PRESENTATION LAYER

- CLI Interface: Command-line interface for system administrators

- Web Interface: Modern responsive web application for end users

- REST API: RESTful web services for external integrations

- Mobile Interface: Responsive design for mobile devices

2. BUSINESS LOGIC LAYER

- Service Layer: Core business logic implementation

- Transaction Management: ACID compliance and rollback mechanisms

- Validation Layer: Input validation and business rule enforcement

- Security Layer: Authentication and authorization

3. DATA ACCESS LAYER

- Repository Pattern: Abstracted data access layer

- MongoDB Integration: Primary database for core entities

- DynamoDB Integration: Audit logs and event storage

- Caching Layer: Performance optimization through caching

4. INFRASTRUCTURE LAYER

- Configuration Management: Environment-specific configurations

- Logging Framework: Comprehensive logging and monitoring

- Security Framework: Authentication and authorization

- Monitoring and Alerting: System health monitoring

ENTITY RELATIONSHIP DIAGRAM

---------------------------

CUSTOMER ENTITY

---------------

Attributes:

- customerId (String, Primary Key)

- firstName (String, Required)

- lastName (String, Required)

- email (String, Unique, Required)

- phoneNumber (String, Required)

- address (String, Required)

- city (String, Required)

- state (String, Required)

- pincode (String, Required)

- panNumber (String, Unique, Optional)

- aadharNumber (String, Unique, Optional)

- dateOfBirth (Date, Optional)

- createdDate (Date, Auto-generated)

- lastModifiedDate (Date, Auto-updated)

- status (Enum: ACTIVE, INACTIVE, SUSPENDED)

Relationships:

- One-to-Many with Account entity

- One-to-Many with Transaction entity (as source customer)

- One-to-Many with AuditLog entity

ACCOUNT ENTITY

--------------

Attributes:

- accountId (String, Primary Key)

- accountNumber (String, Unique, Required)

- customerId (String, Foreign Key to Customer)

- accountType (Enum: SAVINGS, CURRENT, FIXED\_DEPOSIT, RECURRING\_DEPOSIT)

- balance (BigDecimal, Required)

- minimumBalance (BigDecimal, Required)

- interestRate (BigDecimal, Optional)

- status (Enum: ACTIVE, INACTIVE, SUSPENDED, CLOSED)

- openedDate (Date, Auto-generated)

- lastTransactionDate (Date, Optional)

- createdDate (Date, Auto-generated)

- lastModifiedDate (Date, Auto-updated)

Relationships:

- Many-to-One with Customer entity

- One-to-Many with Transaction entity

- One-to-Many with AuditLog entity

TRANSACTION ENTITY

------------------

Attributes:

- transactionId (String, Primary Key)

- accountId (String, Foreign Key to Account)

- transactionType (Enum: DEPOSIT, WITHDRAWAL, TRANSFER)

- amount (BigDecimal, Required)

- description (String, Optional)

- sourceAccountId (String, Foreign Key to Account, Optional)

- destinationAccountId (String, Foreign Key to Account, Optional)

- status (Enum: PENDING, COMPLETED, FAILED, CANCELLED)

- transactionDate (Date, Auto-generated)

- processedDate (Date, Optional)

- referenceNumber (String, Optional)

- userId (String, Required)

- createdDate (Date, Auto-generated)

- lastModifiedDate (Date, Auto-updated)

Relationships:

- Many-to-One with Account entity (source account)

- Many-to-One with Account entity (destination account)

- One-to-Many with AuditLog entity

AUDITLOG ENTITY

---------------

Attributes:

- auditId (String, Primary Key)

- actionId (String, Unique, Required)

- userId (String, Required)

- actionType (String, Required)

- entityType (String, Required)

- entityId (String, Optional)

- actionDetails (String, Required)

- timestamp (Date, Auto-generated)

- ipAddress (String, Optional)

- userAgent (String, Optional)

- status (Enum: SUCCESS, FAILURE)

- errorMessage (String, Optional)

- createdDate (Date, Auto-generated)

Relationships:

- Many-to-One with Customer entity (optional)

- Many-to-One with Account entity (optional)

- Many-to-One with Transaction entity (optional)

CLASS DIAGRAM

-------------

CUSTOMER CLASS

--------------

public class Customer {

private String id;

private String firstName;

private String lastName;

private String email;

private String phoneNumber;

private String address;

private String city;

private String state;

private String pincode;

private String panNumber;

private String aadharNumber;

private LocalDate dateOfBirth;

private LocalDateTime createdDate;

private LocalDateTime lastModifiedDate;

private CustomerStatus status;

// Constructors, getters, setters

// Validation methods

// Business logic methods

}

ACCOUNT CLASS

-------------

public class Account {

private String id;

private String accountNumber;

private Customer customer;

private AccountType accountType;

private BigDecimal balance;

private BigDecimal minimumBalance;

private BigDecimal interestRate;

private AccountStatus status;

private LocalDateTime openedDate;

private LocalDateTime lastTransactionDate;

private LocalDateTime createdDate;

private LocalDateTime lastModifiedDate;

// Constructors, getters, setters

// Transaction methods (deposit, withdraw)

// Validation methods

// Business logic methods

}

TRANSACTION CLASS

-----------------

public class Transaction {

private String id;

private String transactionId;

private Account account;

private TransactionType type;

private BigDecimal amount;

private String description;

private Account sourceAccount;

private Account destinationAccount;

private TransactionStatus status;

private LocalDateTime transactionDate;

private LocalDateTime processedDate;

private String referenceNumber;

private String userId;

private LocalDateTime createdDate;

private LocalDateTime lastModifiedDate;

// Constructors, getters, setters

// Processing methods

// Validation methods

// Business logic methods

}

AUDITLOG CLASS

--------------

public class AuditLog {

private String id;

private String actionId;

private String userId;

private String actionType;

private String entityType;

private String entityId;

private String actionDetails;

private LocalDateTime timestamp;

private String ipAddress;

private String userAgent;

private AuditStatus status;

private String errorMessage;

private LocalDateTime createdDate;

// Constructors, getters, setters

// Logging methods

// Validation methods

}

SERVICE LAYER DESIGN

--------------------

DEPOSIT SERVICE

---------------

public class DepositService {

private AccountRepository accountRepository;

private TransactionRepository transactionRepository;

private AuditService auditService;

private UndoRedoStack undoRedoStack;

public Transaction processDeposit(String accountNumber, BigDecimal amount, String description, String userId);

private void validateDepositAmount(BigDecimal amount);

private void updateAccountBalance(Account account, BigDecimal amount);

private void logDepositTransaction(Transaction transaction, String userId);

}

WITHDRAW SERVICE

----------------

public class WithdrawService {

private AccountRepository accountRepository;

private TransactionRepository transactionRepository;

private AuditService auditService;

private UndoRedoStack undoRedoStack;

public Transaction processWithdrawal(String accountNumber, BigDecimal amount, String description, String userId);

private void validateWithdrawalAmount(BigDecimal amount);

private void validateSufficientBalance(Account account, BigDecimal amount);

private void updateAccountBalance(Account account, BigDecimal amount);

private void logWithdrawalTransaction(Transaction transaction, String userId);

}

TRANSFER SERVICE

----------------

public class TransferService {

private AccountRepository accountRepository;

private TransactionRepository transactionRepository;

private AuditService auditService;

private UndoRedoStack undoRedoStack;

public Transaction processTransfer(String sourceAccountNumber, String destinationAccountNumber, BigDecimal amount, String description, String userId);

private void validateTransferAmount(BigDecimal amount);

private void validateAccounts(String sourceAccountNumber, String destinationAccountNumber);

private void validateSufficientBalance(Account sourceAccount, BigDecimal amount);

private void updateAccountBalances(Account sourceAccount, Account destinationAccount, BigDecimal amount);

private void logTransferTransaction(Transaction transaction, String userId);

}

AUDIT SERVICE

-------------

public class AuditService {

private AuditLogRepository auditLogRepository;

private DynamoDBService dynamoDBService;

public void logSuccess(String userId, String actionType, String entityType, String entityId, String details);

public void logFailure(String userId, String actionType, String entityType, String entityId, String details, String errorMessage);

public List<AuditLog> getAuditLogs(String userId, LocalDateTime fromDate, LocalDateTime toDate);

public List<AuditLog> getAuditLogsByActionType(String actionType);

private String generateActionId();

private void logToMongoDB(AuditLog auditLog);

private void logToDynamoDB(AuditLog auditLog);

}

DATABASE DESIGN

---------------

MONGODB COLLECTIONS

-------------------

Customers Collection:

{

"\_id": "ObjectId",

"firstName": "String",

"lastName": "String",

"email": "String",

"phoneNumber": "String",

"address": "String",

"city": "String",

"state": "String",

"pincode": "String",

"panNumber": "String",

"aadharNumber": "String",

"dateOfBirth": "ISODate",

"createdDate": "ISODate",

"lastModifiedDate": "ISODate",

"status": "String"

}

Accounts Collection:

{

"\_id": "ObjectId",

"accountNumber": "String",

"customerId": "ObjectId",

"accountType": "String",

"balance": "Number",

"minimumBalance": "Number",

"interestRate": "Number",

"status": "String",

"openedDate": "ISODate",

"lastTransactionDate": "ISODate",

"createdDate": "ISODate",

"lastModifiedDate": "ISODate"

}

Transactions Collection:

{

"\_id": "ObjectId",

"transactionId": "String",

"accountId": "ObjectId",

"transactionType": "String",

"amount": "Number",

"description": "String",

"sourceAccountId": "ObjectId",

"destinationAccountId": "ObjectId",

"status": "String",

"transactionDate": "ISODate",

"processedDate": "ISODate",

"referenceNumber": "String",

"userId": "String",

"createdDate": "ISODate",

"lastModifiedDate": "ISODate"

}

AuditLogs Collection:

{

"\_id": "ObjectId",

"actionId": "String",

"userId": "String",

"actionType": "String",

"entityType": "String",

"entityId": "String",

"actionDetails": "String",

"timestamp": "ISODate",

"ipAddress": "String",

"userAgent": "String",

"status": "String",

"errorMessage": "String",

"createdDate": "ISODate"

}

DYNAMODB TABLES

---------------

AuditLogs Table:

- Partition Key: actionId (String)

- Sort Key: timestamp (String)

- Attributes:

- userId (String)

- actionType (String)

- entityType (String)

- entityId (String)

- actionDetails (String)

- ipAddress (String)

- userAgent (String)

- status (String)

- errorMessage (String)

SECURITY DESIGN

---------------

1. AUTHENTICATION

- User authentication through secure login

- Session management with timeout

- Password encryption and hashing

- Multi-factor authentication support

2. AUTHORIZATION

- Role-based access control (RBAC)

- Permission-based authorization

- Resource-level security

- API endpoint protection

3. DATA SECURITY

- Encryption at rest and in transit

- Sensitive data masking

- Audit trail for all access

- Data privacy compliance

4. INPUT VALIDATION

- Server-side validation for all inputs

- SQL injection prevention

- XSS protection

- CSRF protection

PERFORMANCE DESIGN

------------------

1. CACHING STRATEGY

- Redis for session caching

- Application-level caching

- Database query result caching

- Static content caching

2. DATABASE OPTIMIZATION

- Proper indexing strategy

- Query optimization

- Connection pooling

- Read replica configuration

3. SCALABILITY DESIGN

- Horizontal scaling capability

- Load balancing

- Microservices architecture

- Containerization support

INTEGRATION DESIGN

------------------

1. EXTERNAL SYSTEMS

- Payment gateway integration

- SMS/Email notification services

- Third-party authentication providers

- Regulatory reporting systems

2. API DESIGN

- RESTful API architecture

- JSON data format

- API versioning strategy

- Rate limiting and throttling

3. MESSAGE QUEUES

- Asynchronous processing

- Event-driven architecture

- Message reliability

- Dead letter queue handling

This design document provides the foundation for implementing the Banking Transaction System with proper architecture, security, and scalability considerations.