Banking Core - API Documentation

Functional Implementation & API Routes Reference

Version: 1.0.0
Last Updated: January 2026
Target Audience: DBMS Undergraduates, Faculty, Software Engineers

1. System Architecture Overview

Banking Core uses a **layered architecture** pattern:

```
┌─────────────────────────────────────────────────────────┐
│                      Client Layer                       │
│          (Next.js Pages / External API Consumers)       │
└─────────────────────────────────────────────────────────┘
                           │
                           ▼
┌─────────────────────────────────────────────────────────┐
│                      API Layer                          │
│                /api/v1/* Route Handlers                 │
│    ┌─────────────────────────────────────────────┐     │
│    │ withErrorHandler → withAuth → validateBody │     │
│    │         (Middleware Chain)                  │     │
│    └─────────────────────────────────────────────┘     │
└─────────────────────────────────────────────────────────┘
                           │
                           ▼
┌─────────────────────────────────────────────────────────┐
│                     Service Layer                       │
│    auth-service │ transaction-service │ audit-service  │
│              account-service │ analytics-service        │
└─────────────────────────────────────────────────────────┘
                           │
                           ▼
┌─────────────────────────────────────────────────────────┐
│                    Database Layer                       │
│    ┌───────────────────────────────────────────────┐   │
│    │  MySQL 8 │ Stored Procedures │ Transactions   │   │
│    └───────────────────────────────────────────────┘   │
└─────────────────────────────────────────────────────────┘
```

Technology Stack

Component	Technology
Framework	Next.js 16 (App Router)
Database	MySQL 8 with InnoDB
Auth	JWT (access + refresh tokens)
Validation	Zod schemas
PDF Generation	pdf-lib

2. API Versioning & Design Strategy

Base URL

```
/api/v1/*
```

Design Principles

1. **RESTful** resource naming
2. **Role-based** URL prefixes (`/banker/*` , `/auditor/*` , `/admin/*`)
3. **Stored procedures** for all money movement
4. **Idempotency** headers for financial operations

Response Format

All responses follow this structure:

```
{
  "success": true|false,
  "data": { ... },
  "error": "Error message if failed",
  "meta": {
    "page": 1,
    "limit": 25,
    "total": 100,
    "totalPages": 4
  }
}
```

3. Authentication & Authorization Flow

Login Flow

```
Client                    API                    Database
  │                        │                        │
  │──POST /auth/login─────►│                        │
  │  {email,password,type} │                        │
  │                        │──SELECT users/customers─►│
  │                        │◄────credentials row────│
  │                        │                        │
  │                        │ bcrypt.compare()       │
  │                        │ signToken()            │
  │                        │                        │
  │◄──{token, refreshToken,│                        │
  │    user}───────────────│                        │
```

Token Structure

Access Token (JWT):

```
{
  "sub": 1,          // User/Customer ID
  "type": "user",    // "user" or "customer"
  "email": "...",
  "iat": 1706000000,
  "exp": 1706003600  // 1 hour expiry
}
```

Middleware Chain

Every protected route passes through:

```
withErrorHandler(
  withAuth(request, handler, {
    requiredType: 'user',
    requiredRoles: ['BANKER'],
    requiredPermissions: ['transactions:*']
  })
)
```

4. Role-Based API Access Control

Role	Code	Permission	Accessible Modules
Admin	ADMIN	`["*"]`	All endpoints
Banker	BANKER	`["transactions:*"]`	/banker/, */core/*
Auditor	AUDITOR	`["read:*"]`	/auditor/* (read-only)
Customer	CUSTOMER	`["own:*"]`	/accounts/, */customer/*, /transactions/transfer

Access Matrix

Endpoint Prefix	Admin	Banker	Auditor	Customer
/auth/*	✅	✅	✅	✅
/banker/*	✅	✅	❌	❌
/auditor/*	✅	❌	✅	❌
/admin/*	✅	❌	❌	❌
/accounts/*	✅	✅	❌	✅ (own)
/customer/*	❌	❌	❌	✅

5. API Endpoints by Module

5.1 Auth Module

POST /api/v1/auth/login

Purpose: Authenticate user or customer

Property	Value
Auth Required	No
Request Body	`{ email, password, type: 'user'
Response	`{ user, token, refreshToken }`
DB Tables	`users` or `customers` , `roles`
Service	`authenticateUser()` / `authenticateCustomer()`

SQL Operations:

```
SELECT u.*, r.code, r.permissions
FROM users u JOIN roles r ON u.role_id = r.id
WHERE email = ? AND status = 'ACTIVE'
```

POST /api/v1/auth/logout

Purpose: Invalidate session

Property	Value
Auth Required	Yes (any)
Response	`{ message: 'Logged out' }`
DB Tables	`audit_logs`

POST /api/v1/auth/refresh

Purpose: Refresh access token

Property	Value
Auth Required	No (uses refresh token)
Request Body	`{ refreshToken }`
Response	`{ token, refreshToken }`

5.2 Banker Module

GET /api/v1/banker/customers

Purpose: List all customers

Property	Value
Auth Required	Yes (BANKER)
Query Params	`?limit=25&offset=0&status=ACTIVE&search=...`
Response	`{ customers: [...], total }`
DB Tables	`customers`

POST /api/v1/banker/customers/create

Purpose: Onboard new customer with account

Property	Value
Auth Required	Yes (BANKER)
Request Body	`{ email, firstName, lastName, phone, nationalId, accountType, initialDeposit }`

Property	Value
Response	`{ customer, account, credentials }`
DB Tables	`customers` , `accounts` , `account_balances` , `transactions` , `ledger_entries`
Transaction	Yes (atomic)

SQL Operations:

```
INSERT INTO customers (...);
INSERT INTO accounts (...);
INSERT INTO account_balances (...);
CALL sp_deposit(account_id, initial_deposit, ...);
```

GET /api/v1/banker/accounts

Purpose: List accounts (with optional filters)

Property	Value
Auth Required	Yes (BANKER)
Query Params	`?customerId=&status=&accountType=`
Response	`{ accounts: [...], total }`
DB Tables	`accounts` , `account_balances` , `customers` , `account_types`

POST /api/v1/banker/deposits

Purpose: Process cash deposit

Property	Value
Auth Required	Yes (BANKER)
Request Body	`{ accountId, amount, description }`
Response	`{ transactionId, receipt: {...} }`

Property	Value
Stored Procedure	`sp_deposit`
DB Tables	`transactions` , `ledger_entries` , `account_balances`

Stored Procedure Call:

```
CALL sp_deposit(
  p_account_id,
  p_amount,
  p_description,
  p_banker_id,
  @p_transaction_id,
  @p_status,
  @p_message
);
```

POST /api/v1/banker/withdrawals

Purpose: Process cash withdrawal

Property	Value
Auth Required	Yes (BANKER)
Request Body	`{ accountId, amount, description }`
Response	`{ transactionId, receipt: {...} }`
Stored Procedure	`sp_withdraw`
Validation	Checks sufficient balance

POST /api/v1/banker/accounts/[id]/freeze

Purpose: Freeze account

Property	Value
Auth Required	Yes (BANKER)
Response	`{ account }`
DB Tables	`accounts`
Audit	Creates ACCOUNT_FROZEN log

SQL:

```
UPDATE accounts SET status = 'FROZEN' WHERE id = ?
```

POST /api/v1/banker/accounts/[id]/unfreeze

Purpose: Unfreeze account

Property	Value
Auth Required	Yes (BANKER)
DB Tables	`accounts`
Audit	Creates ACCOUNT_UNFROZEN log

POST /api/v1/banker/accounts/[id]/close

Purpose: Close account

Property	Value
Auth Required	Yes (BANKER)
Validation	Balance must be zero
DB Tables	`accounts`
Audit	Creates ACCOUNT_CLOSED log

5.3 Customer Module

GET /api/v1/customer/profile

Purpose: Get current customer profile

Property	Value
Auth Required	Yes (customer)
Response	`{ customer }`
DB Tables	`customers`

PUT /api/v1/customer/profile

Purpose: Update customer profile

Property	Value
Auth Required	Yes (customer)
Request Body	`{ phone, addressLine1, ... }`
DB Tables	`customers`

POST /api/v1/customer/profile/password

Purpose: Change password

Property	Value
Auth Required	Yes (customer)
Request Body	`{ currentPassword, newPassword }`
DB Tables	`customers` (password_hash)
Audit	Creates PASSWORD_CHANGED log

5.4 Accounts Module

GET /api/v1/accounts

Purpose: Get customer's own accounts

Property	Value
Auth Required	Yes (customer)
Response	`{ accounts: [...] }`
DB Tables	`accounts` , `account_balances` , `account_types`

SQL:

```
SELECT a.*, ab.available_balance, at.name as account_type_name
FROM accounts a
JOIN account_balances ab ON a.id = ab.account_id
JOIN account_types at ON a.account_type_id = at.id
WHERE a.customer_id = ? AND a.status = 'ACTIVE'
```

GET /api/v1/accounts/[id]

Purpose: Get single account details

Property	Value
Auth Required	Yes (customer, own account only)
Response	`{ account }`

GET /api/v1/accounts/[id]/statement

Purpose: Get account statement (transactions)

Property	Value
Auth Required	Yes (customer, own account)
Query Params	`?from=YYYY-MM-DD&to=YYYY-MM-DD`
Response	`{ transactions: [...], summary }`
DB Tables	`transactions` , `ledger_entries`

GET /api/v1/accounts/[id]/statement/pdf

Purpose: Download statement as PDF

Property	Value
Auth Required	Yes (customer, own account)
Response	Binary PDF file
Headers	`Content-Type: application/pdf`

5.5 Transactions Module

POST /api/v1/transactions/transfer

Purpose: Transfer funds between accounts

Property	Value
Auth Required	Yes (customer)
Request Body	`{ fromAccountId, toAccountNumber, amount, description }`
Headers	`Idempotency-Key: <uuid>` (optional)
Stored Procedure	`sp_transfer`
Response	`{ transactionId, status, message }`

Flow:

```
1. Validate request body (Zod schema)
2. Check idempotency key
3. Verify fromAccount belongs to customer
4. Lookup destination account by number
5. Call sp_transfer(from, to, amount, desc, key, user_id)
6. Return transaction result
```

GET /api/v1/transactions/[id]

Purpose: Get transaction details

Property	Value
Auth Required	Yes (customer, must be involved)
DB Tables	`transactions` , `ledger_entries`

5.6 Auditor Module (Read-Only)

GET /api/v1/auditor/summary

Purpose: Dashboard summary stats

Property	Value
Auth Required	Yes (AUDITOR)
Response	`{ totalCustomers, totalAccounts, totalTransactions, ... }`

GET /api/v1/auditor/transactions

Purpose: View all system transactions

Property	Value
Auth Required	Yes (AUDITOR)

Property	Value
Query Params	`?limit=25&offset=0&from=&to=&type=&status=`
Response	`{ transactions: [...], total }`
DB Tables	`transactions` , `ledger_entries` , `accounts`

GET /api/v1/auditor/ledger

Purpose: View all ledger entries

Property	Value
Auth Required	Yes (AUDITOR)
Query Params	`?limit=25&offset=0&entryType=DEBIT
Response	`{ entries: [...], total }`
DB Tables	`ledger_entries` , `accounts`

GET /api/v1/auditor/audit-logs

Purpose: View system audit logs

Property	Value
Auth Required	Yes (AUDITOR)
Query Params	`?actionType=&entityType=&from=&to=`
Response	`{ entries: [...], total }`
DB Tables	`audit_logs`

GET /api/v1/auditor/export-pdf/transactions

Purpose: Export transactions as PDF

Property	Value
Auth Required	Yes (AUDITOR/ADMIN)
Query Params	`?from=YYYY-MM-DD&to=YYYY-MM-DD`
Response	Binary PDF with watermark
Audit	Creates PDF_EXPORTED log

GET /api/v1/auditor/export-pdf/ledger

GET /api/v1/auditor/export-pdf/audit-logs

GET /api/v1/auditor/export-pdf/daily-totals

GET /api/v1/auditor/export-pdf/monthly-summary

GET /api/v1/auditor/export-pdf/top-accounts

(Same pattern as transactions PDF export)

5.7 Admin Module

GET /api/v1/admin/users

Purpose: List system users

Property	Value
Auth Required	Yes (ADMIN)
Response	`{ users: [...] }`
DB Tables	`users` , `roles`

POST /api/v1/admin/users

Purpose: Create system user

Property	Value
Auth Required	Yes (ADMIN)
Request Body	`{ email, password, firstName, lastName, roleId }`
DB Tables	`users`

GET /api/v1/admin/roles

Purpose: List roles

Property	Value
Auth Required	Yes (ADMIN)
Response	`{ roles: [...] }`
DB Tables	`roles`

GET /api/v1/admin/config

Purpose: Get system configuration

Property	Value
Auth Required	Yes (ADMIN)
Response	`{ configs: [...] }`
DB Tables	`system_config`

PUT /api/v1/admin/config

Purpose: Update system configuration

Property	Value
Auth Required	Yes (ADMIN)
Request Body	`{ key, value }`
DB Tables	`system_config`

POST /api/v1/admin/eod/run

Purpose: Trigger end-of-day processing

Property	Value
Auth Required	Yes (ADMIN)
Response	`{ jobId, status }`
DB Tables	`system_jobs` , `daily_account_totals`

POST /api/v1/admin/interest/post

Purpose: Post interest for eligible accounts

Property	Value
Auth Required	Yes (ADMIN)
DB Tables	`accounts` , `ledger_entries` , `transactions`

POST /api/v1/admin/reports/rebuild

Purpose: Rebuild aggregated reports

Property	Value
Auth Required	Yes (ADMIN)

Property	Value
Stored Procedure	`sp_rebuild_balance`

5.8 Reports Module

GET /api/v1/reports/daily

Purpose: Get daily totals

Property	Value
Auth Required	Yes (BANKER/ADMIN)
Query Params	`?date=YYYY-MM-DD`
Response	`{ totals: [...] }`

GET /api/v1/reports/monthly

Purpose: Get monthly summary

Property	Value
Auth Required	Yes (BANKER/ADMIN)
Query Params	`?year=&month=`
Response	`{ summary: {...} }`

6. Request Lifecycle

```
┌──────────────────────────────────────────────────────────────┐
│                     1. HTTP Request                          │
└──────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌──────────────────────────────────────────────────────────────┐
│                 2. withErrorHandler()                        │
│         Catches all errors, returns 500 on failure          │
└──────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌──────────────────────────────────────────────────────────────┐
│                    3. withAuth()                             │
│  • Extract Bearer token from Authorization header            │
│  • Verify JWT signature and expiry                           │
│  • Load user/customer from database                          │
│  • Check role/permission requirements                        │
│  • Attach user to request object                             │
└──────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌──────────────────────────────────────────────────────────────┐
│                  4. validateBody()                           │
│         Parse JSON, validate against Zod schema              │
└──────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌──────────────────────────────────────────────────────────────┐
│                 5. Business Logic                            │
│     Service functions / Stored procedure calls               │
└──────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌──────────────────────────────────────────────────────────────┐
│               6. successResponse()                           │
│            Format and return JSON response                   │
└──────────────────────────────────────────────────────────────┘
```

7. Error Handling Strategy

HTTP Status Codes

Code	Meaning	Usage
200	Success	Successful GET/PUT/POST
201	Created	Resource created (POST)
400	Bad Request	Validation error, business rule violation
401	Unauthorized	Missing or invalid token
403	Forbidden	Insufficient permissions
404	Not Found	Resource doesn't exist
500	Server Error	Unhandled exception

Error Response Format

```json
{
  "success": false,
  "error": "Human-readable error message"
}
```

Validation Error Response

```json
{
  "success": false,
  "error": "Validation failed",
  "errors": [
    { "field": "amount", "message": "Amount must be positive" }
  ]
}
```

8. Transaction Management

Stored Procedure Atomicity

All financial operations use stored procedures with explicit transaction control:

```
START TRANSACTION;
-- Lock rows with SELECT ... FOR UPDATE
-- Perform validations
-- Insert/update records
-- On error: ROLLBACK
-- On success: COMMIT
```

Database Tables Affected Per Operation

Operation	Tables Modified
Transfer	transactions, ledger_entries (×2), account_balances (×2), idempotency_keys, events, outbox
Deposit	transactions, ledger_entries (×2), account_balances (×2), events, outbox
Withdrawal	transactions, ledger_entries (×2), account_balances (×2), events, outbox

Idempotency Pattern

```
1. Check idempotency_keys table for existing key
2. If found and not expired → return cached response
3. If not found → proceed with operation
4. On success → store result in idempotency_keys
```

9. Security Considerations

Authentication Security

Measure	Implementation
Password Hashing	bcrypt with salt rounds = 10
Token Expiry	Access: 1 hour, Refresh: 7 days
SQL Injection	Parameterized queries via mysql2

Authorization Security

Measure	Implementation
Role Enforcement	`withAuth({ requiredRoles: [...] })`
Resource Ownership	Customer can only access own accounts
Hidden Endpoints	`hideFailure: true` returns 404 instead of 403

Financial Integrity

Measure	Implementation
Atomic Transactions	Stored procedures with TRANSACTION blocks
Row Locking	`SELECT ... FOR UPDATE` prevents race conditions
Append-Only Ledger	No UPDATE/DELETE on ledger_entries
Idempotency	Prevents duplicate financial operations

10. Summary Tables

Complete API Route List (63 Routes)

Module	Count	Prefix
Auth	3	/auth/*
Accounts	5	/accounts/*
Banker	12	/banker/*
Customer	3	/customer/*
Auditor	12	/auditor/*
Admin	8	/admin/*
Core	8	/core/*
Reports	4	/reports/*
Transactions	5	/transactions/*
Health	1	/health
Customers	2	/customers/*

Stored Procedures Used

Procedure	API Endpoints
`sp_transfer`	POST /transactions/transfer
`sp_deposit`	POST /banker/deposits, POST /core/deposit
`sp_withdraw`	POST /banker/withdrawals, POST /core/withdraw
`sp_rebuild_balance`	POST /admin/reports/rebuild

Appendix: API Testing Quick Reference

Headers Required

```
Authorization: Bearer <jwt_token>
Content-Type: application/json
Idempotency-Key: <uuid>  (for financial operations)
```

Sample cURL - Login

```
curl -X POST http://localhost:3000/api/v1/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"banker1@bnkcore.com","password":"password123","type":"user"}'
```

Sample cURL - Deposit

```
curl -X POST http://localhost:3000/api/v1/banker/deposits \
  -H "Authorization: Bearer <token>" \
  -H "Content-Type: application/json" \
  -d '{"accountId":1,"amount":1000,"description":"Cash deposit"}'
```

This documentation is suitable for academic review, software engineering courses, and viva preparation.