**PROMPT:** You are an expert software architect and engineer. Based on the provided PRD Produce a single, concise "Technical Details Document" (TDD) for the BankingApp full-stack project (Backend(Python FastAPI) + Frontend(React)). The TDD should be high-level and rely on the user's separate requirements document for low-level technical specifics. Provide a clear, code-generation-ready blueprint that maps features to components, APIs, interfaces, and file-level responsibilities so a developer or a code-generation tool — given the requirements document — can implement the system.

The TDD must contain these sections:

1. Executive summary

Purpose, target users, core features

1. System components and data flows

* List components (Frontend app, Backend API, Database, Auth provider).
* For each component, state responsibilities, protocols (HTTP), and deployment unit (container/process).
* Show data-flow steps for main user

1. Functional mapping (features → components)
2. API contract summary

* Provide clear API endpoints (path, method, purpose), example request & response JSON shapes (short examples), status codes, and auth requirements.

1. Security and auth approach.
2. Data model guidelines

* For each domain model, list attributes with types and which are primary keys/foreign keys.
* Indicate constraints (uniqueness, not-null) at a high level.

1. Testing strategy

* List the tests to be written (unit and integration), key scenarios to cover, and test data/fixtures needed.

1. Deployment, containerization, and local run

* Provide recommended Docker/Docker Compose structure and key environment variables (names and purpose). No need to provide full files—just the blueprint.

1. Non-functional requirements

**Technical Details Document (TDD)**

**Executive Summary**

The Banking System Internal Admin Portal is a secure, admin-only full-stack application designed to enable internal operations teams to manage customer accounts, process deposits and withdrawals, and maintain comprehensive transaction histories. The system leverages a FastAPI backend for robust API services, a React frontend for a clean user interface, and implements dual authentication (SSO + fallback) for maximum security and availability. This document provides a code-generation-ready blueprint mapping all functional requirements to technical components, APIs, data models, and deployment strategies.

**Target Users:** Internal bank administrators and operations staff

**Core Features:**

* Dual authentication (SSO primary, credential-based fallback)
* Account dashboard with search and filtering capabilities
* Deposit and withdrawal operations with balance validation
* Comprehensive transaction history tracking and filtering
* Secure HTTPS-only communication with encrypted credential storage

**System Components and Data Flows**

**Component Architecture**

**1. Frontend Application (React + Vite)**

* **Responsibilities:** User interface rendering, form validation, state management, API consumption, session management
* **Protocol:** HTTPS REST API calls to Backend
* **Deployment Unit:** Static files served via Nginx container
* **Key Dependencies:** React Router, Axios/Fetch, Context API, Tailwind CSS
* **Port:** 3000 (development), 80/443 (production)

**2. Backend API (FastAPI)**

* **Responsibilities:** Business logic, authentication/authorization, data validation, transaction processing, balance calculations, API endpoint exposure
* **Protocol:** HTTPS REST APIs, SSO integration via OAuth2/OIDC
* **Deployment Unit:** Uvicorn ASGI server in container
* **Key Dependencies:** SQLAlchemy, Pydantic, JWT libraries, bcrypt
* **Port:** 8000

**3. Database (PostgreSQL)**

* **Responsibilities:** Persistent storage for accounts, transactions, admin users, session tokens
* **Protocol:** PostgreSQL wire protocol (TCP)
* **Deployment Unit:** PostgreSQL container with persistent volume
* **Port:** 5432

**4. Authentication Provider (SSO + Internal)**

* **Responsibilities:** Primary SSO authentication (external), fallback credential verification (internal)
* **Protocol:** OAuth2/OIDC for SSO, JWT for session tokens
* **Deployment Unit:** External SSO service + internal auth module in Backend

**Data Flow - Main User Journey**

**Admin Login Flow:**

1. User accesses Frontend (React app) → Login page rendered
2. User selects SSO → Frontend redirects to SSO Provider
3. SSO Provider authenticates → Returns authorization code
4. Frontend sends code to Backend /auth/sso/callback
5. Backend validates code, creates session, returns JWT token
6. Frontend stores JWT, redirects to Dashboard

**Account Management Flow:**

1. Frontend sends authenticated request to Backend /accounts with JWT
2. Backend validates token, queries Database for accounts
3. Database returns account records
4. Backend filters/formats data, returns JSON response
5. Frontend renders account list with search/filter capabilities

**Deposit/Withdrawal Flow:**

1. Admin selects account → Frontend fetches account details from Backend /accounts/{id}
2. Admin initiates deposit/withdrawal → Frontend sends POST to /transactions/deposit or /transactions/withdraw
3. Backend validates request (amount, balance check for withdrawal)
4. Backend starts database transaction:
   1. Insert transaction record
   2. Update account balance
   3. Commit or rollback on validation failure
5. Backend returns success/error response
6. Frontend displays confirmation message and refreshed balance

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Frontend Component** | **Backend API** | **Database Tables** |
| SSO Authentication | LoginPage.jsx, SSOCallback.jsx | /auth/sso/login, /auth/sso/callback | admins, sessions |
| Fallback Login | LoginPage.jsx, FallbackLogin.jsx | /auth/login | admins, sessions |
| Account Dashboard | Dashboard.jsx, AccountList.jsx | /accounts, /accounts/search | accounts |
| Account Search/Filter | SearchBar.jsx, FilterPanel.jsx | /accounts?name={name}&status={status} | accounts |
| Account Details | AccountDetails.jsx | /accounts/{id} | accounts, transactions |
| Transaction History | TransactionHistory.jsx, TransactionFilter.jsx | /accounts/{id}/transactions | transactions |
| Deposit Operation | DepositForm.jsx | /transactions/deposit | transactions, accounts |
| Withdrawal Operation | WithdrawForm.jsx | /transactions/withdraw | transactions, accounts |
| Session Management | AuthContext.jsx, ProtectedRoute.jsx | /auth/verify, /auth/logout | sessions |

**Functional Mapping**

**API Contract Summary**

**Authentication APIs**

**POST /auth/sso/login**

* **Purpose:** Initiate SSO authentication flow
* **Request:** {}
* **Response:** {"redirect\_url": "<https://sso.provider.com/authorize?client_id=>..."}
* **Status Codes:** 200 (Success), 500 (SSO unavailable)
* **Auth Required:** No

**POST /auth/sso/callback**

* **Purpose:** Handle SSO callback and issue JWT
* **Request:** {"code": "auth\_code\_from\_sso", "state": "csrf\_token"}
* **Response:** {"access\_token": "jwt\_token", "token\_type": "Bearer", "expires\_in": 3600}
* **Status Codes:** 200 (Success), 401 (Invalid code), 403 (Unauthorized domain)
* **Auth Required:** No

**POST /auth/login**

* **Purpose:** Fallback credential-based login
* **Request:** {"username": "[admin@example.com](mailto:admin@example.com)", "password": "hashed\_password"}
* **Response:** {"access\_token": "jwt\_token", "token\_type": "Bearer", "expires\_in": 3600}
* **Status Codes:** 200 (Success), 401 (Invalid credentials)
* **Auth Required:** No

**POST /auth/logout**

* **Purpose:** Invalidate session token
* **Request:** {}
* **Response:** {"message": "Logout successful"}
* **Status Codes:** 200 (Success)
* **Auth Required:** Yes (JWT Bearer token)

**GET /auth/verify**

* **Purpose:** Verify token validity
* **Request:** {}
* **Response:** {"valid": true, "admin\_id": "uuid", "username": "[admin@example.com](mailto:admin@example.com)"}
* **Status Codes:** 200 (Valid), 401 (Invalid/expired)
* **Auth Required:** Yes (JWT Bearer token)

**Account Management APIs**

**GET /accounts**

* **Purpose:** Retrieve all accounts with optional filtering
* **Query Params:** ?name={string}&account\_number={string}&status={active|frozen}&page={int}&limit={int}
* **Response:** {"accounts": [{"id": "uuid", "account\_number": "ACC001", "holder\_name": "John Doe", "balance": 1000.00, "status": "active", "created\_at": "2025-10-01T10:00:00Z"}], "total": 50, "page": 1, "limit": 20}
* **Status Codes:** 200 (Success), 401 (Unauthorized)
* **Auth Required:** Yes

**GET /accounts/{id}**

* **Purpose:** Retrieve detailed account information
* **Request:** {}
* **Response:** {"id": "uuid", "account\_number": "ACC001", "holder\_name": "John Doe", "balance": 1000.00, "status": "active", "created\_at": "2025-10-01T10:00:00Z", "updated\_at": "2025-10-20T14:30:00Z"}
* **Status Codes:** 200 (Success), 404 (Not found), 401 (Unauthorized)
* **Auth Required:** Yes

**Transaction APIs**

**GET /accounts/{id}/transactions**

* **Purpose:** Retrieve transaction history for an account
* **Query Params:** ?type={deposit|withdrawal}&start\_date={ISO8601}&end\_date={ISO8601}&page={int}&limit={int}
* **Response:** {"transactions": [{"id": "uuid", "account\_id": "uuid", "type": "deposit", "amount": 500.00, "reference": "Cash deposit", "balance\_after": 1500.00, "created\_at": "2025-10-20T10:00:00Z"}], "total": 100, "page": 1, "limit": 20}
* **Status Codes:** 200 (Success), 404 (Account not found), 401 (Unauthorized)
* **Auth Required:** Yes

**POST /transactions/deposit**

* **Purpose:** Process deposit transaction
* **Request:** {"account\_id": "uuid", "amount": 500.00, "reference": "Cash deposit", "admin\_id": "uuid"}
* **Response:** {"transaction\_id": "uuid", "account\_id": "uuid", "type": "deposit", "amount": 500.00, "balance\_after": 1500.00, "message": "Deposit successful", "created\_at": "2025-10-20T10:00:00Z"}
* **Status Codes:** 201 (Created), 400 (Invalid amount), 403 (Account frozen), 404 (Account not found), 401 (Unauthorized)
* **Auth Required:** Yes

**POST /transactions/withdraw**

* **Purpose:** Process withdrawal transaction
* **Request:** {"account\_id": "uuid", "amount": 300.00, "reference": "Cash withdrawal", "admin\_id": "uuid"}
* **Response:** {"transaction\_id": "uuid", "account\_id": "uuid", "type": "withdrawal", "amount": 300.00, "balance\_after": 1200.00, "message": "Withdrawal successful", "created\_at": "2025-10-20T10:15:00Z"}
* **Status Codes:** 201 (Created), 400 (Invalid amount/Insufficient funds), 403 (Account frozen), 404 (Account not found), 401 (Unauthorized)
* **Auth Required:** Yes

**Error Response Format**

All error responses follow this structure:

json

{

"error": "Error type",

"message": "User-friendly error message",

"timestamp": "2025-10-20T10:00:00Z"

}

**Security and Auth Approach**

**Authentication Strategy:**

* **Primary:** OAuth2/OIDC SSO integration with organizational identity provider
* **Fallback:** Credential-based authentication using bcrypt-hashed passwords stored in database
* **Session Management:** JWT tokens with 1-hour expiration, stored in HTTP-only cookies or Authorization header
* **Token Refresh:** Implement refresh token mechanism for extended sessions

**Authorization:**

* Role-based access control (RBAC) with admin role verification on every API call
* Domain restriction enforcement for SSO logins (whitelist allowed domains)
* All endpoints except /auth/login and /auth/sso/\* require valid JWT

**Security Measures:**

* HTTPS-only communication (HTTP redirects to HTTPS)
* CORS configuration to allow only frontend domain
* Rate limiting on authentication endpoints (5 attempts per 15 minutes)
* Secure credential storage using environment variables, never in code
* Password hashing with bcrypt (cost factor 12)
* SQL injection prevention via parameterized queries (SQLAlchemy ORM)
* Input validation using Pydantic models
* Audit logging for all deposit/withdrawal operations
* Clear error messages without system detail exposure

**Data Model Guidelines**

**Domain Models**

**Country Schema**

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| CountryCode | SERIAL | Auto-incrementing country identifier. |
| CountryName | TEXT | Country name. Not null. |

* Primary key: CountryCode
* Foreign keys: None
* Referenced by: State.CountryCode (ON DELETE CASCADE)

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| StateCode | SERIAL | Auto-incrementing state identifier. |
| StateName | TEXT | State name. Not null. |
| CountryCode | INT | References Country(CountryCode). Deletes cascade. |

**State schema**

* Primary key: StateCode
* Foreign keys: CountryCode → Country.CountryCode (ON DELETE CASCADE)
* Referenced by: City.StateCode (ON DELETE CASCADE)

**City schema**

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| CityCode | SERIAL | Auto-incrementing city identifier. |
| CityName | TEXT | City name. Not null. |
| StateCode | INT | References State(StateCode). Deletes cascade. |

* Primary key: CityCode
* Foreign keys: StateCode → State.StateCode (ON DELETE CASCADE)
* Referenced by: PostalCode.CityCode (ON DELETE CASCADE)

**PostalCode schema**

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| ZIPCode | VARCHAR(10) | Postal/ZIP code. Primary key. |
| CityCode | INT | References City(CityCode). Deletes cascade. |

* Primary key: ZIPCode
* Foreign keys: CityCode → City.CityCode (ON DELETE CASCADE)
* Referenced by: CustomerDetail.ZIPCode

**CustomerDetail schema**

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| CustID | SERIAL | Auto-incrementing customer identifier. |
| FirstName | TEXT | First name. Not null. |
| LastName | TEXT | Last name. Not null. |
| Address1 | TEXT | Address line 1. |
| Address2 | TEXT | Address line 2. |
| EmailID | TEXT | Unique email. Not null. |
| Phone | VARCHAR(15) | Landline or secondary phone. |
| Mobile | VARCHAR(15) | Mobile number. |
| DOB | DATE | Date of birth. |
| MaritalStatus | TEXT | Marital status. |
| ZIPCode | VARCHAR(10) | References PostalCode(ZIPCode). |

* Primary key: CustID
* Unique: EmailID
* Foreign keys: ZIPCode → PostalCode.ZIPCode
* Referenced by: CustomerAccounts.CustID (ON DELETE CASCADE)

**AccountType schema**

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| AccountTypeID | SERIAL | Auto-incrementing account type identifier. |
| AccountType | TEXT | Major type, e.g., Savings, Loan. Not null. |
| AccSubType | TEXT | Subtype, e.g., Current, Student, Home Loan. |

* Primary key: AccountTypeID
* Foreign keys: None
* Referenced by: CustomerAccounts.AccountTypeID (ON DELETE CASCADE), SavingAccountDetail.SavingAccTypeId, LoanAccountDetail.LoanAccountTypeId

**CustomerAccounts schema**

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| CustID | INT | References CustomerDetail(CustID). Deletes cascade. |
| AcctNum | BIGINT | Account number. Primary key. |
| AccountTypeID | INT | References AccountType(AccountTypeID). Deletes cascade. |

* Primary key: AcctNum.
* Foreign keys: CustID → CustomerDetail.CustID (ON DELETE CASCADE) AccountTypeID → AccountType.AccountTypeID (ON DELETE CASCADE)
* Referenced by: SavingAccountDetail.AcctNum (ON DELETE CASCADE) LoanAccountDetail.AcctNum (ON DELETE CASCADE)
* Notes: Allows multiple accounts per customer

**SavingAccountDetail schema**

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| AcctNum | BIGINT | References CustomerAccounts(AcctNum). Primary key. |
| SavingAccTypeId | INT | References AccountType(AccountTypeID). |
| Balance | NUMERIC(18,2) | Current balance. Default 0.00. |
| TransferLimit | NUMERIC(18,2) | Per-transaction or daily transfer limit. |
| BranchCode | VARCHAR(20) | Branch identifier/code. |

* Primary key: AcctNum
* Foreign keys: AcctNum → CustomerAccounts.AcctNum (ON DELETE CASCADE) SavingAccTypeId → AccountType.AccountTypeID
* Referenced by: SavingAccountTxnHistory.AcctNum (ON DELETE CASCADE)
* Notes: One saving account detail per account number

**SavingAccountTxnHistory schema**

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| TxnID | SERIAL | Auto-incrementing transaction identifier. |
| TxnDate | TIMESTAMPTZ | Transaction timestamp. Default NOW(). |
| AcctNum | BIGINT | References SavingAccountDetail(AcctNum). Deletes cascade. |
| TxnDetail | TEXT | Narrative or description. |
| WithdrawAmount | NUMERIC(18,2) | Withdrawal amount. Default 0.00. |
| DepositAmount | NUMERIC(18,2) | Deposit amount. Default 0.00. |
| Balance | NUMERIC(18,2) | Account balance after this transaction. |

* Primary key: TxnID
* Foreign keys: AcctNum → SavingAccountDetail.AcctNum (ON DELETE CASCADE)
* Relationships: Many transactions per saving account

**LoanAccountDetail schema**

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| AcctNum | BIGINT | References CustomerAccounts(AcctNum). Primary key. |
| EMIID | INT | EMI grouping identifier (no FK defined). |
| BalanceAmount | NUMERIC(18,2) | Current outstanding balance. |
| BranchCode | VARCHAR(20) | Branch identifier/code. |
| RateOfInterest | NUMERIC(5,2) | Annual interest rate (percent). |
| LoanDuration | INT | Loan term in months. |
| TotalLoanAmount | NUMERIC(18,2) | Original sanctioned/principal amount. |
| LoanAccountTypeId | INT | References AccountType(AccountTypeID). |

* Primary key: AcctNum
* Foreign keys: AcctNum → CustomerAccounts.AcctNum (ON DELETE CASCADE) LoanAccountTypeId → AccountType.AccountTypeID
* Referenced by: LoanEMIDetail.AcctNum (ON DELETE CASCADE)
* Notes: EMIID is not constrained as a foreign key in this schema

**LoanEMIDetail schema**

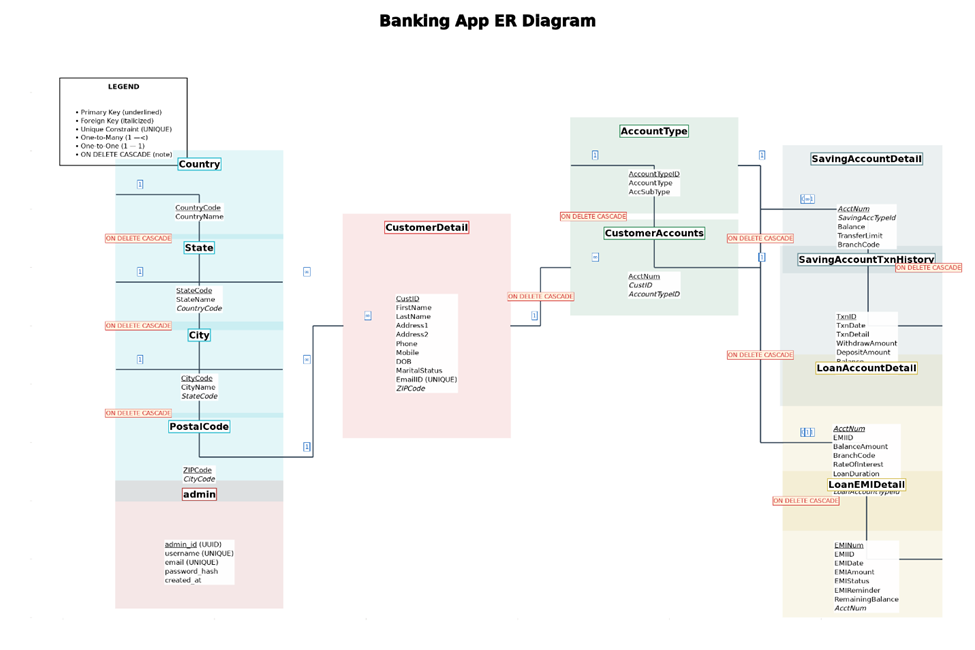
|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| EMINum | SERIAL | Auto-incrementing EMI row identifier. |
| EMIID | INT | EMI grouping identifier (no FK defined). |
| AcctNum | BIGINT | References LoanAccountDetail(AcctNum). Deletes cascade. |
| EMIDate | DATE | Due date for the EMI. |
| EMIAmount | NUMERIC(18,2) | Scheduled EMI amount. |
| EMIStatus | TEXT | Status, e.g., Paid or Pending. |
| EMIReminder | BOOLEAN | Reminder flag. Default FALSE. |
| RemainingBalance | NUMERIC(18,2) | Balance remaining after applying this EMI. |

* Primary key: EMINum
* Foreign keys: AcctNum → LoanAccountDetail.AcctNum (ON DELETE CASCADE)
* Relationships: Many EMIs per loan account; EMIID is a free-form grouping value unless constrained elsewhere

**admin schema**

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| admin\_id | UUID | Administrator identifier. Primary key. |
| username | VARCHAR(100) | Unique username. Not null. |
| email | VARCHAR(255) | Unique email. Not null. |
| password\_hash | VARCHAR(255) | Password hash. Not null. |
| created\_at | TIMESTAMPTZ | Creation timestamp. Default now(). |

* Primary key: admin\_id
* Unique: email, username
* Foreign keys: None
* Relationships: Standalone admin/auth table



**Testing Strategy**

**Unit Tests**

**Backend (pytest)**

* **Authentication Module:**
  + SSO token validation and JWT generation
  + Fallback login with correct/incorrect credentials
  + Password hashing verification
  + Token expiration handling
  + Domain restriction validation
* **Transaction Processing:**
  + Deposit amount validation (positive values)
  + Withdrawal balance validation (sufficient funds check)
  + Transaction record creation
  + Balance calculation accuracy
  + Concurrent transaction handling (race conditions)
* **Account Management:**
  + Account search by name/number/status
  + Pagination logic
  + Account status filtering

**Frontend (Jest + React Testing Library)**

* **Component Rendering:**
  + LoginPage with SSO and fallback options
  + Dashboard account list rendering
  + Account details display
  + Deposit/withdrawal forms
* **Form Validation:**
  + Amount input validation (positive, numeric)
  + Required field validation
  + Error message display
* **State Management:**
  + Authentication context state updates
  + API response handling
  + Loading and error states

**Integration Tests**

**API Integration (pytest + TestClient)**

* **Complete User Flows:**
  + Full authentication flow (SSO callback → JWT → protected endpoint access)
  + Account search and filter combinations
  + Deposit workflow (request → validation → database update → response)
  + Withdrawal with insufficient funds scenario
  + Transaction history filtering by date and type
* **Database Transactions:**
  + Rollback on validation failure
  + Balance consistency after concurrent operations
  + Foreign key constraint enforcement
* **Security Tests:**
  + Unauthorized access attempts to protected endpoints
  + HTTPS enforcement verification
  + Rate limiting behavior

**End-to-End Tests (Playwright/Cypress)**

* Admin login via SSO and fallback
* Search and filter accounts from dashboard
* View account details and transaction history
* Perform deposit and verify balance update
* Attempt withdrawal with insufficient funds and verify error message
* Logout and verify session termination

**Test Data & Fixtures**

* **Seed Data:** 20 sample accounts (15 active, 5 frozen) with varying balances
* **Transaction History:** 100 historical transactions (deposits and withdrawals) distributed across accounts
* **Admin Users:** 3 test admin accounts (1 SSO-only, 1 fallback-only, 1 both)
* **Mock SSO Provider:** Stub OAuth2 server for SSO integration testing
* **Database Fixtures:** SQLAlchemy fixtures for rollback after each test
* **Edge Cases:** Accounts with zero balance, maximum balance values, concurrent transaction scenarios

**Deployment, Containerization, and Local Run**

**Docker Architecture**

**Multi-Container Setup (Docker Compose)**

**1. Frontend Container (Nginx + React build)**

* **Base Image:** node:18-alpine (build stage), nginx:alpine (runtime)
* **Build Process:** npm install → npm run build → copy to Nginx
* **Port Mapping:** 3000:80 (development), 80:80 (production)
* **Environment Variables:**
  + VITE\_API\_BASE\_URL - Backend API base URL (e.g., <http://localhost:8000>)
  + VITE\_SSO\_ENABLED - Enable/disable SSO (true/false)

**2. Backend Container (FastAPI + Uvicorn)**

* **Base Image:** python:3.11-slim
* **Command:** uvicorn main:app --host 0.0.0.0 --port 8000 --reload
* **Port Mapping:** 8000:8000
* **Environment Variables:**
  + DATABASE\_URL - PostgreSQL connection string (e.g., postgresql://user:pass@db:5432/banking\_db)
  + JWT\_SECRET\_KEY - Secret key for JWT signing (min 32 chars)
  + JWT\_ALGORITHM - JWT algorithm (HS256)
  + JWT\_EXPIRATION\_MINUTES - Token expiration time (60)
  + SSO\_CLIENT\_ID - OAuth2 client ID from SSO provider
  + SSO\_CLIENT\_SECRET - OAuth2 client secret
  + SSO\_REDIRECT\_URI - OAuth2 callback URL (e.g., <http://localhost:8000/auth/sso/callback>)
  + SSO\_AUTHORIZE\_URL - SSO authorization endpoint
  + SSO\_TOKEN\_URL - SSO token endpoint
  + ALLOWED\_DOMAINS - Comma-separated list of allowed SSO domains
  + CORS\_ORIGINS - Allowed frontend origins (e.g., <http://localhost:3000>)
  + BCRYPT\_ROUNDS - Password hashing rounds (12)
* **Volumes:** ./backend:/app (development hot-reload)

**3. Database Container (PostgreSQL)**

* **Base Image:** postgres:15-alpine
* **Port Mapping:** 5432:5432
* **Environment Variables:**
  + POSTGRES\_USER - Database user (banking\_admin)
  + POSTGRES\_PASSWORD - Database password (strong password)
  + POSTGRES\_DB - Database name (banking\_db)
* **Volumes:** postgres\_data:/var/lib/postgresql/data (persistent storage)

**4. Migration Service (Alembic)**

* **Purpose:** Database schema initialization and migrations
* **Command:** alembic upgrade head
* **Dependencies:** Waits for PostgreSQL to be ready

**Local Development Run**

**Prerequisites:**

* Docker 24+ and Docker Compose v2
* Node.js 18+ (for frontend development without Docker)
* Python 3.11+ (for backend development without Docker)

**Steps:**

1. Clone repository and navigate to project root
2. Copy .env.example to .env and configure environment variables
3. Run docker-compose up -d to start all services
4. Access frontend at <http://localhost:3000>
5. Access backend API docs at <http://localhost:8000/docs>
6. Run migrations: docker-compose exec backend alembic upgrade head
7. Seed demo data: docker-compose exec backend python scripts/seed\_data.py

**Development Workflow:**

* Frontend changes auto-reload via Vite HMR
* Backend changes auto-reload via Uvicorn --reload
* Database persists via named volume
* Logs accessible via docker-compose logs -f [service\_name]

**Production Deployment Blueprint**

**Container Orchestration:** Kubernetes or cloud-managed containers (AWS ECS, Azure Container Apps)   
 **Reverse Proxy:** Nginx/Traefik for HTTPS termination and load balancing  
 **Database:** Managed PostgreSQL (AWS RDS, Azure Database for PostgreSQL)  
 **Secrets Management:** Kubernetes Secrets, AWS Secrets Manager, or HashiCorp Vault  
 **CI/CD Pipeline:** GitHub Actions/GitLab CI → Build images → Push to registry → Deploy to cluster  
 **Monitoring:** Prometheus + Grafana for metrics, ELK/Loki for logs  
 **Backup Strategy:** Automated database backups with point-in-time recovery

**Non-Functional Requirements**

**Performance:**

* API response time < 200ms for account queries (95th percentile)
* Support 100 concurrent admin users
* Database connection pooling (min 5, max 20 connections)
* Frontend lazy loading for transaction history pagination

**Scalability:**

* Horizontal scaling for backend API (stateless design)
* Database read replicas for reporting queries
* CDN for static frontend assets in production

**Availability:**

* 99.5% uptime target (excluding maintenance windows)
* Database automatic failover in production
* Health check endpoints: /health (backend), / (frontend)

**Reliability:**

* Database transaction atomicity for all balance-modifying operations
* Automatic retry logic for transient failures (3 retries with exponential backoff)
* Audit logs retained for 7 years (compliance)

**Maintainability:**

* Code documentation with docstrings (backend) and JSDoc (frontend)
* API versioning strategy (/api/v1/ prefix)
* Database migration scripts via Alembic
* Environment-specific configuration (dev/staging/prod)

**Security:**

* Regular dependency vulnerability scanning (Dependabot, Snyk)
* Secrets rotation policy (quarterly for production)
* Database encryption at rest and in transit
* Regular security audits and penetration testing

**Compliance:**

* Audit trail for all financial transactions (immutable logs)
* GDPR/data privacy compliance for admin data
* Secure credential storage per industry standards

**Usability:**

* Clean, intuitive UI with standard organizational branding
* Clear error messaging without technical exposure
* Responsive design for desktop browsers (tablet/mobile optional in v1)
* Accessibility compliance (WCAG 2.1 Level AA target)