vBank: Simplified Virtual Banking System Project Description

This document outlines the architecture and functional specifications for vBank, a simplified virtual banking system designed with a microservices-based approach. Each core banking domain is implemented as an independent, loosely coupled microservice, orchestrated by a Backend-for-Frontend (BFF) layer to optimize communication between the frontend and backend.

System Architecture Overview

The vBank system is composed of the following independent microservices:

- **User Service:** Manages user registration, authentication, and profile information.
- **Account Service:** Handles user accounts, including creation, balance management, and status updates.
- **Transaction Service:** Processes financial transactions between accounts.
- **Logging Service:** Captures and stores system-wide request and response logs.
- **BFF (Backend-for-Frontend) Service:** Acts as an aggregation layer, orchestrating calls to the core microservices to reduce chattiness and simplify frontend interactions.

1. User Service

The User Service is responsible for managing user identities and profiles within the vBank system.

- **Database Name:** user_db
- **Table Name:** users
- **Schema:**

Column Name	Data Type	Constraints	Description
id	UUID	PRIMARY KEY, Generated	Unique identifier for the user
username	VARCHAR(50)	Unique, Not Null	User's unique username
email	VARCHAR(100)	Unique, Not Null	User's unique email address
password_hash	VARCHAR(255)	Not Null	Hashed password of the user

first_name	VARCHAR(50)	Not Null	User's first name
last_name	VARCHAR(50)	Not Null	User's last name
created_at	TIMESTAMP	Default CURRENT_TIMESTAMP	Timestamp of user creation
updated_at	TIMESTAMP	Default CURRENT_TIMESTAMP	Timestamp of last user update

- **Endpoints:**
  - **A. POST /users/register**
    - **Description:** Registers a new user in the system.
    - **Request Body (JSON):**

```
{
  "username": "john.doe",
  "password": "securePassword123", // To be stored in DB hashed
  "email": "john.doe@example.com",
  "firstName": "John",
  "lastName": "Doe"
}
```

    - **Response (201 Created):**

```
{
  "userId": "a1b2c3d4-e5f6-7890-1234-567890abcdef",
  "username": "john.doe",
  "message": "User registered successfully."
}
```

    - **Response (409 Conflict):**

```
{
  "status": 409,
  "error": "Conflict",
  "message": "Username or email already exists."
}
```

  - **B. POST /users/login**
    - **Description:** Authenticates a user and provides user details upon

successful login.

- **Request Body (JSON):**

```
{
  "username": "john.doe",
  "password": "securePassword123"
}
```

- **Response (200 OK):**

```
{
  "userId": "a1b2c3d4-e5f6-7890-1234-567890abcdef",
  "username": "john.doe"
}
```

- **Response (401 Unauthorized):**

```
{
  "status": 401,
  "error": "Unauthorized",
  "message": "Invalid username or password."
}
```

- **C. GET /users/{userId}/profile**
  - **Description:** Retrieves the profile information for a specific user.
  - **Response (200 OK):**

```
{
  "userId": "a1b2c3d4-e5f6-7890-1234-567890abcdef",
  "username": "john.doe",
  "email": "john.doe@example.com",
  "firstName": "John",
  "lastName": "Doe"
}
```

  - **Response (404 Not Found):**

```
{
  "status": 404,
  "error": "Not Found",
  "message": "User with ID a1b2c3d4-e5f6-7890-1234-567890abcdef not found."
}
```

2. Account Service

The Account Service manages banking accounts for users, including creation, balance, and status.

- **Database Name:** account_db
- **Table Name:** accounts
- **Schema:**

Column Name	Data Type	Constraints	Description
id	UUID	PRIMARY KEY, Generated	Unique identifier for the account
user_id	UUID	Not Null	Foreign key referencing the user
account_number	VARCHAR(20)	Unique, Not Null	Unique account number
account_type	ENUM('SAVINGS', 'CHECKING')	Not Null	Type of account (e.g., SAVINGS, CHECKING)
balance	DECIMAL(15,2)	Default 0.00	Current balance of the account
status	ENUM('ACTIVE', 'INACTIVE')	Default 'ACTIVE'	Current status of the account
created_at	TIMESTAMP	Default CURRENT_TIMESTAMP	Timestamp of account creation
updated_at	TIMESTAMP	Default CURRENT_TIMESTAMP	Timestamp of last account update

- **Endpoints:**
  - **A. PUT /accounts/transfer**
    - **Description:** Initiates a transfer of funds between two accounts.
    - **Request Body (JSON):**

```
{
  "fromAccountId": "f1e2d3c4-b5a6-9876-5432-10fedcba9876",
  "toAccountId": "a1b2c3d4-e5f6-7890-1234-567890abcdef",
  "amount": 100.00
}
```

- **Response (200 OK):**

```
{
  "message": "Transfer initiated successfully."
}
```

- **Response (400 Bad Request):**

```
{
  "status": 400,
  "error": "Bad Request",
  "message": "Insufficient funds."
}
```

- **Response (404 Not Found):**

```
{
  "status": 404,
  "error": "Not Found",
  "message": "One or both accounts not found."
}
```

- **B. GET /accounts/{accountId}**
  - **Description:** Retrieves details for a specific account.
  - **Response (200 OK):**

```
{
  "accountId": "f1e2d3c4-b5a6-9876-5432-10fedcba9876",
  "accountNumber": "1234567890",
  "accountType": "SAVINGS",
  "balance": 100.00,
  "status": "ACTIVE"
}
```

  - **Response (404 Not Found):**

```
{
  "status": 404,
```

"error": "Not Found",
"message": "Account with ID f1e2d3c4-b5a6-9876-5432-10fedcba9876 not found."
}

- **C. POST /accounts**
  - **Description:** Creates a new banking account for a specified user.
  - **Request Body (JSON):**
{
"userId": "a1b2c3d4-e5f6-7890-1234-567890abcdef",
"accountType": "SAVINGS",
"initialBalance": 100.00
}
  - **Response (201 Created):**
{
"accountId": "f1e2d3c4-b5a6-9876-5432-10fedcba9876",
"accountNumber": "1234567890",
"message": "Account created successfully."
}
  - **Response (400 Bad Request):**
{
"status": 400,
"error": "Bad Request",
"message": "Invalid account type or initial balance."
}

- **D. GET /{userId}/accounts**
  - **Description:** Retrieves all accounts associated with a specific user ID.
  - **Response (200 OK):**
[
{
"accountId": "f1e2d3c4-b5a6-9876-5432-10fedcba9876",
"accountNumber": "1234567890",
"accountType": "SAVINGS",
"balance": 100.00,
"status": "ACTIVE"
},

```
 {
  "accountId": "g7h8i9j0-k1l2-3456-7890-abcdef123456",
  "accountNumber": "0987654321",
  "accountType": "CHECKING",
  "balance": 500.50,
  "status": "ACTIVE"
 }
]
```


- **Response (404 Not Found):**


```
{
 "status": 404,
 "error": "Not Found",
 "message": "No accounts found for user ID
a1b2c3d4-e5f6-7890-1234-567890abcdef."
}
```


- **Scheduled Job:**
  - **Frequency:** Every 1 hour.
  - **Action:** Checks for idle accounts and sets their status to INACTIVE. An account is considered idle if its updated_at timestamp is older than 24 hours from the current time.

3. Transaction Service

The Transaction Service manages the initiation and execution of financial transactions between accounts.

- **Database Name:** transaction_db
- **Table Name:** transactions
- **Schema:**

Column Name	Data Type	Constraints	Description
id	UUID	PRIMARY KEY, Generated	Unique identifier for the transaction
from_account_id	UUID	Not Null	ID of the account from which funds are debited

to_account_id	UUID	Not Null	ID of the account to which funds are credited
amount	DECIMAL(15,2)	Not Null	Amount of the transaction
description	VARCHAR(255)	Nullable	Optional description of the transaction
status	ENUM('INITIATED', 'SUCCESS', 'FAILED')	Default 'INITIATED'	Current status of the transaction
timestamp	TIMESTAMP	Default CURRENT_TIMESTAMP	Timestamp of transaction creation

- **Endpoints:**
  - **A. POST /transfer/initiation**
    - **Description:** Initiates a new transfer transaction, setting its status to INITIATED.
    - **Request Body (JSON):**

```
{
  "fromAccountId": "f1e2d3c4-b5a6-9876-5432-10fedcba9876",
  "toAccountId": "g7h8i9j0-k1l2-3456-7890-abcdef123456",
  "amount": 30.00,
  "description": "Transfer to checking account"
}
```

    - **Response (200 OK):**

```
{
  "transactionId": "t1r2a3n4-s5a6-7890-1234-567890abcdef",
  "status": "Initiated",
  "timestamp": "2025-07-15T07:16:49.822Z"
}
```

    - **Response (400 Bad Request):**

```
{
  "status": 400,
```

"error": "Bad Request",
"message": "Invalid 'from' or 'to' account ID." // or "Insufficient funds"
}

- **B. POST /transfer/execution**
  - **Description:** Executes a previously initiated transfer transaction, updating its status to SUCCESS or FAILED.
  - **Request Body (JSON):**
    {
    "transactionId": "t1r2a3n4-s5a6-7890-1234-567890abcdef"
    }
  - **Response (200 OK):**
    {
    "transactionId": "t1r2a3n4-s5a6-7890-1234-567890abcdef",
    "status": "Success",
    "timestamp": "2025-07-15T07:16:49.822Z"
    }
  - **Response (400 Bad Request):**
    {
    "status": 400,
    "error": "Bad Request",
    "message": "Transaction cannot be executed."
    }
  - **Response (404 Not Found):**
    {
    "status": 404,
    "error": "Not Found",
    "message": "Transaction with ID t1r2a3n4-s5a6-7890-1234-567890abcdef not found."
    }
- **C. GET /{accountId}/transactions**
  - **Description:** Retrieves all transactions associated with a specific account ID.
  - **Response (200 OK):**
    [

```
 {
  "transactionId": "t1r2a3n4-s5a6-7890-1234-567890abcdef",
  "accountId": "f1e2d3c4-b5a6-9876-5432-10fedcba9876",
  "amount": 50.00,
  "description": "Cash deposit",
  "timestamp": "2025-06-30T10:05:00Z"
 },
 {
  "transactionId": "x1y2z3a4-b5c6-7890-1234-567890fedcba",
  "accountId": "f1e2d3c4-b5a6-9876-5432-10fedcba9876",
  "amount": -30.00,
  "description": "Transfer to checking account",
  "timestamp": "2025-06-30T10:10:00Z"
 }
]
```


- **Response (404 Not Found):**


```
{
 "status": 404,
 "error": "Not Found",
 "message": "No transactions found for account ID
f1e2d3c4-b5a6-9876-5432-10fedcba9876."
}
```


4. BFF (Backend-for-Frontend) Service

The BFF Service acts as an aggregation and orchestration layer, reducing the number of direct calls from the frontend to individual microservices.

- **Endpoints:**
  - **A. GET /dashboard/{userId}**
    - **Description:** Aggregates user profile, account details, and transaction history to provide a comprehensive dashboard view for a given user.
    - **Orchestration Logic:**
      1. Calls GET /users/{userId}/profile from the User Service.
      2. Calls GET /{userId}/accounts from the Account Service.
      3. For each account retrieved, asynchronously calls GET /{accountId}/transactions from the Transaction Service.
      4. Combines all responses into a single aggregated JSON response.
    - **Response (200 OK):**

```
{
  "userId": "a1b2c3d4-e5f6-7890-1234-567890abcdef",
  "username": "john.doe",
  "email": "john.doe@example.com",
  "firstName": "John",
  "lastName": "Doe",
  "accounts": [
   {
    "accountId": "f1e2d3c4-b5a6-9876-5432-10fedcba9876",
    "accountNumber": "1234567890",
    "accountType": "SAVINGS",
    "balance": 120.00,
    "transactions": [
     {
      "transactionId": "t1r2a3n4-s5a6-7890-1234-567890abcdef",
      "amount": 50.00,
      "toAccountId": "t1r2a3n4-s5a6-7890-1234-567890abcdef",
      "description": "Cash deposit",
      "timestamp": "2025-06-30T10:05:00Z"
     }
    ]
   }
  ]
}
```


- **Response (404 Not Found):**


```
{
  "status": 404,
  "error": "User not found",
  "message": "Failed to retrieve dashboard data due to an issue with downstream services."
}
```


- **Response (500 Internal Server Error):**


```
{
  "status": 500,
  "error": "Internal Server Error",
  "message": "Failed to retrieve dashboard data due to an issue with downstream services."
```

}

- **B. POST /transfer**
  - **Description:** Orchestrates the multi-step process of initiating and executing a fund transfer.
  - **Request Body (JSON):**
    {
     "fromAccountId": "f1e2d3c4-b5a6-9876-5432-10fedcba9876",
     "toAccountId": "g7h8i9j0-k1l2-3456-7890-abcdef123456",
     "amount": 30.00,
     "description": "Transfer to checking account"
    }
  - **Orchestration Logic:**
    1. Calls POST /transfer/initiation on the Transaction Service.
    2. If the initiation is successful (does not return an error), it then calls POST /transfer/execution on the Transaction Service.
    3. If any step returns an error, the BFF service propagates that error back to the client.
  - **Response (200 OK):**
    {
     "message": "Transfer completed successfully."
    }
  - **Response (400 Bad Request):**
    {
     "status": 400,
     "error": "Bad Request",
     "message": "Invalid 'from' or 'to' account ID." // or "Insufficient funds"
    }
  - **Response (500 Internal Server Error):**
    {
     "status": 500,
     "error": "Internal Server Error",
     "message": "Failed to complete transfer due to an issue with downstream services."
    }

5. Logging Service

The Logging Service is responsible for capturing and persisting all request and response data flowing through the BFF service.

- **Consumption Mechanism:** Consumes messages from a Kafka topic.
- **Database Name:** logging_db
- **Table Name:** logs
- **Schema:**

Column Name	Data Type	Constraints	Description
id	BIGINT	PRIMARY KEY, Auto Increment	Unique identifier for the log entry
message	TEXT	Not Null	The JSON text of the request or response
message_type	ENUM('Request', 'Response')	Not Null	Indicates if the log is a Request or Response
date_time	TIMESTAMP	Not Null	Timestamp when the log entry was recorded