Phân tích Thiết kế Backend cho SmartBank A

Tài liệu này trình bày chi tiết về kiến trúc, luồng dữ liệu, và cách tích hợp CSDL MongoDB vào hệ thống backend của ứng dụng SmartBank A. Mục tiêu là xây dựng một hệ thống an toàn, hiệu quả và có khả năng mở rộng bằng cách kết hợp thế mạnh của Hyperledger Fabric (on-chain) và MongoDB (off-chain).

1. Nguyên tắc Thiết kế: Kiến trúc Hybrid On-chain và Off-chain

Để xây dựng một ứng dụng blockchain thực tế, việc lưu trữ tất cả dữ liệu lên blockchain là một sai lầm nghiêm trọng. Thay vào đó, chúng ta áp dụng mô hình Hybrid, phân chia rõ ràng vai trò của từng nền tảng:

- **Hyperledger Fabric (On-chain): Nơi lưu trữ "Sự thật Chung"**
  - **Dữ liệu lưu trữ:** Trạng thái cốt lõi của sổ cái, bao gồm AccountId và Balance (số dư). Lịch sử các giao dịch cũng được lưu một cách bất biến trong chuỗi các khối.
  - **Tại sao?:** Dữ liệu này đòi hỏi tính minh bạch, bất biến, khả năng kiểm toán và sự đồng thuận của tất cả các ngân hàng tham gia. Một khi số dư được ghi nhận, không ai có thể đơn phương thay đổi nó.
- **MongoDB (Off-chain): Nơi lưu trữ Dữ liệu Ứng dụng và Thông tin Nhạy cảm**
  - **Dữ liệu lưu trữ:**
    1. **Thông tin định danh cá nhân (PII):** fullName, số điện thoại, email...
    2. **Thông tin xác thực ứng dụng:** hashedPin (Mã PIN đã được hash).
    3. **Dữ liệu hỗ trợ trải nghiệm người dùng:** Danh sách người nhận tin cậy.
    4. **Bản ghi giao dịch để truy vấn nhanh:** Lưu lại transactionId từ Fabric cùng các thông tin thân thiện khác.
  - **Tại sao?:**
    - **Bảo mật & Quyền riêng tư:** Tuyệt đối không đưa thông tin cá nhân hay mã PIN lên một sổ cái công khai hoặc bán công khai.
    - **Hiệu suất:** Truy vấn dữ liệu từ MongoDB nhanh hơn rất nhiều so với việc quét qua các khối trên blockchain, đặc biệt là cho các tính năng như hiển thị lịch sử giao dịch.
    - **Linh hoạt:** Dễ dàng thêm, sửa, xóa các dữ liệu ứng dụng mà không ảnh hưởng đến sổ cái.
    - **Chi phí lưu trữ:** Lưu trữ dữ liệu lớn trên blockchain rất tốn kém.

2. Luồng Xử lý của Backend API

Backend Node.js/Express.js đóng vai trò là "tổng đài viên" thông minh, điều phối thông tin giữa các hệ thống. Ta sẽ phân tích luồng xử lý của nghiệp vụ quan trọng

nhất: **Chuyển tiền (POST /api/transfer)**.

1. **Nhận yêu cầu:** Frontend gửi một yêu cầu chứa currentUser, recipient, amount, và pin đến API.
2. **Xác thực PIN (Tương tác với MongoDB):**
   - Backend kết nối đến MongoDB, truy vấn collection users để tìm document có userId khớp với currentUser.
   - Lấy ra trường hashedPin.
   - Sử dụng một thư viện (ví dụ bcrypt) để hash mã pin mà người dùng gửi lên.
   - So sánh chuỗi hash vừa tạo với chuỗi hashedPin trong CSDL.
   - Nếu không khớp, ngay lập tức trả về lỗi **401 Unauthorized** và dừng quy trình.
3. **Gửi Giao dịch (Tương tác với Fabric):**
   - Nếu PIN hợp lệ, backend sử dụng Fabric SDK để kết nối tới mạng lưới, sử dụng đúng danh tính (chứng chỉ) của currentUser.
   - Gửi một yêu cầu thực thi (submit transaction) đến hàm Transfer(from, to, amount) trong chaincode.
4. **Xử lý Kết quả từ Fabric:**
   - Fabric SDK sẽ chờ cho đến khi giao dịch được xác thực và commit lên sổ cái.
   - SDK trả về kết quả, bao gồm trạng thái (thành công/thất bại) và **mã giao dịch (Transaction ID)**.
5. **Ghi nhận Lịch sử (Tương tác với MongoDB):**
   - Nếu giao dịch Fabric thành công, backend sẽ tạo một document mới trong collection transaction_logs của MongoDB.
   - Document này sẽ chứa fabricTxId vừa nhận được, cùng các thông tin khác như người gửi, người nhận, số tiền, thời gian.
6. **Phản hồi cho Frontend:**
   - Backend gửi về cho Frontend một thông báo thành công (ví dụ: 200 OK) cùng với transactionId để hiển thị cho người dùng.

3. Cấu hình và Tích hợp MongoDB

3.1. Cài đặt và Khởi chạy

Cách đơn giản nhất để chạy MongoDB cho môi trường phát triển là sử dụng Docker:

```
docker run --name smartbank-mongo -p 27017:27017 -e
MONGO_INITDB_ROOT_USERNAME=admin -e
MONGO_INITDB_ROOT_PASSWORD=password -d mongo
```

Lệnh này sẽ tạo một container MongoDB, mở cổng 27017, và tạo một user quản trị với

username/password là admin/password.

3.2. Thiết kế Schema (Cấu trúc Collection)

Chúng ta sẽ sử dụng 2 collection chính:

- **Collection users:**

```
{
  "userId": "User1@smartbanka.com", // Khớp với danh tính on-chain
  "fullName": "Nguyễn Văn Tuấn",
  "bankName": "SmartBank A",
  "hashedPin": "...", // Chuỗi hash của mã PIN
  "trustedRecipients": ["User1@smartbankb.com"],
  "createdAt": "..."
}
```

- **Collection transaction_logs:**

```
{
  "fabricTxId": "a1b2c3...", // Mã giao dịch từ Fabric
  "fromAccount": "User1@smartbanka.com",
  "toAccount": "User1@smartbankb.com",
  "amount": 500000,
  "timestamp": "...",
  "status": "COMPLETED"
}
```

3.3. Kết nối từ Node.js và Bảo mật

Trong ứng dụng Node.js, ta sử dụng thư viện mongoose hoặc mongodb để kết nối.

```
// Ví dụ kết nối dùng thư viện mongodb
const { MongoClient } = require('mongodb');

const uri = "mongodb://admin:password@localhost:27017";
const client = new MongoClient(uri);

async function connectToDB() {
  try {
    await client.connect();
    console.log("Kết nối MongoDB thành công!");
  } catch (e) {
```

```
    console.error(e);
  }
}
connectToDB();
```

Về bảo mật mã PIN:
Việc hash mã PIN là bắt buộc. Không bao giờ lưu PIN dưới dạng văn bản thuần. Sử dụng thư viện bcrypt là một lựa chọn tốt vì nó đã tích hợp sẵn "salt", giúp chống lại các cuộc tấn công bảng cầu vồng (rainbow table).

```
const bcrypt = require('bcrypt');
const saltRounds = 10;

// Khi người dùng đăng ký PIN
const plainPin = '123456';
const hashedPin = await bcrypt.hash(plainPin, saltRounds);
// -> Lưu hashedPin vào MongoDB

// Khi người dùng nhập PIN để xác thực
const isMatch = await bcrypt.compare(plainPin, hashedPinFromDB);
// -> isMatch sẽ là true nếu PIN đúng
```