**Mô tả: "Trong ngân hàng XYZ, khách hàng có thể đăng ký tài khoản và làm thẻ (trong trường hợp này là thẻ VISA) dựa trên một thẻ căn cước. Khi tiến hành giao dịch tại các chi nhánh của ngân hàng, khách hàng dùng thông tin nhân thân (căn cước) để có thể giao dịch. Khi thực hiện các giao dịch trên các máy ATM, người dùng cần một thẻ để tiến hành giao dịch. Thông tin giao dịch sẽ được lưu giữ sau mỗi lần giao dịch dù thành công hay thất bại. Ngoài ra, để tạo thuận lợi cho khách hàng, ngân hàng còn tạo một trang web và một ứng dụng di động cho phép khách hàng đăng ký thành viên và thực hiện các công việc cần thiết."**

**Yêu cầu:**

**1. Hãy chọn/thiết kế một kiến trúc cho phép ngân hàng thực hiện được công việc này. Bạn có thể thâm vào các giả thuyết khác để việc thiết kế được tốt hơn.**

**2. Hãy chọn một chức năng trong phần thiết kế để cài đặt nó.**

**Mặc dù kiến trúc Layered mang lại nhiều lợi ích, nhưng cũng tồn tại một số nhược điểm:**

* Tính ràng buộc giữa các tầng: Các tầng trong kiến trúc Layered thường giao tiếp theo hướng đơn điệu từ tầng trên xuống tầng dưới. Điều này có thể dẫn đến tính ràng buộc cao giữa các tầng và làm giảm tính linh hoạt của hệ thống.
* Khó khăn trong việc mở rộng theo chiều ngang: Kiến trúc Layered thường không linh hoạt khi cần mở rộng theo chiều ngang (horizontal scaling), tức là thêm các phiên bản của cùng một tầng để xử lý tải cao. Việc này có thể gây ra vấn đề về đồng bộ hóa và quản lý.
* Hiệu suất: Việc thông qua nhiều tầng có thể làm giảm hiệu suất của hệ thống do các cuộc gọi hàm trung gian và việc truy cập dữ liệu qua các tầng khác nhau.
* Tính bảo mật: Mặc dù kiến trúc Layered có thể cung cấp tính bảo mật tốt hơn, nhưng nếu một tầng bị xâm nhập, nó có thể dễ dàng lan rộng sang các tầng khác qua các giao tiếp giữa các tầng.
* Khả năng quản lý: Kiến trúc Layered đòi hỏi một quản lý kỹ thuật cao để duy trì sự phân tách giữa các tầng và đảm bảo tính nhất quán của hệ thống.
* Khó khăn trong việc thử nghiệm và gỡ lỗi: Vì mỗi tầng hoạt động độc lập, việc thử nghiệm và gỡ lỗi có thể phức tạp hơn so với các kiểu kiến trúc khác.

Tóm lại, mặc dù kiến trúc Layered có nhiều ưu điểm, nhưng cũng có nhược điểm, đặc biệt là khi không được thiết kế hoặc quản lý một cách cẩn thận, có thể dẫn đến sự phức tạp và hiệu suất kém của hệ thống.

**Khi thiết kế theo kiến trúc Layered, có một số trade-off mà chúng ta cần phải đánh đổi:**

* Tính linh hoạt và tính nhất quán:
  + Lợi ích: Kiến trúc Layered tạo ra một môi trường phân tầng, giúp dễ dàng quản lý, bảo trì và mở rộng hệ thống.
  + Chi phí: Tính linh hoạt và tính nhất quán đôi khi có thể giảm đi một chút do sự ràng buộc giữa các tầng và khó khăn trong việc mở rộng theo chiều ngang.
* Hiệu suất và Tính bảo mật:
  + Lợi ích: Kiến trúc Layered có thể cung cấp tính bảo mật tốt hơn bằng cách phân tách rõ ràng giữa các tầng.
  + Chi phí: Tính ràng buộc giữa các tầng có thể làm giảm hiệu suất của hệ thống do các cuộc gọi hàm trung gian và việc truy cập dữ liệu qua các tầng khác nhau.
* Dễ quản lý và Dễ bảo trì:
  + Lợi ích: Kiến trúc Layered làm cho việc quản lý và bảo trì hệ thống trở nên dễ dàng hơn, do mỗi tầng có trách nhiệm cụ thể và độc lập.
  + Chi phí: Cần có một quản lý kỹ thuật cao để duy trì sự phân tách giữa các tầng và đảm bảo tính nhất quán của hệ thống.
* Tính mở rộng và Khả năng tái sử dụng:
  + Lợi ích: Kiến trúc Layered cung cấp khả năng mở rộng và tái sử dụng mã nguồn dễ dàng hơn.
  + Chi phí: Việc mở rộng theo chiều ngang có thể gặp khó khăn và đòi hỏi sự đồng bộ giữa các tầng.

Tóm lại, khi thiết kế theo kiến trúc Layered, chúng ta phải đánh đổi giữa tính linh hoạt, hiệu suất, tính bảo mật, dễ quản lý và khả năng mở rộng. Có những ưu và nhược điểm riêng của kiến trúc này, và việc lựa chọn phù hợp phụ thuộc vào yêu cầu cụ thể của dự án.

Chọn kiến trúc nào: Kiến trúc theo lớp (Layered Architecture).

Vì sao chọn kiến trúc đó: Kiến trúc này chia hệ thống thành các lớp với chức năng rõ ràng và khả năng phân chia trách nhiệm, giúp cho việc phát triển và bảo trì trở nên đơn giản hơn. Nó phù hợp với các ngân hàng có nhiều loại hình dịch vụ và cần một cấu trúc rõ ràng để quản lý giao dịch và thông tin khách hàng.

Lợi ích là gì: Lợi ích chính của kiến trúc theo lớp là việc cô lập các phần khác nhau của hệ thống, giảm thiểu sự phụ thuộc lẫn nhau. Nó giúp đơn giản hóa việc phát triển và bảo trì bằng cách cho phép các nhà phát triển tập trung vào một phần nhất định của hệ thống mà không ảnh hưởng đến các phần khác. Kiến trúc này cũng dễ dàng tích hợp với các hệ thống sẵn có và cho phép tái sử dụng mã nguồn.

Cái giá phải trả là gì: Mặc dù kiến trúc theo lớp cung cấp sự cô lập và tổ chức tốt cho hệ thống, nó có thể dẫn đến hiệu suất không tối ưu do chi phí điều phối giữa các lớp. Trong một số trường hợp, việc thay đổi hay cập nhật một lớp có thể yêu cầu thay đổi ở các lớp khác. Điều này cũng có thể làm tăng độ phức tạp khi triển khai và có thể gây khó khăn trong việc kiểm thử do các phụ thuộc không rõ ràng giữa các lớp.

* Giả sử ngân hàng lưu rất nhiều thông tin người dùng 20 triệu record, và chỉ lưu database để truy xuất chính ở cụm Hồ Chí Minh. Lúc người dùng đi rút tiền ở 1 cây ATM nào đó trên địa bàn của đất nước Việt Nam ví dụ ở tỉnh Quảng Nam, khi đó lượng dữ liệu chỉ đang lưu trữ ở 1 nơi là Hồ Chí Minh sẽ làm cho việc truy xuất database lâu hơn. Cho nên có thể chia database lưu trữ thành các cụm redis database ở Bắc-Trung-Nam dựa vào địa chỉ khách hàng đã đăng kí trên thẻ ngân hàng ở 3 thành phố chính là Hà Nội, Sài Gòn, Đà Nẵng, khi đó sẽ tránh tải cho việc truy xuất data làm cho các giao dịch trở nên nhanh hơn. Đặc biệt là thông tin tài khoản ngân hàng của khách hàng rất là quan trọng nên chúng ta không thể cho trực tiếp truy xuất đến các database mà phải nên tạo ra các view ảo để tránh ảnh hưởng mất dữ liệu người dùng. Ngoài view thì có thể thêm các biện pháp như ghi cache data, redis data để có thể backup dữ liệu.

A diagram of a computer

Description automatically generated

Flow giao dịch trực tiếp tại quầy (người dùng giao dịch bằng thông tin cccd):

* Tại phòng giao dịch, người dùng nhập vào thông tin cccd (mã số cccd) -> Người dùng đọc thông tin cccd cho nhân viên -> Nhân viên nhập số cccd (ở đây sẽ thực hiện hàm check thông tin căn cước trong dữ liệu ngân hàng như flow trước đã làm).
* Tại các cây ATM, người dùng sẽ dùng thẻ cho vào máy, máy check thẻ hợp lệ. Màn hình hiển thị loại giao dịch mà người dùng chọn (có thể rút tiền, chuyển tiền...). Ví dụ người dùng chọn rút tiền -> nhập số tiền cần rút -> check tài khoản coi còn đủ số dư không (gọi function checkBalancer(double money) ) -> nếu số tiền rút nhỏ hơn hoặc bằng tiền dư trong tài khoản thì cho phép rút -> Thực hiện function withDraw() -> lưu dữ liệu vào database giao dịch, đồng thời cập nhật dữ liệu tại database tài khoản. Thông báo khách hàng “rút tiền thành công” -> khách hàng nhận tiền, lấy lại thẻ, và đi về. (Cái flow này chỉ đang mô tả cho việc check số dư true, còn false thì sẽ thông báo cho khách hàng là số tiền không hợp lệ, số tiền rút vượt quá số tiền còn trong tk).