Viết Use Case Hiệu Quả cho Kiến Trúc Sư Giải Pháp trong Bối Cảnh Tích Hợp Hệ Thống Ngân Hàng Phức Tạp

I. Giới thiệu: Vai trò Quan trọng của Use Case trong Tích hợp Hệ thống Ngân hàng

Bối cảnh: Ngành ngân hàng đang trải qua một quá trình chuyển đổi số mạnh mẽ, dẫn đến sự gia tăng phức tạp trong kiến trúc hệ thống. Nhu cầu cung cấp trải nghiệm khách hàng liền mạch, chẳng hạn như quy trình onboarding (mở tài khoản và dịch vụ) nhanh chóng và hoàn toàn trực tuyến [1], cùng với các yêu cầu tuân thủ pháp lý ngày càng nghiêm ngặt như KYC (Know Your Customer) và AML (Anti-Money Laundering) [2], đòi hỏi sự tích hợp chặt chẽ giữa nhiều hệ thống khác nhau. Các hệ thống này thường bao gồm các hệ thống lõi (core banking) truyền thống, các nền tảng quản lý khách hàng (Customer Platform/CRM), các ứng dụng di động hiện đại, các dịch vụ eKYC của bên thứ ba, và các hệ thống phụ trợ khác.[1]

Thách thức của Kiến trúc sư Giải pháp: Trong môi trường phức tạp này, Kiến trúc sư Giải pháp (Solution Architect - SA) đối mặt với thách thức lớn trong việc thiết kế và tài liệu hóa các luồng tích hợp. SA cần đảm bảo sự rõ ràng, nhất quán và đồng bộ giữa yêu cầu nghiệp vụ và việc triển khai kỹ thuật, đồng thời phải giao tiếp hiệu quả với nhiều bên liên quan có nền tảng khác nhau (nghiệp vụ, kỹ thuật, tuân thủ).[5] Việc thiếu một phương pháp tài liệu hóa chuẩn mực và hiệu quả có thể dẫn đến hiểu lầm, thiết kế sai lệch và thất bại trong dự án tích hợp.

Use Case như một Công cụ Giải pháp: Use case, khi được xây dựng đúng cách, trở thành một công cụ nền tảng không thể thiếu cho SA. Nó không chỉ đơn thuần là một phương tiện để thu thập yêu cầu, mà còn là một bản thiết kế chi tiết cho các tương tác, định nghĩa giao diện và là cơ sở cho các quyết định kiến trúc quan trọng.[12] Một use case hiệu quả cung cấp cái nhìn rõ ràng về cách các hệ thống tương tác với nhau để đạt được một mục tiêu nghiệp vụ cụ thể.

Ví dụ xuyên suốt: Để minh họa các khái niệm và phương pháp, báo cáo này sẽ sử dụng một ví dụ xuyên suốt: Quy trình Onboarding Khách hàng qua Mobile Banking. Trong kịch bản này, khách hàng sử dụng ứng dụng Mobile Banking để đăng ký mở tài khoản. Quá trình này yêu cầu ứng dụng Mobile App tương tác với hệ thống Backend, hệ thống Backend tiếp tục tương tác với Customer Platform để lấy thông tin khách hàng, kiểm tra trạng thái eKYC (có thể thông qua một hệ thống eKYC riêng biệt), và cuối cùng là tạo tài khoản trong hệ thống Core Banking.[1] Mục tiêu là khách hàng có thể hoàn thành quá trình onboarding một cách nhanh chóng và thuận tiện qua kênh di

động.

Lộ trình Báo cáo: Báo cáo này sẽ đi sâu vào các khía cạnh chính của việc viết use case hiệu quả cho SA trong bối cảnh tích hợp ngân hàng, bao gồm:

1. Nền tảng về use case trong hệ thống tích hợp.
2. Xác định tác nhân và hệ thống liên quan trong kịch bản onboarding.
3. Mô tả chi tiết các luồng tương tác (luồng chính, luồng thay thế, ngoại lệ).
4. Cấu trúc, mẫu và định dạng use case phù hợp cho SA.
5. Trực quan hóa luồng bằng Sơ đồ Tuần tự UML (Sequence Diagram).
6. Định nghĩa giao diện và hợp đồng dữ liệu giữa các hệ thống.
7. Tích hợp các yêu cầu phi chức năng (NFRs).
8. Các thực tiễn tốt nhất trong ngành ngân hàng.

II. Nền tảng: Use Case trong Hệ thống Ngân hàng Tích hợp (Đáp ứng Q1)

Định nghĩa Use Case:
Một use case mô tả một chuỗi các tương tác giữa một hoặc nhiều tác nhân (actors) - có thể là người dùng hoặc các hệ thống khác - và hệ thống đang được thiết kế, nhằm đạt được một mục tiêu cụ thể, có thể quan sát được và mang lại giá trị cho tác nhân khởi tạo.10 Nó tập trung vào hành vi bên ngoài của hệ thống, mô tả cái gì hệ thống làm và cách tác nhân sử dụng nó, thay vì làm thế nào hệ thống thực hiện điều đó bên trong.12
Trong bối cảnh phát triển phần mềm, cần phân biệt giữa *System Use Case* và *Business Use Case*. System Use Case tập trung vào các tương tác với một hệ thống phần mềm cụ thể, mô tả các chức năng mà hệ thống đó cung cấp. Đây là loại use case chủ yếu được đề cập trong báo cáo này. Ngược lại, Business Use Case mô tả các quy trình nghiệp vụ ở mức độ cao hơn, tập trung vào cách một tổ chức hoạt động để đạt được mục tiêu kinh doanh, không nhất thiết liên quan đến một hệ thống phần mềm cụ thể.[10]

Mục đích đối với Kiến trúc sư Giải pháp:
Đối với SA, use case không chỉ là tài liệu yêu cầu mà còn là công cụ thiết kế kiến trúc mạnh mẽ:

- **Thu thập & Đặc tả Yêu cầu:** Ghi lại một cách có hệ thống *những gì* hệ thống tích hợp phải làm từ góc độ tương tác.[10] Use case giúp tập trung vào người dùng và mục tiêu của họ, đảm bảo rằng các chức năng được phát triển mang lại giá trị thực sự.[10]
- **Xác định Phạm vi:** Giúp định nghĩa rõ ràng ranh giới của hệ thống hoặc thành phần đang được xem xét và phạm vi chức năng của nó.[11] Điều này đặc biệt quan trọng trong các hệ thống tích hợp phức tạp.
- **Tạo điều kiện Giao tiếp:** Cung cấp một ngôn ngữ chung, dễ hiểu (thông qua mô

tả văn bản và sơ đồ trực quan) cho tất cả các bên liên quan, bao gồm nghiệp vụ, kỹ thuật, kiểm thử và quản lý, giúp giảm thiểu hiểu lầm.[10]

- **Động lực cho Thiết kế Kiến trúc:** Các tương tác chính được mô tả trong use case giúp SA xác định các thành phần kiến trúc cần thiết, các giao diện giữa chúng và các luồng dữ liệu quan trọng.[8] Các use case quan trọng thường định hình kiến trúc ứng dụng.[13]
- **Xác định Kịch bản Kiểm thử:** Use case là cơ sở để xác định các kịch bản kiểm thử hệ thống và kiểm thử tích hợp, đảm bảo rằng hệ thống hoạt động đúng như mong đợi trong các tình huống sử dụng khác nhau.[12]

Các Thành phần Cốt lõi của Use Case:
Một use case điển hình bao gồm các thành phần sau:

- **Tác nhân (Actors):** Bất kỳ thực thể nào tương tác với hệ thống, có thể là người dùng (với các vai trò khác nhau), các hệ thống phần mềm khác, thiết bị phần cứng hoặc thậm chí là yếu tố thời gian.[10] Trong bối cảnh tích hợp, việc xác định rõ ràng các *hệ thống* khác là tác nhân đóng vai trò cực kỳ quan trọng.
- **Hệ thống (System/Subject):** Ranh giới của hệ thống hoặc thành phần đang được mô tả trong use case.[11] SA cần định nghĩa ranh giới này một cách chính xác trong ngữ cảnh nhiều hệ thống.
- **Mục tiêu (Goal):** Kết quả thành công mà tác nhân chính mong muốn đạt được thông qua việc tương tác với hệ thống.[10] Mục tiêu phải mang lại giá trị có thể quan sát được.[12]
- **Tương tác/Luồng (Interactions/Flow):** Chuỗi các bước hoặc hành động tuần tự giữa tác nhân và hệ thống để đạt được mục tiêu.[10]
- **Tiền điều kiện (Preconditions):** Trạng thái của hệ thống hoặc môi trường *trước khi* use case có thể bắt đầu. Chúng là các điều kiện phải đúng để use case được thực thi.[16] Tiền điều kiện rất quan trọng để xác định sự phụ thuộc giữa các use case hoặc các bước trong luồng tích hợp.
- **Hậu điều kiện (Postconditions):** Trạng thái của hệ thống *sau khi* use case hoàn thành (thành công hoặc thất bại). Chúng mô tả kết quả được đảm bảo hoặc sự thay đổi trạng thái trên các hệ thống liên quan.[16]
- **Kịch bản/Luồng (Scenarios/Flows):** Bao gồm luồng thành công chính (main success scenario hay happy path), các luồng thay thế (alternative flows) và cách xử lý ngoại lệ (exception handling).[16]

Việc các use case định nghĩa các tương tác, tiền điều kiện và hậu điều kiện liên quan đến nhiều hệ thống [12] và kiến trúc tích hợp đòi hỏi các giao diện rõ ràng và sự tách biệt [4] cho thấy một vai trò sâu sắc hơn của use case. Trong các kịch bản tích hợp phức tạp, một use case không chỉ mô tả chức năng mà còn hoạt động như một *hợp đồng kiến trúc* ngầm, phác thảo hành vi dự kiến và các thay đổi trạng thái cần thiết

trên các ranh giới hệ thống. Các tiền điều kiện và hậu điều kiện trở thành những khẳng định trạng thái quan trọng mà kiến trúc phải đảm bảo thực hiện được trên toàn bộ các hệ thống tham gia. Ví dụ, hậu điều kiện thành công của use case onboarding không chỉ là "tài khoản được tạo" mà phải là "bản ghi khách hàng được cập nhật trạng thái KYC trong Customer Platform VÀ tài khoản được tạo thành công trong Core Banking". Điều này nâng tầm use case từ mô tả chức năng đơn thuần thành một đặc tả về tính nhất quán trạng thái phân tán, đòi hỏi SA phải đảm bảo các điều kiện này phản ánh chính xác trạng thái phối hợp cần thiết giữa tất cả các hệ thống liên quan.

III. Định nghĩa Kịch bản: Onboarding qua Mobile Banking (Đáp ứng Q2)

Xác định Tác nhân:
Trong kịch bản onboarding khách hàng qua Mobile Banking, các tác nhân tham gia bao gồm:

- **Tác nhân Chính (Primary Actor):**
  - **Khách hàng (Customer):** Người dùng cuối khởi tạo và thực hiện quá trình onboarding thông qua ứng dụng di động.[16]
- **Tác nhân Phụ/Hỗ trợ (Secondary/Supporting Actors - Chủ yếu là Hệ thống):**
  - **Ứng dụng Mobile Banking (Mobile App):** Giao diện người dùng (frontend) mà khách hàng tương tác trực tiếp [User Query]. Đây vừa là điểm bắt đầu, vừa là một tác nhân tương tác với Backend.
  - **Mobile Banking Backend:** Lớp API hoặc tầng điều phối (orchestration layer) xử lý logic nghiệp vụ, điều phối các cuộc gọi đến các hệ thống khác [User Query]. Đây là trung tâm của quá trình tích hợp.
  - **Customer Platform:** Hệ thống lưu trữ dữ liệu gốc của khách hàng (master data), có thể là hệ thống CRM hoặc một nền tảng dữ liệu khách hàng chuyên dụng.[1]
  - **Hệ thống eKYC:** Dịch vụ nội bộ hoặc của bên thứ ba thực hiện việc xác minh danh tính điện tử.[1]
  - **Hệ thống Core Banking:** Hệ thống ghi nhận chính (system of record) cho các tài khoản, số dư và các giao dịch tài chính cốt lõi.[1]

Việc liệt kê rõ ràng *tất cả* các hệ thống tương tác như là các tác nhân là cực kỳ quan trọng đối với SA để hình dung bức tranh toàn cảnh của việc tích hợp.[11] Điều này buộc SA phải xem xét mọi điểm tương tác và giao diện cần thiết.

Định nghĩa Ranh giới Hệ thống:
Khái niệm "ranh giới hệ thống" (system boundary) giúp phân định rõ ràng những gì nằm "bên trong" phạm vi đang xét và những gì là các thực thể bên ngoài tương tác với nó (tác nhân).11 Ranh giới này có thể được biểu diễn trực quan bằng hộp chữ nhật trong sơ đồ use case.11

Trong bối cảnh tích hợp nhiều hệ thống, việc xác định ranh giới là rất quan trọng nhưng cũng mang tính ngữ cảnh:

- Nếu xem xét toàn bộ *quy trình nghiệp vụ* onboarding, ranh giới có thể bao gồm tất cả các hệ thống tham gia: Mobile App, Backend, Customer Platform, eKYC, và Core Banking.
- Tuy nhiên, khi viết một *use case cụ thể* cho một bước trong quy trình (ví dụ: "Xác minh trạng thái eKYC"), "hệ thống" (subject) có thể chỉ là Mobile Banking Backend. Khi đó, Mobile App (gửi yêu cầu), Customer Platform (cung cấp dữ liệu cần thiết), và Hệ thống eKYC (thực hiện xác minh) sẽ đóng vai trò là các tác nhân tương tác *với* Backend.[12]

Việc xác định rõ ràng ranh giới giúp SA hiểu rõ họ đang thiết kế thành phần nào và đang tích hợp với thành phần nào, từ đó định hướng thiết kế giao diện và phân chia trách nhiệm.[4] Đối với SA khi tài liệu hóa một quy trình đa hệ thống, "hệ thống đang thiết kế" (system under design) cho một bước use case *cụ thể* có thể chỉ là một thành phần đơn lẻ (ví dụ: Backend orchestrator), trong khi các hệ thống liên quan khác lại đóng vai trò "tác nhân" từ góc nhìn của thành phần đó. Toàn bộ *quy trình nghiệp vụ* liên quan đến tất cả các hệ thống, nhưng mô tả use case riêng lẻ thường tập trung vào một phạm vi hẹp hơn. Do đó, SA phải nêu rõ *phạm vi* hoặc "hệ thống đang thiết kế" cho mỗi tài liệu use case chi tiết trong luồng tích hợp lớn hơn. Điều này tránh sự mơ hồ về việc thành phần nào chịu trách nhiệm thực thi logic cụ thể so với việc chỉ phản hồi yêu cầu, và sự lựa chọn ranh giới này ảnh hưởng trực tiếp đến cách mô tả các tương tác.

Danh sách các Hệ thống Liên quan (Tóm tắt):
Như đã xác định, các hệ thống chính tham gia vào ví dụ onboarding bao gồm: Mobile App, Mobile Banking Backend, Customer Platform, Hệ thống eKYC, và Hệ thống Core Banking. Cần lưu ý rằng trong thực tế, các quy trình onboarding có thể phức tạp hơn và liên quan đến nhiều hệ thống khác nữa, ví dụ như Hệ thống Quản lý Tài liệu (Document Management System), Hệ thống Phát hiện Gian lận (Fraud Detection System), Hệ thống Thông báo (Notification System).1

IV. Mô tả Chi tiết Tương tác: Các Luồng Use Case (Đáp ứng Q3 & Q4)

Việc mô tả chi tiết các luồng tương tác là trọng tâm của use case, đặc biệt là đối với SA cần hiểu rõ cách các hệ thống phối hợp với nhau.

Tài liệu hóa Luồng Thành công Chính (Happy Path):
Đây là kịch bản phổ biến nhất, mô tả chuỗi các bước diễn ra khi mọi thứ hoạt động đúng như mong đợi và dẫn đến kết quả thành công.16

- **Định dạng:** Sử dụng định dạng rõ ràng, đánh số thứ tự các bước, theo cấu trúc "Hành động của Tác nhân" / "Phản hồi của Hệ thống".[24] Điều quan trọng là trong phần "Phản hồi của Hệ thống", cần nêu rõ *thành phần hệ thống nào* đang thực hiện hành động và tương tác với các hệ thống khác.
- **Ví dụ các bước (Minh họa - Mức cao) cho Onboarding Happy Path:**
  1. **Khách hàng** (Tác nhân) bắt đầu onboarding trên **Mobile App** (Tác nhân/Hệ thống).
  2. **Mobile App** yêu cầu nhập thông tin ban đầu (ví dụ: tên, email, SĐT).
  3. **Khách hàng** nhập thông tin.
  4. **Mobile App** gửi dữ liệu đến **Mobile Banking Backend** (Hệ thống).
  5. **Mobile Banking Backend** gọi API getCustomerByContactInfo của **Customer Platform** (Tác nhân) để kiểm tra khách hàng tồn tại.
  6. **Customer Platform** phản hồi (ví dụ: 'Khách hàng mới').
  7. **Mobile Banking Backend** gọi API initiateVerification của **Hệ thống eKYC** (Tác nhân), truyền dữ liệu cần thiết.
  8. **Hệ thống eKYC** thực hiện quy trình xác minh (chi tiết có thể được mô tả trong use case con hoặc trừu tượng hóa).
  9. **Hệ thống eKYC** trả về trạng thái 'Thành công' cho **Mobile Banking Backend** qua callback hoặc API getVerificationStatus.
  10. **Mobile Banking Backend** gọi API createCustomer hoặc updateCustomerKYCStatus của **Customer Platform** để tạo/cập nhật bản ghi khách hàng với trạng thái KYC.
  11. **Customer Platform** xác nhận thành công.
  12. **Mobile Banking Backend** gọi API openAccount của **Hệ thống Core Banking** (Tác nhân) để tạo tài khoản mới.
  13. **Hệ thống Core Banking** tạo tài khoản và trả về thông tin tài khoản (số tài khoản).
  14. **Mobile Banking Backend** lưu trữ thông tin liên quan và xác nhận thành công cho **Mobile App**.
  15. **Mobile App** hiển thị thông báo onboarding thành công cho **Khách hàng**.
- **Trọng tâm Kiến trúc:** Trong các bước liên quan đến tương tác backend, cần nêu rõ hệ thống khởi tạo cuộc gọi, hệ thống nhận cuộc gọi và (nếu có thể) tên API hoặc loại thông điệp được sử dụng (ví dụ: "Backend gọi API 'getCustomerDetails' của Customer Platform"). Điều này cung cấp thông tin đầu vào trực tiếp cho việc thiết kế giao diện.

Trong bối cảnh tích hợp, việc chỉ mô tả chung chung như "Hệ thống xác minh dữ liệu" là không đủ cho SA.[25] Luồng use case *phải* chỉ định *thành phần hệ thống nào* thực hiện hành động, *hệ thống nào khác* mà nó tương tác, và lý tưởng nhất là *cách thức*

tương tác (ví dụ: "Backend gọi API 'validateAddress' của Customer Platform qua REST/JSON").[7] SA cần mức độ chi tiết này để xác định các API/giao diện cần thiết, hiểu yêu cầu trao đổi dữ liệu, định nghĩa logic điều phối (orchestration), và đánh giá các điểm lỗi tiềm ẩn giữa các hệ thống.[4] Use case trở thành đầu vào trực tiếp cho thiết kế giao diện và lựa chọn các mẫu tích hợp (ví dụ: request-response, nhắn tin bất đồng bộ [29]).

Xác định và Tài liệu hóa Luồng Thay thế (Alternative Flows):
Đây là các biến thể so với luồng chính, nhưng vẫn dẫn đến một kết quả thành công hoặc một kết quả thay thế đã được xác định trước.16 Chúng không phải là lỗi, mà là các kịch bản hợp lệ khác.

- **Đánh số:** Đánh số rõ ràng, thường liên kết với bước trong luồng chính nơi sự phân nhánh xảy ra.[24]
- **Ví dụ cho Onboarding:**
  - *Tại bước 6:* Khách hàng đã tồn tại trên Customer Platform. Luồng thay thế có thể bao gồm các bước xác minh danh tính bổ sung (ví dụ: gửi OTP) và liên kết yêu cầu onboarding mới với hồ sơ hiện có, thay vì tạo mới.
  - *Tại bước 9:* Hệ thống eKYC trả về trạng thái 'Cần xem xét thủ công' (Manual Review Required). Luồng thay thế có thể bao gồm việc thông báo cho khách hàng về việc tạm dừng, lưu trạng thái chờ, và kích hoạt một quy trình xử lý nội bộ cho bộ phận nghiệp vụ.
  - Khách hàng chọn một sản phẩm/loại tài khoản khác trong quá trình onboarding.
- **Trọng tâm Kiến trúc:** Mô tả cách các phản hồi khác nhau từ các hệ thống phụ trợ kích hoạt các nhánh logic thay thế trong thành phần điều phối (thường là Backend).

Xử lý Ngoại lệ và Lỗi (Exception Handling):
Đây là phần mô tả các điều kiện lỗi dự kiến có thể xảy ra và cách hệ thống phải phản hồi.16 Việc xử lý ngoại lệ bao gồm cả lỗi nghiệp vụ và lỗi kỹ thuật.

- **Ví dụ cho Onboarding:**
  - *Tại bước 9:* Hệ thống eKYC trả về trạng thái 'Thất bại' (Failed). Phản hồi hệ thống: Thông báo cho khách hàng, ghi log lỗi, có thể đề xuất phương thức xác minh thay thế (ví dụ: tại quầy), kết thúc luồng onboarding.
  - *Tại bước 5:* Customer Platform không khả dụng (lỗi kết nối, timeout). Phản hồi hệ thống: Áp dụng cơ chế thử lại (retry)? Thông báo cho khách hàng về sự cố tạm thời? Chấm dứt quy trình và yêu cầu thử lại sau?
  - *Tại bước 13:* Hệ thống Core Banking từ chối tạo tài khoản do vi phạm quy tắc dữ liệu (ví dụ: thông tin không hợp lệ). Phản hồi hệ thống: Ghi log lỗi chi tiết, xem xét việc rollback cập nhật trạng thái KYC trên Customer Platform (nếu

cần thiết để đảm bảo tính nhất quán), thông báo lỗi cho khách hàng, có thể tạo yêu cầu xử lý nội bộ.
    - Lỗi mạng giữa Backend và Hệ thống eKYC.
- **Trọng tâm Kiến trúc:** Xử lý ngoại lệ là yếu tố sống còn đối với sự ổn định của hệ thống tích hợp. Use case phải chỉ định hành vi mong muốn của hệ thống khi gặp lỗi để SA có thể thiết kế các cơ chế xử lý lỗi phù hợp, bao gồm ghi log chi tiết, cơ chế thử lại (retry policies), thông báo lỗi, và quan trọng nhất là các chiến lược đảm bảo tính nhất quán dữ liệu giữa các hệ thống (ví dụ: cơ chế bù trừ - compensation logic, rollback). Cần xác định rõ trạng thái mong muốn của từng hệ thống sau khi một ngoại lệ xảy ra.

V. Cấu trúc để Rõ ràng: Mẫu và Định dạng Use Case (Đáp ứng Q5, phần đầu)

Sự cần thiết của Cấu trúc: Một mẫu (template) chuẩn hóa là rất quan trọng, đặc biệt trong các dự án phức tạp, để đảm bảo tính nhất quán, đầy đủ và dễ đọc giữa các use case khác nhau và giữa các thành viên trong nhóm.[10]

Xem xét các Mẫu Hiện có: Có nhiều mẫu use case khác nhau đã được đề xuất và sử dụng.[16] Tuy nhiên, điều quan trọng là phải điều chỉnh hoặc chọn một mẫu phù hợp với nhu cầu cụ thể của việc tài liệu hóa tích hợp hệ thống và cung cấp đủ thông tin cho SA.

Các Trường Mẫu Đề xuất cho Kiến trúc sư Giải pháp: Dưới đây là một cấu trúc mẫu được đề xuất, kết hợp các thực tiễn tốt nhất và bổ sung các trường thông tin quan trọng cho SA trong bối cảnh tích hợp:

1. **Use Case ID:** Mã định danh duy nhất (ví dụ: UC-OB-01).
2. **Use Case Name:** Tên ngắn gọn, theo định dạng Động từ-Danh từ, hướng tới mục tiêu [24] (ví dụ: Onboard Khách hàng Mobile Banking Mới).
3. **Version/History:** Thông tin về phiên bản và lịch sử thay đổi (Người tạo, Ngày tạo, Người cập nhật cuối, Ngày cập nhật cuối).[24]
4. **Priority:** Mức độ ưu tiên nghiệp vụ (ví dụ: Cao, Trung bình, Thấp).[24]
5. **Frequency of Use:** Tần suất sử dụng ước tính (ví dụ: 1000 lượt/ngày).[24] Thông tin này hữu ích cho việc xác định NFR về hiệu năng và khả năng mở rộng.
6. **Actors:** Liệt kê *tất cả* các tác nhân tham gia, *nhấn mạnh việc xác định rõ các hệ thống* [11] (ví dụ: Khách hàng (Người dùng), Mobile App (Hệ thống), Backend (Hệ thống), Customer Platform (Hệ thống), Hệ thống eKYC (Hệ thống), Core Banking (Hệ thống)).
7. **Description/Goal:** Mô tả tóm tắt mục tiêu của use case.[16]

8. **Trigger:** Sự kiện khởi động use case [19] (ví dụ: Khách hàng nhấn nút 'Bắt đầu Onboarding' trên Mobile App).
9. **Preconditions:** Các điều kiện *phải đúng* trên các hệ thống liên quan *trước khi* use case bắt đầu. Cần mô tả cụ thể trạng thái của từng hệ thống.[16]
    - Ví dụ:
        1. Khách hàng đã cài đặt Mobile App phiên bản X.Y.Z.
        2. Dịch vụ Mobile Banking Backend đang hoạt động.
        3. Hệ thống Customer Platform sẵn sàng nhận yêu cầu.
        4. Hệ thống Core Banking sẵn sàng nhận yêu cầu.
        5. Hệ thống eKYC sẵn sàng nhận yêu cầu.
10. **Postconditions (Success):** Trạng thái của *tất cả* các hệ thống liên quan *sau khi* use case hoàn thành thành công.[16]
    - Ví dụ:
        1. Bản ghi khách hàng được tạo/cập nhật trong Customer Platform với trạng thái 'eKYC Verified'.
        2. Tài khoản thanh toán được tạo thành công trong Core Banking với số tài khoản ABC.
        3. Thông báo chào mừng đã được gửi tới khách hàng.
        4. Log kiểm toán (audit log) đã được ghi nhận tại Backend, Customer Platform, Core Banking.
11. **Postconditions (Failure):** Trạng thái của các hệ thống sau các kịch bản thất bại đã xác định.[24]
    - Ví dụ (Trường hợp eKYC thất bại):
        1. Bản ghi khách hàng trong Customer Platform được cập nhật trạng thái 'eKYC Failed'.
        2. Không có tài khoản nào được tạo trong Core Banking.
        3. Thông báo lỗi được gửi tới khách hàng.
        4. Log kiểm toán ghi nhận sự kiện thất bại.
12. **Main Success Scenario (Basic Flow):** Các bước được đánh số, mô tả Hành động Tác nhân / Phản hồi Hệ thống, nêu rõ tên hệ thống tương tác và API/thông điệp.[16]
13. **Alternative Flows:** Các phần được đánh số mô tả các luồng thay thế, liên kết với bước trong luồng chính.[16]
14. **Exception Flows:** Các phần được đánh số mô tả cách xử lý lỗi.[18]
15. **Includes/Extends:** Mối quan hệ với các use case khác (nếu có, thường ít dùng hơn trong các use case hệ thống chi tiết).[24]
16. **Interface References:** Liên kết hoặc tham chiếu rõ ràng đến tài liệu đặc tả giao diện chi tiết (ví dụ: link tài liệu API Swagger/OpenAPI, định nghĩa topic Kafka) cho mỗi tương tác hệ thống-hệ thống được đề cập trong các luồng. (Đáp ứng Q6 một phần, sẽ mở rộng ở Mục VII).

17. **Data Contract References:** Liên kết hoặc tham chiếu đến định nghĩa hợp đồng dữ liệu cho dữ liệu được trao đổi giữa các hệ thống. (Đáp ứng Q6 một phần, sẽ mở rộng ở Mục VII).
18. **Related Non-Functional Requirements:** Liệt kê hoặc tham chiếu đến các NFR chính ảnh hưởng đến use case này (ví dụ: Thời gian phản hồi tối đa cho cuộc gọi Core Banking, yêu cầu mã hóa dữ liệu PII).[35] (Đáp ứng Q7 một phần, sẽ mở rộng ở Mục VIII).
19. **Open Issues/Notes:** Các điểm cần xác định (TBDs), giả định, câu hỏi còn bỏ ngỏ.[24]

Bảng Ví dụ: Mẫu Use Case cho Onboarding (Minh họa)

Việc cung cấp một ví dụ cụ thể về cách điền thông tin vào mẫu này là rất hữu ích. Bảng dưới đây minh họa cấu trúc đề xuất và cách điền một số trường chính cho luồng thành công của use case "Onboard Khách hàng Mobile Banking Mới".

Trường Mẫu	**Ví dụ Nội dung (Minh họa)**
Use Case ID:	UC-OB-01
Use Case Name:	Onboard Khách hàng Mobile Banking Mới
Actors:	Khách hàng (Người dùng), Mobile App (Hệ thống), Backend (Hệ thống), Customer Platform (Hệ thống), Hệ thống eKYC (Hệ thống), Core Banking (Hệ thống)
Goal:	Khách hàng hoàn thành đăng ký và mở tài khoản thanh toán thành công qua Mobile App.
Preconditions:	1. Khách hàng đã cài Mobile App. 2. Backend, Customer Platform, eKYC, Core Banking sẵn sàng.
Postconditions (Success):	1. Bản ghi KH trong Customer Platform cập nhật 'eKYC Verified'. 2. Tài khoản tạo thành công trong Core Banking. 3. Thông báo chào mừng gửi đi. 4. Audit log ghi nhận.
Main Success Scenario:	...

	5. **Backend** gọi API getCustomerByContactInfo của **Customer Platform**.
	6. **Customer Platform** phản hồi 'Khách hàng mới'.
	7. **Backend** gọi API initiateVerification của **Hệ thống eKYC**.
	...
	12. **Backend** gọi API openAccount của **Core Banking**.
	13. **Core Banking** tạo TK và trả về số TK.
	...
Interface References:	- Customer Platform API: - eKYC API: [Link tới tài liệu] - Core Banking API: [Link tới đặc tả]
Data Contract References:	- Customer Data (Backend <-> CP): [Link tới Data Contract v1.2] - Account Creation Request (Backend <-> Core): [Link tới Data Contract v1.0]
Related NFRs:	- NFR-PERF-001: Thời gian phản hồi end-to-end < 1 phút. - NFR-SEC-005: Mã hóa PII khi truyền tải. - NFR-REL-002: Tỷ lệ thành công onboarding > 99.5%.

Bảng ví dụ này cung cấp một cái nhìn trực quan, cụ thể về cấu trúc được đề xuất, giúp SA dễ dàng áp dụng và hiểu cách tài liệu hóa các chi tiết quan trọng, đặc biệt là việc xác định các hệ thống là tác nhân, mô tả trạng thái liên hệ thống trong tiền/hậu điều kiện, và chỉ rõ các tương tác hệ thống trong luồng.

VI. Trực quan hóa Luồng: Sơ đồ Tuần tự UML (Đáp ứng Q5, phần thứ hai)

Mục đích của Sơ đồ Tuần tự trong Tích hợp:

Trong khi use case văn bản mô tả cái gì xảy ra và các bước tuần tự, Sơ đồ Tuần tự (Sequence Diagram) UML lại trực quan hóa cách thức các tương tác đó diễn ra theo thời gian giữa các đối tượng hoặc thực thể hệ thống cụ thể.14 Chúng đặc biệt hữu ích để:

- **Mô tả chi tiết luồng thông điệp:** Hiển thị rõ ràng chuỗi các thông điệp (ví dụ: lời gọi API, sự kiện) được gửi và nhận giữa các thành phần hệ thống trong một *kịch bản cụ thể* (luồng chính, luồng thay thế hoặc luồng ngoại lệ) của use case.[14] Một use case có thể được minh họa bằng nhiều sơ đồ tuần tự, mỗi sơ đồ cho một kịch bản quan trọng.
- **Xác thực logic điều phối:** Giúp SA hình dung và kiểm tra logic điều phối (orchestration) trong thành phần trung gian (như Mobile Banking Backend), đảm bảo các cuộc gọi đến hệ thống khác diễn ra đúng thứ tự và xử lý phản hồi chính xác.[14]
- **Xác định điểm nghẽn tiềm ẩn:** Việc nhìn thấy các cuộc gọi tuần tự, đặc biệt là các cuộc gọi đồng bộ kéo dài, có thể giúp phát hiện sớm các vấn đề về hiệu năng.
- **Giao tiếp hiệu quả:** Truyền đạt các tương tác phức tạp một cách trực quan, dễ hiểu hơn so với mô tả văn bản thuần túy, đặc biệt với đội ngũ kỹ thuật.[14]

Ánh xạ Kịch bản Use Case sang Sơ đồ Tuần tự:
Quá trình chuyển đổi từ mô tả use case sang sơ đồ tuần tự khá trực tiếp 14:

- **Tác nhân Hệ thống -> Đường sống (Lifeline):** Mỗi tác nhân là một hệ thống được xác định trong use case (ví dụ: Mobile Banking Backend, Customer Platform, Core Banking System) sẽ trở thành một đường sống (lifeline) trên sơ đồ tuần tự, biểu thị sự tồn tại của thành phần đó theo thời gian.[14]
- **Bước Tương tác -> Thông điệp (Message):** Mỗi bước trong luồng use case mô tả một tương tác giữa hai hệ thống (ví dụ: "Backend gọi API openAccount của Core Banking") sẽ được biểu diễn bằng một mũi tên thông điệp (message arrow) từ đường sống của hệ thống gửi đến đường sống của hệ thống nhận.[14]

Ký hiệu UML Sequence Diagram Quan trọng cho Kiến trúc sư:
SA cần nắm vững các ký hiệu sau để vẽ và đọc sơ đồ tuần tự hiệu quả:

- **Đường sống (Lifeline):** Biểu diễn các thực thể tham gia tương tác (thường là các thành phần hệ thống hoặc đối tượng). Được vẽ dưới dạng hộp chữ nhật chứa tên (ví dụ: :MobileBankingBackend, :CustomerPlatform) với đường nét đứt thẳng đứng đi xuống.[14] Tên có thể theo định dạng instanceName:ClassName hoặc chỉ ClassName nếu là vai trò hoặc thể hiện ẩn danh.[14]
- **Thông điệp (Message):**
    - **Đồng bộ (Synchronous):** Mũi tên liền, đầu mũi tên đặc (->). Biểu thị người gửi đợi phản hồi trước khi tiếp tục. Phù hợp cho các cuộc gọi API request-response.[14] Ví dụ: Backend gọi createAccount và chờ Core Banking trả về kết quả.

    - **Bất đồng bộ (Asynchronous):** Mũi tên liền, đầu mũi tên hở (->>). Biểu thị người gửi không đợi phản hồi và tiếp tục xử lý ngay. Phù hợp cho việc gửi sự kiện (event publishing) hoặc các lệnh không cần phản hồi tức thì.[43] Ví dụ: Backend gửi sự kiện "OnboardingCompleted" lên message queue.
    - **Thông điệp Trả về (Return):** Mũi tên nét đứt, đầu mũi tên hở (-->). Biểu thị dữ liệu hoặc tín hiệu trả về từ một cuộc gọi đồng bộ. Có thể bỏ qua nếu không cần thiết, nhưng hữu ích để hiển thị dữ liệu trả về quan trọng.[14]
    - **Thông điệp Tạo/Hủy (Create/Destroy):** Dùng để biểu diễn việc tạo hoặc hủy các đối tượng/thành phần một cách động trong quá trình tương tác (ít phổ biến hơn ở mức kiến trúc hệ thống cao).[43]
- **Thanh Kích hoạt (Activation Bar):** Hình chữ nhật hẹp trên đường sống, biểu thị khoảng thời gian mà đối tượng/hệ thống đang hoạt động (xử lý thông điệp).[43] Giúp hình dung luồng kiểm soát.
- **Mảnh Tương tác (Interaction Fragment):** Các hộp bao quanh một phần của sơ đồ để mô tả logic phức tạp. Rất quan trọng cho SA:
    - **alt (Alternative):** Mô hình hóa các điều kiện if-else. Dùng để biểu diễn các luồng thay thế hoặc xử lý lỗi dựa trên phản hồi hoặc dữ liệu.[14] Ví dụ: phân nhánh dựa trên verificationStatus là 'Success' hay 'Failed'.
    - **opt (Optional):** Mô hình hóa các bước chỉ xảy ra nếu một điều kiện nhất định đúng.[14]
    - **loop (Loop):** Mô hình hóa các hành động lặp đi lặp lại.[14] Ví dụ: lặp lại việc gọi API nếu gặp lỗi tạm thời.
    - **ref (Reference):** Cho phép tham chiếu đến một sơ đồ tuần tự khác, giúp chia nhỏ các sơ đồ phức tạp thành các phần dễ quản lý hơn.[14] Ví dụ: tham chiếu đến sơ đồ chi tiết "Xác minh eKYC".
- **Điều kiện Bảo vệ (Guard):** Biểu thức logic (Boolean) đặt trong ngoặc vuông `` trên một thông điệp, xác định điều kiện để thông điệp đó được gửi. Thường dùng kết hợp với các mảnh alt hoặc opt.[14] Ví dụ: createAccount().

Ví dụ: Sơ đồ Tuần tự cho Onboarding Happy Path (Đoạn trích)
Sơ đồ sau minh họa một phần luồng thành công chính, tập trung vào các tương tác giữa các hệ thống backend:

Đoạn mã

```
sequenceDiagram
    participant BE as Mobile Banking Backend
    participant CP as Customer Platform
```

```
    participant EK as eKYC System
    participant CB as Core Banking System

    BE->>+CP: checkCustomer(email)
    CP-->>-BE: customerStatus='New'

    BE->>+EK: initiateVerification(userData)
    Note right of EK: eKYC process occurs...
    EK-->>-BE: verificationStatus='Success'

    BE->>+CP: updateCustomerKYCStatus(userId, 'Verified')
    CP-->>-BE: status='Success'

    BE->>+CB: openAccount(customerDetails)
    CB-->>-BE: accountDetails={accountId: '12345'}

    Note over BE: Store results, confirm to Mobile App...
```

Việc chuyển đổi từ mô tả use case sang sơ đồ tuần tự không chỉ là một bài tập tài liệu hóa. Nó buộc SA phải đưa ra các quyết định kiến trúc rõ ràng mà có thể chỉ tiềm ẩn trong văn bản use case.[14] Ví dụ, khi vẽ sơ đồ, SA phải quyết định liệu cuộc gọi giữa Backend và Core Banking là đồng bộ hay bất đồng bộ.[43] Đây là một lựa chọn kiến trúc cơ bản ảnh hưởng đến hiệu năng, độ tin cậy và sự ghép nối (coupling) giữa các hệ thống.[29] Sơ đồ tuần tự làm cho lựa chọn này trở nên tường minh. Do đó, SA nên sử dụng việc vẽ sơ đồ tuần tự không chỉ để ghi lại mà còn là một công cụ thiết kế chủ động để xem xét và chỉ định các mẫu tương tác (sync/async), luồng xử lý lỗi (sử dụng mảnh alt), và các điểm nghẽn hiệu năng tiềm ẩn. Các ký hiệu được chọn phản ánh trực tiếp các quyết định kiến trúc về cách các hệ thống giao tiếp trong kịch bản đó.

Bảng Ví dụ: Ánh xạ Bước Use Case sang Thông điệp Sơ đồ Tuần tự

Để đảm bảo tính nhất quán và dễ theo dõi giữa mô tả use case văn bản và sơ đồ tuần tự trực quan, việc tạo ra một bảng ánh xạ là hữu ích.

Bước trong Luồng Use Case (Văn bản)	Thông điệp tương ứng trên Sơ đồ Tuần tự
5. Backend gọi API getCustomerByContactInfo của Customer Platform.	Thông điệp đồng bộ checkCustomer(email) từ đường sống :MobileBankingBackend đến đường sống :CustomerPlatform.

6. Customer Platform phản hồi 'Khách hàng mới'.	Thông điệp trả về customerStatus='New' từ :CustomerPlatform đến :MobileBankingBackend.
7. Backend gọi API initiateVerification của Hệ thống eKYC.	Thông điệp đồng bộ (hoặc bất đồng bộ tùy thiết kế) initiateVerification(userData) từ :MobileBankingBackend đến :eKYCSystem.
9. Hệ thống eKYC trả về trạng thái 'Thành công'.	Thông điệp trả về (hoặc callback/webhook) verificationStatus='Success' từ :eKYCSystem đến :MobileBankingBackend.
12. Backend gọi API openAccount của Core Banking.	Thông điệp đồng bộ openAccount(customerDetails) từ :MobileBankingBackend đến :CoreBankingSystem.
13. Core Banking tạo TK và trả về số TK.	Thông điệp trả về accountDetails={accountId: '...'} từ :CoreBankingSystem đến :MobileBankingBackend.

Bảng ánh xạ này tạo ra một liên kết rõ ràng, giúp người đọc hiểu cách sơ đồ thực thi logic của use case và giúp người tạo đảm bảo sơ đồ phản ánh chính xác use case, tăng cường sự rõ ràng và nhất quán giữa hai loại tài liệu.

VII. Định nghĩa các Điểm Chạm: Giao diện và Hợp đồng Dữ liệu (Đáp ứng Q6)

Nhu cầu về Sự Chính xác của Kiến trúc sư: Đối với việc tích hợp hệ thống, đặc biệt là trong ngành ngân hàng với yêu cầu cao về độ tin cậy và bảo mật, việc hiểu rõ các "đường nối" (seams) chính xác giữa các hệ thống là tối quan trọng. Các bước mô tả chức năng chung trong use case là chưa đủ; SA cần các định nghĩa giao diện (interface) chi tiết để thiết kế và triển khai tích hợp thành công.[4]

Tài liệu hóa Giao diện:
Mỗi tương tác hệ thống-hệ thống được xác định trong luồng use case và sơ đồ tuần tự đều ngụ ý sự tồn tại của một giao diện. Giao diện này có thể là:

- **API (Application Programming Interface):** Phổ biến nhất hiện nay, đặc biệt là RESTful APIs sử dụng JSON hoặc SOAP services sử dụng XML.
- **Message Queues/Topics:** Đối với giao tiếp bất đồng bộ (ví dụ: Kafka, RabbitMQ).
- **File Transfers:** Trong một số trường hợp, đặc biệt với các hệ thống cũ hoặc xử lý

batch.
- **Database Links/Direct Access:** Ít được khuyến khích do tạo ra sự ghép nối chặt, nhưng vẫn có thể tồn tại.

SA nên đảm bảo rằng use case tham chiếu đến tài liệu đặc tả giao diện chính thức. Điều này có thể được thực hiện bằng cách:

- Thêm trường "Interface References" vào mẫu use case, chứa liên kết đến tài liệu API (ví dụ: link tới file Swagger/OpenAPI cho REST API, WSDL cho SOAP, tài liệu định nghĩa topic/message cho queue).
- Chú thích trực tiếp trong các bước của luồng use case hoặc trên thông điệp của sơ đồ tuần tự tên của API/endpoint/message cụ thể được sử dụng.

Vai trò của các cổng API (API gateways) và nền tảng tích hợp (integration platforms) như MuleSoft cũng cần được xem xét trong kiến trúc tổng thể để quản lý, bảo mật và điều phối các giao diện này.[4]

Giới thiệu Hợp đồng Dữ liệu (Data Contracts):
Trong khi đặc tả giao diện định nghĩa cách các hệ thống nói chuyện với nhau (endpoints, methods, protocols), Hợp đồng Dữ liệu (Data Contract) định nghĩa nội dung của cuộc nói chuyện đó.

- **Định nghĩa:** Hợp đồng dữ liệu là một thỏa thuận chính thức, rõ ràng giữa bên sản xuất dữ liệu (data producer - ví dụ: Core Banking cung cấp thông tin tài khoản) và bên tiêu thụ dữ liệu (data consumer - ví dụ: Backend sử dụng thông tin đó), xác định cấu trúc, định dạng, ngữ nghĩa, chất lượng, các quy tắc truy cập và vòng đời của dữ liệu được trao đổi thông qua một giao diện.[54] Chúng mã hóa các kỳ vọng và loại bỏ sự không chắc chắn hoặc các giả định ngầm về dữ liệu.[55]
- **Tại sao quan trọng trong Tích hợp Ngân hàng:**
  - **Đảm bảo Tính nhất quán & Chất lượng:** Ngăn chặn lỗi do định dạng hoặc kiểu dữ liệu không khớp giữa các hệ thống.[55]
  - **Ngăn chặn Thay đổi Gây lỗi (Breaking Changes):** Cung cấp cơ chế quản lý phiên bản và thông báo thay đổi, giúp người tiêu thụ không bị ảnh hưởng bất ngờ.[55]
  - **Tạo dựng Niềm tin & Hợp tác:** Thiết lập sự hiểu biết chung và trách nhiệm rõ ràng giữa các nhóm phát triển/vận hành các hệ thống khác nhau.[55]
  - **Hỗ trợ Tuân thủ:** Giúp đảm bảo dữ liệu nhạy cảm (PII) được xử lý đúng cách theo quy định.[5]
  - **Tăng độ tin cậy Xử lý:** Dữ liệu đáng tin cậy là nền tảng cho các quy trình nghiệp vụ đáng tin cậy.[58]

Các Yếu tố Chính của Hợp đồng Dữ liệu (cho Kiến trúc sư):

Một hợp đồng dữ liệu hiệu quả nên bao gồm các yếu tố sau:

- **Schema (Lược đồ):** Định nghĩa cấu trúc dữ liệu: tên trường, kiểu dữ liệu (string, integer, boolean, date, object, array), các ràng buộc (bắt buộc/tùy chọn - nullable, độ dài, giá trị tối thiểu/tối đa, định dạng cụ thể như email), và mối quan hệ giữa các trường. Thường được biểu diễn bằng các định dạng chuẩn như JSON Schema, Avro, Protobuf.[54] *Ví dụ:* Định nghĩa trường accountStatus phải là một trong các giá trị 'ACTIVE', 'INACTIVE', 'PENDING'.
- **Semantics (Ngữ nghĩa):** Ý nghĩa nghiệp vụ của từng trường dữ liệu. Các quy tắc xác thực logic vượt ra ngoài kiểu dữ liệu cơ bản (ví dụ: mã quốc gia phải hợp lệ theo ISO 3166-1 alpha-2). Thông tin về nguồn gốc dữ liệu (data lineage).[55] *Ví dụ:* Trường transactionTimestamp luôn ở múi giờ UTC.
- **Data Quality / SLAs (Chất lượng Dữ liệu / Thỏa thuận Mức độ Dịch vụ):** Các chỉ số và ngưỡng chất lượng dữ liệu mong đợi: độ đầy đủ (completeness), độ chính xác (accuracy), tính kịp thời (timeliness), tần suất cập nhật.[54] *Ví dụ:* Dữ liệu số dư tài khoản phải được cập nhật trong vòng 5 phút so với hệ thống Core Banking; tỷ lệ trường customerId bị thiếu phải dưới 0.1%.
- **Governance/Security (Quản trị/Bảo mật):** Phân loại dữ liệu (ví dụ: Công khai, Nội bộ, Mật, PII), yêu cầu mã hóa (khi lưu trữ và truyền tải), phương thức kiểm soát truy cập (RBAC), các yêu cầu tuân thủ (GDPR, PCI-DSS).[55] *Ví dụ:* Trường nationalId phải được mã hóa khi lưu trữ và che dấu (masked) trong log.
- **Versioning & Change Management (Quản lý Phiên bản & Thay đổi):** Cách đánh số phiên bản cho hợp đồng, quy trình thông báo và xử lý các thay đổi (đặc biệt là các thay đổi không tương thích ngược - backward-incompatible), chính sách ngừng hỗ trợ (deprecation).[54] *Ví dụ:* Sử dụng Semantic Versioning; thông báo trước 30 ngày cho các thay đổi breaking.
- **Stakeholders/Ownership (Các bên liên quan/Sở hữu):** Xác định rõ ai là người sở hữu dữ liệu, ai chịu trách nhiệm duy trì hợp đồng.[54]

Tích hợp Hợp đồng Dữ liệu với Use Case:
Có nhiều cách để liên kết hợp đồng dữ liệu với use case:

1. **Tham chiếu:** Cách tiếp cận phổ biến và gọn gàng nhất là thêm trường "Data Contract References" vào mẫu use case, chứa liên kết đến tài liệu hoặc kho lưu trữ hợp đồng dữ liệu tập trung.
2. **Nhúng (Embedding):** Đối với các trao đổi dữ liệu cực kỳ quan trọng hoặc đơn giản, có thể trích dẫn các yếu tố chính của hợp đồng (ví dụ: các trường cốt lõi, định dạng mong đợi) trực tiếp trong mô tả luồng use case hoặc dưới dạng phụ lục.
3. **Chú thích Sơ đồ Tuần tự:** Hiển thị các tham số dữ liệu chính được truyền trong thông điệp trên sơ đồ tuần tự và chú thích tham chiếu đến hợp đồng dữ liệu đầy

đủ.

Trong khi use case xác định *luồng chức năng* và nhu cầu trao đổi dữ liệu (ví dụ: Backend cần lấy chi tiết khách hàng từ Customer Platform) [10], hợp đồng dữ liệu lại định nghĩa *thỏa thuận* chi tiết về chính dữ liệu đó.[54] Những thách thức trong tích hợp thường không nằm ở việc gọi API mà ở dữ liệu trả về không đúng định dạng, không đầy đủ, không chính xác hoặc có ngữ nghĩa bị hiểu sai.[4] Hợp đồng dữ liệu làm cho các giả định ngầm về dữ liệu trở nên rõ ràng và có thể thực thi được.[55] Do đó, SA phải thúc đẩy việc sử dụng hợp đồng dữ liệu song song với use case cho các luồng tích hợp quan trọng. Use case xác định *nhu cầu* trao đổi dữ liệu, còn hợp đồng dữ liệu chỉ định *quy tắc* cho việc trao đổi đó, đảm bảo tích hợp được xây dựng trên nền tảng dữ liệu đáng tin cậy, chứ không chỉ dựa vào tính đúng đắn về mặt chức năng. Điều này chủ động giải quyết các điểm lỗi tích hợp phổ biến liên quan đến dữ liệu.

VIII. Vượt ra ngoài Chức năng: Tích hợp Yêu cầu Phi Chức năng (NFRs) (Đáp ứng Q7)

Định nghĩa Yêu cầu Phi Chức năng (NFRs):
NFRs mô tả cách thức hệ thống nên hoạt động hoặc các thuộc tính chất lượng mà nó phải sở hữu, thay vì những gì nó làm (được mô tả bởi yêu cầu chức năng/use case).35 Chúng thường được gọi là các "-ilities" (ví dụ: performance, security, reliability, scalability, usability, maintainability). NFRs là yếu tố quyết định đến trải nghiệm người dùng, sự ổn định vận hành và khả năng đáp ứng mục tiêu kinh doanh.39
Tại sao NFRs Quan trọng với Kiến trúc sư:
NFRs có ảnh hưởng sâu sắc đến các quyết định thiết kế kiến trúc, bao gồm lựa chọn công nghệ, mẫu thiết kế (design patterns), kiến trúc tổng thể (architectural patterns) và cơ sở hạ tầng.40 Bỏ qua NFRs có thể dẫn đến một hệ thống hoạt động về mặt chức năng nhưng không đáp ứng được yêu cầu về hiệu năng, không thể mở rộng, không an toàn hoặc khó bảo trì.36
Các Loại NFR Chính cho Tích hợp Ngân hàng:
Trong bối cảnh tích hợp hệ thống ngân hàng, các loại NFR sau đây đặc biệt quan trọng:

- **Hiệu năng (Performance):**
  - *Mô tả:* Tốc độ phản hồi, thông lượng xử lý, độ trễ.
  - *Ví dụ:* Thời gian phản hồi của các cuộc gọi API giữa các hệ thống (ví dụ: Backend gọi Core Banking), số lượng giao dịch onboarding có thể xử lý mỗi giờ, thời gian tải giao diện trên Mobile App.[35]
  - *NFR cụ thể:* "Cuộc gọi API từ Mobile Backend đến Core Banking để tạo tài khoản phải hoàn thành dưới 500ms trong 95% trường hợp dưới tải cao điểm (định nghĩa là 100 phiên onboarding đồng thời)." [39]
- **Bảo mật (Security):**
  - *Mô tả:* Xác thực giữa các hệ thống, mã hóa dữ liệu (lưu trữ và truyền tải, đặc biệt là PII), tuân thủ quy định (GDPR, PCI-DSS, luật pháp địa phương), ghi log

kiểm toán (audit logging), phòng chống tấn công.[4]
    - *NFR cụ thể:* "Tất cả dữ liệu định danh cá nhân (PII) của khách hàng trao đổi giữa Mobile Backend và Customer Platform phải được mã hóa bằng TLS 1.2 trở lên khi truyền tải và AES-256 khi lưu trữ." [39]
- **Độ tin cậy/Sẵn sàng (Reliability/Availability):**
    - *Mô tả:* Thời gian hoạt động của hệ thống (uptime), thời gian trung bình giữa các lỗi (MTBF), thời gian trung bình để phục hồi (MTTR), khả năng chịu lỗi.[35]
    - *NFR cụ thể:* "Quy trình onboarding end-to-end phải có độ sẵn sàng 99.9% trong giờ làm việc (8:00 - 18:00, Thứ Hai - Thứ Sáu)." [35]
- **Khả năng Mở rộng (Scalability):**
    - *Mô tả:* Khả năng của hệ thống xử lý khối lượng công việc tăng lên (số lượng người dùng, giao dịch) mà không làm giảm hiệu năng, thông qua việc mở rộng theo chiều dọc (thêm tài nguyên) hoặc chiều ngang (thêm node).[4]
    - *NFR cụ thể:* "Hệ thống onboarding phải có khả năng mở rộng theo chiều ngang để xử lý lượng người dùng đồng thời tăng 50% mỗi năm mà thời gian phản hồi API trung bình không tăng quá 10%." [37]
- **Khả năng Bảo trì (Maintainability):**
    - *Mô tả:* Mức độ dễ dàng trong việc sửa lỗi, cập nhật, nâng cấp hoặc thay đổi các thành phần của hệ thống.[35]
    - *NFR cụ thể:* "Thời gian để triển khai một bản vá lỗi bảo mật quan trọng cho Mobile Banking Backend không được quá 4 giờ làm việc."
- **Khả năng Sử dụng (Usability) (Đối với Mobile App):**
    - *Mô tả:* Mức độ dễ dàng và trực quan cho khách hàng khi sử dụng ứng dụng để hoàn thành quy trình onboarding.[35]
    - *NFR cụ thể:* "Người dùng mới phải có thể hoàn thành luồng onboarding thành công chính trong vòng dưới 5 phút mà không cần trợ giúp." [35]

Phương pháp Tài liệu hóa NFRs cùng với Use Case:
Việc liên kết NFRs với các use case cụ thể giúp đảm bảo rằng các yêu cầu chất lượng được xem xét trong ngữ cảnh chức năng mà chúng ảnh hưởng:

1. **Phần NFR riêng trong Mẫu Use Case:** Thêm một trường "Related Non-Functional Requirements" vào mẫu use case (như đã đề xuất ở Mục V). Trường này liệt kê các mã NFR hoặc mô tả ngắn gọn các NFR chính liên quan trực tiếp đến use case đó.
2. **Chú thích trong Luồng Use Case:** Gắn các ràng buộc NFR trực tiếp vào các bước có liên quan trong mô tả luồng. Ví dụ: "Bước 12: Backend gọi API openAccount của Core Banking... **".
3. **Tài liệu NFR Riêng biệt:** Duy trì một tài liệu NFR tập trung, được phân loại theo các thuộc tính chất lượng. Sau đó, từ mỗi use case, tham chiếu đến các NFR liên

quan trong tài liệu này.[35] Đây là cách tiếp cận phổ biến, giúp quản lý NFRs một cách có hệ thống.

4. **NFRs như Hậu điều kiện:** Một số NFR có thể được diễn đạt dưới dạng các hậu điều kiện có thể đo lường được. Ví dụ: "Hậu điều kiện: Mục log kiểm toán được tạo tuân thủ tiêu chuẩn **."

Đảm bảo NFRs có thể Đo lường được:
Điều cốt yếu là NFRs phải cụ thể, định lượng được và có thể kiểm thử. Tránh các thuật ngữ mơ hồ như "nhanh", "an toàn", "dễ sử dụng". Thay vào đó, hãy sử dụng các chỉ số rõ ràng: thời gian phản hồi tính bằng giây hoặc mili giây, thông lượng tính bằng giao dịch/giây, tỷ lệ phần trăm thời gian hoạt động, tuân thủ các tiêu chuẩn cụ thể (WCAG 2.1, HIPAA, ISO 27001), số lần nhấp chuột tối đa để hoàn thành tác vụ.35
Use case mô tả các tương tác chức năng [10], trong khi NFRs xác định các thuộc tính chất lượng cần thiết của các tương tác đó hoặc của hệ thống thực hiện chúng.[35] NFRs không tồn tại độc lập với chức năng; chúng là các ràng buộc hoặc mục tiêu chất lượng được *áp dụng cho* chức năng được mô tả bởi use case.[39] Ví dụ, use case "Onboard Khách hàng" cần phải diễn ra, nhưng NFRs quy định rằng nó phải diễn ra một cách an toàn, đáng tin cậy và trong một khung thời gian nhất định. Do đó, SA phải phân tích các use case *qua lăng kính* của NFRs. Đối với mỗi tương tác quan trọng được mô tả trong use case (đặc biệt là các cuộc gọi liên hệ thống), SA phải xem xét các yêu cầu về hiệu năng, bảo mật, độ tin cậy và khả năng mở rộng liên quan. Việc tài liệu hóa NFRs *trong ngữ cảnh* của use case giúp đảm bảo rằng kiến trúc được thiết kế để đáp ứng đồng thời cả mục tiêu chức năng và mục tiêu chất lượng.

IX. Thực tiễn Tốt nhất cho Use Case Tích hợp Ngân hàng (Đáp ứng Q8)

Dựa trên các phân tích và ví dụ trên, dưới đây là tổng hợp các thực tiễn tốt nhất (best practices) mà SA nên áp dụng khi viết use case cho các dự án tích hợp hệ thống phức tạp trong ngành ngân hàng:

1. **Rõ ràng và Chính xác:** Sử dụng ngôn ngữ không mơ hồ. Định nghĩa các thuật ngữ nghiệp vụ quan trọng (ví dụ: "eKYC Verified", "Khách hàng Hiện hữu") trong một bảng chú giải (glossary) chung.[35] Phân định rõ ràng trách nhiệm của từng hệ thống trong mỗi bước tương tác.
2. **Tập trung vào Giá trị và Mục tiêu:** Đảm bảo mỗi use case mang lại giá trị có thể quan sát được cho một tác nhân và đạt được một mục tiêu nghiệp vụ rõ ràng.[10] Tránh tạo các use case quá kỹ thuật hoặc bị phân mảnh chỉ mô tả các thao tác CRUD đơn lẻ, trừ khi chúng thực sự đại diện cho một mục tiêu người dùng riêng biệt.[22]
3. **Mức độ Chi tiết Phù hợp:** Cân bằng giữa việc cung cấp cái nhìn tổng quan và đi

vào chi tiết cần thiết cho thiết kế. Sử dụng các kỹ thuật như tham chiếu (ref trong sơ đồ tuần tự) hoặc mối quan hệ include (sử dụng cẩn thận [28]) để quản lý sự phức tạp, chia nhỏ các quy trình lớn thành các use case dễ quản lý hơn.[12]

4. **Mô hình hóa Tác nhân Hệ thống Rõ ràng:** Luôn xác định và bao gồm các hệ thống tương tác như là các tác nhân trong use case và các đường sống trong sơ đồ tuần tự đối với các kịch bản tích hợp.[11]
5. **Chi tiết hóa Giao diện và Dữ liệu:** Không dừng lại ở luồng chức năng. Chỉ định hoặc tham chiếu đến các chi tiết API và Hợp đồng Dữ liệu cho giao tiếp liên hệ thống. Đây là thông tin sống còn cho việc tích hợp.[4]
6. **Tích hợp NFRs theo Ngữ cảnh:** Đảm bảo các thuộc tính chất lượng (hiệu năng, bảo mật, độ tin cậy...) được xem xét và tài liệu hóa cùng với các bước chức năng mà chúng ảnh hưởng.[37]
7. **Trực quan hóa bằng Sơ đồ Tuần tự:** Bổ sung use case văn bản bằng sơ đồ tuần tự cho các kịch bản chính hoặc phức tạp để làm rõ logic tương tác và các quyết định kiến trúc.[14]
8. **Xem xét Đặc thù Ngân hàng:**
    - **Quy định & Tuân thủ:** Đảm bảo use case bao hàm (một cách tường minh hoặc ngầm định) các bước cần thiết cho tuân thủ quy định (KYC, AML, báo cáo...).[2] NFRs về bảo mật và khả năng kiểm toán (auditability) là tối quan trọng.
    - **Hệ thống Legacy:** Thừa nhận các ràng buộc do hệ thống lõi cũ gây ra. Use case và kiến trúc có thể cần phải tính đến các hạn chế về API, định dạng dữ liệu, hoặc khả năng xử lý giao dịch.[4]
    - **Độ nhạy cảm Dữ liệu:** Đặc biệt chú ý đến dữ liệu PII và dữ liệu tài chính nhạy cảm trong các luồng, định nghĩa giao diện, hợp đồng dữ liệu và NFRs bảo mật.[5]
    - **Khả năng Phục hồi & Xử lý Lỗi:** Hệ thống ngân hàng đòi hỏi độ tin cậy rất cao. Mô tả chi tiết việc xử lý ngoại lệ, cơ chế dự phòng (fallback) và khả năng phục hồi.[29] Cân nhắc sử dụng các mẫu bất đồng bộ để tăng khả năng phục hồi và giảm ghép nối.[29]
9. **Xem xét và Lặp lại:** Use case là tài liệu sống. Cần được xem xét bởi các bên liên quan (nghiệp vụ, kỹ thuật, bảo mật, tuân thủ) và được tinh chỉnh liên tục trong suốt quá trình thiết kế và phát triển.[8]

Hướng dẫn viết use case tiêu chuẩn thường tập trung nhiều vào mục tiêu của tác nhân và các bước chức năng.[10] Tuy nhiên, tích hợp ngân hàng liên quan đến việc quản lý trạng thái phức tạp trên các hệ thống phân tán và phụ thuộc rất nhiều vào các giao diện được xác định rõ ràng.[1] Các lỗi tích hợp thường xuất phát từ các vấn đề về giao diện, dữ liệu hoặc trạng thái không nhất quán giữa các hệ thống.[4] Do đó, đối với SA

thiết kế tích hợp, các khía cạnh quan trọng nhất của tài liệu use case trở thành việc *đặc tả chi tiết các giao diện liên hệ thống* (API, thông điệp, hợp đồng dữ liệu) và *định nghĩa chính xác trạng thái hệ thống* trong tiền điều kiện, hậu điều kiện (cho cả thành công và thất bại), và các luồng ngoại lệ.[24] Điều này cung cấp thông tin trực tiếp cho việc thiết kế các cơ chế kiến trúc cần thiết để đảm bảo giao tiếp đáng tin cậy và tính nhất quán trạng thái (ví dụ: quản lý giao dịch phân tán, giao dịch bù trừ, event sourcing). SA nên dành nỗ lực đáng kể trong tài liệu use case để xác định tỉ mỉ các "đường nối" giữa các hệ thống và các chuyển đổi trạng thái dự kiến trên tất cả các hệ thống liên quan, đặc biệt là trong các điều kiện lỗi. Trọng tâm này chuyển dịch nhẹ từ mô tả thuần túy lấy người dùng làm trung tâm sang góc nhìn tập trung hơn vào tương tác hệ thống và quản lý trạng thái, điều này là cực kỳ quan trọng để thiết kế các giải pháp tích hợp mạnh mẽ và đáng tin cậy trong ngành ngân hàng.

X. Kết luận

Việc xây dựng các use case chất lượng cao, tập trung vào kiến trúc là một khoản đầu tư quan trọng đối với các Kiến trúc sư Giải pháp làm việc trong môi trường tích hợp hệ thống ngân hàng phức tạp. Chúng không chỉ đơn thuần là tài liệu yêu cầu mà còn là nền tảng cho việc thiết kế, giao tiếp và triển khai thành công các giải pháp công nghệ đáp ứng nhu cầu nghiệp vụ ngày càng khắt khe.

Báo cáo này đã trình bày một phương pháp tiếp cận toàn diện để viết use case hiệu quả, nhấn mạnh các yếu tố then chốt bao gồm:

- **Nền tảng vững chắc:** Hiểu rõ định nghĩa, mục đích và các thành phần cốt lõi của use case.
- **Xác định rõ ràng:** Nhận diện chính xác các tác nhân (bao gồm cả hệ thống) và định nghĩa ranh giới hệ thống một cách tường minh trong bối cảnh tích hợp.
- **Mô tả chi tiết:** Tài liệu hóa cẩn thận các luồng tương tác (chính, thay thế, ngoại lệ), đặc biệt tập trung vào việc mô tả rõ ràng các giao tiếp liên hệ thống.
- **Cấu trúc và Trực quan hóa:** Sử dụng các mẫu use case chuẩn hóa, phù hợp với nhu cầu kiến trúc và bổ sung bằng Sơ đồ Tuần tự UML để trực quan hóa các tương tác phức tạp.
- **Định nghĩa Điểm chạm:** Chú trọng đặc tả các giao diện và Hợp đồng Dữ liệu để đảm bảo sự rõ ràng và tin cậy trong trao đổi thông tin giữa các hệ thống.
- **Tích hợp Yêu cầu Chất lượng:** Lồng ghép các Yêu cầu Phi Chức năng (NFRs) vào ngữ cảnh của use case để đảm bảo các khía cạnh về hiệu năng, bảo mật, độ tin cậy được xem xét ngay từ đầu.
- **Tuân thủ Thực tiễn Ngân hàng:** Áp dụng các kinh nghiệm và lưu ý đặc thù của ngành tài chính - ngân hàng, đặc biệt là về quy định, bảo mật và quản lý trạng

thái.

Bằng cách áp dụng các nguyên tắc và kỹ thuật được trình bày, Kiến trúc sư Giải pháp có thể tạo ra các use case không chỉ đáp ứng yêu cầu chức năng mà còn thực sự hướng dẫn việc xây dựng các kiến trúc tích hợp mạnh mẽ, linh hoạt và đáng tin cậy. Đầu tư thời gian và công sức vào việc tạo ra các use case chất lượng cao sẽ giúp giảm thiểu sự mơ hồ, cải thiện giao tiếp giữa các nhóm, định hướng thiết kế vững chắc và cuối cùng góp phần vào sự thành công của các dự án chuyển đổi số phức tạp trong ngành ngân hàng.

Nguồn trích dẫn

1. Real Time Customer Onboarding Solution for Banks - Maveric Systems, truy cập vào tháng 4 24, 2025, https://maveric-systems.com/real-time-onboarding-solutions/
2. Successful Customer Onboarding in Banks - ServiceNow Blog, truy cập vào tháng 4 24, 2025, https://www.servicenow.com/uk/blogs/2022/successful-customer-onboarding-banks
3. Automate user on-boarding for financial services with a digital assistant powered by Amazon Bedrock | AWS Machine Learning Blog, truy cập vào tháng 4 24, 2025, https://aws.amazon.com/blogs/machine-learning/automate-user-on-boarding-for-financial-services-with-a-digital-assistant-powered-by-amazon-bedrock/
4. Understanding Integration Architecture: A Comprehensive Guide - Ardoq, truy cập vào tháng 4 24, 2025, https://www.ardoq.com/knowledge-hub/integration-architecture
5. Solution Architecture 101: The Essential Guide For SMBs, Mid-Market, And Enterprise Businesses - Euvic, truy cập vào tháng 4 24, 2025, https://www.euvic.com/us/post/solution-architecture-guide/
6. Client onboarding for banks in Portugal: The BPI bank case - Veridas, truy cập vào tháng 4 24, 2025, https://veridas.com/en/digital-onboarding-identity-as-a-catalyst-in-banking/
7. MuleSoft Accelerator for Financial Services - Use case 1 - Core banking foundation, truy cập vào tháng 4 24, 2025, https://www.mulesoft.com/exchange/org.mule.examples/mulesoft-accelerator-for-financial-services/minor/1.6/pages/Use%20case%201%20-%20Core%20banking%20foundation/?no_navbar=1&no_cookie=1&embedded=1&is_marketing=1
8. What is Solution Architecture: Process, Solution Architect - AltexSoft, truy cập vào tháng 4 24, 2025, https://www.altexsoft.com/blog/solution-architecture/
9. Part 2: Solution Design — Design Process of Modern IT Systems and Large Software Integration Projects - SoftwareDominos, truy cập vào tháng 4 24, 2025, https://softwaredominos.com/home/software-design-development-articles/solution-design-of-large-system-integration-projects/
10. Use case - Wikipedia, truy cập vào tháng 4 24, 2025,

https://en.wikipedia.org/wiki/Use_case
11. Use Case Diagram – Unified Modeling Language (UML) | GeeksforGeeks, truy cập vào tháng 4 24, 2025, https://www.geeksforgeeks.org/use-case-diagram/
12. Use-case diagrams in UML modeling - IBM, truy cập vào tháng 4 24, 2025, https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case
13. Use Case Definition | Ivar Jacobson International, truy cập vào tháng 4 24, 2025, https://www.ivarjacobson.com/publications/white-papers-articles/use-case-definition
14. Explore the UML sequence diagram - IBM Developer, truy cập vào tháng 4 24, 2025, https://developer.ibm.com/articles/the-sequence-diagram/
15. Building a delightful digital onboarding experience for consumer and SME banking customers - Plumery, truy cập vào tháng 4 24, 2025, https://plumery.com/building-a-delightful-digital-onboarding-experience-for-consumer-and-sme-banking-customers/
16. What is a Use Case? Definition & Examples - Dovetail, truy cập vào tháng 4 24, 2025, https://dovetail.com/product-development/what-is-a-use-case/
17. Designing Use Cases for a Project | GeeksforGeeks, truy cập vào tháng 4 24, 2025, https://www.geeksforgeeks.org/designing-use-cases-for-a-project/
18. Use Cases: Diagram & Examples (Updated 2024) - Inflectra Corporation, truy cập vào tháng 4 24, 2025, https://www.inflectra.com/Ideas/Topic/Use-Cases.aspx
19. Use case template: Downloadable example & how to write one - LogRocket Blog, truy cập vào tháng 4 24, 2025, https://blog.logrocket.com/product-management/what-is-a-use-case-template-how-to-write/
20. What is a use case and how to write one - Wrike, truy cập vào tháng 4 24, 2025, https://www.wrike.com/blog/what-is-a-use-case/
21. Use Cases in a Nutshell | GEOG 468: GIS Analysis and Design - Dutton Institute, truy cập vào tháng 4 24, 2025, https://www.e-education.psu.edu/geog468/l8_p3.html
22. Use Cases: Best Practices, truy cập vào tháng 4 24, 2025, https://www.eg.bucknell.edu/~cs475/F04-S05/useCases.pdf
23. Use Case Diagram Best Practices and Examples - Justinmind, truy cập vào tháng 4 24, 2025, https://www.justinmind.com/blog/use-case-diagramm-beispiele/
24. Use Case Template - Profinit, truy cập vào tháng 4 24, 2025, https://profinit.eu/wp-content/uploads/2016/03/use_case_template.doc
25. Use-case template - IBM, truy cập vào tháng 4 24, 2025, https://www.ibm.com/docs/en/imdm/11.6?topic=cases-use-case-template
26. Design Use Cases · CSE 110 Software Engineering, truy cập vào tháng 4 24, 2025, http://ieng6.ucsd.edu/~cs110x/static/artifacts/DesignUseCases.html
27. How to Write a Use Case: Template + Tutorial - Bridging the Gap, truy cập vào tháng 4 24, 2025, https://www.bridging-the-gap.com/what-is-a-use-case/
28. Use case scenario based on a use case using "uses" or "extends" - Software Engineering Stack Exchange, truy cập vào tháng 4 24, 2025, https://softwareengineering.stackexchange.com/questions/432945/use-case-scenario-based-on-a-use-case-using-uses-or-extends

29. Defining Integration Best Practices: Template Based - SAP Learning, truy cập vào tháng 4 24, 2025, https://learning.sap.com/learning-journeys/becoming-an-sap-btp-solution-architect/defining-integration-best-practices-template-based
30. Case studies - Moody's, truy cập vào tháng 4 24, 2025, https://www.moodys.com/web/en/us/kyc/resources/case-studies.html
31. MuleSoft Accelerator for Financial Services - Use case 1a - Unlock any core, truy cập vào tháng 4 24, 2025, https://www.mulesoft.com/exchange/org.mule.examples/mulesoft-accelerator-for-financial-services/minor/1.8/pages/a5h-00q/Use%20case%201a%20-%20Unlock%20any%20core/?no_navbar=1&no_cookie=1&embedded=1&is_marketing=1
32. Driving Successful Software Development with the Use Case Driven ..., truy cập vào tháng 4 24, 2025, https://guides.visual-paradigm.com/driving-successful-software-development-with-the-use-case-driven-approach-real-life-templates-and-examples/
33. User Stories vs Use Cases - Key Difference & Examples - UXCam, truy cập vào tháng 4 24, 2025, https://uxcam.com/blog/user-stories-vs-use-cases/
34. Enterprise Application Integration Best Practices Unveiled - DBSync, truy cập vào tháng 4 24, 2025, https://www.mydbsync.com/blogs/application-integration-best-practices
35. NFRs: What is Non Functional Requirements (Example & Types) - BrowserStack, truy cập vào tháng 4 24, 2025, https://www.browserstack.com/guide/non-functional-requirements-examples
36. Non-Functional Requirements Examples: a Full Guide - testomat.io, truy cập vào tháng 4 24, 2025, https://testomat.io/blog/non-functional-requirements-examples-definition-complete-guide/
37. How to Write Non-Functional Requirements in 6 Steps, truy cập vào tháng 4 24, 2025, https://www.modernrequirements.com/blogs/what-are-non-functional-requirements-and-how-to-build-them/
38. Non-Functional Requirements Examples and Templates: A Guide to Defining Performance, Security, and Scalability, truy cập vào tháng 4 24, 2025, https://www.businessanalyststoolkit.com/non-functional-requirements-examples-and-templates/
39. Non-functional Requirements: What They Do, Examples, and Best Practices - Perforce, truy cập vào tháng 4 24, 2025, https://www.perforce.com/blog/alm/what-are-non-functional-requirements-examples
40. Nonfunctional Requirements: Examples, Types and Approaches - AltexSoft, truy cập vào tháng 4 24, 2025, https://www.altexsoft.com/blog/non-functional-requirements/
41. Non-Functional Requirements (with Examples) - Khalil Stemmler, truy cập vào tháng 4 24, 2025, https://khalilstemmler.com/articles/object-oriented/analysis/non-functional-requir

ements/
42. Non-Functional Requirements (NFRs) for Performance Testing - Radview, truy cập vào tháng 4 24, 2025, https://www.radview.com/blog/non-functional-requirements-nfrs-for-performance-testing/
43. UML 2 Tutorial - Sequence Diagram - Sparx Systems, truy cập vào tháng 4 24, 2025, https://sparxsystems.com/resources/tutorials/uml2/sequence-diagram.html
44. UML Sequence Diagram Tutorial | Lucidchart, truy cập vào tháng 4 24, 2025, https://www.lucidchart.com/pages/uml-sequence-diagram
45. Sequence Diagrams – Unified Modeling Language (UML) | GeeksforGeeks, truy cập vào tháng 4 24, 2025, https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/
46. Sequence Diagram vs Usecase Diagram - uml - Stack Overflow, truy cập vào tháng 4 24, 2025, https://stackoverflow.com/questions/32518752/sequence-diagram-vs-usecase-diagram
47. Practical Examples of Sequence Diagrams to Replace BPMN in Business Process Modeling, truy cập vào tháng 4 24, 2025, https://zenuml.com/blog/2024/05/19/2024/practical-examples-sequence-diagrams-replace-bpmn-business-process-modeling/
48. Creating Sequence Diagrams with a Use Case-Driven Approach: A ..., truy cập vào tháng 4 24, 2025, https://guides.visual-paradigm.com/creating-sequence-diagrams-with-a-use-case-driven-approach-a-comprehensive-guide/
49. A single sequence diagram for the whole system? - Software Engineering Stack Exchange, truy cập vào tháng 4 24, 2025, https://softwareengineering.stackexchange.com/questions/433422/a-single-sequence-diagram-for-the-whole-system
50. How to create a sequence diagram - Gleek, truy cập vào tháng 4 24, 2025, https://www.gleek.io/sequence
51. How to Draw a Sequence Diagram (+Examples) | ClickUp, truy cập vào tháng 4 24, 2025, https://clickup.com/blog/sequence-diagram-examples/
52. How to show "if" condition on a sequence diagram? - Stack Overflow, truy cập vào tháng 4 24, 2025, https://stackoverflow.com/questions/8114770/how-to-show-if-condition-on-a-sequence-diagram
53. System Integration Best Practices – Quick Start Guide - Codeless Platforms, truy cập vào tháng 4 24, 2025, https://www.codelessplatforms.com/blog/system-integration-best-practices/
54. Data Contracts 101 - ProfitOptics, truy cập vào tháng 4 24, 2025, https://www.profitoptics.com/blog/data-contracts-101
55. Data Contracts 101: Importance, Validations & Best Practices - Atlan, truy cập vào tháng 4 24, 2025, https://atlan.com/data-contracts/
56. Data mesh round table discussion on data contracts, truy cập vào tháng 4 24,

2025, https://datameshlearning.com/blog/data-mesh-round-table-discussion-on-data-contracts/
57. Understanding Data Contracts and Their Role in Data Management - Airbyte, truy cập vào tháng 4 24, 2025, https://airbyte.com/data-engineering-resources/data-contracts
58. Understanding the Fundamentals of Data Contracts | ACL Digital, truy cập vào tháng 4 24, 2025, https://www.acldigital.com/blogs/fundamentals-data-contracts
59. Data Contracts: 7 Critical Implementation Lessons Learned - Monte Carlo Data, truy cập vào tháng 4 24, 2025, https://www.montecarlodata.com/blog-data-contracts/
60. A Comprehensive Guide to Contract Testing APIs in a Service Oriented Architecture, truy cập vào tháng 4 24, 2025, https://lirantal.com/blog/a-comprehensive-guide-to-contract-testing-apis-in-a-service-oriented-architecture-5695ccf9ac5a
61. What Are Data Contracts? A Beginner Guide with Examples - DataCamp, truy cập vào tháng 4 24, 2025, https://www.datacamp.com/blog/data-contracts