BÀI TẬP 60 PHÚT

Monolith, Microservices, Data Ownership, API Gateway & BFF

Môn: Software Architecture

Họ và tên	Lê Chí Hoàng Long	**MSSV**	23120294
Lớp	CQ2023/3	**Ngày**	28/09/2026

Mục tiêu của bài tập: hiểu bản chất kiến trúc thông qua một tình huống thực tế. Không có một đáp án duy nhất. Điểm chủ yếu đến từ lập luận, trade-off, quyết định và tính nhất quán của toàn bài.

Cơ sở nội dung: slide 26–35 của tài liệu Product Development, tập trung vào Monolith vs Microservices, domain/data ownership, coupling, direct client-to-microservice, API Gateway và BFF.

1. Learning Objectives

- Phân biệt Monolith và Microservices theo boundary, deployment và responsibility thay vì theo "độ hiện đại" của công nghệ.
- Nhận ra rằng tách source code chưa đủ tạo ra service độc lập nếu vẫn dùng chung data/schema.
- Phát hiện coupling qua database, API contract và client biết trực tiếp topology nội bộ.
- Phân tích khi nào direct client-to-microservice trở thành vấn đề.
- Phân biệt vai trò của domain service, API Gateway và Backend-for-Frontend (BFF).
- Đưa ra quyết định kiến trúc theo chuỗi: Problem → Architecture Choice → Trade-off → Decision → Justification.

2. Bối cảnh bài toán — UniRide

UniRide là ứng dụng đặt xe nội bộ cho một trường đại học lớn. Hệ thống có Student Mobile App và Operations Web. Backend hiện tại là một application duy nhất, được chia logic thành bốn module: Passenger, Driver, Trip và Payment. Toàn hệ thống dùng chung một PostgreSQL database khoảng 26 tables; một số module đang đọc/ghi trực tiếp dữ liệu của module khác.

Dữ liệu vận hành

- Khoảng 20.000 active users/ngày.
- Giờ cao điểm khoảng 700 requests/second.
- Backend team: 7 developers.
- Trip thay đổi gần như mỗi tuần; Driver thay đổi ít.
- Mỗi thay đổi backend hiện phải deploy toàn application.
- Dự kiến 12 tháng tới: 100.000 users/ngày, nhiều product squad hơn, mobile và Operations Web phát triển độc lập hơn.

Các đề xuất trong team

Nhân vật	Đề xuất
Developer A	"Chia ngay Passenger, Driver, Trip và Payment thành bốn microservices."
Developer B	"Giữ modular monolith trước, sửa boundary cho sạch rồi mới tách khi cần."
Developer C	"Tách thành microservices nhưng cho cả bốn service dùng chung database để dễ query."

Nhân vật	Đề xuất
Frontend team	"Mobile và Web gọi trực tiếp từng microservice."

Vai trò của bạn: Software Architect của UniRide. Bạn phải đưa ra một quyết định có lý do, không chỉ chọn pattern.

3. Đề bài 60 phút

Phần	Nội dung	Thời gian	Điểm
A	Identify the real problem	8 phút	10
B	Monolith hay Microservices?	12 phút	20
C	Domain boundary, Data Ownership & Coupling	15 phút	25
D	Direct Client, API Gateway hay BFF?	15 phút	25
E	Final Architecture Decision	7 phút	15
F	AI Usage Log	3 phút	5

Mọi phần trả lời phải thể hiện được: Problem → Architecture Choice → Trade-off → Decision → Justification.

PHẦN A — Identify the Real Problem (8 phút — 10 điểm)

CEO nói: *"User sẽ tăng 5 lần, vậy chúng ta phải chuyển sang microservices."*

A1. Xác định 3 vấn đề kiến trúc thực sự mà UniRide đang có hoặc sắp có. Điền bảng:

Problem	Evidence từ case	Tại sao đây là architecture problem?
1. Boundary giữa các domain chưa độc lập	Passenger, Driver, Trip và Payment nằm trong một application và một số module đọc/ghi trực tiếp dữ liệu của module khác.	Thay đổi domain có thể ảnh hưởng đến domain khác. Điều này làm tăng coupling và khiến việc thay đổi/deploy khó độc lập
2. Chia sẻ database tạo coupling dữ liệu	Toàn hệ thống dùng chung database khoảng 26 tables; nhiều module có thể đọc/ghi dữ liệu module khác.	Nếu nhiều service cùng phụ thuộc vào schema/table của nhau thì việc tách source code thành service vẫn chưa tạo ra ownership độc lập. Thay đổi schema có thể ảnh hưởng nhiều thành phần.
3. Một thay đổi backend phải deploy toàn bộ application	Mỗi thay đổi backend hiện phải deploy toàn bộ application; Trip thay đổi gần như mỗi tuần trong khi Driver thay đổi ít.	Deployment boundary không phù hợp với tốc độ thay đổi của các domain. Trip có nhu cầu release thường xuyên nhưng vẫn bị ràng buộc bởi toàn application.

A2. Chọn một vấn đề quan trọng nhất và trả lời tối đa 4 câu: *"Nếu chưa biết từ microservices, bạn vẫn mô tả vấn đề này như thế nào?"*

- Vấn đề quan trọng nhất là các domain đang phụ thuộc lẫn nhau quá nhiều thông qua application và database. Khi một domain thay đổi, các domain khác có thể bị ảnh hưởng vì chúng dùng chung dữ liệu và schema. Điều này đặc biệt rõ với Trip, vì Trip thay đổi gần như mỗi tuần nhưng hiện tại mỗi thay đổi phải deploy lại toàn bộ application. Vì vậy, vấn đề cốt lõi là cần tạo ra boundary và ownership rõ hơn giữa các domain.

PHẦN B — Monolith hay Microservices? (12 phút — 20 điểm)

Bạn được chọn một trong ba hướng:

- **M1 — Keep Modular Monolith:** Một deployable application, nhưng boundary giữa Passenger / Driver / Trip / Payment phải rõ.
- **M2 — Partial Extraction:** Chỉ tách một hoặc hai domain cần độc lập trước.

- **M3 — Full Microservices:** Tách các domain thành independently deployable services.

B1. Chọn M1, M2 hoặc M3 cho UniRide tại thời điểm hiện tại.

B2. Nêu 2 lợi ích, 2 chi phí/rủi ro do chính lựa chọn tạo ra, và 1 alternative bạn đã cân nhắc nhưng không chọn.

B3. Hoàn thành: "Tôi sẽ thay đổi quyết định trên nếu _____ xảy ra, bởi vì _____."

Không được viết: "Microservices tốt hơn vì scalable." Muốn được điểm phải chỉ rõ scalable phần nào, giải quyết problem nào, đổi lại chi phí gì.

Bài làm:

B1. Lựa chọn: Em chọn M2 - Partial Extraction: Chỉ tách một hoặc hai domain cần độc lập trước. Trong case này, Trip là ứng viên ưu tiên nhất. Vì Trip thay đổi gần như mỗi tuần.

B2. Nêu lợi ích và rủi ro:

- Lợi ích:
  - Trip có thể deploy độc lập: Vì trip thay đổi gần như mỗi tuần nên nếu Trip được extraction thành service riêng thì Trip có thể deploy độc lập chứ không cần phải deploy lại toàn bộ hệ thống.
  - Có thể xây dựng boundary trước khi mở rộng thành microservice: Team hiện tại chỉ có 7 thành viên, vì vậy việc lập trúc vận hành 4 microservice sẽ tạo thêm nhiều operational overhead.
- Chi phí/rủi ro:
  - Distribute-system complexity: Sau khi trip thành service riêng, giao tiếp giữa Trip và các domain đi qua API/network. Dẫn đến các vấn đề như: network failure, timeout, retry, latency, API contract.
  - Data consistency phức tạp hơn: Vì Trip không còn dùng database trực tiếp như các domain khác, việc lấy các thông tin khác phải qua API/contract hoặc 1 cơ chế khác dẫn đến phức tạp hơn khi shared database. Tuy nhiên, đổi lại làm ownership rõ hơn.
- Alternative: M1 — Keep Modular Monolith: M1 có ưu điểm là đơn giản hơn để vận hành và phù hợp với team 7 developers. Tuy nhiên, mình không chọn M1 hoàn toàn vì vấn đề deployment coupling đã tồn tại: mỗi thay đổi backend phải deploy toàn application, trong khi Trip thay đổi gần như hàng tuần.

B3. Tôi sẽ thay đổi quyết định trên nếu các domain trong modular monolith có boundary ổn định nhưng nhu cầu deploy độc lập, scale độc lập và tốc độ thay đổi tăng đáng kể, bởi vì khi đó chi phí distributed system của microservices có thể được bù lại bởi nhu cầu độc lập giữa các domain

PHẦN C — Domain Boundary, Data Ownership & Coupling (15 phút — 25 điểm)

Một team implement bốn service: Passenger Service, Driver Service, Trip Service, Payment Service; nhưng cả bốn đều có quyền SELECT / INSERT / UPDATE trực tiếp vào cùng 26 tables.

C1. Ownership map — 8 điểm

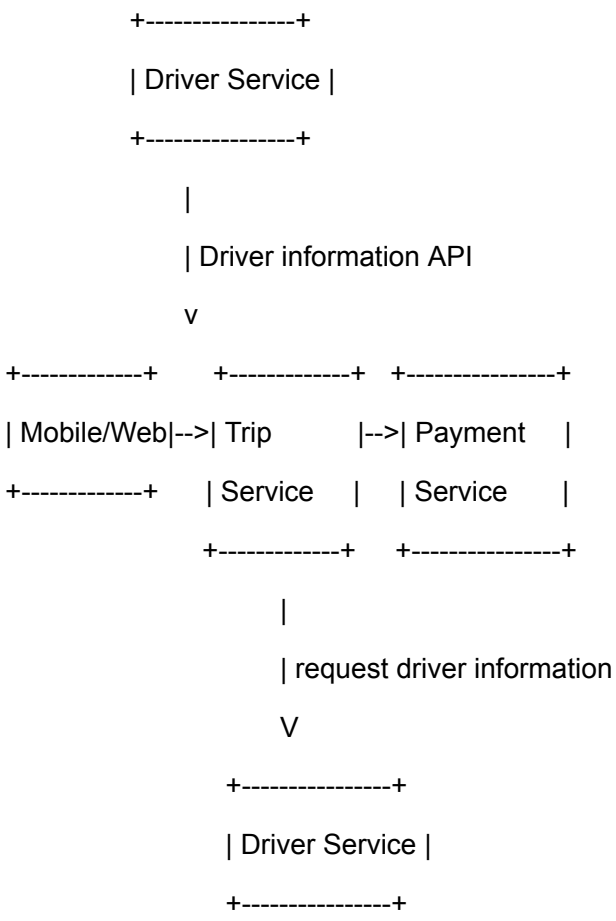
Gán mỗi loại dữ liệu cho một owner. Bạn được quyền thay đổi boundary nếu thấy bốn service trên chưa hợp lý.

Dữ liệu	Owner bạn chọn	Lý do ngắn
Passenger	Passenger	Đây là dữ liệu cốt lõi của passenger domain
PassengerContact	Passenger	Thông tin liên hệ thuộc passenger domain và được quản lý cùng Passenger

Dữ liệu	Owner bạn chọn	Lý do ngắn
Driver	Driver	Driver service chịu trách nhiệm về thông tin tài xế.
DriverAvailability	Driver	Availability phản ánh trạng thái hoạt động của driver
Trip	Trip	Trip là trung tâm của quá trình đặt và thực hiện chuyến đi.
TripStatus	Trip	Trạng thái ACCEPTED → STARTED là business rule của Trip.
Fare	Trip	Fare liên quan trực tiếp đến trip và yêu cầu tính giá của chuyến đi
Payment	Payment	Payment service chịu trách nhiệm về payment domain
PaymentTransaction	Payment	Transaction tài chính phải được sở hữu bởi payment domain.

C2. Cross-domain requirement — 7 điểm

Trip cần hiển thị driverName, driverRating và paymentStatus. Thiết kế cách Trip có được thông tin cần thiết mà bạn cho là phù hợp. Không cần code. Chỉ cần vẽ luồng và ghi điều gì bên A được phép biết về B.



```
[ ? ] -------- ? --------> [ ? ]
```

C3. Schema change thought experiment — 5 điểm

Team Driver muốn đổi driver_name thành first_name + last_name. Giải thích kiến trúc bạn thiết kế ở C1–C2 ảnh hưởng như thế nào đến số thành phần phải thay đổi. Không chỉ viết "coupling thấp hơn/cao hơn"; phải chỉ ra coupling nằm ở đâu.

nếu Driver đổi driver_name thành first_name + last_name, với shared database thì các service đang truy cập trực tiếp bảng Driver cũng phải thay đổi theo, tạo coupling qua database schema. Với data ownership, chỉ Driver Service quản lý schema của Driver; các service khác chỉ giao tiếp qua API contract, nên thay đổi bên trong Driver chủ yếu chỉ cần sửa Driver Service nếu contract bên ngoài vẫn giữ nguyê

C4. Trade-off — 5 điểm

Nêu một lợi ích của data ownership và một cái giá phải trả khi không còn shared database thuận tiện.

Lợi ích

ata ownership giúp mỗi domain có trách nhiệm rõ ràng với dữ liệu của mình. Một domain có thể thay đổi internal schema mà không bắt buộc tất cả domain khác phải truy cập hoặc sửa database trực tiếp.

Cái giá phải trả

Không còn shared database thuận tiện nghĩa là những query cross-domain trước đây rất đơn giản sẽ trở thành API calls hoặc cần một cơ chế dữ liệu khác

PHẦN D — Direct Client, API Gateway hay BFF? (15 phút — 25 điểm)

```
Mobile --------> Passenger Service
       --------> Driver Service
       --------> Trip Service
       --------> Payment Service

Web -----------> Passenger Service
      ----------> Driver Service
      ----------> Trip Service
      ----------> Payment Service
```

Student Mobile Home cần: Current trip, Driver summary, Estimated fare, Payment status. Mạng mobile đôi khi yếu.

Operations Web cần: Detailed trip, Full driver information, Payment exceptions, Advanced filtering.

D1. Compare — 8 điểm

Choice	Một lợi ích trong UniRide	Một risk/cost trong UniRide
Direct Client → Services	Client có thể gọi trực tiếp service cần thiết, không cần thêm gateway layer.	Mobile/Web phải biết topology backend và phải thực hiện nhiều request. Khi service thay đổi, client contract/topology cũng dễ bị ảnh hưởng. Mobile mạng yếu càng dễ bị ảnh hưởng bởi nhiều network calls.

Single API Gateway	Có một entry point cho client, tập trung routing và authentication/token handling.	Mobile và Web có requirements khác nhau nên gateway có thể trở thành nơi chứa nhiều logic đặc thù cho từng client.
BFF Mobile + BFF Web	Mỗi client có API contract phù hợp với nhu cầu riêng; có thể aggregate dữ liệu cho từng UI.	Có thêm hai backend layer cần maintain, test và deploy. Logic có thể bị duplicate nếu không quản lý boundary tốt.

D2. Decide — 5 điểm

Chọn cách bạn muốn dùng và vẽ đường request cho: (1) Student Mobile Home; (2) Operations Web.

Chọn BFF Mobile + BFF Web.

Student Mobile Home:
Student Mobile → Mobile BFF → Trip / Driver / Payment Services
Operations Web:
Operations Web → Web BFF → Trip / Driver / Payment Services

Lý do: Mobile và Web có nhu cầu dữ liệu khác nhau, nên mỗi BFF có thể tổng hợp và trả về API phù hợp với từng client. Mobile cũng giảm số request khi mạng yếu.

D3. Gateway responsibility — 7 điểm

Responsibility	Đặt ở đâu?	Vì sao?
A. Routing request	**BFF / Gateway layer**	Routing là concern của entry point, không phải business domain.
B. Authentication / token handling	**BFF / Gateway layer**	Đây là cross-cutting concern dùng để kiểm tra authentication trước khi request vào domain services.
C. Combining responses needed by one UI	**BFF**	Đây chính là nhu cầu aggregation theo từng client. Mobile và Web có response requirements khác nhau.
D. Calculating trip fare	**Trip Service**	Fare thuộc Trip domain theo ownership đã xác định ở C1.
E. Rules for Trip: ACCEPTED → STARTED	**Trip Service**	Đây là business rule của Trip và phải nằm trong domain service, không nên nằm trong gateway/BFF.

F. Deciding whether a payment can be refunded	**Payment Service**	Refund là business rule thuộc Payment domain.

D4. BFF question — 5 điểm

Mobile và Operations Web dùng cùng Trip Service. Tại sao chúng vẫn có thể cần API contract khác nhau? Trả lời bằng một ví dụ cụ thể từ UniRide, không viết định nghĩa BFF.

Mobile và Operations Web cùng sử dụng Trip Service nhưng vẫn có thể cần API contract khác nhau vì hai client có nhu cầu dữ liệu khác nhau.

Ví dụ Mobile Home chỉ cần:

- Current trip
- Driver summary
- Estimated fare
- Payment status

trong khi Operations Web cần:

- Detailed trip
- Full driver information
- Payment exceptions
- Advanced filtering

Nếu cả hai client phải gọi trực tiếp cùng một API generic của Trip Service, Mobile có thể phải nhận nhiều dữ liệu không cần thiết hoặc thực hiện thêm request. BFF cho phép mỗi client có contract phù hợp với UI của mình. Requirements này được nêu trực tiếp trong đề.

PHẦN E — Final Architecture Decision (7 phút — 15 điểm)

E1. Vẽ kiến trúc cuối cùng — 7 điểm

Sơ đồ bắt buộc có: Clients → Entry point(s), nếu có → Modules/Services → Data ownership. Dùng tối đa 12 boxes. Không cần UML chuẩn.

[Student Mobile] ----HTTP----> [Mobile BFF]

[Operations Web] ----HTTP----> [Web BFF]

[Mobile BFF] ----HTTP----> [Trip Service]

[Web BFF] ----HTTP----> [Trip Service]

[Trip Service] ----HTTP----> [Driver Module]

[Trip Service] ----HTTP----> [Payment Module]

[Passenger Module] ----owns----> [Passenger Data]

[Driver Module] ----owns----> [Driver Data]

[Trip Service] ----owns----> [Trip Data]

[Payment Module] ----owns----> [Payment Data]

```
[Component A] ----HTTP----> [Component B]
      |
      +----owns----> [Data X]
```

E2. Architecture Decision Record mini — 8 điểm

Problem

UniRide hiện có coupling giữa các domain thông qua shared application và shared PostgreSQL database. Mọi backend change phải deploy toàn application, trong khi Trip thay đổi gần như hàng tuần và hệ thống dự kiến tăng từ khoảng 20.000 lên 100.000 users/ngày.

Architecture Choice

Chọn M2 — Partial Extraction, trước mắt tách Trip thành independently deployable service và giữ Passenger, Driver, Payment trong modular structure với data ownership rõ ràng. Sử dụng Mobile BFF và Web BFF làm entry points cho hai loại client.

Trade-off accepted

Chấp nhận thêm network communication, API contracts, monitoring và distributed-system complexity để đổi lấy khả năng deploy Trip độc lập và giảm coupling qua shared database.

Decision

Không chuyển toàn bộ bốn domain thành microservices ngay lập tức. Trước tiên thiết lập domain boundaries và data ownership, sau đó extraction Trip vì đây là domain có tốc độ thay đổi cao nhất. Client không gọi trực tiếp từng domain service; Mobile và Web đi qua BFF tương ứng.

Justification

Quyết định này giải quyết vấn đề thực tế hiện tại thay vì giả định rằng tăng users tự động có nghĩa là phải dùng full microservices. Nó cũng phù hợp với team 7 developers và cho phép UniRide tăng mức độ phân tách dần khi nhu cầu deploy/scale độc lập xuất hiện.

Biggest architecture risk I am knowingly accepting

Rủi ro lớn nhất là complexity của distributed communication sau khi Trip được tách ra. Trip phải giao tiếp với các domain khác thông qua API/contract thay vì shared database, nên timeout, network failure, latency và contract changes trở thành những vấn đề phải quản lý.

PHẦN F — AI Usage Log (3 phút — 5 điểm)

Nếu không dùng AI, ghi: “AI not used.” Nếu có dùng, ghi tối đa 5 entries.

Tôi hỏi AI	Tôi dùng AI để làm gì?	Tôi hiểu được gì bằng lời của mình?
“What is data ownership in microservices?”	Hiểu khái niệm data ownership	Mỗi domain/service nên có owner rõ ràng cho dữ liệu của mình; service khác không nên sửa trực tiếp database của owner.
“What is the difference between API Gateway and BFF?”	Phân biệt Gateway và BFF	Gateway tập trung các concern ở entry point, còn BFF cho phép tạo API contract phù hợp với từng loại client.
“Why is shared database coupling in microservices?”	Hiểu database coupling	Tách code thành nhiều service nhưng cùng sửa một schema vẫn khiến các service phụ thuộc nhau.
“What is direct client-to-microservice communication?”	Hiểu vấn đề của direct client	Client phải biết nhiều service và topology backend, khiến client coupling và số network requests tăng.
“What is data ownership?”	Kiểm tra cách phân chia owner	Mỗi domain chịu trách nhiệm với dữ liệu của mình và service khác nên tương tác thông qua contract thay vì database trực tiếp.

4. Quy định sử dụng AI

Được phép

- Tra cứu/giải thích keyword hoặc khái niệm chưa hiểu, ví dụ: data ownership, bounded context, coupling, API Gateway, BFF, independent deployment.
- Yêu cầu AI đưa ví dụ đơn giản để hiểu khái niệm.

- Yêu cầu AI so sánh hai thuật ngữ ở mức khái niệm.

Không được phép

- Yêu cầu AI giải toàn bộ bài UniRide.
- Yêu cầu AI quyết định toàn bộ kiến trúc thay cho bạn.
- Yêu cầu AI viết toàn bộ phần A–F hoặc tạo nguyên bài nộp.
- Copy câu trả lời AI mà không thể giải thích lại bằng lời của mình.

Nguyên tắc: AI = reference / explainer, không phải decision maker. Bạn chịu trách nhiệm về architecture choice, trade-off, diagram và justification.

Cơ chế chấm để hạn chế copy AI

- Điểm cao yêu cầu các phần A–E cross-consistent với nhau.
- Nếu ownership ở C mâu thuẫn với responsibility ở D hoặc sơ đồ ở E, bài sẽ mất điểm dù từng câu nghe có vẻ hợp lý.
- Bài bắt buộc có: alternative bị loại, trade-off chấp nhận, điều kiện khiến đổi quyết định và biggest remaining risk.

5. Rubric chấm điểm

Hạng mục	Điểm	Tiêu chí điểm cao
A. Problem framing	10	Phân biệt problem với solution; dùng evidence từ case; chỉ ra vấn đề kiến trúc thực sự.
B. Monolith vs Microservices	20	Choice phù hợp assumptions; có trade-offs thực chất; có alternative và condition làm đổi decision.
C. Domain & Data Ownership	25	Boundary/ownership có lý do; nhận diện coupling qua shared data/schema; interaction nhất quán với ownership.
D. Client/Gateway/BFF	25	Nhìn ra client coupling/chattiness; so sánh theo mobile/web requirements; phân bố responsibility có lập luận.
E. Final Decision	15	Diagram và ADR nhất quán với A–D; không tự mâu thuẫn; nêu được risk đang chấp nhận.
F. AI Usage Log	5	Trung thực, ngắn gọn; ghi câu hỏi/keyword và kiến thức tự diễn giải.
TỔNG	**100**	

Nguyên tắc chấm quan trọng: Không có điểm cho việc chỉ dùng đúng thuật ngữ. Ví dụ “BFF reduces coupling” chỉ có giá trị khi bạn chỉ ra coupling giữa ai với ai, dependency nào thay đổi và chi phí mới là gì.

6. Nguồn tham khảo uy tín

Nguồn chính của đề: slide 26–35 trong tài liệu Product Development được cung cấp. Các nguồn dưới đây dùng để kiểm chứng/đọc thêm; không phải là “đáp án mẫu” của bài tập.

- **Microsoft .NET Microservices — Direct client-to-microservice vs API Gateway**
  https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern
- **Microsoft .NET Microservices — Data sovereignty per microservice**
  https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/data-sovereignty-per-microservice
- **Azure Architecture Center — Data considerations for microservices**
  https://learn.microsoft.com/en-us/azure/architecture/microservices/design/data-considerations
- **Martin Fowler — Microservices**
  https://martinfowler.com/articles/microservices.html
- **Martin Fowler — Monolith First**
  https://martinfowler.com/bliki/MonolithFirst.html
- **AWS Prescriptive Guidance — API integration / Backend-for-Frontend**
  https://docs.aws.amazon.com/prescriptive-guidance/latest/micro-frontends-aws/api-integration-data-fetching.html

- **Google Cloud Architecture Center — Scalable and resilient apps**
  https://cloud.google.com/architecture/scalable-and-resilient-apps

Kết thúc — Hãy ưu tiên lập luận kiến trúc hơn việc gọi đúng tên pattern.