SOFTWARE ARCHITECTURE / PRODUCT DEVELOPMENT

BÀI TẬP PHÂN TÍCH KIẾN TRÚC HỆ THỐNG

Case Study: CampusQ&A tăng trưởng từ startup nhỏ đến hệ thống có tải cao

Thời gian		**Tổng điểm**	10 điểm
Họ và tên	Lê Chí Hoàng Long	**MSSV**	23120294

1. Mục tiêu học tập

- Nhận diện đúng bottleneck/vấn đề kỹ thuật từ dữ kiện thay vì đoán theo buzzword.
- Giải thích nguyên nhân kỹ thuật, so sánh phương án và phân tích trade-off.
- Đưa ra quyết định kiến trúc phù hợp với constraint về tải, team, ngân sách và availability.
- Bảo vệ quyết định bằng evidence cụ thể; nhận diện assumption, rủi ro và điều kiện khiến quyết định phải thay đổi.
- Thực hành tư duy xuyên suốt: Problem → Evidence → Alternatives → Trade-off → Decision → Risk.

2. Quy định sử dụng AI

ĐƯỢC PHÉP	KHÔNG ĐƯỢC PHÉP
• Hỏi nghĩa thuật ngữ/kỹ thuật. • Hỏi ví dụ đơn giản để hiểu khái niệm. • Tra cứu ý nghĩa của Scale-out, replication, CDN, cache, index, v.v.	• Đưa nguyên case vào AI để yêu cầu giải toàn bộ. • Yêu cầu AI chọn kiến trúc/ra quyết định thay mình. • Copy nguyên câu trả lời do AI tạo hoặc dùng AI viết toàn bộ phần phân tích.

Lưu ý chấm điểm: Không có điểm thưởng cho việc dùng nhiều công nghệ. Một giải pháp đơn giản nhưng bám sát evidence và trade-off có thể đạt điểm cao hơn một kiến trúc phức tạp không cần thiết.

3. Case Study — CampusQ&A

Bối cảnh. CampusQ&A là nền tảng hỏi–đáp và chia sẻ tài liệu cho sinh viên. Khi mới ra mắt, hệ thống chỉ có khoảng 500 người dùng/ngày và được xây như một monolithic web application. Sau một năm, lượng người dùng tăng mạnh. Nhóm phát triển muốn cải thiện performance và availability trong 6 tuần tới, nhưng không muốn biến hệ thống thành một "showcase công nghệ" khó vận hành.

Kiến trúc hiện tại. Browser (SPA/CSR) → 01 Application Server → 01 Relational Database Server. Static resources (JS/CSS/images) cũng được phục vụ từ Application Server. Chưa có Load Balancer, CDN, external cache, database replica hoặc Kubernetes.

4. Dữ liệu và constraints của hệ thống

Dữ kiện	Giá trị hiện tại	Tại sao con số này có thể quan trọng?
Người dùng	35.000 DAU; peak 1.200 concurrent users	Quy mô tải và mức tăng trưởng
Traffic API	Peak 350 requests/second; 80% read, 20% write	Ảnh hưởng scale/caching/database
Traffic spike	2 lần/tuần, kéo dài ~15 phút; peak có thể tăng 4×	Ảnh hưởng capacity và elasticity
Static resources	62% tổng request; 74% tổng số byte trả về	Ảnh hưởng client delivery/CDN
Client load	92 HTTP requests; 3,8 MB cho first load; median 3,8 s Wi-Fi, 7,1 s 4G	Phân biệt client-side vs server-side bottleneck
API latency	p95 = 420 ms off-peak; p95 = 1,7 s lúc peak	Dấu hiệu degradation theo tải
Application Server	8 vCPU / 16 GB RAM; CPU 92% lúc peak; RAM 58%	Scale-up/scale-out app tier
Database Server	8 vCPU / 32 GB RAM SSD; CPU 61%; disk I/O 68% lúc peak	Không tự động kết luận DB là bottleneck
Hot query	Query danh sách câu hỏi theo tag chiếm ~38% DB time; scan ~1,4 triệu rows; chưa có composite index phù hợp	Index/query plan
Hot reads	3 endpoint đọc chiếm 52% API traffic; chấp nhận dữ liệu cũ tối đa 60 giây	Cache feasibility
Writes	Đăng bài/answer/vote phải ghi bền vững; nội dung mới cần thấy gần như ngay	Consistency constraint
Deployment	4 lần/tuần; restart app gây 2–4 phút 503	Availability và rolling deployment
Availability	Mục tiêu 99,9%/tháng; hiện app hoặc DB hỏng đều gây outage	Single point of failure
Recovery	Backup DB mỗi đêm; restore thử nghiệm mất ~2 giờ	RPO/RTO và HA
Team	6 dev (4 full-stack/backend, 1 frontend, 1 QA); không có SRE/DevOps chuyên trách	Operational complexity
Ngân sách	Tăng tối đa 900 USD/tháng trong 6 tháng	Giới hạn giải pháp
Tăng trưởng	Dự báo traffic trung bình tăng ~3× trong 6 tháng	Target architecture horizon

Nguyên tắc: Không giả định rằng CDN, Redis, replica, microservices, Kubernetes hoặc NoSQL đều cần thiết. Mỗi thành phần được thêm vào phải gắn với một vấn đề cụ thể trong case.

5. Nhiệm vụ

Phân bổ gợi ý: 5 phút đọc case + 65 phút làm 6 nhiệm vụ. Trả lời ngắn gọn, ưu tiên reasoning hơn văn phong.

Nhiệm vụ 1 — Diagnose: Vấn đề thật sự nằm ở đâu?

Xác định tối đa 4 vấn đề kỹ thuật đáng ưu tiên trong hệ thống hiện tại và xếp hạng 3 vấn đề cần xử lý trước. Với mỗi vấn đề, hãy chỉ ra: (a) tầng bị ảnh hưởng — Client / Application / Database / Reliability; (b) ít nhất 01 dữ kiện cụ thể trong case; (c) quan hệ nguyên nhân → hệ quả; (d) 01 thông tin còn thiếu hoặc assumption có thể làm bạn thay đổi thứ tự ưu tiên.

Không được trả lời chỉ bằng tên công nghệ hoặc định nghĩa.

Vấn đề 1 (ưu tiên 1): Application Server quá tải

a. Tầng bị ảnh hưởng: Application
b. Dữ liệu cụ thể:
   - 350 requests/s lúc peak.
   - CPU Application Server lên 92% lúc peak.
   - API p95 tăng từ 420 ms off-peak → 1,7 s peak.
   - Peak có thể tăng 4×.
   - Chỉ có 01 Application Server.
c. Quan hệ nguyên nhân → hệ quả: CPU gần bão hòa → không còn headroom xử lý request khi traffic tăng → p95 latency nhảy từ 420 ms (off-peak) lên 1,7 s (peak); khi traffic spike 4× (2 lần/tuần), server gần như chắc chắn nghẽn hoàn toàn.
d. Thông tin còn thiếu: Chưa biết CPU bị chiếm bởi phần nào (business logic, serialization hay việc phục vụ static resource ngay trên app server). Nếu phần lớn CPU dùng để serve static files, thứ tự ưu tiên sẽ đổi sang vấn đề 4 (static resource) thay vì scale app server.

Vấn đề 2 (ưu tiên 2): Hot query thiếu composite index

a. Tầng bị ảnh hưởng: Database
b. Dữ liệu cụ thể: Query danh sách câu hỏi theo tag chiếm ~38% DB time, scan ~1,4 triệu rows, chưa có composite index phù hợp.
c. Quan hệ nguyên nhân → hệ quả: Thiếu composite index → DB phải scan nhiều rows → query tốn DB time → API phải chờ DB → latency API tăng.
d. Thông tin còn thiếu: Tần suất gọi thực tế của query này trong 350 req/s peak. Nếu đây là 1 trong 3 hot-read endpoint chiếm 52% traffic, việc thêm cache có thể quan trọng hơn việc thêm index.

Vấn đề 3 (ưu tiên 3): Không có Load Balancer, single Application Server là SPOF

a. Tầng bị ảnh hưởng: Reliability
b. Dữ liệu cụ thể: Chỉ có 1 Application Server; deploy 4 lần/tuần gây restart, mỗi lần 2–4 phút trả 503; mục tiêu availability 99,9%/tháng nhưng app hoặc DB hỏng đều gây outage toàn bộ.
c. Quan hệ nguyên nhân → hệ quả: Không có redundancy ở tầng app → mọi lần deploy hoặc crash đều là downtime trực tiếp cho toàn bộ user, không có cách nào rolling-update mà không mất traffic.
d. Thông tin còn thiếu: Ứng dụng hiện có đang lưu state (session, upload tạm...) trực tiếp trên app server hay không. Nếu có, phải giải quyết statelessness trước khi scale-out phía sau LB, nếu không quyết định ưu tiên sẽ đổi thành "chuyển session ra ngoài" trước khi thêm LB.

Vấn đề 4:

a. Tầng bị ảnh hưởng: Client/Application
b. Dữ liệu cụ thể:
   - Static resources chiếm 62% tổng request.
   - Chiếm 74% tổng số byte trả về.
   - JS/CSS/images đều được Application Server phục vụ.
   - First load có 92 HTTP requests và 3,8 MB.
c. Quan hệ nguyên nhân → hệ quả: Application Server vừa phải xử lý logic vừa phải phục vụ file tĩnh → tốn băng thông + một phần CPU/IO không cần thiết, góp phần vào tình trạng CPU 92%.
d. Thông tin còn thiếu:
   - static assets có immutable/versioned hay không;
   - cache-control hiện tại;
   - người dùng phân bố địa lý thế nào;
   - CDN latency thực tế.

Nhiệm vụ 2 — Compare & Decide: Scale Up hay Scale Out?

Đưa ra quyết định riêng cho Application Server và Database trong 4–6 tuần tới. Với mỗi tầng, so sánh ít nhất hai phương án (ví dụ Scale Up vs Scale Out hoặc tối ưu trước khi scale), sau đó chọn phương án phù hợp với dữ liệu, team, ngân sách và availability hiện tại.

Bắt buộc nêu: Evidence → Reasoning → Trade-off; đồng thời chỉ ra khi nào phương án còn lại sẽ trở nên hợp lý hơn.

Application Server

	Scale Up (nâng cấp server hiện tại)	Scale Out (LB + nhiều instance)
Evidence	CPU 92% peak, RAM chỉ 58% → vẫn có thể tăng vCPU trong ngắn hạn	CPU 92% peak + là SPOF + deploy gây 2–4 phút 503
Reasoning	Đơn giản, không cần đổi kiến trúc, triển khai nhanh trong 4–6 tuần	Giải quyết đồng thời cả bài toán capacity (traffic dự báo tăng 3× trong 6 tháng) lẫn availability/rolling deploy
Trade-off	Vẫn là SPOF, vẫn crash toàn bộ khi deploy, chi phí server lớn hơn tăng phi tuyến	Cần app stateless (rủi ro kỹ thuật), tăng độ phức tạp vận hành cho team không có SRE

Quyết định: Chọn Scale Out (thêm Load Balancer + tối thiểu 2 Application Server instance) cho 4–6 tuần tới.

- Evidence: CPU đã gần bão hòa (92%), hệ thống là SPOF, deploy gây downtime thật (503 lặp lại), traffic dự báo tăng ~3× trong 6 tháng, ngân sách cho phép tăng tối đa 900 USD/tháng — đủ cho 1 LB nhỏ + 1 app instance thêm, rẻ hơn nhiều so với việc liên tục nâng cấp lên instance CPU lớn hơn.
- Reasoning: Scale out là lựa chọn duy nhất giải quyết luôn cả 2 vấn đề ưu tiên #1 và #3 cùng lúc trong cùng một ngân sách.
- Trade-off: Phải đảm bảo app server không lưu state cục bộ (session, file tạm); nếu hiện tại có, team phải tốn thời gian chuyển session ra cache/DB dùng chung trước.
- Risk: Nếu chưa kiểm tra kỹ statelessness, scale-out có thể gây lỗi (user bị mất session khi LB route sang instance khác).
- Điều kiện chuyển sang Scale Up: Nếu app đang gắn chặt với state cục bộ và không thể tách trong 4–6 tuần, nên scale-up làm giải pháp tạm thời để giảm áp lực ngay, rồi làm scale-out sau khi refactor statelessness.

Database

	Tối ưu trước khi scale (composite index)	Scale Out (Read Replica)
Evidence	CPU DB chỉ 61% (chưa bão hòa); hot query cụ thể đã xác định được (38% DB time, scan 1,4 triệu rows) thiếu index	80% traffic là read; traffic dự báo tăng 3× trong 6 tháng
Reasoning	Sửa đúng nguyên nhân gốc, chi phí gần như 0, triển khai nhanh	Tăng khả năng chịu tải read trong dài hạn
Trade-off	Không giải quyết SPOF của DB, không chuẩn bị cho tăng trưởng 3×	Thêm replication lag — rủi ro với yêu cầu "nội dung mới cần thấy gần như ngay"; tốn thêm chi phí vận hành

Quyết định: Chọn tối ưu trước (thêm composite index cho hot query) trong sprint này; chưa thêm read replica ngay.

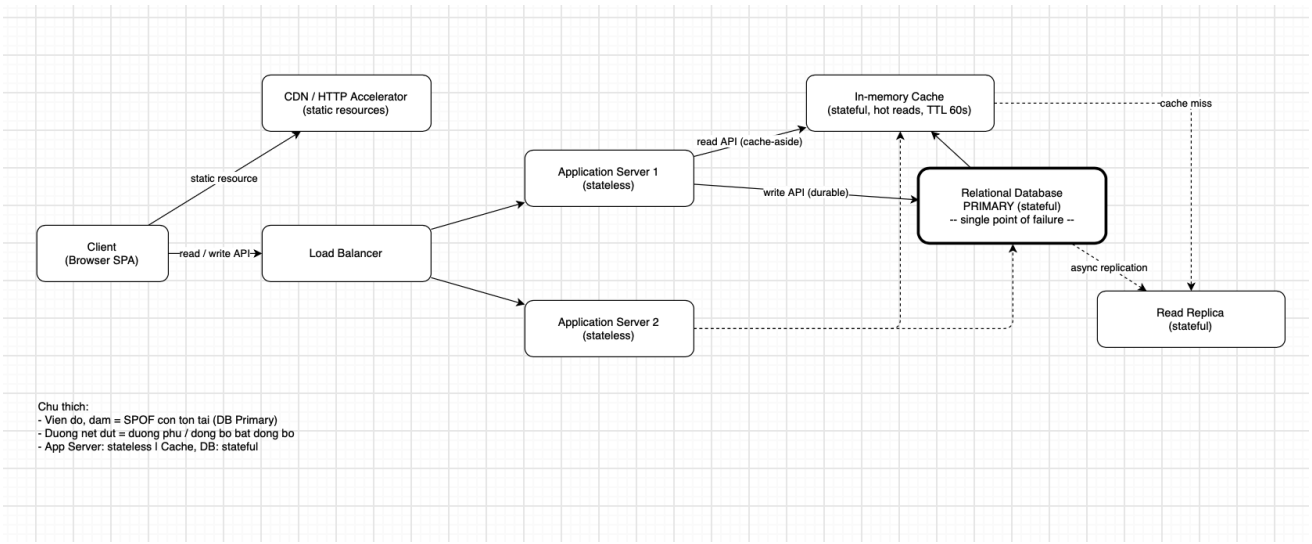
- Evidence: DB CPU 61%, disk I/O 68% — chưa phải bottleneck cấp bách; nguyên nhân chính đã xác định rõ là thiếu index, không phải do thiếu năng lực tổng thể của DB.
- Reasoning: Nguyên tắc của case là "mỗi thành phần thêm vào phải gắn với vấn đề cụ thể" — DB chưa cho thấy dấu hiệu cần thêm node, chỉ cần sửa đúng query.
- Trade-off: DB vẫn là SPOF và chưa có phương án cho tăng trưởng 3× trong 6 tháng — cần theo dõi tiếp.
- Risk: Việc build index trên bảng lớn (1,4 triệu rows) có thể khóa bảng tạm thời — nên chạy off-peak.
- Điều kiện chuyển sang Scale Out: Khi traffic tăng theo dự báo (3× trong 6 tháng) khiến CPU/disk I/O của DB tiệm cận bão hòa dù đã có index, hoặc khi cần failover cho DB (giải quyết luôn SPOF) — lúc đó thêm read replica là hợp lý (đã đưa vào kiến trúc mục tiêu 6 tháng ở Nhiệm vụ 3).

Nhiệm vụ 3 — Design: Kiến trúc mục tiêu 6 tháng

Vẽ kiến trúc bạn đề xuất cho CampusQ&A trong 6 tháng tới. Bạn có thể cân nhắc các thành phần sau, nhưng KHÔNG bắt buộc dùng hết: Client, CDN/HTTP Accelerator, Load Balancer, Application Server(s), Cache, Relational Database, Read Replica, Object/Static Storage.

- Đánh dấu đường đi của ít nhất 3 loại request: static resource, read API, write API.
- Ghi rõ thành phần nào stateful / stateless nếu quyết định đó ảnh hưởng đến scale-out.
- Khoanh hoặc đánh dấu các single point of failure còn tồn tại.
- Bên dưới sơ đồ, chọn 3 quyết định kiến trúc quan trọng nhất và bảo vệ bằng Decision + Evidence + Reasoning + Trade-off + Risk.
- Ít nhất 01 thành phần phổ biến phải được bạn chủ động KHÔNG dùng hoặc trì hoãn, kèm lý do.

KHU VỰC VẼ / HOÀN THIỆN SƠ ĐỒ KIẾN TRÚC



Ba quyết định chính: có thể viết ngắn theo cấu trúc dưới đây (không cần lặp lại toàn bộ cho mọi mũi tên trong sơ đồ).

	Thêm CDN/HTTP Accelerator cho static resources	**Load Balancer + Application Server scale-out (2 instance, stateless)**	**Thêm in-memory Cache cho 3 hot-read endpoint + composite index cho hot query**
Decision	Tách toàn bộ static resource ra khỏi Application Server, phục vụ qua CDN.	Thêm Load Balancer, chạy tối thiểu 2 Application Server instance stateless phía sau.	Cache kết quả của 3 endpoint đọc chiếm 52% API traffic (chấp nhận dữ liệu cũ tối đa 60 giây); song song thêm composite index cho hot query tag.
Evidence	Static chiếm 62% request, 74% byte trả về; client load 3,8 s (Wi-Fi)/7,1 s (4G).	CPU app 92% lúc peak; hiện là SPOF; deploy gây 2–4 phút 503; traffic dự báo tăng 3× trong 6 tháng.	3 endpoint = 52% traffic, tolerance 60s staleness; hot query chiếm 38% DB time, scan 1,4 triệu rows.
Reasoning	Đây là phần tải lớn nhất về băng thông nhưng không cần logic xử lý — tách ra giải phóng tài nguyên Application Server và giảm latency phía client.	Giải quyết đồng thời capacity, availability (99,9% mục tiêu) và rolling deploy không downtime, trong ngân sách 900 USD/tháng.	Cache giảm tải lặp lại lên DB cho phần dữ liệu chấp nhận độ trễ; index sửa trực tiếp nguyên nhân gốc của hot query

Trade-off	Thêm chi phí CDN hàng tháng; cần quản lý cache-invalidation mỗi lần deploy version mới của asset.	Bắt buộc app phải stateless — tăng effort refactor ban đầu; thêm một thành phần vận hành (LB) cho team không có SRE.	Cache thêm độ phức tạp về invalidation và khả năng dữ liệu cũ vượt quá 60s nếu cấu hình TTL sai; index làm chậm nhẹ thao tác ghi (write) và cần build off-peak.
Risk	Asset cũ bị cache "dính" (stale) nếu invalidation cấu hình sai.	Nếu statelessness không hoàn chỉnh, có thể phát sinh lỗi mất session khi request bị route sang instance khác.	Cache stampede khi entry hết hạn đồng loạt lúc peak; index build khóa bảng tạm thời nếu chạy sai thời điểm.
Điều kiện khiến quyết định thay đổi	Nếu tỷ lệ static resource giảm mạnh (ví dụ chuyển sang server-rendering ít asset hơn) hoặc ngân sách không còn đủ, có thể lùi lại dùng browser cache header thay vì CDN.	Nếu traffic tăng chậm hơn dự báo và budget hạn chế, có thể tạm hoãn instance thứ 2, chỉ scale-up instance hiện tại làm giải pháp tạm.	Nếu yêu cầu độ tươi dữ liệu (staleness) giảm xuống dưới ngưỡng cache TTL, phải giảm phụ thuộc cache và đọc thẳng từ Read Replica.

Nhiệm vụ 4 — Performance & Data Path: Chọn đúng đòn bẩy

Bạn chỉ có đủ thời gian triển khai 3 cải tiến trong sprint này. Chọn tối đa 3 hành động từ danh sách dưới đây hoặc đề xuất hành động tương đương:

- CDN / HTTP caching cho static resources
- Minify / compress / lazy-load / giảm số request phía client
- Application/data cache (ví dụ in-memory/Redis)
- Thêm index / điều chỉnh query plan cho hot query
- Thêm read replica
- Scale-up Application Server
- Thêm Load Balancer + nhiều App instances
- Chuyển JSON sang Protobuf
- Chuyển toàn bộ database sang NoSQL

Với mỗi lựa chọn: chỉ ra dữ kiện nào khiến hành động đó có giá trị, tầng nào được cải thiện, và trade-off/rủi ro mới. Sau đó chọn 01 phương án "nghe có vẻ hiện đại" nhưng bạn KHÔNG ưu tiên trong sprint này và giải thích vì sao evidence của case chưa đủ.

3 hành động chọn triển khai trong sprint này:

1. Thêm composite index cho hot query
   - Dữ kiện khiến hành động có giá trị: Query theo tag chiếm ~38% DB time, scan ~1,4 triệu rows, chưa có index phù hợp.
   - Tầng được cải thiện: Database.
   - Trade-off/rủi ro mới: Ghi (write) chậm đi nhẹ do thêm index; build index trên bảng lớn có thể khóa bảng tạm thời — cần chạy off-peak.
2. CDN/HTTP caching cho static resources
   - Dữ kiện khiến hành động có giá trị: Static chiếm 62% request, 74% byte; client load 3,8–7,1 giây tùy mạng.
   - Tầng được cải thiện: Client delivery, đồng thời giảm tải Application Server (giải phóng một phần CPU/băng thông).
   - Trade-off/rủi ro mới: Thêm chi phí hàng tháng; cần quản lý cache-invalidation khi deploy version asset mới.
3. Load Balancer + nhiều App instances
   - Dữ kiện khiến hành động có giá trị: CPU app 92% lúc peak (gần bão hòa); single app server là SPOF; deploy gây 2–4 phút 503 lặp lại 4 lần/tuần.
   - Tầng được cải thiện: Application + Reliability.
   - Trade-off/rủi ro mới: Yêu cầu app phải stateless trước khi scale-out; tăng độ phức tạp vận hành cho team không có SRE.

Phương án "nghe có vẻ hiện đại" nhưng KHÔNG ưu tiên trong sprint này:
Chuyển toàn bộ database sang NoSQL.

- Vì sao evidence chưa đủ: Không có dữ kiện nào trong case cho thấy giới hạn đến từ mô hình dữ liệu quan hệ (schema, join, transaction) — nguyên nhân đã xác định rõ là thiếu một composite index cụ thể, không phải bản chất relational của DB. DB hiện tại cũng chưa bão hòa (CPU 61%). Chuyển sang NoSQL sẽ tốn effort migration rất lớn, rủi ro mất tính nhất quán cho các thao tác ghi (đăng bài/answer/vote cần ghi bền vững và thấy gần như ngay) trong khi không giải quyết đúng vấn đề đã chẩn đoán — đúng dạng "dùng công nghệ vì nghe hiện đại" mà đề bài cảnh báo tránh.

Ví dụ về câu trả lời KHÔNG đạt: "Dùng cache vì cache làm nhanh hơn"; "Dùng microservices vì scale tốt"; "Thêm index vì database sẽ nhanh hơn".

Tiêu chuẩn tối thiểu: Nêu dữ kiện cụ thể trong case và giải thích vì sao dữ kiện đó liên quan đến lựa chọn.