**Fundamentals of Software Architecture**

**Buổi 1 — Architectural Thinking & Trade-offs**

**Chủ đề:** Architectural characteristics, trade-offs, constraints

### ***Mục tiêu***

* Hiểu tư duy kiến trúc và cách đưa ra quyết định dựa trên trade-off, chứ không phải “best practice”.

### ***Bài tập thực hành***

Cho một hệ thống “E-commerce mini”, hãy:

1. Xác định 7–10 architectural characteristics (availabilty, elasticity, performance, modifiability…).

**Performance** là đặc tính quan trọng vì người dùng tương tác trực tiếp với hệ thống thông qua việc xem video, tải bài học và thanh toán, tất cả đều rất nhạy cảm với độ trễ. Nếu hệ thống phản hồi chậm, người dùng sẽ nhanh chóng mất kiên nhẫn, bỏ học hoặc rời khỏi trang mua khóa học, dẫn đến giảm doanh thu.

Ví dụ, nếu video mất vài giây dài để bắt đầu phát hoặc trang checkout load chậm trong giờ cao điểm, người dùng có xu hướng thoát ra và không quay lại.

**Availability** cần thiết vì nền tảng học trực tuyến hoạt động liên tục 24/7 và bất kỳ thời gian gián đoạn nào cũng trực tiếp gây mất tiền và mất niềm tin người dùng. Nếu hệ thống không sẵn sàng, người dùng không thể học hoặc mua khóa học dù họ đang có nhu cầu, lâu dài sẽ hình thành cảm giác nền tảng không đáng tin cậy.

Ví dụ, hệ thống bị downtime trong lúc diễn ra livestream bán khóa học có thể làm thất bại toàn bộ chiến dịch marketing.

**Scalability** quan trọng vì lượng truy cập của hệ thống không ổn định, thường tăng đột biến trong các chiến dịch quảng cáo hoặc sự kiện livestream. Nếu hệ thống không scale tốt, backend sẽ bị quá tải dẫn đến chậm hoặc sập, trong khi việc đầu tư hạ tầng lớn ngay từ đầu lại gây lãng phí chi phí.

Ví dụ, khi hàng nghìn người cùng truy cập xem livestream hoặc mua khóa học cùng lúc, hệ thống không đủ khả năng xử lý sẽ khiến trải nghiệm người dùng xuống cấp nghiêm trọng.

**Cost efficiency** được ưu tiên vì nội dung video tiêu tốn rất nhiều tài nguyên lưu trữ và băng thông, và nếu không kiểm soát tốt chi phí, hệ thống có thể càng nhiều người dùng thì càng lỗ. Nếu bỏ qua yếu tố này, chi phí hạ tầng sẽ tăng nhanh hơn doanh thu, đặc biệt khi mở rộng quy mô.

Ví dụ, tự host video hoặc không giới hạn băng thông có thể khiến hóa đơn vận hành tăng đột biến mà không mang lại giá trị tương xứng.

**Modifiability** cần thiết vì sản phẩm giáo dục thường xuyên thay đổi, liên tục bổ sung khóa học, bài kiểm tra, chứng chỉ hoặc các tính năng mới. Nếu hệ thống khó thay đổi, mỗi yêu cầu mới sẽ mất rất nhiều thời gian và dễ gây lỗi lan sang các phần khác.

Ví dụ, việc thêm một loại quiz mới nhưng phải sửa nhiều module liên quan cho thấy kiến trúc thiếu linh hoạt và làm chậm tốc độ phát triển sản phẩm.

**Security** quan trọng vì video là tài sản cốt lõi của nền tảng và hệ thống còn xử lý dữ liệu người dùng cũng như thanh toán. Nếu không đảm bảo an ninh, video có thể bị leak, tài khoản người dùng bị chiếm quyền hoặc dữ liệu nhạy cảm bị lộ, gây thiệt hại lớn về uy tín và pháp lý.

Ví dụ, nếu link video không được bảo vệ tốt, nội dung có thể bị chia sẻ tràn lan trên các nền tảng khác.

**Consistency** cần được đảm bảo cho các nghiệp vụ quan trọng như thanh toán và kích hoạt khóa học, vì sai lệch dữ liệu ở những khu vực này sẽ ảnh hưởng trực tiếp đến niềm tin người dùng. Nếu thiếu consistency, người dùng có thể đã trả tiền nhưng không được cấp quyền học hoặc trạng thái đơn hàng hiển thị sai.

Ví dụ, payment thành công nhưng khóa học chưa được mở ngay sẽ gây khiếu nại và trải nghiệm tiêu cực.

**Elasticity** quan trọng vì hệ thống cần tự động thích ứng với mức tải thay đổi theo thời gian mà không cần can thiệp thủ công. Nếu không có elasticity, hệ thống sẽ hoặc bị quá tải khi traffic tăng, hoặc lãng phí tài nguyên khi traffic thấp.

Ví dụ, trong thời gian chạy chiến dịch marketing, hệ thống không kịp scale sẽ gây chậm và lỗi, trong khi ban đêm ít người dùng nhưng tài nguyên vẫn bị sử dụng dư thừa.

**Observability** cần thiết để đội vận hành hiểu được hệ thống đang hoạt động như thế nào trong môi trường production. Nếu thiếu khả năng quan sát, việc phát hiện và xử lý sự cố sẽ rất chậm và chủ yếu dựa vào phản ánh của người dùng.

Ví dụ, video không phát được ở một khu vực nhất định hoặc payment thất bại ngẫu nhiên nhưng không có log hay trace sẽ khiến việc debug trở nên cực kỳ khó khăn.

1. Sắp xếp ưu tiên đặc tính bằng kỹ thuật **utility tree**.  
    Performance

 Availability

 Cost efficiency

 Scalability

 Modifiability

 Security

 Consistency

 Observability

1. Viết 5 quyết định kiến trúc có trade-off (ví dụ: SQL vs NoSQL, sync vs async).

**🔹 Decision 1: Video self-host vs CDN/Cloud Storage**

**Option A**: Tự host video (server riêng)  
**Option B**: Cloud Storage + CDN (Cloudinary / S3 + CloudFront) ✅

**Decision**: **Cloud + CDN**

**Trade-off**

| **Ưu điểm** | **Nhược điểm** |
| --- | --- |
| Video load nhanh toàn cầu | Chi phí CDN |
| Scale tự động | Ít kiểm soát |
| Reliability cao | Vendor lock-in |

👉 **Ưu tiên performance & availability hơn cost tuyệt đối**

**🔹 Decision 2: Strong consistency vs Eventual consistency**

**Decision**:

* Payment: **Strong consistency**
* Progress học: **Eventual consistency**

**Trade-off**

| **Ưu điểm** | **Nhược điểm** |
| --- | --- |
| Tránh sai tiền | Progress có thể trễ |
| UX mượt | Logic phức tạp hơn |

👉 Tiến độ học **không critical** → cho phép delay

**🔹 Decision 3: Monolith vs Microservices**

**Decision**: **Modular Monolith**

**Trade-off**

| **Ưu điểm** | **Nhược điểm** |
| --- | --- |
| Phát triển nhanh | Scale từng phần khó |
| Ít DevOps | Coupling tăng nếu thiết kế kém |

👉 Course platform **KHÔNG cần microservices sớm**

**🔹 Decision 4: Sync vs Async cho Video Progress**

**Decision**:

* Sync: load lesson
* Async: save progress (Kafka / Queue)

**Trade-off**

| **Ưu điểm** | **Nhược điểm** |
| --- | --- |
| UX nhanh | Có độ trễ |
| Giảm DB write | Phức tạp hơn |

**🔹 Decision 5: Auth Session vs JWT**

**Decision**: **JWT + short-lived token**

**Trade-off**

| **Ưu điểm** | **Nhược điểm** |
| --- | --- |
| Scale dễ | Revoke khó |
| Phù hợp stateless | Cần refresh token |

**Architectural Decision Record (ADR)**

**📄 ADR-001: Use CDN for Video Delivery**

**Status**: Accepted

**Context**  
Video là tài nguyên nặng, nhiều user truy cập đồng thời.

**Decision**  
Lưu video trên cloud storage và phân phối qua CDN.

**Consequences**

* (+) Giảm tải backend
* (+) Tăng performance toàn hệ thống
* (−) Tăng chi phí CDN

**📄 ADR-002: Eventual Consistency for Learning Progress**

**Status**: Accepted

**Decision**  
Tiến độ học được ghi nhận bất đồng bộ.

**Consequences**

* (+) UX mượt
* (+) Giảm write load
* (−) Dữ liệu có thể trễ vài giây

### **📄 *Deliverable:***

* Utility tree
* Trade-off analysis
* Architectural decision record (ADR)

**Buổi 2 — Component-Based Thinking**

**Chủ đề:** Modular decomposition, domain partitioning.

### **🧪 *Bài tập thực hành***

Thiết kế hệ thống “Course Management” và:

+ Hệ thống **Course Management** cho phép:

* Quản lý khóa học
* Quản lý giảng viên
* Quản lý học viên
* Đăng ký khóa học
* Quản lý bài giảng, nội dung
* Theo dõi tiến độ học tập

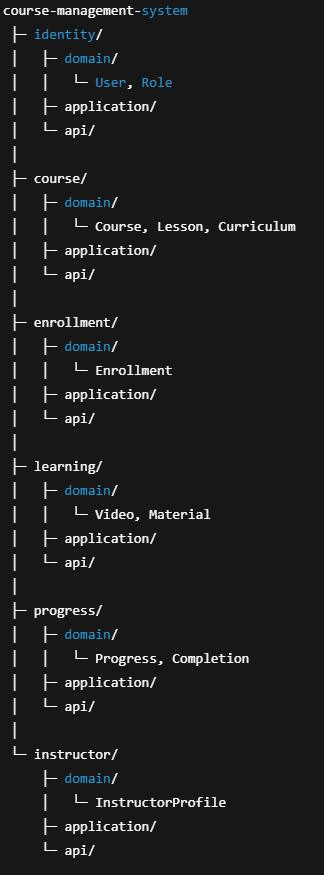
1. Phân chia thành module theo **Domain Decomposition**.

**Nguyên tắc áp dụng**  
 Chia theo **nghiệp vụ (business capability)**

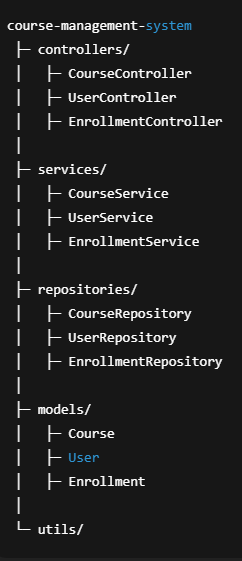
 Mỗi module chịu trách nhiệm **một miền nghiệp vụ rõ ràng**

 Không chia theo kỹ thuật (controller/service/db)

* Identity Domain
* Course Domain
* Enrollment Domain
* Learning Domain
* Progress Domain
* Instructor Domain



1. Phân chia lại theo **Technical Partitioning**.  
   **Nguyên tắc áp dụng**
   * Chia theo **kỹ thuật / tầng kiến trúc**
   * Phổ biến trong monolith truyền thống



1. Giải thích tại sao hai cách cho kết quả khác nhau.  
   **1. Khác nhau về tư duy thiết kế**

| **Tiêu chí** | **Domain Decomposition** | **Technical Partitioning** |
| --- | --- | --- |
| Góc nhìn | Business | Kỹ thuật |
| Câu hỏi chính | Business cần gì? | Code tổ chức thế nào? |
| Trọng tâm | Nghiệp vụ | Tầng kiến trúc |

**2. Khác nhau về mức độ coupling**

**Technical Partitioning**

* + Service gọi chéo
  + Chia nhỏ về mặt code nhưng **không chia trách nhiệm**

**Domain Decomposition**

* + Mỗi domain tự chịu trách nhiệm

- Coupling thấp, cohesion cao

**3. Khác nhau về khả năng mở rộng**

Technical Partitioning:

- Scale toàn hệ thống

- Khó tách microservices

Domain Decomposition:

- Scale theo domain

- Tách microservices dễ dàng

**4. Khác nhau về tổ chức team**

Technical:

* + Team làm chung nhiều module

Domain:

* + Mỗi team phụ trách 1–2 domain

1. Đưa ra quyết định chọn kiến trúc mô-đun nào.  
   => Chọn Domain Decomposition làm kiến trúc chính

Lý do:

* Phù hợp với hệ thống nghiệp vụ phức tạp
* Dễ mở rộng và bảo trì lâu dài
* Hỗ trợ tốt cho:
* Modular Monolith
* Microservices (khi cần)
* Tách biệt rõ ràng trách nhiệm nghiệp vụ

### **📄 *Deliverable:***

* Module diagram cho 2 hướng
* Lý do chọn 1 hướng

# 

# **Buổi 3 — Architecture Styles (Layered, Microkernel)**

**Chủ đề:** Layered Architecture, Microkernel.

### **🧪 *Bài tập***

Với hệ thống “Plugin-based CMS”:

1. Vẽ sơ đồ Layered Architecture.
2. Vẽ sơ đồ Microkernel Architecture.
3. So sánh ưu/nhược điểm dựa trên sách.
4. Chọn style thích hợp nhất cho CMS và giải thích.

### **📄 *Deliverable:***

* 2 architecture diagrams
* Style trade-off

# **Buổi 4 — Architecture Styles (Microservices, SOA)**

**Chủ đề:** Distributed architecture style.

### **🧪 *Bài tập***

Với hệ thống “Online Food Delivery”:

1. Thiết kế bản **monolith → microservices migration**.
2. Xác định bounded context.
3. Mô tả giao tiếp Sync vs Async giữa services.
4. Tạo event messaging design (Kafka/RabbitMQ, chỉ cần mô hình).

### **📄 *Deliverable:***

* Service map
* Context map
* Communication diagram

# **Buổi 5 — Event-driven Architecture**

**Chủ đề:** Event choreography vs orchestration.

### **🧪 *Bài tập***

Thiết kế workflow đặt đơn thực phẩm:

1. Sử dụng **Event Choreography**.
2. Sử dụng **Event Orchestration** (có 1 orchestrator).
3. So sánh ưu nhược điểm.
4. Quyết định mô hình phù hợp với scaling + resilience.

### **📄 *Deliverable:***

* Event flow diagrams
* Trade-off notes

# **Buổi 6 — Architecture Patterns for Deployment**

**Chủ đề:** Deployment patterns, modular monolith, pipelines.

### **🧪 *Bài tập***

Cho hệ thống “Video Streaming Service”:

1. Thiết kế mô hình deploy:  
   * Single deployable unit
   * Modular monolith
   * Microservices
2. Xác định rủi ro vận hành cho mỗi mô hình.
3. Đưa ra khuyến nghị cuối.

### **📄 *Deliverable:***

* Deployment diagrams
* Operational risk assessment

# **Buổi 7 — Architecture Decisions (ADRs + Decision Trap Avoidance)**

**Chủ đề:** Architecture decisions, decision log.

### **🧪 *Bài tập***

1. Chọn 5 quyết định quan trọng cho hệ thống “Banking Mini”:  
   * Database
   * API style
   * Transaction consistency
   * Security strategy
   * Deployment
2. Viết ADR theo template của sách.
3. Identify **decision drift** và đề xuất cách ngăn chặn.

### **📄 *Deliverable:***

* 5 ADRs
* Risk of decision drift

# **Buổi 8 — Architecture Analysis & Fitness Functions**

**Chủ đề:** Fitness functions, evolutionary architecture.

### **🧪 *Bài tập***

Với hệ thống “Billing & Subscription”:

1. Chọn 4 đặc tính kiến trúc quan trọng (security, latency, modifiability…).
2. Định nghĩa mỗi đặc tính bằng 1 fitness function:  
   * Atomic fitness function
   * Holistic fitness function
3. Thiết kế pipeline CI/CD chạy fitness function tự động.

### **📄 *Deliverable:***

* Fitness function definitions
* Fitness pipeline diagram

# **Buổi 9 — Contract Evolution & Versioning**

**Chủ đề:** API versioning, schema evolution, backward compatibility.

### **🧪 *Bài tập***

1. Cho API “User Profile”, thiết kế 3 phiên bản API:  
   * v1: basic
   * v2: add email
   * v3: add address + optional fields
2. Thực hiện backward compatibility matrix.
3. Thiết kế chiến lược migration an toàn.

### **📄 *Deliverable:***

* API evolution diagram
* Compatibility matrix
* Migration plan

# 

# 

# 

# **Buổi 10 — Architectural Kata (Final Project)**

**Chủ đề:** Tổng hợp toàn bộ kiến thức trong sách.

### **🧪 *Bài tập — Full System Architecture***

Thiết kế kiến trúc cho hệ thống:  
 💡 **“Smart Inventory Management Platform”** (IoT + API + Analytics)

### **Yêu cầu:**

1. Xác định architectural characteristics.
2. Chọn architecture style (layered / event-driven / microservices / modular monolith).
3. Phân chia module + service.
4. Chọn giao tiếp: sync/async, event flow.
5. Thiết kế deployment architecture.
6. Viết 3 ADR quan trọng.
7. Xây 3 fitness functions.

### **📄 *Deliverable:***

* Architecture diagram
* Context & module diagram
* ADRs
* Fitness functions
* Decision rationale document