Layered Architecture

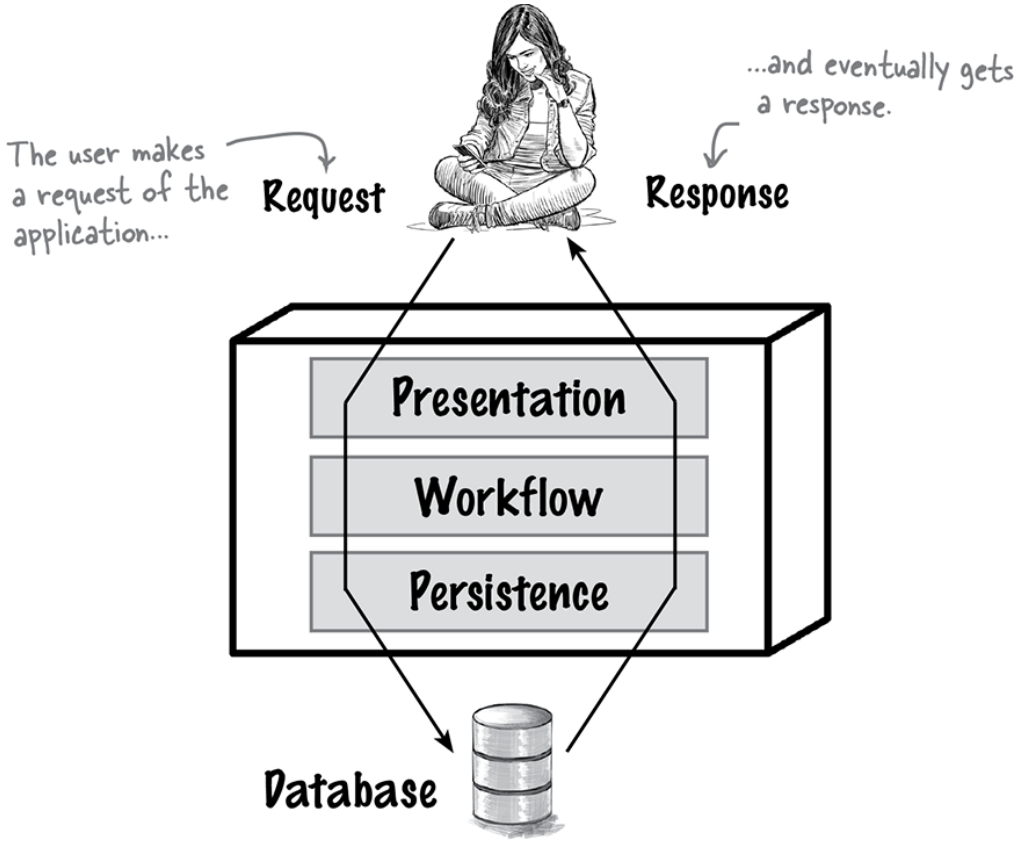
UNSW SYDNEY

Introduction to Layered Architecture

- Layered Architecture separates technical responsibilities into distinct layers.
- Simplifies the design by dividing the system into manageable, logical parts.

Key benefits:

- Easy to understand and implement.
- Promotes reuse and separation of concerns.

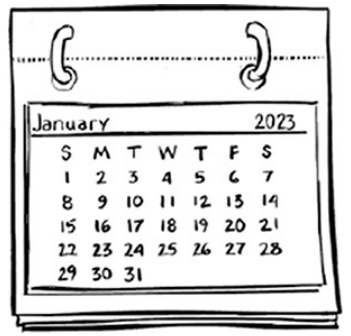


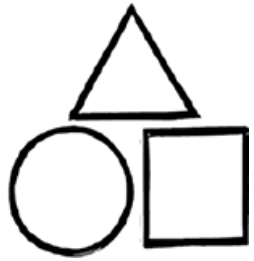
Case Study: Naan & Pop Restaurant

- Startup restaurant serving Indian-inspired flatbread sandwiches.
- Needs a simple website for online ordering quickly.

Requirements:

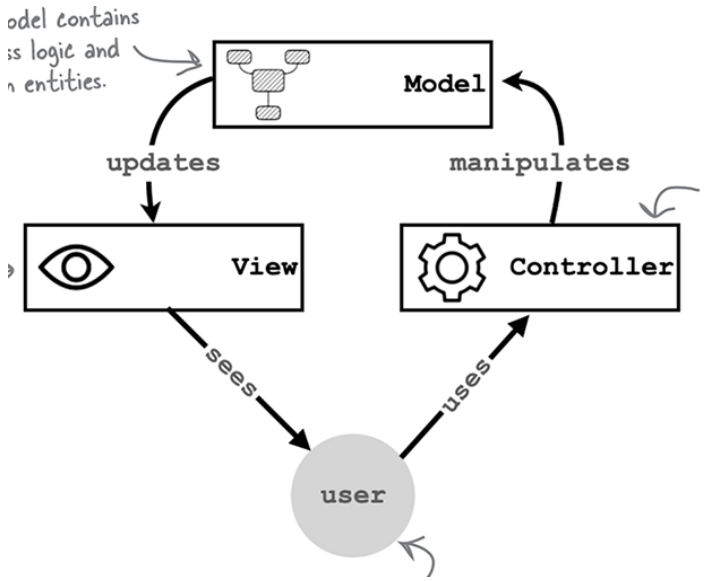
- Time to market: Quick launch.
- Separation of responsibilities: Clear division for UI specialists and database administrators.
- Extensible: Allow future enhancements easily.





Why Choose Layered Architecture?

- Matches Naan & Pop's needs: simplicity, fast delivery, separation of technical roles.
- Aligns closely with familiar design patterns like MVC.

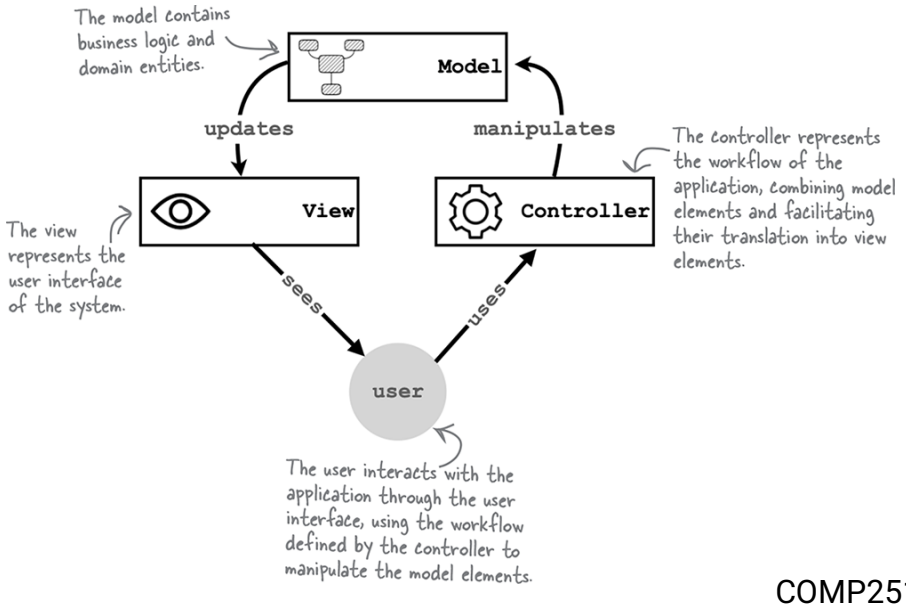


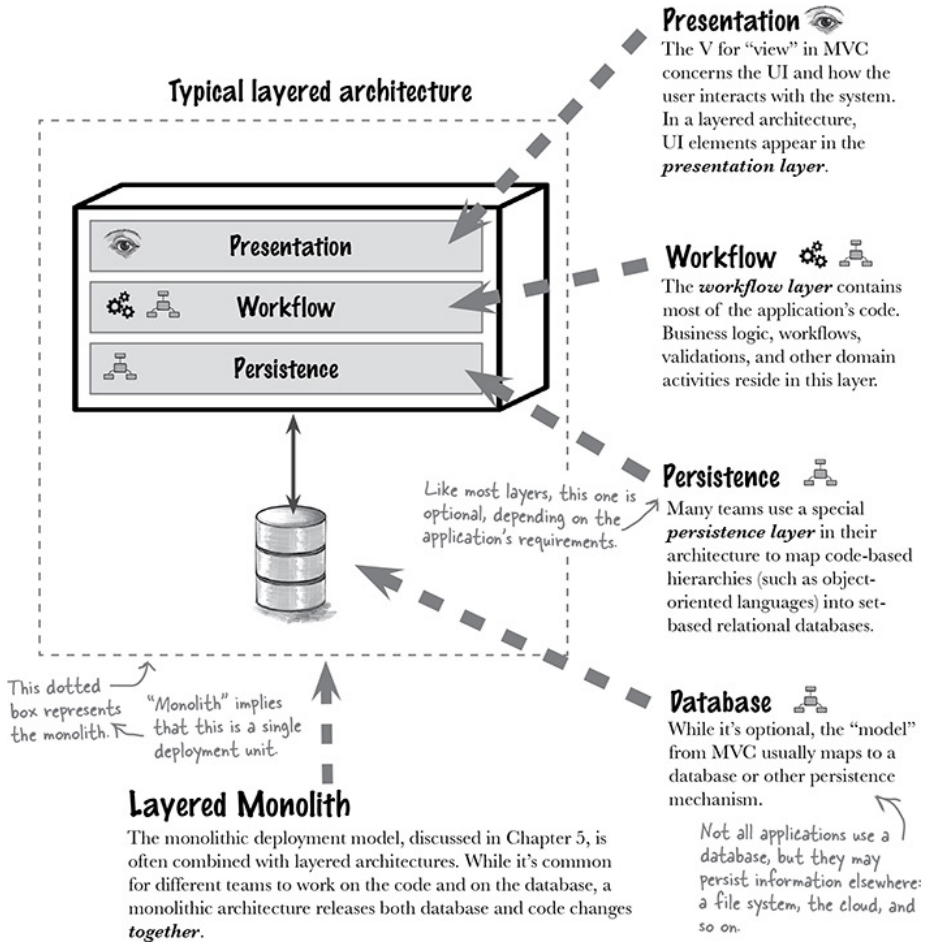
Trade-offs involved:

- Simplicity vs. extensibility.
- Speed vs. maintainability.

Mapping MVC to Layered Architecture

- MVC concepts translate naturally into architectural layers.
- Additional layers may be introduced based on real-world constraints (e.g., integration).



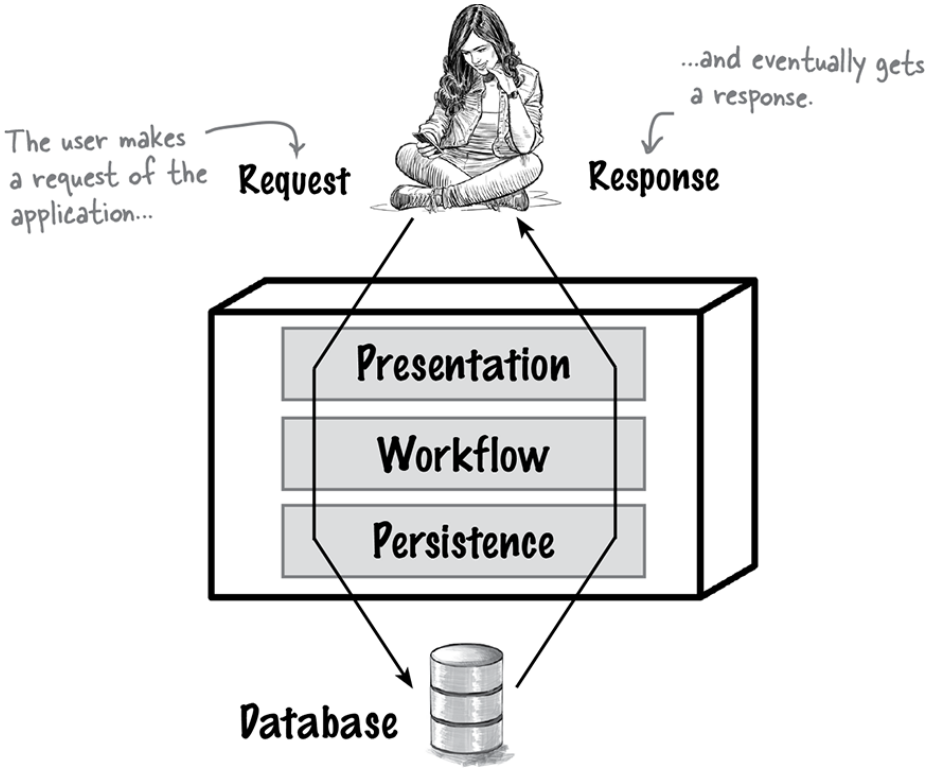


Layered Architecture – Philosophy

- Technically partitioned and usually monolithic.
- Domain logic spans multiple layers:
  - Presentation (UI components).
  - Workflow (business logic components).
  - Persistence (database schemas and operations).

Implication:

- Domain changes affect multiple layers.



Drivers for Layered Architecture

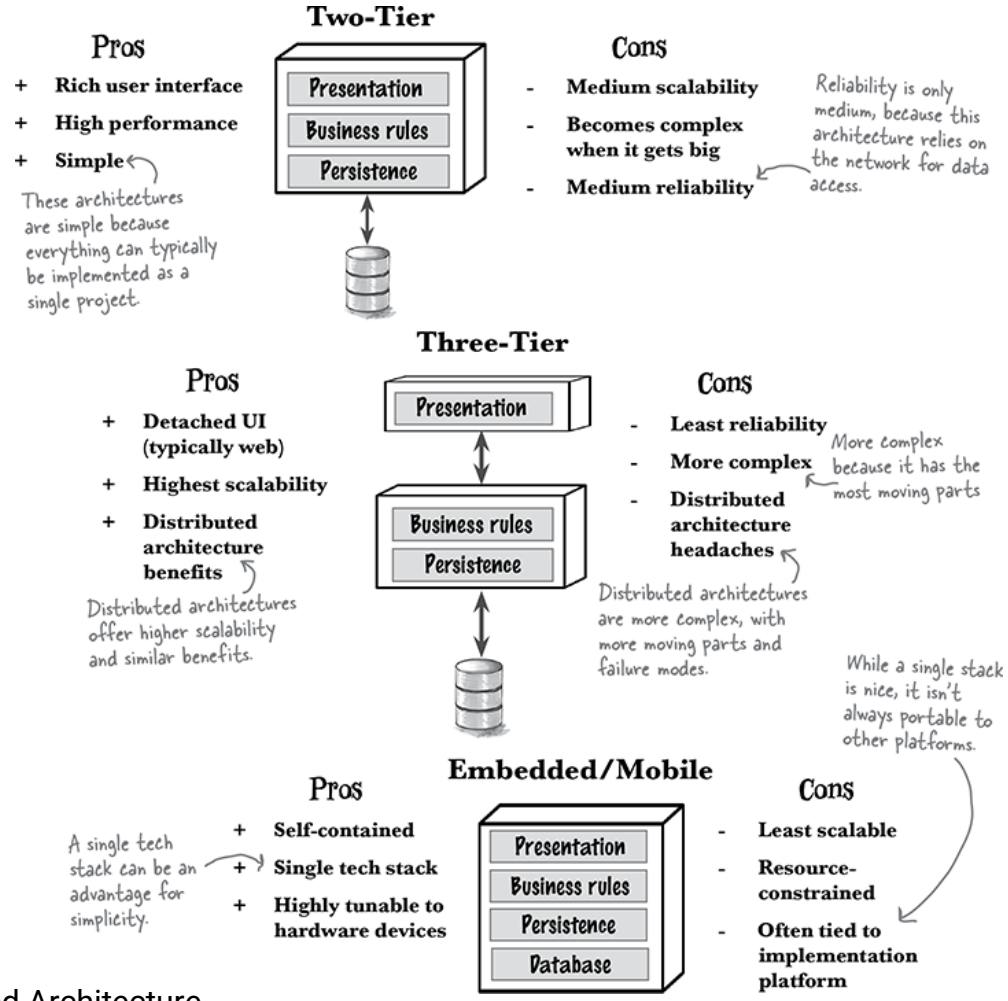
Why choose layered architecture?

- Specialization: Separates UI, business logic, and database, allowing team specialisation.
- Physical separation: Matches real-world technology separation (frontend/backend/database).
- Ease of reuse: Technical reuse across multiple projects.
- Familiarity: Mirrors MVC, easy for developers to grasp.

Physical Architectures in Layered Systems

Common physical architectures:

- Two-tier (Client/Server):
  - Client UI directly accesses the database.
- Three-tier (Web):
  - Browser (presentation),
  - App server (business logic)
  - Database server (persistence)
- Embedded/Mobile:
  - All layers bundled into one deployable unit.



Physical Architecture – Pros and Cons

Physical Architecture	Pros	Cons
Two-tier (Client/Server)	Simple, quick to build	Less secure, poor scalability
Three-tier (Web)	Scalable, flexible	Complex infrastructure
Embedded/Mobile	High performance, simple deployment	Limited scalability

Adding Layers – Integration Layer Example

- Additional layers can be introduced for specialised tasks (e.g., Integration layer for delivery partners).
- Clearly isolates integration code from core business logic.

Example:

- Integration with Uber Eats API resides entirely within an Integration Layer.

Caveats – Domain Changes Impact Multiple Layers

- Layered architecture easily supports changes in technical capabilities.
- However, changes in the domain (e.g., adding pizzas to menu) will affect multiple layers:
  - Presentation layer (new UI)
  - Workflow layer (processing new item)
  - Persistence layer (storing item data)

Trade-off:

- Ease of technical changes vs. difficulty of domain-wide changes.

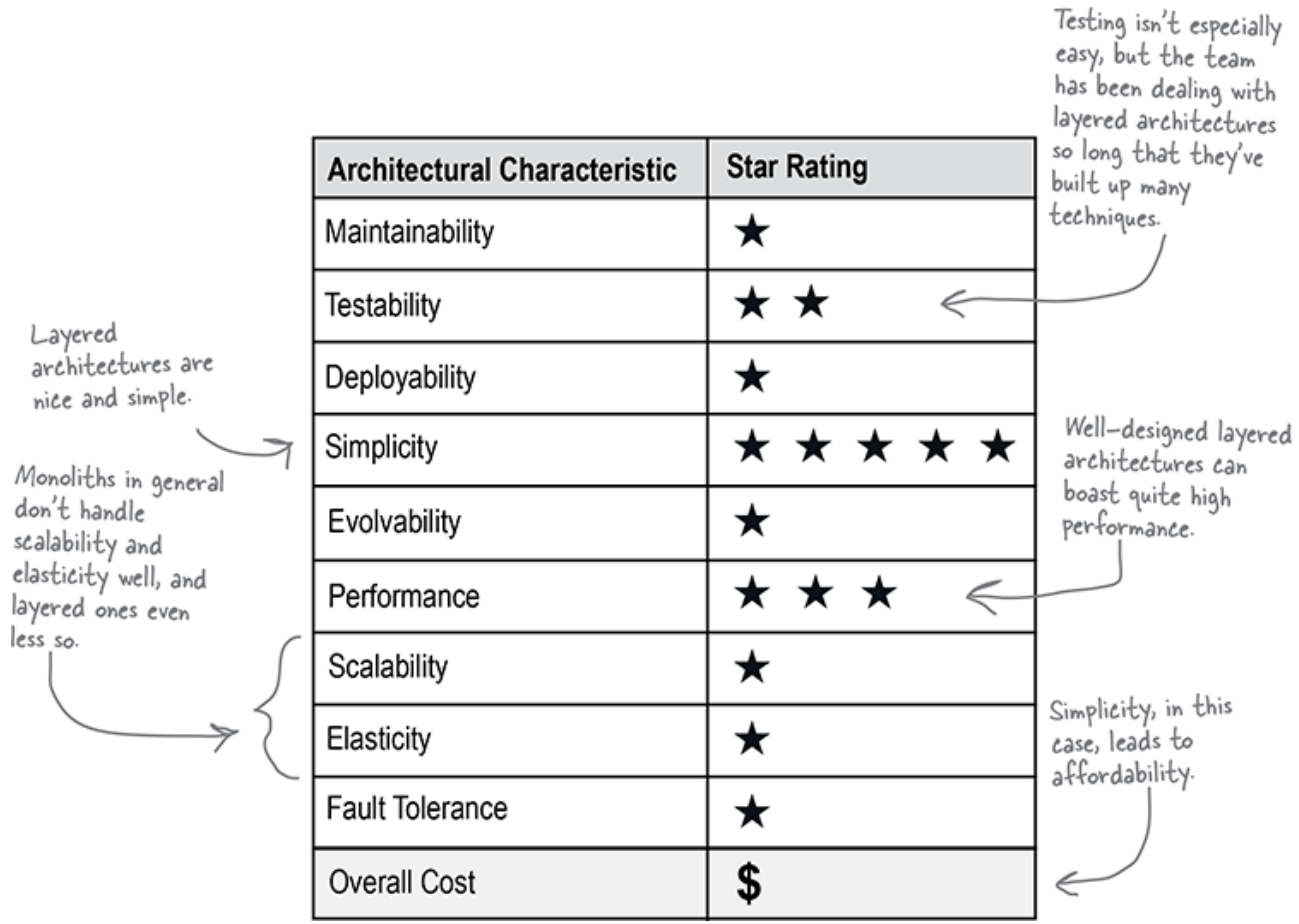
Layered Architecture: Strengths

- Feasibility: Quick, cost-effective solutions.
- Technical partitioning: Easy technical reuse.
- Data-intensive operations: Efficient local data processing.
- Performance: High internal performance without network overhead.
- Fast development: Ideal for MVPs and small systems.

Layered Architecture: Weaknesses

- Deployability: Monolith deployments become cumbersome as systems grow.
- Coupling: High risk of tight coupling (“big ball of mud”).
- Scalability: Difficult to scale individual functionalities independently.
- Elasticity: Poor performance under bursty traffic conditions.
- Testability: Increasingly difficult testing as codebase grows.

Layered Architecture – Rating Chart (Example)



Architectural Characteristic	Star Rating
Maintainability	★
Testability	★ ★
Deployability	★
Simplicity	★ ★ ★ ★ ★
Evolvability	★
Performance	★ ★ ★
Scalability	★
Elasticity	★
Fault Tolerance	★
Overall Cost	$

Layered Architecture – Exercises

An online auction system where users can bid on items

Why? _____

_____

_____

- ☐ Well suited for layered monolith
- ☐ Might be a fit for layered monolith
- ☐ Not well suited for layered monolith

A large backend financial system for processing and settling international wire transfers overnight

Why? _____

_____

_____

- ☐ Well suited for layered monolith
- ☐ Might be a fit for layered monolith
- ☐ Not well suited for layered monolith

A company entering a new line of business that expects constant changes to its system

Why? _____

_____

_____

- ☐ Well suited for layered monolith
- ☐ Might be a fit for layered monolith
- ☐ Not well suited for layered monolith

A small bakery that wants to start taking online orders

Why? _____

_____

_____

- ☐ Well suited for layered monolith
- ☐ Might be a fit for layered monolith
- ☐ Not well suited for layered monolith

A trouble ticket system for electronics purchased with a support plan, in which field technicians come to customers to fix problems

Why? _____

_____

_____

- ☐ Well suited for layered monolith
- ☐ Might be a fit for layered monolith
- ☐ Not well suited for layered monolith

Suitable Scenarios for Layered Architecture

Ideal Use Cases:

- Small, simple systems requiring quick delivery (e.g., small business websites).
- Data-intensive applications with local database storage

  (e.g., desktop CRM apps).
- Applications needing clear specialization boundaries

  (e.g., separate UI, backend, DB teams).

Summary of Layered Architecture

Key points:

- Simple, fast to implement.
- Clearly separates technical concerns.
- Ideal for stable domains with minimal changes.
- Challenging to adapt when domain changes significantly.