# Arquitectura de Capas

- Problema **-/> sub-problemas**.
- Fomentan uso **interfaces**.
- Intercambiar componentes **reutilizando conectores y protocolos**.

## Layers (capas lógicas)

Agrupación lógica de componentes y funcionalidades de un sistema.

- Verticales y horizontales.
- Suelen hacer `downcalls` (con response).
  - Las `upcalls` son excepcionales.
- **Layer:** módulo con responsabilidades limitadas, coherencia y cohesión.
- C/ capa provee servicios a la capa superior y consume de la inferior.

## Tiers (capas físicas)

Describen la distribución física de componentes y funcionalidades de un sistema.

- 2-Tier deployment, 3-Tier.
- Despliegue de Layers dentro de cada tier.

# Interfaces

- **Permiten comunicación** entre dos o más componentes/servicios/sistemas.
- Diferentes **contratos**.
- Se expone **una parte** del sistema.
- Esconden implementación.
  - Cambiarla sin modificar contrato.
  - Cambio contrato -> nueva versión.

## Tipos de contrato

- **Inter-Aplicaciones.**
  - **API (Application Program Interface).** Interfaz que permite que dos aplicaciones hablen entre sí.
  - Punto de acceso para que una aplicacion que vive por si sola permita que otra aplicación pueda reutilizar la funcionalidad que expone.
  - Eco-sistema entre aplicaciones.
- **Intra-Aplicaciones.**
  - Patrón(es) de diseño (Facades, Mediators, Interfaces).
  - Layers hablandose entre sí.
  - Mensajes entre objetos.

## Problemas a resolver

Software es:

- **Dificil de integrar:**
  - Complejidad aumenta exponencialmente con la cantidad de elementos expuestos.
  - Interfaces pequeñas, esconder cosas.
  - Empezar exponiendo poco.
  - Extender con nuevas versiones si es necesario.
- **Dificil de cambiar:**
  - Fina linea entre interfaz flexible vs. cerrada y que no se adapta a los cambios.

  - Ojo con complejizar una interfaz de más.

### Modelado de contratos p/ APIs

- Orientados a **Entidades:**
  - Desacoplamiento entre sistemas.
  - Objetivo = flexibilidad.
  - Admite extensiones.
- Orientados a **Procesos:**
  - Alto acoplamiento.
  - Alta performance como objetivo.

### Clasificación

- **Web APIs.**
  - REST based (HTTP + JSON).
  - Web Services based (HTTP + SOAP).
- **Remote APIs.** Object-Procedure oriented.
- **Library-based / Frameworks.** Java-Android API.
- **OS related.** POSIX, WinAPI.

## Protocolos

### Modelo HTTP

- Client-Server.
- Request-Reply.
- Sin estado.

#### Protocol Data Unit (PDUs)

**Encapsulación** de PDUs entre capas.

1. **Encapsulación** exacta.
2. **Segmentación** de paquetes.
3. **Blocking** de paquetes.

## RESTful

- **Entidades**.
- Web resource **representado por URL**.
- **HTTP/S** protocolo de comunicación.
- **JSON/XML** protocolo de serialización.
- Operaciones **CRUD** p/ cambio de estado.
- **Principios de Arquitectura:**
  - Cliente-Servidor.
  - Cacheability.
  - Interface Uniforme (HATEOAS).
  - Stateless.
  - Layered.
- **Identidad.** Unívocamente entre sistemas.
- **Relaciones.** Integración con otros sistemas.

## Versionado de API

- **Semantic Versioning (semver).**
- **Incremento:**
  - `+Major` = cambios incompatibles.
  - `+Minor` = agregar funcionalidad manteniendo retrocompatibilidad.
  - `+Patch` (*build*) = correcciones sin afectar interfaz.
- **Tipos:**
  - Explícito en URL.
  - HTTP Custom Header. Incorrecto.
  - HTTP Accept Header. Correcto.
- **!= versionado de objetos!**
  - **Format Versioning.** API puede brindar distintas representaciones de **la misma entidad**.
  - **Historical Versioning.**