PATRONES DE ARQUITECTURA DE SOFTWARE

introducción

¿Qué es la arquitectura de software?

Son los bloques fundamentales del sistema, la estructura de la solución.

"Lo significativo se mide por el costo de cambio"

- Forma del sistema.
- Decisiones de diseño significativas medidas por el costo de cambio.
- División del sistema en componentes y la comunicación entre ellos.

¿Por qué importa la arquitectura?

Define si nuestro sistema puede evolucionar favorablemente a largo plazo.

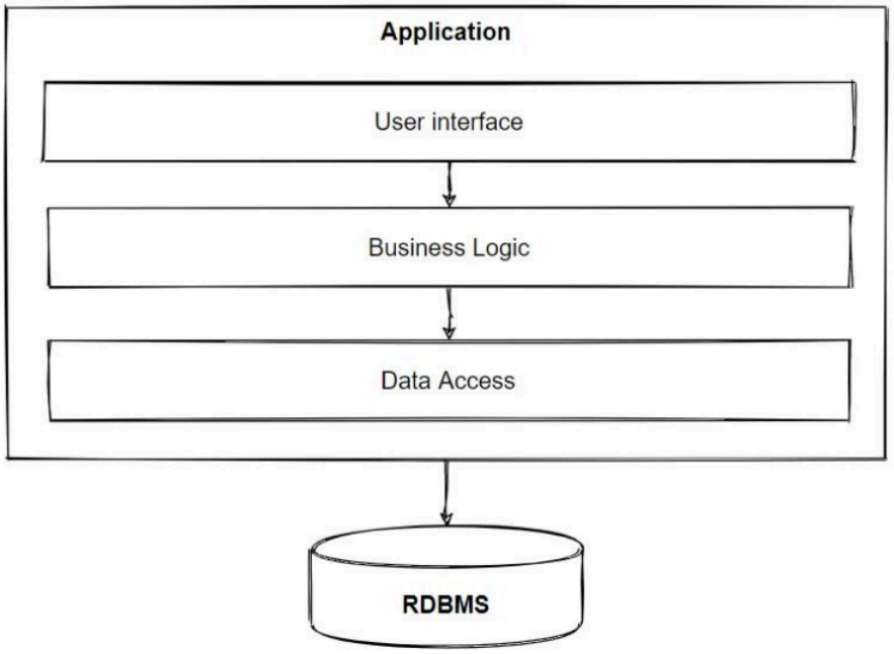
PATRONES

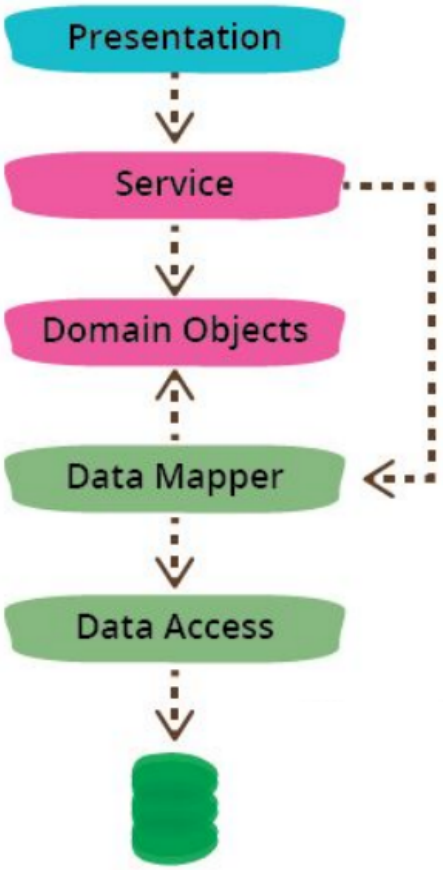
LAYERS (parecido al MVC)

Dividimos el sistema en capas, donde las capas exteriores dependen de las interiores, pero las interiores no pueden depender de las exteriores.

Ejemplo: sistema de bases relacionales.

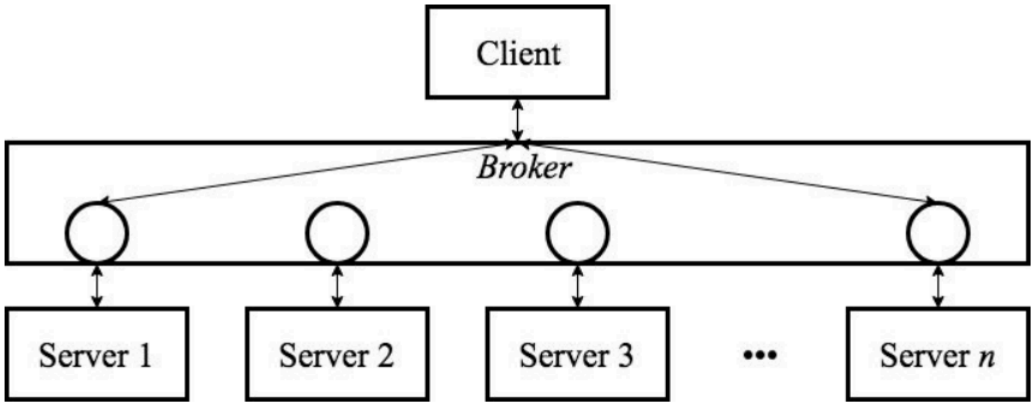
- User interface: comunicación con el usuario.
- Business logic: lógica de negocio, casos de uso del programa.
- Data access: detalles de dónde se encuentra cada dato.


Application
User interface
Business Logic
Data Access
RDBMS


Presentation
Service
Domain Objects
Data Mapper
Data Access

BROKER (intermediario)

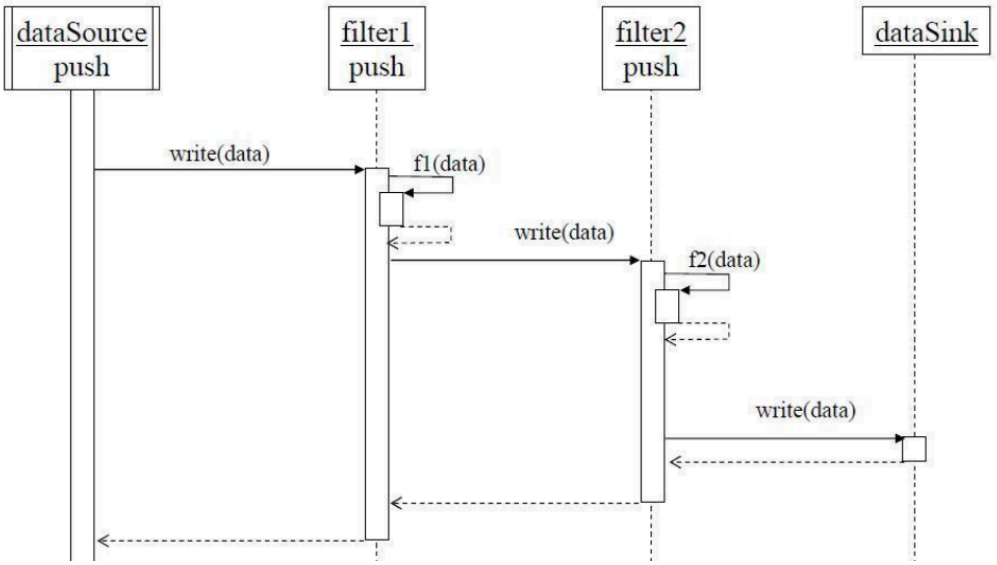
Cumple la función de una vez ingresada dada una petición, asignarla a un determinado servidor/cliente. La asignación puede ser aleatoria o cumpliendo "Round Robin".



PIPE & FILTERS

Cadena de eslabones que se comunican entre sí para resolver una determinada tarea.

ETL: Extracción, Transformación y Carga (Load)



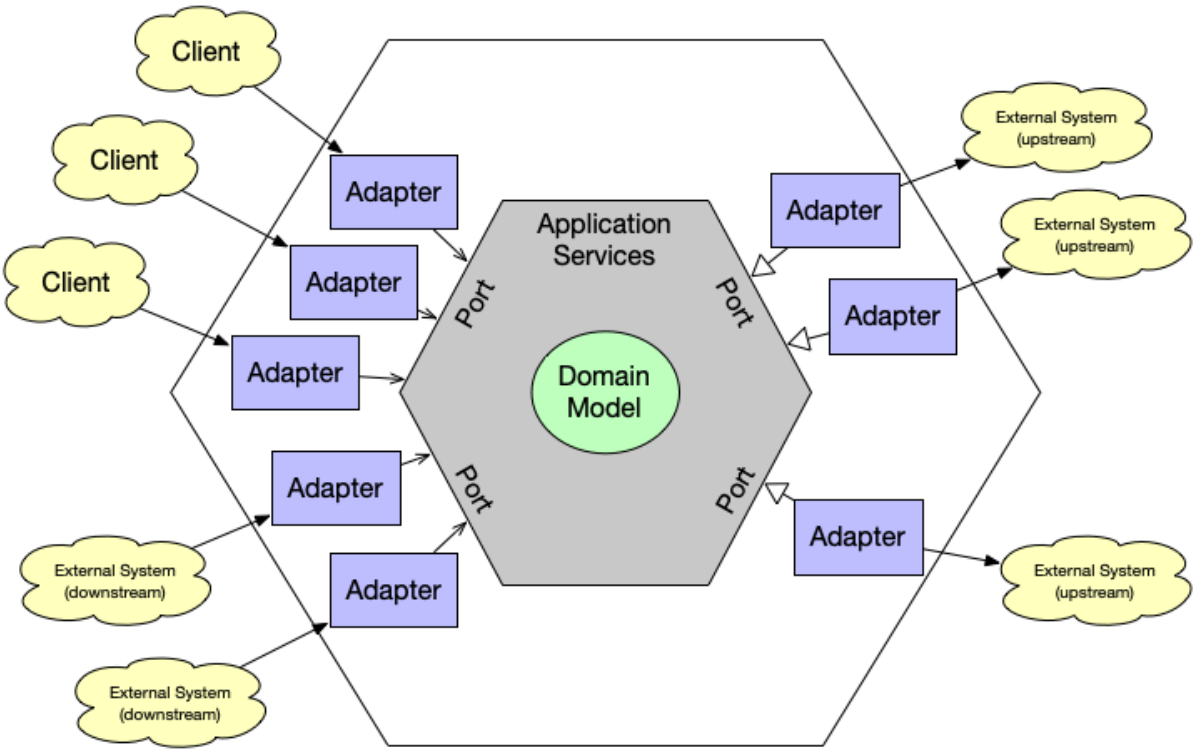
- dataSource: obtengo los datos.
- filter1, filter2: filtro los datos.
- dataSink: "vuelco" los datos, escribo un archivo, etc.

Ejemplos: HTTP Filters, UNIX commands, compiler.

HEXAGONAL ARCHITECTURE (PORTS & ADAPTERS)

"Lo que importa de una aplicación son los casos de uso"

PORTS: interfaces de adaptación (no sé qué tienen adentro).
ADAPTERS: implementación de los ports.

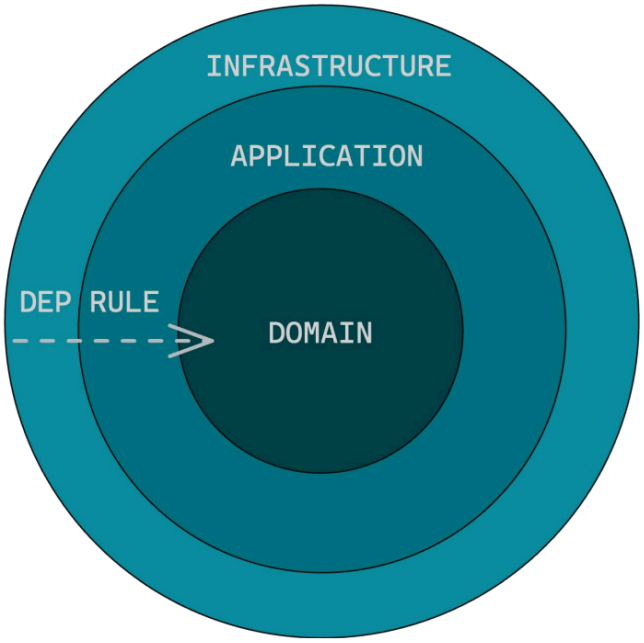


El core del sistema es la aplicación (lógica de negocio + dominio) y alrededor tenemos la implementación.

Diagrama circular

- Dominio: entidades + reglas de negocio.
- Aplicación: casos de uso (procesos de negocio).
- Infraestructura: detalles de implementación (periferia).

DEP RULE: regla de dependencias.



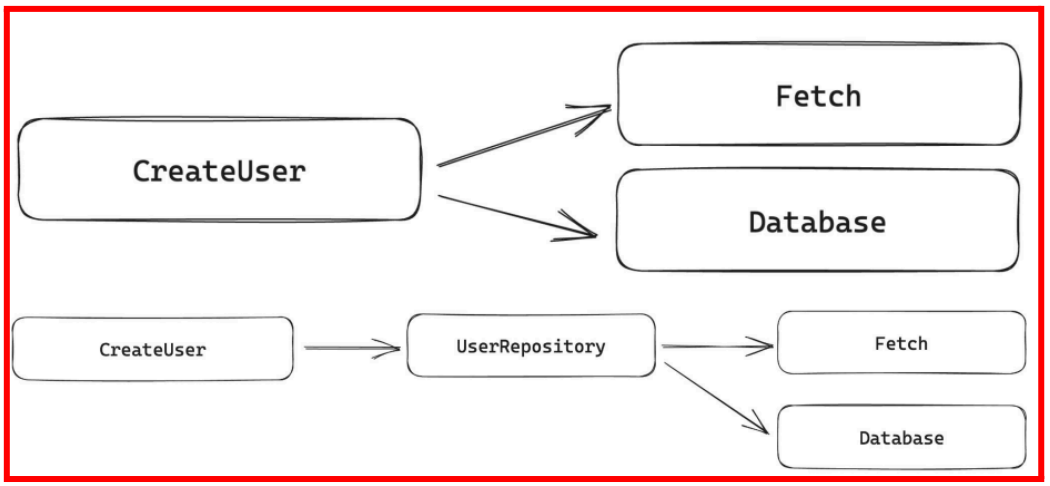
Ejemplo de uso:

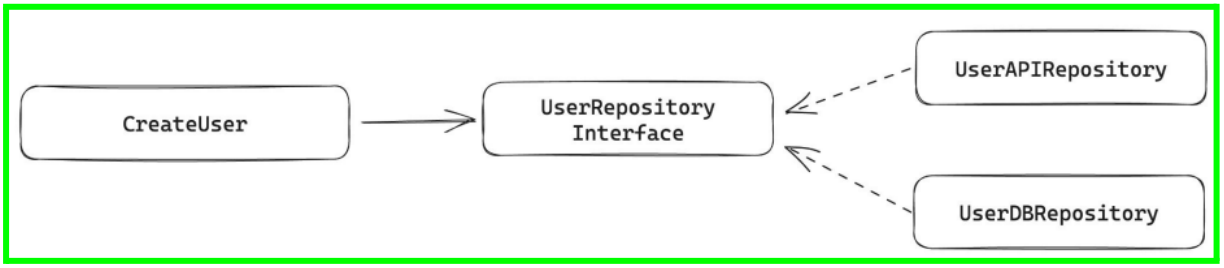
```
export function createUser(user: User) {
  return db.collection('users').add(user);
}

export function findUser(userId: string) {
  return db.collection('users').doc(userId).get();
}

export function updateUser(userId: string, user: User) {
  return db.collection('users').doc(userId).update(user);
}

export function deleteUser(userId: string) {
  return db.collection('users').doc(userId).delete();
}
```

Tenemos acoplamiento a un servicio de la infraestructura (db, database). Esto se extendería de la misma manera si quiero testear. ¿Cómo lo resuelvo? encapsulamiento: crear una abstracción (una clase, un port) del fetch, que simule lo que hace.

Repository pattern

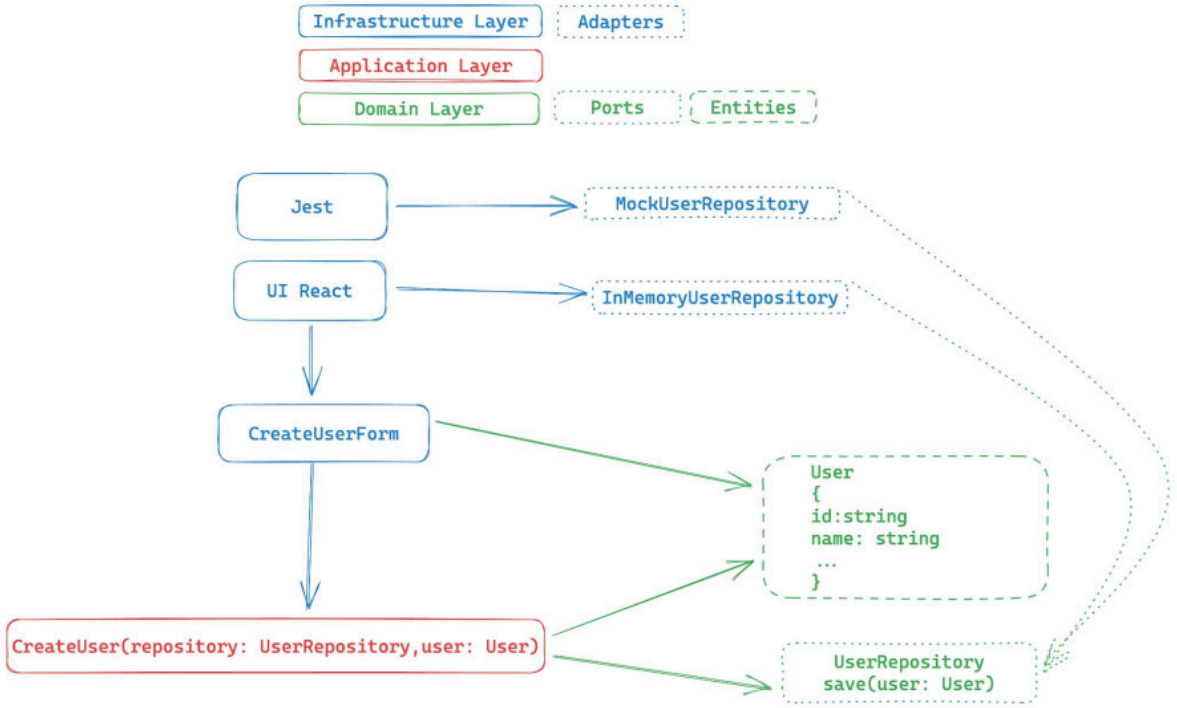
Todavía tengo un new UserRepository() que puedo solucionar envíandolo por parámetro a la función createUser, esto es aplicar el patrón *Dependency Inversion Principle* (módulos de alto nivel no deberían depender de módulos de bajo nivel).





¿y el new() ahora dónde se hace? usamos el Dependency Injection Principle, cuando arranque el sistema se hace por única vez el new() en el main.

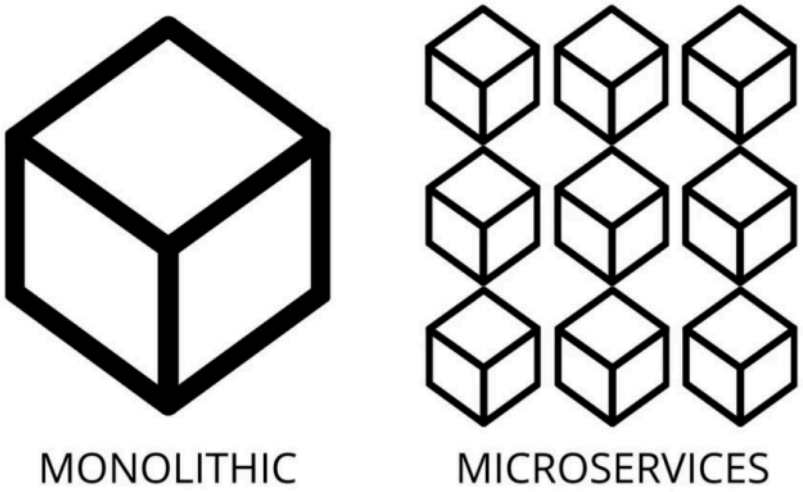
```
function createUser(user: User, repository: UserRepository): Promise<void>
@param user — a user object
@param repository — a repository that implements the UserRepository interface
@returns — a promise that resolves to the created user
@example
const user = await createUser({ name: 'John', email: 'xx@xx.com', age: 30 }, new
UserAPIRepository());
console.log(user);
// { name: 'John', email: 'xx@xx', age: 30 }
@example
const user = await createUser({ name: 'John', email: 'xx@xx.com', age: 30 }, new
UserDBRepository());
return repository.save(user);
```



FOLDER STRUCTURE

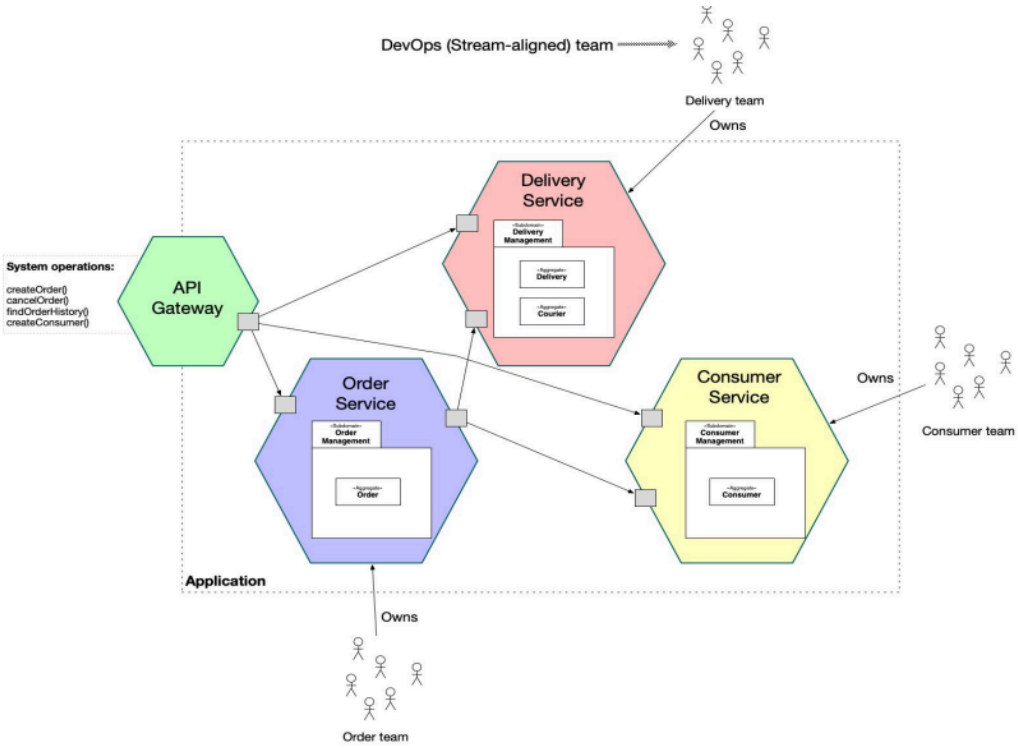
1 carpeta por capa.

MICROSERVICES (servicios más pequeños)



Cada microservicio tiene un diseño propio (arquitectura, patrones, etc.) y se pueden comunicar entre sí.

En el modelo del ejemplo la API Gateway controla el acceso y es el único servicio accesible al usuario, el resto solo puede acceder el equipo de desarrollo del mismo.



MICROFRONTEDS