Repositorios

- Principios Solid
- Definición
- Análisis
- Diferentes Approach

S.O.L.I.D.

Los principios SOLID son un conjunto de cinco principios de diseño de software que se utilizan para crear código limpio, mantenible y escalable en la programación orientada a objetos. Estos principios fueron introducidos por Robert C. Martin y se convirtieron en pautas esenciales para el desarrollo de software de calidad. Cada letra en "SOLID" representa un principio específico:

S - Principio de Responsabilidad Única (Single Responsibility Principle, SRP)

O - Principio de Abierto/Cerrado (Open/Closed Principle, OCP)

L - Principio de Sustitución de Liskov (Liskov Substitution Principle, LSP)

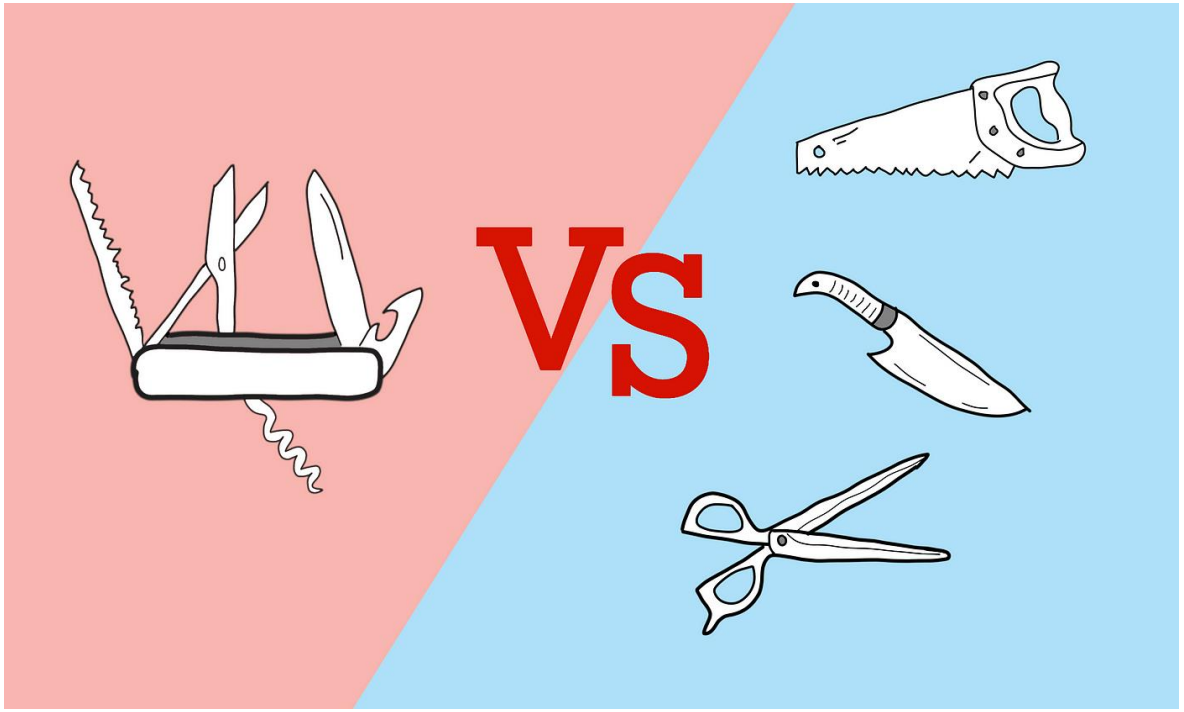
I - Principio de Segregación de Interfaces (Interface Segregation Principle, ISP)

D - Principio de Inversión de Dependencia (Dependency Inversion Principle, DIP)

Single responsibility principle.

Una clase o módulo debe tener una sola razón para cambiar

The Single Responsibility Principle (SRP) states that each software module should have one and only one reason to change.
— Robert C. Martin



https://levelup.gitconnected.com/the-single-responsibility-principle-made-simple-4e1597a44d7d

Single responsibility principle.

Ayuda a mantener el código más organizado, modular y fácil de mantener. Si una clase tiene múltiples responsabilidades, se vuelve más difícil de entender, modificar y probar.

Single responsibility principle.

Interacción entre partes

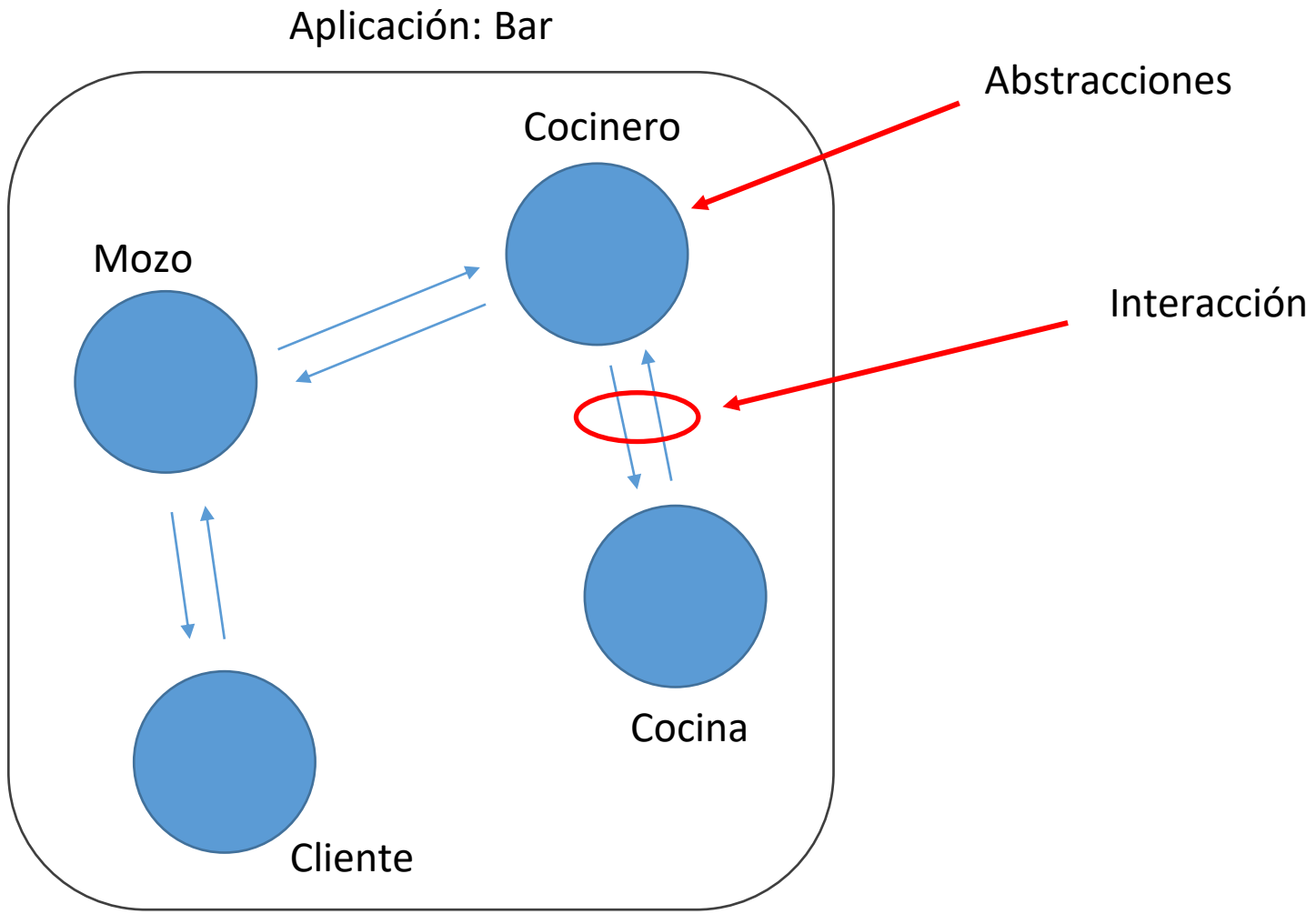
Cocinero

Mozo

Cliente

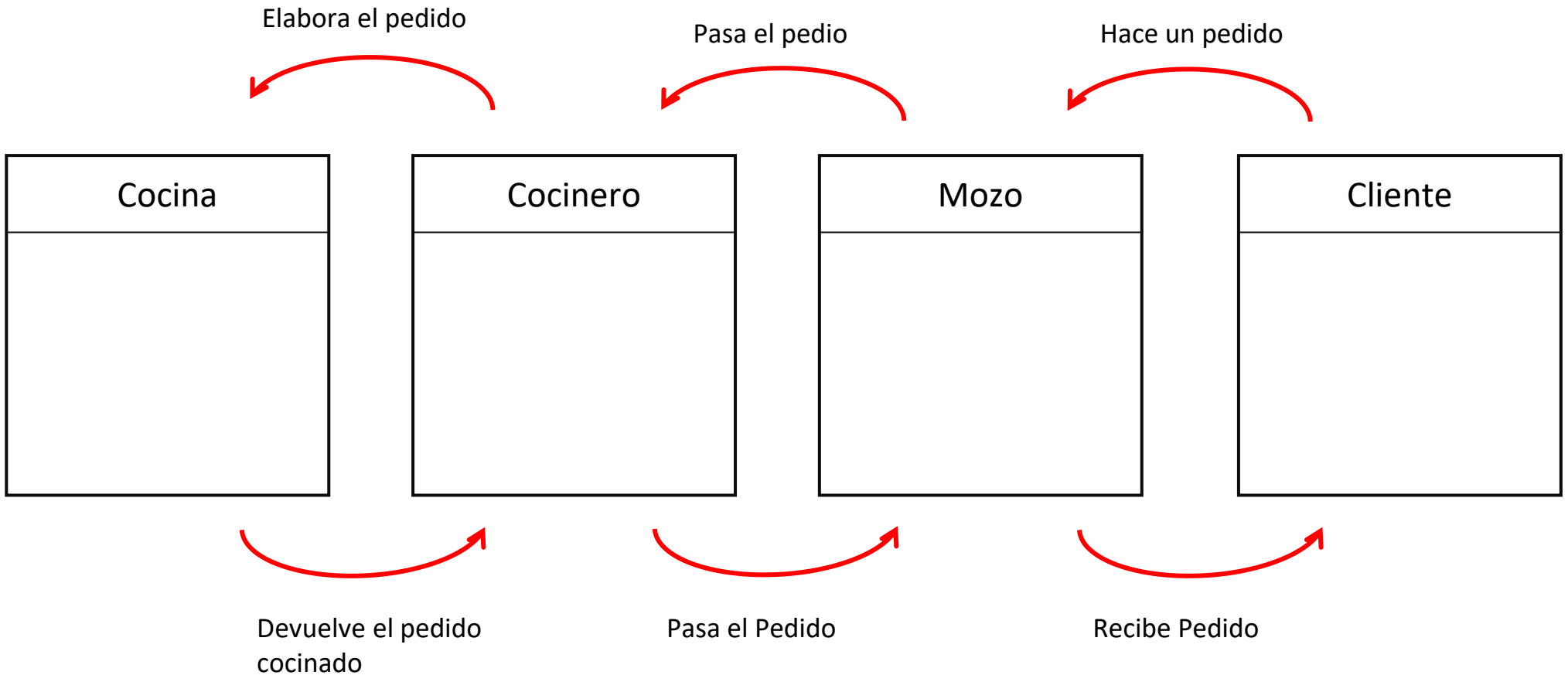
Arquitectura de una aplicación POO

Simulando un bar con clases



Arquitectura de una aplicación POO

Interacción entre partes



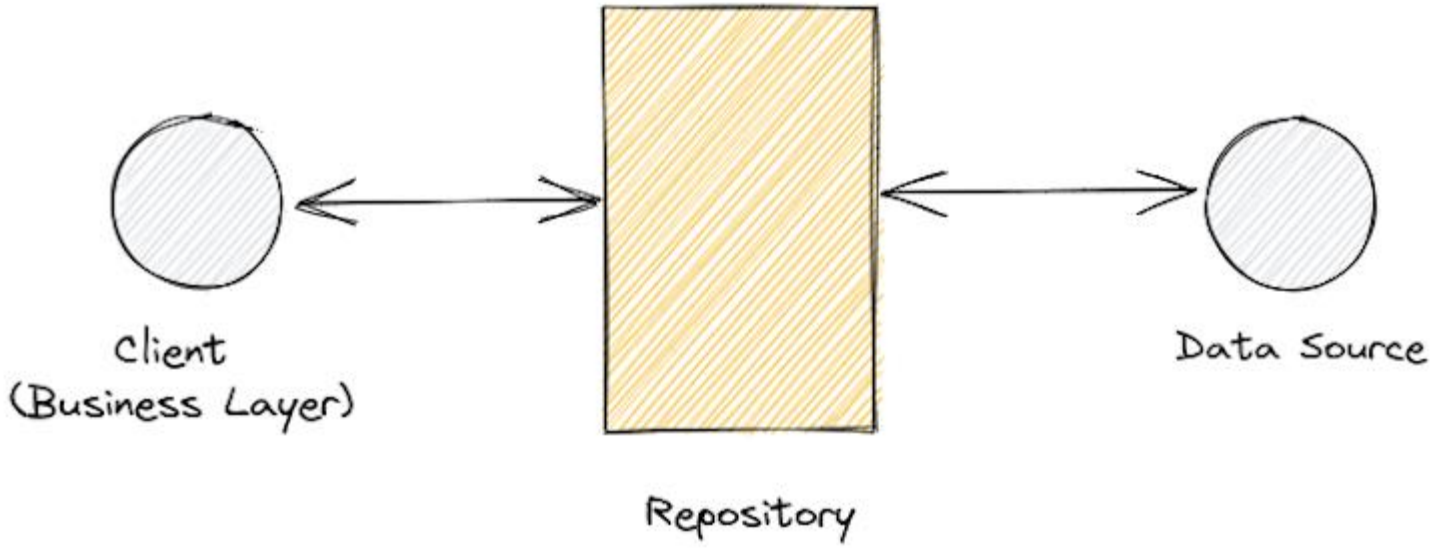
Aplicando el principio de responsabilidad única al uso de bases de datos: El patrón repositorio.

Repositorio

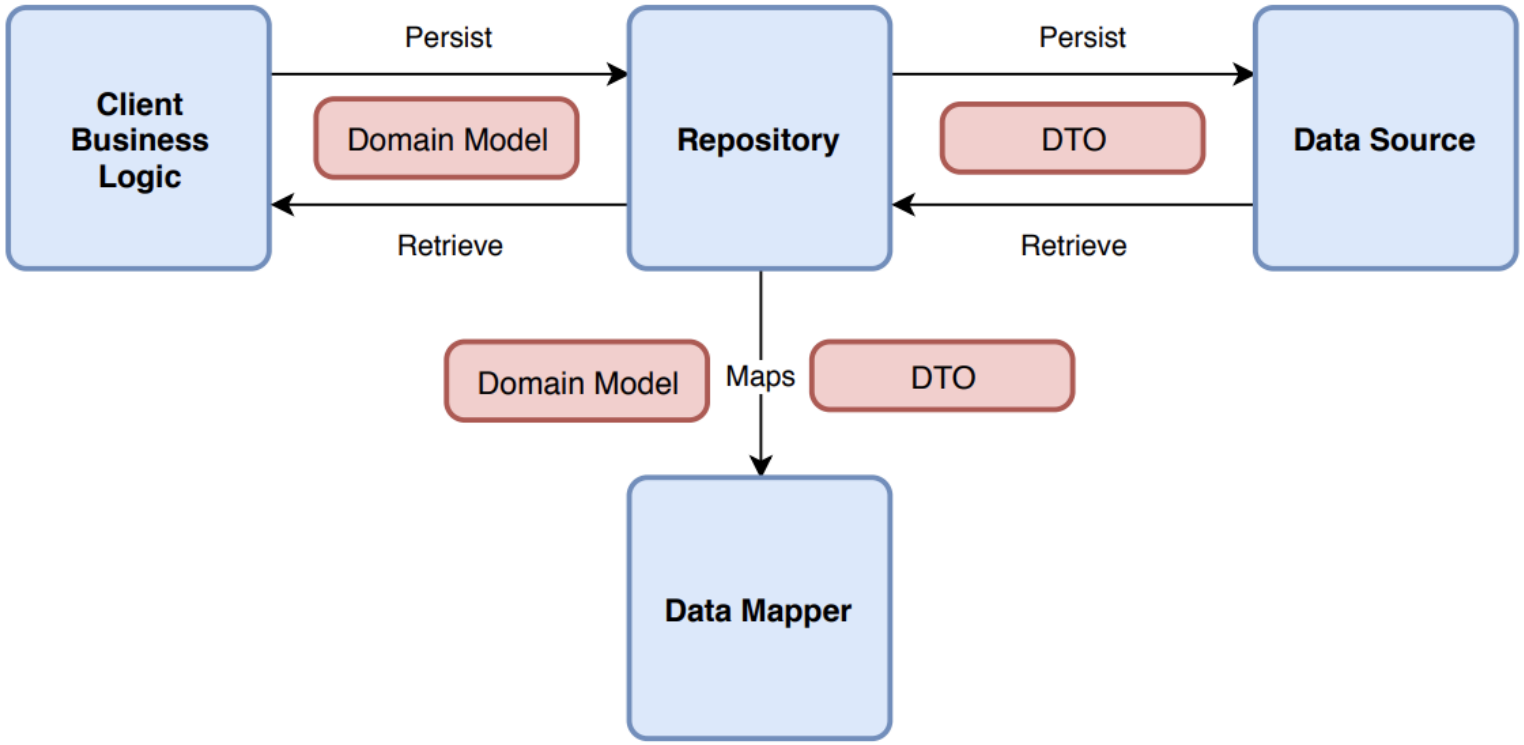
Es un patrón para trabajar con datos externos a la aplicación. Como ser: Json, Servicios Web, Documentos, Bases de datos, etc

Repositorio

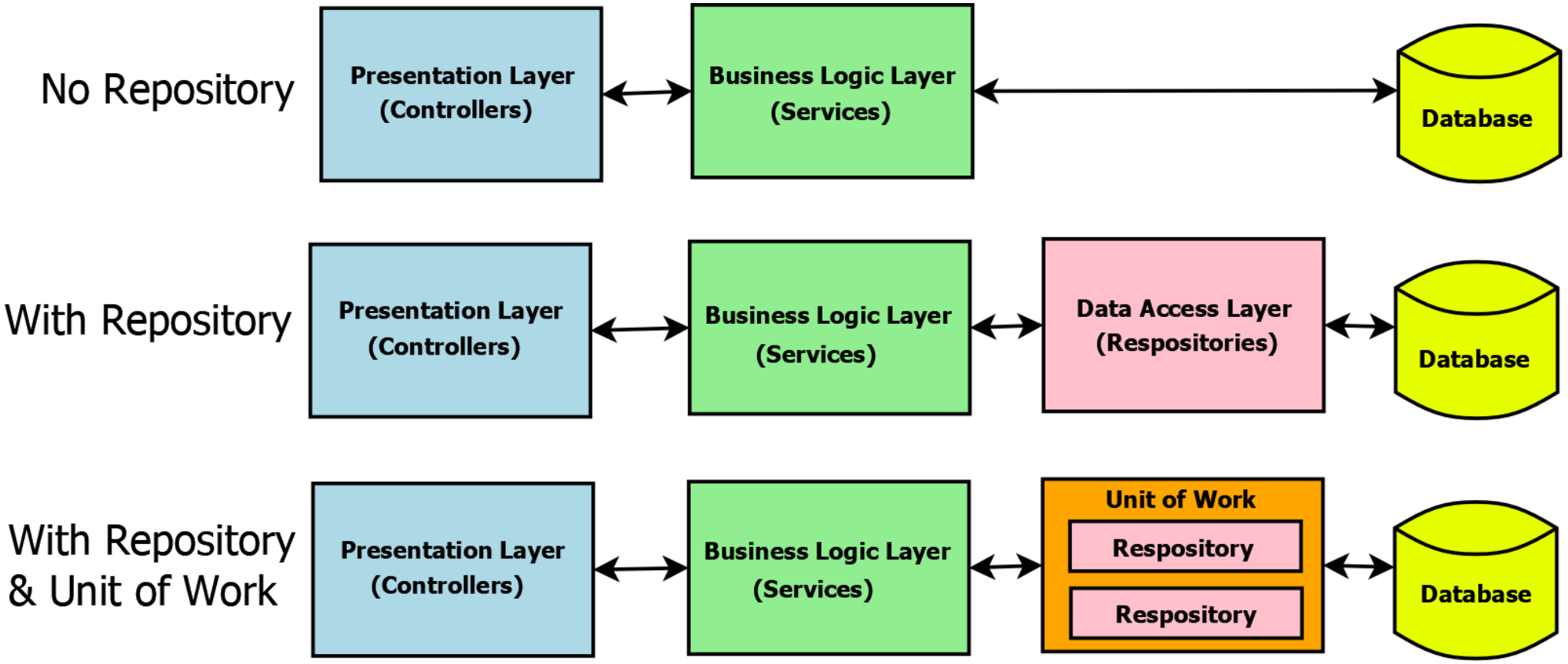
El repositorio está diseñado para crear una capa de abstracción entre la capa de acceso a datos y la capa de lógica de negocios de una aplicación.



Repositorio



Repositorio



```
public class UserRepository : IRepository
 {
    public List<User> GetAll()
    {
       //devolver todos los usuarios
    }
    public User GetById(int id)
    {
       //código que devuelve un usuario del repositorio filtrando por by Id
    }
    public void Create(User usuario)
    {
       //Código para crear un usuario
    }
    public void Update(User usuario)
    {
       //Codigo para actualizar un usuario
    }
    public void Delete(User usuario)
    {
       //Codigo para eliminar un usuario
    }
 }
```

Repositorio

```
public interface IUserRepository
{
        User FindById(int id);
        IEnumerable FindAll();
        void Insert(User usuario);
        void Update(User usuario);
        void Delete(int idUsuario);
}
```

Bibliografía

https://exceptionnotfound.net/dependency-injection-in-dotnet-6-service-lifetimes/

https://exceptionnotfound.net/dependency-injection-in-dotnet-6-adding-and-injecting-dependencies/

https://exceptionnotfound.net/dependency-injection-in-dotnet-6-intro-and-background/

https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff649690(v=pandp.10)?redirectedfrom=MSDN

https://learn.microsoft.com/es-es/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application