**📌 Arquitectura Backend – Plataforma de Publicidad**

Este diseño de la arquitectura del backend **modular, escalable y seguro** para la plataforma de publicidad, aplicando:

* **DDD (Domain-Driven Design)** → el negocio como núcleo de la aplicación.
* **Principios SOLID** → bajo acoplamiento y alta cohesión.
* **Monolito Modular** → estructura organizada, rápida de implementar y preparada para crecer.

**📌 Capas de la Arquitectura**

**1. Domain Layer (Dominio)**

* Representa el **modelo del negocio**.
* **Contiene:**
  + **Entidades:** User, Provider, Rubro, Rating, Comment.
  + **Value Objects:** encapsulan atributos con reglas propias
    - Email → valida formato de correo.
    - Password → asegura políticas de seguridad.
    - Score → garantiza calificaciones entre 1 y 5.
  + **Interfaces de Repositorio:** contratos que definen cómo acceder a las entidades (ej. ProviderRepository).

**Justificación:**  
Esta capa asegura que el **núcleo del negocio sea independiente de frameworks o bases de datos**.  
Gracias a las interfaces de repositorio, el dominio no depende de la tecnología usada para la persistencia.

**2. Service Layer (Servicios)**

* Orquesta la lógica del negocio.
* **Contiene:**
  + Interfaces de servicio (UserService, ProviderService).
  + Implementaciones (UserServiceImpl, ProviderServiceImpl).
  + Reglas de negocio y orquestación de entidades y repositorios.
  + Servicios de dominio adicionales para lógica compleja.

**Justificación:**  
Permite **centralizar las reglas de negocio** y mantener claridad sin generar un exceso de clases.  
Favorece la cohesión, ya que cada servicio agrupa funcionalidades relacionadas con un módulo específico (usuarios, prestadores, rubros, etc.).

**3. Infrastructure Layer (Infraestructura)**

* Implementa los contratos definidos en el dominio.
* **Contiene:**
  + Repositorios JPA (implementaciones de ProviderRepository, UserRepository, etc.).
  + Configuración de seguridad (Spring Security con JWT/OAuth2).
  + Persistencia y acceso a la base de datos.
  + Integraciones externas (ej. Google/Facebook para login, envío de emails).

**Justificación:**  
Encapsula la interacción con frameworks y servicios externos, manteniendo el dominio libre de dependencias técnicas.

**4. Controller Layer (Controladores)**

* Expone la API REST para el frontend.
* **Contiene:**
  + Controladores REST (UserController, ProviderController).
  + DTOs de entrada y salida.
  + Validaciones (@Valid).
  + Manejadores globales de excepciones (@ControllerAdvice).

**Justificación:**  
Separa la lógica de presentación y validación de la lógica de negocio, asegurando un flujo claro entre el frontend y el backend.

**5. Shared Layer (Capa Común)**

* Contiene elementos **reutilizables y transversales**.
* **Contiene:**
  + Excepciones personalizadas (BusinessException, NotFoundException).
  + DTOs genéricos (ApiResponse, ErrorResponse).
  + Utilidades (validaciones, formateo de datos).
  + Constantes globales (mensajes de error, configuraciones compartidas).

**Justificación:**  
Centraliza el código compartido entre módulos, reduciendo duplicación y facilitando el mantenimiento.

**📌 Módulos de Dominio**

* **Usuarios** → Registro, login, autenticación.
* **Prestadores** → Gestión de datos de prestadores.
* **Rubros** → Alta, edición, baja lógica y listado de categorías.
* **Calificaciones** → Sistema de estrellas 1–5.
* **Comentarios** → Opiniones de usuarios sobre prestadores.
* **Administración** → Panel de control y estadísticas.
* **Shared** → Utilidades, excepciones y DTOs comunes.
* **Configuración** → Beans, seguridad y properties de la aplicación.

**📌 Principios Aplicados**

**Single Responsibility Principle (SRP)** → cada capa tiene una responsabilidad clara.

**Dependency Inversion Principle (DIP)** → el dominio depende de interfaces, no de implementaciones técnicas.

**Open/Closed Principle (OCP)** → el sistema puede extenderse con nuevos módulos sin modificar los existentes.

**DDD** → el modelo refleja fielmente las reglas del negocio.

**Monolito Modular** → organización por módulos independientes dentro de un mismo despliegue, facilitando la escalabilidad futura.

📃VISUALIZACION GRAFICA DE LA ESTRCUTURA:

Src/

shared/ # Código común y reutilizable

dto/ # DTOs genéricos (ApiResponse, ErrorResponse)

exceptions/ # Excepciones personalizadas

utils/ # Utilidades y helpers

constants/ # Constantes globales (mensajes, configuraciones)

config/ # Configuración global de la aplicación

security/ # Seguridad (JWT, OAuth2, roles)

persistence/ # Configuración JPA, DataSource

general/ # Beans, properties, logging

users/ # Módulo de Usuarios

domain/ # Entidades, Value Objects, Repositorios (interfaces)

service/ # Interfaces y lógica de negocio

infrastructure/ # Implementaciones concretas de repositorios

controller/ # Controladores REST y DTOs

providers/ # Módulo de Prestadores

domain/

service/

infrastructure/

controller/

rubros/ # Módulo de Rubros

domain/

service/

infrastructure/

controller/

ratings/ # Módulo de Calificaciones

domain/

service/

infrastructure/

controller/

comments/ # Módulo de Comentarios

domain/

service/

infrastructure/

controller/

admin/ # Módulo de Administración (estadísticas y panel)

domain/

service/

infrastructure/

controller/