Contenido

¿Qué es la Lógica de Negocio?

- Definición: Reglas y procesos que definen cómo opera una organización
- Importancia en aplicaciones fullstack
- Evolución en el contexto de arquitecturas modernas

Tendencias en Arquitecturas Fullstack

- Microservicios
- Serverless
- JAMstack
- Edge Computing

Microservicios y Lógica de Negocio

- Descomposición de la lógica de negocio
- Beneficios: escalabilidad, mantenibilidad
- Desafíos: consistencia de datos, latencia

Patrones de Diseño para Lógica de Negocio

- Domain-Driven Design (DDD)
- CQRS (Command Query Responsibility Segregation)
- Event Sourcing

Ejemplo: Domain-Driven Design

```typescript
// Dominio: Gestión de Pedidos
class Pedido {
  private items: LineaPedido[] = [];

  agregarItem(producto: Producto, cantidad: number) {
    this.items.push(new LineaPedido(producto, cantidad));
  }

  calcularTotal(): number {
    return this.items.reduce((total, item) =>
      total + item.calcularSubtotal(), 0);
  }
}

class LineaPedido {
  constructor(
    private producto: Producto,
    private cantidad: number
  ) {}

  calcularSubtotal(): number {
    return this.producto.precio * this.cantidad;
  }
}
```

Lógica de Negocio en el Frontend

- State Management (Redux, MobX, Recoil)
- Hooks personalizados en React
- Computados y Watchers en Vue.js

Ejemplo: Custom Hook en React

```
import { useState, useEffect } from 'react';

function usePedido(pedidoInicial) \{
  const [pedido, setPedido] = useState(pedidoInicial);
  const [total, setTotal] = useState(0);

  useEffect(() => \{
    const nuevoTotal = pedido.items.reduce(
      (sum, item) => sum + item.precio * item.cantidad,
      0
    );
    setTotal(nuevoTotal);
  \}, [pedido]);

  const agregarItem = (item) => \{
    setPedido(prevPedido => (\{
      ...prevPedido,
      items: [...prevPedido.items, item]
    \}));
  \};
```

Lógica de Negocio en el Backend

- Clean Architecture
- Inyección de Dependencias
- Servicios y Repositorios

Ejemplo: Clean Architecture en Node.js

```
// Domain
class Pedido \{
  constructor(id, items) \{
    this.id = id;
    this.items = items;
  \}

  calcularTotal() \{
    return this.items.reduce((total, item) =>
      total + item.precio * item.cantidad, 0);
  \}
\}

// Use Case
class CrearPedidoUseCase \{
  constructor(pedidoRepository) \{
    this.pedidoRepository = pedidoRepository;
  \}

  async execute(pedidoData) \{
    const pedido = new Pedido(null, pedidoData.items);
    return this.pedidoRepository.save(pedido);
  \}
\}
```

```
// Controller
class PedidoController \{
  constructor(crearPedidoUseCase) \{
    this.crearPedidoUseCase = crearPedidoUseCase;
  \}

  async crearPedido(req, res) \{
    const pedido = await this.crearPedidoUseCase.execute(req.body);
    res.json(pedido);
  \}
\}
```

Tendencias Emergentes en Lógica de Negocio

- AI y Machine Learning integrados
- Blockchain para lógica de negocio descentralizada
- Low-Code y No-Code para definición de reglas de negocio

Ejemplo: Integración de AI en Lógica de Negocio

```
def recomendar(self, descripcion_cliente) : resultado =
    self.modelo(descripcion_cliente)categoria = resultado[0][´label´]
if categoria == 'tech':  return "Smartphone
última generación" elif categoria == 'outdoor':  return "Equipo de
    camping" else:  return "Libro best-seller"  Uso recomendador =
    RecomendacionProducto() sugerencia = recomendador.recomendar(
    "Cliente joven interesado en aventuras al aire libre" )
    print(sugerencia)  Output:  "Equipo de camping"
```

Mejores Prácticas

- Separación clara de responsabilidades
- Tests unitarios y de integración
- Documentación exhaustiva
- Monitoreo y logging de la lógica de negocio
- Versionado de API y lógica de negocio