Principios SOLID en la vida real con Node.js + NestJS

S – Single Responsibility Principle

Una clase debe tener una sola razón para cambiar.

Ejemplo malo:

```
@Injectable()
export class OrdersService {
  createOrder(orderDto: CreateOrderDto) {
    // 1. Validar datos
    // 2. Calcular impuestos
    // 3. Guardar en BD
    // 4. Enviar email
  }
}
```

Ejemplo bueno:

```
@Injectable()
export class OrdersService {
  constructor(
    private readonly taxService: TaxService,
    private readonly repository: OrdersRepository,
    private readonly emailService: EmailService,
  ) {}

  async createOrder(orderDto: CreateOrderDto) {
    const totalWithTax = this.taxService.applyTax(orderDto);
    const order = await this.repository.save(orderDto, totalWithTax);
    await this.emailService.sendOrderConfirmation(order);
  }
}
```

O – Open/Closed Principle

Abierto para extensión, cerrado para modificación.

```
export abstract class ShippingStrategy {
  abstract calculate(order: Order): number;
}

export class NationalShipping extends ShippingStrategy {
  calculate(order: Order) {
    return 5;
  }
}
```

```
export class InternationalShipping extends ShippingStrategy {
  calculate(order: Order) {
    return 15 + order.weight * 2;
  }
}
```

L – Liskov Substitution Principle

Las subclases deben poder reemplazar a la clase base sin alterar el comportamiento.

```
export interface PaymentGateway {
  pay(amount: number): Promise<boolean>;
}

export class PayPalGateway implements PaymentGateway {
  async pay(amount: number) {
    return true;
  }
}

export class StripeGateway implements PaymentGateway {
  async pay(amount: number) {
    return true;
  }
}
```

I – Interface Segregation Principle

Es mejor tener muchas interfaces específicas que una genérica gigante.

```
export interface EmailNotifier {
  sendEmail(): void;
}

export interface SMSNotifier {
  sendSMS(): void;
}
```

D – Dependency Inversion Principle

Depender de abstracciones, no de implementaciones concretas.

```
export interface OrdersRepository {
  save(order: Order): Promise<Order>;
}

@Injectable()
export class TypeOrmOrdersRepository implements OrdersRepository {
  async save(order: Order) {
    return order;
  }
}
```

```
@Injectable()
export class OrdersService {
  constructor(private readonly repository: OrdersRepository) {}

  createOrder(order: Order) {
    return this.repository.save(order);
  }
}
```