PRUEBA TÉCNICA CLARISA

Explicación de los practicas y principios de ingeniería

Practicas y principios de ingeniería en front

Prácticas aplicadas

1. Separación de responsabilidades (Separation of Concerns)

Modelos en core/models: Centraliza la definición de interfaces de datos (CufePayload, ResultCufe), promoviendo reutilización y mantenimiento.

Servicios en services: Aislas la lógica de comunicación con el backend (CufeService) del componente, permitiendo pruebas unitarias e independencia de la vista.

Componentes separados de estilos y lógica: El HTML (register-bill.component.html) está desacoplado del SCSS y del TypeScript, siguiendo una estructura clara.

2. Uso adecuado de Angular Forms (Reactive Forms)

Implementación de formGroup, validación reactiva ([disabled]="!cufeForm.valid", invalid, touched), lo cual mejora la robustez y facilita pruebas.

3. Estilos con nomenclatura BEM

Las clases siguen una convención de tipo BEM (.cufe-form__field, .cufe-form__error, etc.), lo cual mejora la legibilidad y evita colisiones de CSS.

4. Inyección de dependencias

El servicio CufeService usa inyección de dependencias a través del constructor: principio de inversión de dependencias aplicado correctamente.

Arquitectura del proyecto aplicada

Arquitectura Modular por Features (Feature-Sliced Architecture)
features/cufe/...: indica una arquitectura por dominio funcional (modularización basada en funcionalidades).

core/models: carpeta centralizada para modelos reutilizables en todo el proyecto.

Principios de Ingeniería de Software aplicados

1. SOLID

S: Single Responsibility Principle
Cada archivo cumple una única responsabilidad (el servicio genera CUFE, el componente muestra el formulario, etc.).

D: Dependency Inversion Principle

El componente depende de una abstracción (CufeService) en lugar de una implementación directa.

2. DRY (Don't Repeat Yourself)

Campos del formulario renderizados dinámicamente mediante *ngFor → no hay repetición innecesaria de HTML.

Estilos reutilizados con clases bien nombradas.

3. KISS (Keep It Simple, Stupid)

Lógica del componente, servicio y modelos es directa, sencilla y mantenible.

No se sobrecomplica la implementación.

4. YAGNI (You Aren't Gonna Need It)

Prácticas y principios de ingeniería en back

Practicas aplicadas

- Principios Solid
- Clean code
- Uso de anotaciones

Arquitectura del proyecto aplicada

- Arquitectura Hexagonal (Ports & Adapters): Aunque no se especificaba una arquitectura hexagonal, quise aplicar clean architecture.

Principios de Ingeniería de Software aplicados

1. Principios de Clean Code
   Legibilidad: El método generateCufe es directo, conciso y fácil de entender.
   Nombres significativos: generateCufe, concatenated, hashBytes, etc., describen bien lo que hacen.
   Evita comentarios innecesarios: El código se explica solo.
   Separación lógica clara: Primero concatena, luego genera el hash.

2. Principios SOLID aplicados
   - S — Responsabilidad Única (SRP)

   CufeGeneratorService solo tiene una responsabilidad: generar el hash CUFE a partir de los datos de entrada.

   - D — Inversión de Dependencias (DIP)

   El dominio define el contrato (CufeUseCase) y la clase de aplicación lo implementa.

Datos de prueba

Datos para llenar formulario

ciTec: ABC1234567890DEF"
fecFac: 2025-07-03
horFac: 10:45:00-05:00
nitFE → 900123456
numAdq → 123456789
numFac → FE12345
tipoAmbiente → 2
valFac → 1000.00
valTot → 1190.00
valImp1 → 190.00
valImp2 → 0.00
valImp3 → 0.00

Probar BackEnd

```
curl --location 'http://localhost:8080/api/cufe' \
--data '{
  "numFac": "FE12345",
  "fecFac": "2025-07-03",
  "horFac": "10:45:00-05:00",
  "valFac": "1000.00",
  "codImp1": "01",
  "valImp1": "190.00",
  "codImp2": "04",
  "valImp2": "0.00",
  "codImp3": "03",
  "valImp3": "0.00",
  "valTot": "1190.00",
  "nitFE": "900123456",
  "numAdq": "123456789",
  "ciTec": "ABC1234567890DEF",
  "tipoAmbiente": "2"
}
'
```