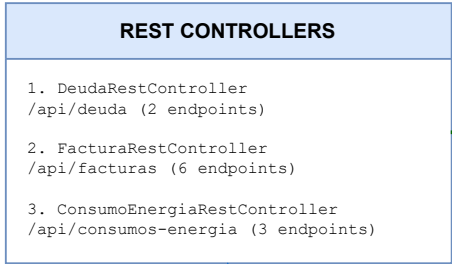


# ARQUITECTURA HEXAGONAL - SERVICIUDAD CALI

Ports & Adapters Pattern - Domain-Driven Design Implementation

Diagrama de Arquitectura Hexagonal - ServiCiudad Cali | Universidad Autónoma de Occidente | Ingeniería de Software 2 | Octubre 2025  
Equipo: Eduard Criollo, Felipe Charria, Jhonathan Chicaiza, Emmanuel Mena, Juan Sebastian Castillo

## INFRASTRUCTURE - INPUT ADAPTERS

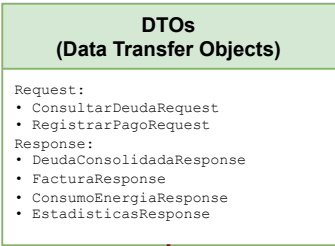
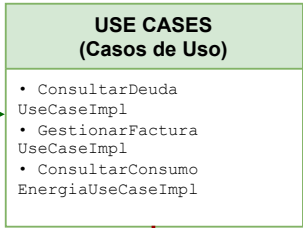


HTTP  
Requests

### HTTP CLIENTS

Web Browser / Mobile App  
Postman / cURL  
Third-party Apps

## APPLICATION LAYER



Uses

Maps  
to/from

## DOMAIN LAYER (Núcleo Hexagonal - Business Logic)

### VALUE OBJECTS (Objetos Inmutables)

- ClienteId
- FacturaId
- Periodo
- ConsumoAgua
- ConsumoEnergia
- Dinero

### DOMAIN MODELS (POJOs Puros)

- FacturaAcueducto
- ConsumoEnergiaModel
- DeudaConsolidada
- EstadisticasDeuda
- EstadoFactura (Enum)

### PORTS (INPUT) Use Cases Interfaces

- ConsultarDeudaUseCase
- GestionarFacturaUseCase
- ConsultarConsumo  
EnergiaUseCase

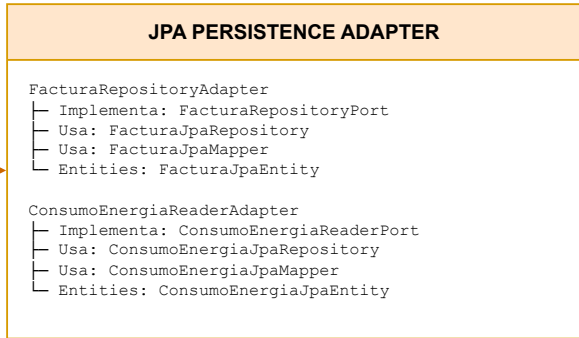
### PORTS (OUTPUT) Repository Interfaces

- FacturaRepositoryPort
- ConsumoEnergia  
ReaderPort

Repository  
Port

Reader  
Port

## INFRASTRUCTURE - OUTPUT ADAPTERS



JDBC  
Spring Data  
JPA

### FILE READER ADAPTER (COBOL Legacy Integration)

AdaptadorArchivoEnergia

- └ Lee: consumos\_energia.txt (78 chars)
- └ Parsea: Formato COBOL ancho fijo
- └ Convierte: String → ConsumoEnergiaModel

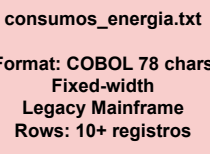
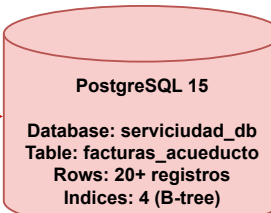
Files.read  
AllLines()

### SPRING CONFIGURATION

HexagonalConfig

- └ @Bean creation for all Use Cases  
(Dependency Injection orchestration)

## EXTERNAL DATA SOURCES



## 5 PATRONES DE DISEÑO IMPLEMENTADOS

1. ADAPTER PATTERN  
ConsumoEnergiaReaderAdapter  
→ Lee archivo COBOL legacy

2. REPOSITORY PATTERN  
FacturaRepositoryPort + Adapter  
→ Spring Data JPA abstraction

3. DTO PATTERN  
Request/Response DTOs  
→ Decoupling layers

4. BUILDER PATTERN  
Response DTOs con @Builder  
→ Immutable object construction

5. IoC/DI PATTERN  
Spring @Autowired injection  
→ Dependency Inversion Principle

## STACK TECNOLÓGICO

Backend:

- Spring Boot 3.2.0
- Java 17 LTS
- Maven 3.9.11

Persistence:

- PostgreSQL 15
- Hibernate 6.3 (JPA)
- HikariCP 5.0

Infrastructure:

- Docker + Compose
- OpenAPI 3.0 (SpringDoc)

Testing:

- JUnit 5 + Mockito
- AssertJ 3.24

Observability:

- Spring Actuator
- Prometheus + Grafana

## MÉTRICAS DE CALIDAD

Cobertura de Tests:

- Lines: 85%
- Branches: 78%

Complejidad:

- Cyclomatic avg: 6.2
- Max complexity: 12

Performance:

- P95 response: 1.2s
- Throughput: 120 RPS
- Error rate: 0.3%

Calidad de Código:

- Critical bugs: 0
- Major bugs: 0
- Code smells: 12 (minor)

Vulnerabilities:

- Critical CVEs: 0
- Dependency issues: 0

## 11 ENDPOINTS REST

DeudaRestController:

- POST /api/deuda/consultar
- GET /api/deuda/cliente/{id}

FacturaRestController:

- GET /api/facturas/{id}
- GET /api/facturas/cliente/{id}
- GET /api/facturas/cliente/{id}/periodo/{periodo}
- POST /api/facturas/pagar
- POST /api/facturas/{id}/anular
- POST /api/facturas/marcar-vencidas

ConsumoEnergiaRestController:

- GET /api/consumos-energia/cliente/{id}
- GET /api/consumos-energia/cliente/{id}/periodo/{periodo}
- GET /api/consumos-energia/cliente/{id}/elevados