CODE DIAGRAM - MICROSERVICIO DE PAGOS SERVICIUDAD

CONTROLLER LAYER

```
@RestController
@RequestMapping("/api/v1/pagos")
@Validated
public class PaymentController {

@Autowired
private PaymentService paymentService;

@Autowired
private PaymentMapper paymentMapper;

@PostMapping("/unificado")
@Operation(summary = "Procesar pago unificado")
public ResponseEntity
procesarPagoUnificado(
@Valid @RequestBody PagoUnificadoRequest request) {

try {
PaymentCommand command = paymentMapper
.toCommand(request);

PaymentResult result = paymentService
.procesarPagoUnificado(command);

PaymentResponseDTO response = paymentMapper
.toResponseDTO(result);

return ResponseEntity.ok(response);

} catch (PaymentValidationException e) {
return ResponseEntity.badRequest()
.body(PaymentResponseDTO.error(e.getMessage()));
} catch (PaymentProcessingException e) {
return ResponseEntity.status(500)
.body(PaymentResponseDTO.error(
"Error procesando pago"));
}
}

@GetMapping("/{id}")
public ResponseEntity
consultarEstadoPago(@PathVariable String id) {

Payment payment = paymentService
.consultarEstadoPago(id);

PaymentStatusDTO status = paymentMapper
.toStatusDTO(payment);

return ResponseEntity.ok(status);
}

@PostMapping("/{id}/cancelar")
public ResponseEntity cancelarPago(
@PathVariable String id) {

paymentService.cancelarPago(id);
return ResponseEntity.ok().build();
}
}
```

DOMAIN ENTITIES

```
@Entity
@Table(name = "payments")
@Builder
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Payment {

@Id
private String id;

@Column(name = "cliente_id", nullable = false)
private String clienteId;

@Column(name = "monto_total", precision = 19, scale = 2)
private BigDecimal montoTotal;

@OneToMany(cascade = CascadeType.ALL,
fetch = FetchType.LAZY,
mappedBy = "payment")
private List servicios;

@Enumerated(EnumType.STRING)
@Column(name = "estado")
private PaymentStatus estado;

@Column(name = "fecha_creacion")
private LocalDateTime fechaCreacion;

@Column(name = "fecha_procesamiento")
private LocalDateTime fechaProcesamiento;

@Column(name = "transaction_id")
private String transactionId;

@Enumerated(EnumType.STRING)
@Column(name = "metodo_pago")
private MetodoPago metodoPago;

// Métodos de negocio
public void marcarComoProcesado() {
this.estado = PaymentStatus.COMPLETED;
this.fechaProcesamiento = LocalDateTime.now();
}

public void marcarComoFallido() {
this.estado = PaymentStatus.FAILED;
}

public BigDecimal calcularComision() {
return montoTotal.multiply(
new BigDecimal("0.01")); // 1% comisión
}

public boolean tieneServicio(TipoServicio tipo) {
return servicios.stream()
.anyMatch(s -> s.getTipoServicio() == tipo);
}

public ServicioPago getServicioPorTipo(TipoServicio tipo) {
return servicios.stream()
.filter(s -> s.getTipoServicio() == tipo)
.findFirst()
.orElseThrow(() -> new IllegalArgumentException(
"Servicio no encontrado: " + tipo));
}

public PaymentRequest toPaymentRequest() {
return PaymentRequest.builder()
.amount(montoTotal)
.clienteId(clienteId)
.paymentMethod(metodoPago.toString())
.description("Pago ServiCiudad - " +
servicios.size() + " servicios")
.build();
}
}
```

SERVICE LAYER

```
@Service
@Transactional
@Slf4j
public class PaymentService {

@Autowired
private PaymentRepository paymentRepository;

@Autowired
private SagaOrchestrator sagaOrchestrator;

@Autowired
private PaymentValidator paymentValidator;

@Autowired
private ClienteService clienteService;

@CircuitBreaker(name = "payment-service")
@Retry(name = "payment-service")
@TimeLimiter(name = "payment-service")
public PaymentResult procesarPagoUnificado(
PaymentCommand command) {

log.info("Iniciando pago unificado para cliente: {}",
command.getClienteId());

// 1. Validar request
paymentValidator.validate(command);

// 2. Validar cliente ServiCiudad
boolean clienteValido = clienteService
.validarClienteServiCiudad(command.getClienteId());

if (!clienteValido) {
throw new ClienteInvalidoException(
"Cliente no válido en ServiCiudad");
}

// 3. Crear entidad Payment
Payment payment = Payment.builder()
.clienteId(command.getClienteId())
.servicios(command.getServicios())
.montoTotal(command.calcularMontoTotal())
.metodoPago(command.getMetodoPago())
.estado(PaymentStatus.PENDING)
.fechaCreacion(LocalDateTime.now())
.build();

// 4. Persistir payment
payment = paymentRepository.save(payment);

// 5. Iniciar Saga de pago distribuido
SagaResult sagaResult = sagaOrchestrator
.coordinarPagoDistribuido(payment);

// 6. Actualizar estado según resultado
if (sagaResult.isSuccess()) {
payment.marcarComoProcesado();
log.info("Pago completado exitosamente: {}",
payment.getId());
} else {
payment.marcarComoFallido();
log.error("Pago falló: {}, razón: {}",
payment.getId(), sagaResult.getError());
}

paymentRepository.save(payment);

return PaymentResult.from(payment, sagaResult);
}

@Cacheable(value = "payments", key = "#id")
public Payment consultarEstadoPago(String id) {
return paymentRepository.findById(id)
.orElseThrow(() -> new PaymentNotFoundException(
"Pago no encontrado: " + id));
}

public void cancelarPago(String id) {
Payment payment = consultarEstadoPago(id);

if (payment.getEstado() == PaymentStatus.COMPLETED) {
throw new PaymentStateException(
"No se puede cancelar un pago completado");
}

payment.setEstado(PaymentStatus.CANCELLED);
paymentRepository.save(payment);

// Ejecutar compensaciones si es necesario
if (payment.getEstado() == PaymentStatus.PROCESSING) {
sagaOrchestrator.ejecutarCompensaciones(
payment.getId());
}
}
}
```

CONFIGURATION

```
# application.yml
resilience4j:
circuitbreaker:
instances:
payment-service:
failure-rate-threshold: 50
wait-duration-in-open-state: 30s
sliding-window-size: 10
minimum-number-of-calls: 5
legacy-energy:
failure-rate-threshold: 60
wait-duration-in-open-state: 60s
slow-call-duration-threshold: 30s
legacy-water:
failure-rate-threshold: 60
wait-duration-in-open-state: 60s
slow-call-duration-threshold: 30s
retry:
instances:
payment-service:
max-attempts: 3
wait-duration: 2s
enable-exponential-backoff: true
timelimiter:
instances:
payment-service:
timeout-duration: 30s

# Kafka Configuration
spring:
kafka:
bootstrap-servers: localhost:9092
producer:
key-serializer: org.apache.kafka.common.serialization.StringSerializer
value-serializer: org.springframework.kafka.support.serializer.JsonSerializer
consumer:
group-id: payment-service
key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer

# Database Configuration
datasource:
url: jdbc:postgresql://localhost:5432/serviciudad_pagos
username: ${DB_USER:serviciudad}
password: ${DB_PASSWORD:password123}
jpa:
hibernate:
ddl-auto: validate
properties:
hibernate:
dialect: org.hibernate.dialect.PostgreSQLDialect
format_sql: true
show-sql: false

# Cache Configuration
cache:
type: redis
redis:
host: localhost
port: 6379
timeout: 2000ms
jedis:
pool:
max-active: 8
max-idle: 8
min-idle: 0

# Management/Monitoring
management:
endpoints:
web:
exposure:
include: health,info,metrics,prometheus
endpoint:
health:
show-details: always
metrics:
export:
prometheus:
enabled: true
```

SAGA ORCHESTRATOR

```
@Component
@Slf4j
public class SagaOrchestrator {

@Autowired
private PaymentGateway paymentGateway;

@Autowired
private LegacyEnergyAdapter energyAdapter;

@Autowired
private LegacyWaterAdapter waterAdapter;

@Autowired
private TelecomAdapter telecomAdapter;

@Autowired
private EventPublisher eventPublisher;

@Autowired
private SagaStateRepository sagaStateRepository;

public SagaResult coordinarPagoDistribuido(
Payment payment) {

String sagaId = UUID.randomUUID().toString();

SagaState sagaState = SagaState.builder()
.sagaId(sagaId)
.paymentId(payment.getId())
.estado(SagaStatus.STARTED)
.steps(new ArrayList<>())
.build();

sagaStateRepository.save(sagaState);

try {
// Paso 1: Procesar pago en PSE
log.info("Saga {}: Procesando pago en PSE", sagaId);

PaymentResult pseResult = paymentGateway
.procesarPago(payment.toPaymentRequest());

sagaState.addStep(SagaStep.completed(
"PSE_PAYMENT", pseResult));

if (!pseResult.isSuccess()) {
return SagaResult.failed(
"Error en procesamiento PSE: " +
pseResult.getError());
}

// Paso 2: Actualizar saldo energía
if (payment.tieneServicio(TipoServicio.ENERGIA)) {
log.info("Saga {}: Actualizando saldo energía",
sagaId);

boolean energiaOk = energyAdapter
.actualizarSaldoEnergia(
payment.getServicioPorTipo(
TipoServicio.ENERGIA));

sagaState.addStep(SagaStep.from(
"ENERGY_UPDATE", energiaOk));

if (!energiaOk) {
ejecutarCompensaciones(sagaState);
return SagaResult.failed(
"Error actualizando saldo energía");
}
}

// Paso 3: Actualizar saldo agua
if (payment.tieneServicio(TipoServicio.ACUEDUCTO)) {
log.info("Saga {}: Actualizando saldo agua",
sagaId);

boolean aguaOk = waterAdapter
.actualizarSaldoAcueducto(
payment.getServicioPorTipo(
TipoServicio.ACUEDUCTO));

sagaState.addStep(SagaStep.from(
"WATER_UPDATE", aguaOk));

if (!aguaOk) {
ejecutarCompensaciones(sagaState);
return SagaResult.failed(
"Error actualizando saldo agua");
}
}

// Paso 4: Actualizar saldo telecom
if (payment.tieneServicio(
TipoServicio.TELECOMUNICACIONES)) {

log.info("Saga {}: Actualizando saldo telecom",
sagaId);

boolean telecomOk = telecomAdapter
.actualizarSaldoTelecom(
payment.getServicioPorTipo(
TipoServicio.TELECOMUNICACIONES));

sagaState.addStep(SagaStep.from(
"TELECOM_UPDATE", telecomOk));

if (!telecomOk) {
ejecutarCompensaciones(sagaState);
return SagaResult.failed(
"Error actualizando saldo telecom");
}
}

// Paso 5: Publicar evento de éxito
PagoCompletadoEvent event =
PagoCompletadoEvent.builder()
.paymentId(payment.getId())
.clienteId(payment.getClienteId())
.montoTotal(payment.getMontoTotal())
.servicios(payment.getServicios())
.timestamp(LocalDateTime.now())
.build();

eventPublisher.publish(event);

sagaState.setEstado(SagaStatus.COMPLETED);
sagaStateRepository.save(sagaState);

log.info("Saga {} completada exitosamente", sagaId);

return SagaResult.success(pseResult.getTransactionId());

} catch (Exception e) {
log.error("Error en Saga {}: {}", sagaId, e.getMessage());

ejecutarCompensaciones(sagaState);

return SagaResult.failed(
"Error inesperado en saga: " + e.getMessage());
}
}

private void ejecutarCompensaciones(SagaState sagaState) {
log.warn("Ejecutando compensaciones para saga: {}",
sagaState.getSagaId());

// Compensar en orden inverso
List steps = sagaState.getSteps();
Collections.reverse(steps);

for (SagaStep step : steps) {
if (step.isCompleted()) {
try {
switch (step.getStepName()) {
case "PSE_PAYMENT":
paymentGateway.revertirTransaccion(
step.getTransactionId());
break;
case "ENERGY_UPDATE":
energyAdapter.revertirActualizacion(
sagaState.getPaymentId());
break;
case "WATER_UPDATE":
waterAdapter.revertirActualizacion(
sagaState.getPaymentId());
break;
case "TELECOM_UPDATE":
telecomAdapter.revertirActualizacion(
sagaState.getPaymentId());
break;
}

log.info("Compensación ejecutada para: {}",
step.getStepName());

} catch (Exception e) {
log.error("Error en compensación {}: {}",
step.getStepName(), e.getMessage());
}
}
}

sagaState.setEstado(SagaStatus.COMPENSATED);
sagaStateRepository.save(sagaState);
}
}
```