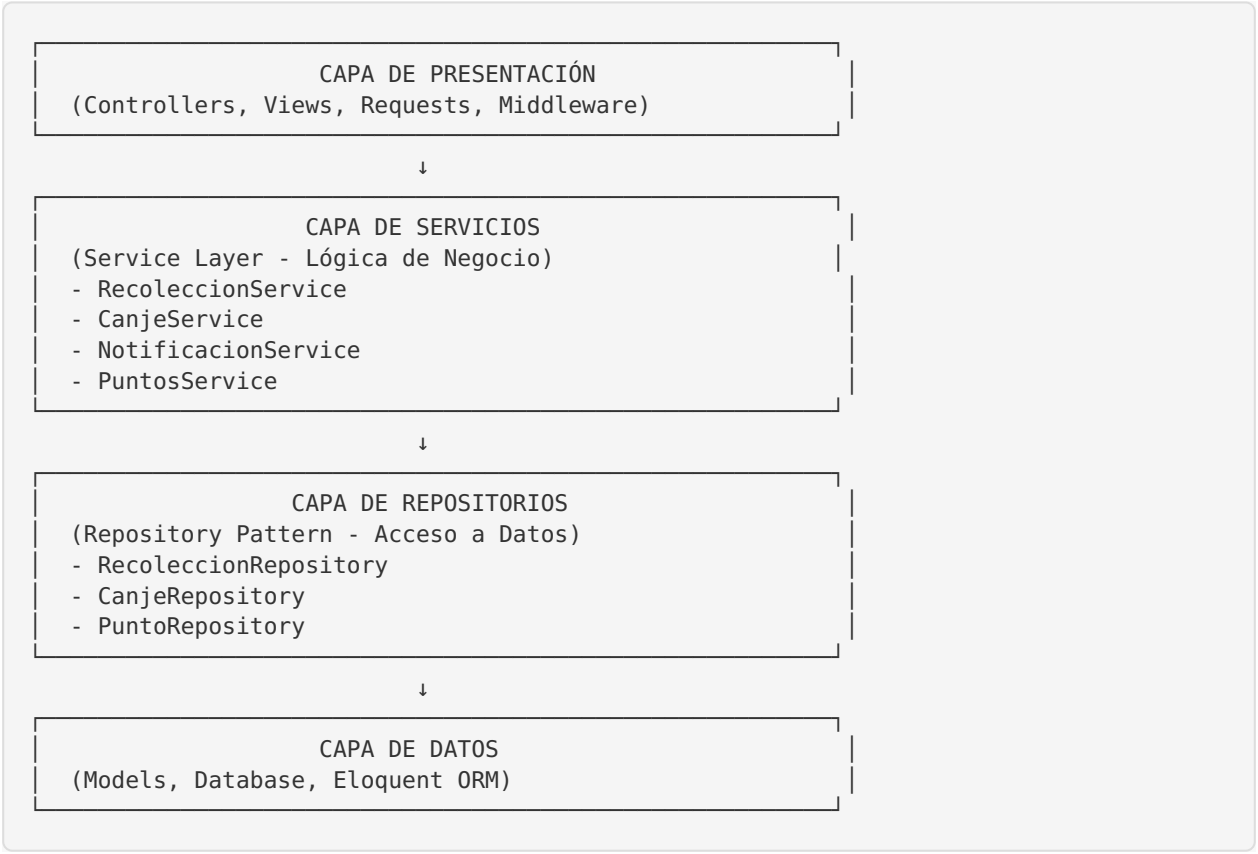
Arquitectura del Sistema EcoResiduos

📐 Visión General

EcoResiduos implementa una arquitectura en capas basada en patrones de diseño modernos para garantizar mantenibilidad, escalabilidad y testabilidad.

🏗️ Diagrama de Capas



🎯 Patrones de Diseño Implementados

1. Service Layer Pattern

Propósito: Centralizar la lógica de negocio fuera de los controladores.

Implementación:

```
// app/Services/RecoleccionService.php
class RecoleccionService
{
    public function crearRecoleccionOrganica(array $data): Collection
    {
        // Lógica de negocio compleja
        // Validaciones
        // Transacciones
        // Notificaciones
    }
}
```

Beneficios:
- Controladores más ligeros y enfocados en HTTP
- Lógica de negocio reutilizable
- Más fácil de testear
- Mejor separación de responsabilidades

Servicios Implementados:

RecoleccionService

- `crearRecoleccionOrganica()` : Crea recolección de orgánicos con programación automática
- `crearRecoleccionInorganica()` : Crea recolección de inorgánicos (programada o demanda)
- `crearSolicitudPeligrosos()` : Crea solicitud de peligrosos con aprobación
- `asignarRutaAutomatica()` : Asigna ruta según localidad y tipo
- `completarRecoleccion()` : Completa recolección y asigna puntos

CanjeService

- `validarPuntosSuficientes()` : Valida disponibilidad de puntos
- `procesarCanje()` : Procesa canje completo con transacción
- `generarCodigoCanje()` : Genera código único
- `marcarComoUsado()` : Marca canje como utilizado

NotificacionService

- `enviarConfirmacion()` : Notifica confirmación de recolección
- `enviarRecordatorio()` : Envía recordatorio previo
- `enviarNotificacionCompletado()` : Notifica completado
- `enviarNotificacionSolicitud()` : Notifica solicitud recibida
- `enviarNotificacionCanje()` : Notifica canje realizado

PuntosService

- `calcularPuntos()` : Calcula puntos usando estrategia configurada
- `asignarPuntos()` : Asigna puntos a usuario
- `descontarPuntos()` : Descuenta puntos para canjes
- `cambiarEstrategia()` : Cambia estrategia de cálculo dinámicamente

2. Repository Pattern

Propósito: Abstraer el acceso a datos y queries complejas.

Implementación:

```
// app/Repositories/RecoleccionRepository.php
class RecoleccionRepository
{
    public function findByUser(int $userId)
    {
        return Collection::where('user_id', $userId)
            ->with(['tipoResiduo', 'empresa', 'localidad'])
            ->orderBy('fecha_recoleccion', 'desc')
            ->get();
    }

    public function getEstadisticasPorLocalidad(int $localidadId, array $filters)
    {
        // Query compleja con joins y agregaciones
    }
}
```

Beneficios:
- Queries complejas centralizadas
- Más fácil de mantener y optimizar
- Reutilización de queries
- Mejor testabilidad

Repositorios Implementados:

RecoleccionRepository

- `findByUser()`: Recolecciones de un usuario
- `findByLocalidad()`: Recolecciones por localidad con filtros
- `findByEmpresa()`: Recolecciones por empresa con filtros
- `getEstadisticasPorLocalidad()`: Estadísticas agregadas por localidad
- `getEstadisticasPorEmpresa()`: Estadísticas agregadas por empresa
- `getPendientesAprobacion()`: Solicitudes pendientes
- `getRecoleccionesDelDia()`: Recolecciones programadas para hoy

PuntoRepository

- `getTotalPuntosByUser()`: Total de puntos de usuario
- `getHistorialPuntos()`: Historial de puntos ganados
- `getEstadisticasPorTipo()`: Estadísticas por tipo de residuo
- `getRankingUsuarios()`: Ranking de usuarios por puntos

CanjeRepository

- `findByUser()`: Canjes de un usuario
- `getEstadisticasCanjes()`: Estadísticas de canjes
- `getPendientes()`: Canjes pendientes de uso
- `findByCodigo()`: Buscar canje por código
- `getTotalPuntosCanjeados()`: Total de puntos canjeados

3. Strategy Pattern

Propósito: Permitir cambiar algoritmos de cálculo de puntos dinámicamente.

Implementación:

```
// app/Contracts/PuntosCalculatorInterface.php
interface PuntosCalculatorInterface
{
    public function calcular(float $pesoKg): float;
}

// app/Services/PuntosCalculators/SimplePuntosCalculator.php
class SimplePuntosCalculator implements PuntosCalculatorInterface
{
    public function calcular(float $pesoKg): float
    {
        return round($pesoKg, 2); // 1 kg = 1 punto
    }
}

// app/Services/PuntosCalculators/ConfigurablePuntosCalculator.php
class ConfigurablePuntosCalculator implements PuntosCalculatorInterface
{
    public function calcular(float $pesoKg): float
    {
        $multiplicador = config('puntos.multiplicador', 1.0);
        $bonus = config('puntos.bonus', 0);
        return round(($pesoKg * $multiplicador) + $bonus, 2);
    }
}

// app/Services/PuntosService.php
class PuntosService
{
    protected $calculator;

    public function __construct()
    {
        $this->setCalculator();
    }

    public function cambiarEstrategia(string $estrategia): void
    {
        $this->calculator = match($estrategia) {
            'configurable' => new ConfigurablePuntosCalculator(),
            'simple' => new SimplePuntosCalculator(),
            default => new SimplePuntosCalculator(),
        };
    }
}
```

Beneficios:
- Fácil agregar nuevas estrategias de cálculo
- Cambio dinámico de algoritmo
- Configuración flexible
- Preparado para fórmulas complejas futuras

Estrategias Disponibles:

1. **SimplePuntosCalculator**
   - Fórmula: `puntos = peso_kg`
   - Uso: Cálculo básico 1:1

2. **ConfigurablePuntosCalculator**
   - Fórmula: `puntos = (peso_kg * multiplicador) + bonus`
   - Uso: Cálculo configurable desde `.env` o admin
   - Configuración en `config/puntos.php`

4. Dependency Injection

Propósito: Inyectar dependencias para mejor testabilidad y desacoplamiento.

Implementación:

```
class CanjeController extends Controller
{
    protected $canjeService;

    public function __construct(CanjeService $canjeService)
    {
        $this->canjeService = $canjeService;
    }

    public function store(Request $request)
    {
        $resultado = $this->canjeService->procesarCanje(
            Auth::id(),
            $request->tienda_id
        );
        // ...
    }
}
```

🔄 Flujos Principales

Flujo de Recolección de Orgánicos

```
Usuario → Controller → RecoleccionService
                              ↓
                      asignarRutaAutomatica()
                              ↓
                      crear Collection + Detail
                              ↓
                      NotificacionService
                              ↓
                      enviarConfirmacion()
```

Flujo de Canje de Puntos

```
Usuario → CanjeController → CanjeService
                                ↓
                        validarPuntosSuficientes()
                                ↓
                        PuntosService.descontarPuntos()
                                ↓
                        generarCodigoCanje()
                                ↓
                        crear Canje
                                ↓
                        NotificacionService
                                ↓
                        enviarNotificacionCanje()
```

Flujo de Completar Recolección

```
Admin → Controller → RecoleccionService
                            ↓
                    actualizar estado
                            ↓
                    PuntosService.asignarPuntos()
                            ↓
                    calcular con Strategy
                            ↓
                    actualizar Point
                            ↓
                    NotificacionService
                            ↓
                    enviarNotificacionCompletado()
```

Flujo de Generación de Reportes

```
Admin → ReporteController → RecoleccionRepository
                                    ↓
                            getEstadisticasPorLocalidad()
                                    ↓
                            queries con joins y agregaciones
                                    ↓
                            retornar datos
                                    ↓
                            generar PDF/CSV
```

🗄️ Modelo de Datos

Entidades Principales

```
User (usuarios)
├── hasMany → Collection (recolecciones)
├── hasOne → Point (puntos)
└── hasMany → Canje (canjes)

Collection (recolecciones)
├── belongsTo → User
├── belongsTo → TipoResiduo
├── belongsTo → Empresa
├── belongsTo → Localidad
├── belongsTo → Ruta
└── hasOne → CollectionDetail

Tienda (tiendas)
└── hasMany → Canje

Canje (canjes)
├── belongsTo → User
└── belongsTo → Tienda

Localidad (localidades)
└── hasMany → Ruta

Ruta (rutas)
├── belongsTo → Localidad
└── hasMany → Collection
```

🔒 Seguridad

Capas de Seguridad

1. **Autenticación**: Laravel Sanctum
2. **Autorización**: Spatie Permission (roles y permisos)
3. **Validación**: Form Requests
4. **Middleware**: Verificación de roles
5. **CSRF Protection**: Tokens en formularios
6. **SQL Injection**: Eloquent ORM con prepared statements

Middleware Stack

```
web → auth → role:Administrador → controller
```

📊 Escalabilidad

Estrategias Implementadas

1. **Eager Loading**: Prevenir N+1 queries
2. **Query Optimization**: Índices en BD
3. **Caching**: Config, routes, views
4. **Queue Jobs**: Notificaciones asíncronas (preparado)

5. **Repository Pattern**: Optimización centralizada

Preparado para Futuro

- **Queue System**: Notificaciones en cola
- **Cache Layer**: Redis para datos frecuentes
- **API REST**: Endpoints para mobile app
- **Microservicios**: Separación de módulos

🧪 Testabilidad

Ventajas de la Arquitectura

1. **Service Layer**: Testear lógica sin HTTP
2. **Repository Pattern**: Mock de datos fácil
3. **Dependency Injection**: Inyectar mocks
4. **Strategy Pattern**: Testear algoritmos aislados

Ejemplo de Test

```
public function test_procesar_canje_exitoso()
{
    $canjeService = new CanjeService(
        new MockCanjeRepository(),
        new MockNotificacionService()
    );

    $resultado = $canjeService->procesarCanje(1, 1);

    $this->assertTrue($resultado['success']);
}
```

📈 Métricas y Monitoreo

Puntos de Monitoreo

1. **Performance**: Tiempo de respuesta de servicios
2. **Errores**: Logs centralizados
3. **Uso**: Estadísticas de canjes y recolecciones
4. **Disponibilidad**: Uptime del sistema

🔮 Evolución Futura

Fase 4 (Planificada)

- API REST completa
- Aplicación móvil
- Dashboard en tiempo real
- Integración con IoT (sensores de peso)
- Machine Learning para predicción de rutas

Fase 5 (Planificada)

- Microservicios

- Event Sourcing
- CQRS Pattern
- GraphQL API
- Blockchain para trazabilidad

📚 Referencias

- Laravel Documentation (https://laravel.com/docs)
- Repository Pattern (https://designpatternsphp.readthedocs.io/en/latest/More/Repository/README.html)
- Service Layer Pattern (https://martinfowler.com/eaaCatalog/serviceLayer.html)
- Strategy Pattern (https://refactoring.guru/design-patterns/strategy)