Mejoras de Backend - INMOVA API

Resumen Ejecutivo

Se han implementado mejoras significativas en los endpoints de la API backend de INMOVA, siguiendo las mejores prácticas de desarrollo de APIs RESTful. Este documento detalla las mejoras realizadas y proporciona guías de implementación.

Índice

Mejoras Implementadas

1. Sistema de Validación Centralizado

Archivo: `/lib/validations/index.ts`

- **Esquemas Zod completos** para todas las entidades:
- Proveedores (Providers)
- Gastos (Expenses)
- Tareas (Tasks)
- Documentos (Documents)
- Edificios, Unidades, Inquilinos, Contratos, Pagos, Mantenimiento
- **Validaciones robustas**:
- Tipos de datos
- Rangos y límites
- Formatos (email, teléfono, CIF, DNI, códigos postales)
- Relaciones entre datos
- Mensajes de error descriptivos en español

Ejemplo:

```
export const providerCreateSchema = z.object({
  nombre: z.string()
    .min(1, 'El nombre es requerido')
    .max(200, 'El nombre no puede exceder 200 caracteres')
    .trim(),
  email: z.string()
    .email('Email inválido')
    .toLowerCase()
    .trim(),
  cif: z.string()
    .regex(/^[A-Z][0-9]{7}[A-Z0-9]$/, 'CIF inválido (formato: A12345678)')
    .toUpperCase()
    .trim()
    .optional(),
  // ... más campos
});
```

2. Endpoints Mejorados

Proveedores (`/api/providers`)

Mejoras:
- ✅ Validación Zod completa
- ✅ Verificación de duplicados (email, CIF)
- ✅ Filtros de búsqueda (tipo, search)
- ✅ Verificación de relaciones antes de eliminar
- ✅ Inclusión de contadores de recursos relacionados
- ✅ Control de acceso por compañía

Endpoints:
- `GET /api/providers` - Lista con filtros
- `POST /api/providers` - Crear con validación
- `GET /api/providers/[id]` - Obtener con relaciones
- `PUT /api/providers/[id]` - Actualizar con validación
- `DELETE /api/providers/[id]` - Eliminar con verificaciones

Gastos (`/api/expenses`)

Mejoras:
- ✅ Validación Zod completa
- ✅ Paginación (limit, offset)
- ✅ Filtros múltiples (edificio, unidad, proveedor, categoría, fechas)
- ✅ Verificación de pertenencia a compañía
- ✅ Validación de relaciones (edificio, unidad, proveedor)
- ✅ Respuestas con metadatos de paginación

Endpoints:
- `GET /api/expenses` - Lista paginada con filtros
- `POST /api/expenses` - Crear con validaciones
- `GET /api/expenses/[id]` - Obtener con relaciones
- `PUT /api/expenses/[id]` - Actualizar con validaciones
- `DELETE /api/expenses/[id]` - Eliminar con verificaciones

Tareas (`/api/tasks`)

Mejoras:
- ✅ Validación Zod completa
- ✅ Paginación y filtros
- ✅ Asignación de usuarios con verificaciones
- ✅ Relación con edificios y unidades
- ✅ Ordenamiento por prioridad y fecha límite
- ✅ Inclusión de datos de usuarios (asignado y creador)

Endpoints:
- `GET /api/tasks` - Lista paginada con filtros
- `POST /api/tasks` - Crear con validaciones
- `GET /api/tasks/[id]` - Obtener con relaciones
- `PUT /api/tasks/[id]` - Actualizar con validaciones
- `DELETE /api/tasks/[id]` - Eliminar con verificaciones

3. Utilidades de API

Archivo: `/lib/api-utils.ts`

Funciones Helper:

```
// Respuestas estandarizadas
successResponse<T>(data: T, status?: number)
errorResponse(error: string, message: string, status: number, details?: any)

// Respuestas de error por tipo
badRequestResponse(message?: string, details?: any)       // 400
unauthorizedResponse(message?: string)                    // 401
forbiddenResponse(message?: string)                       // 403
notFoundResponse(resource?: string)                       // 404
conflictResponse(message?: string)                        // 409
internalServerErrorResponse(message?: string)             // 500

// Manejo de errores
handleZodError(error: z.ZodError)
handleError(error: any, context: string)

// Paginación
paginatedResponse<T>(data: T[], total: number, limit?: number, offset?: number)
parsePaginationParams(searchParams: URLSearchParams)

// Validaciones
isValidUUID(uuid: string)
isValidDate(date: string)

// Utilidades
sanitizeObject<T>(obj: T)
convertDatesToObjects(data: any)
```

4. Manejo Consistente de Errores

Estructura de try/catch estandarizada:

```
try {
  // 1. Autenticación/Autorización
  const user = await requirePermission('create');

  // 2. Obtener y validar datos
  const body = await req.json();
  const validatedData = schema.parse(body);

  // 3. Verificaciones de negocio
  // - Verificar duplicados
  // - Verificar relaciones
  // - Verificar permisos

  // 4. Operación de base de datos
  const result = await prisma.model.create({ ... });

  // 5. Logging y respuesta
  logger.info('Resource created', { userId, resourceId });
  return successResponse(result, 201);

} catch (error: any) {
  logger.error('Error context:', error);

  // Manejo de errores por tipo
  if (error instanceof z.ZodError) {
    return handleZodError(error);
  }

  if (error.message === 'No autenticado') {
    return unauthorizedResponse();
  }

  if (error.message?.includes('permiso')) {
    return forbiddenResponse(error.message);
  }

  return internalServerErrorResponse();
}
```

Arquitectura

Estructura de Directorios

```
app/api/
├── providers/
│   ├── route.ts              # GET, POST
│   ├── [id]/
│   │   └── route.ts          # GET, PUT, DELETE
│   ├── performance/
│   ├── recommend/
│   └── stats/
├── expenses/
│   ├── route.ts
│   └── [id]/
│       └── route.ts
├── tasks/
│   ├── route.ts
│   ├── [id]/
│   │   └── route.ts
│   ├── my-day/
│   └── prioritized/
└── ...

lib/
├── validations/
│   └── index.ts              # Esquemas Zod centralizados
├── api-utils.ts              # Utilidades de API
├── permissions.ts            # Control de permisos
├── logger.ts                 # Logging estructurado
├── db.ts                     # Cliente Prisma
└── auth-options.ts           # Configuración NextAuth
```

Flujo de una Petición

```
Cliente
  ↓
  v
Endpoint API
  ↓
  v
Autenticación (requireAuth / requirePermission)
  ↓
  v
Validación Zod
  ↓
  v
Verificaciones de Negocio
  ↓
  v
Operación en Base de Datos (Prisma)
  ↓
  v
Logging
  ↓
  v
Respuesta HTTP
```

Validación de Datos

Tipos de Validaciones Implementadas

1. Validaciones de Tipo y Formato

```
// Strings
z.string()
  .min(1, 'Campo requerido')
  .max(200, 'Máximo 200 caracteres')
  .trim()

// Email
z.string()
  .email('Email inválido')
  .toLowerCase()

// Números
z.number()
  .positive('Debe ser positivo')
  .max(10000000, 'Máximo 10,000,000')

// UUIDs
z.string()
  .uuid('ID inválido')

// Fechas
z.string()
  .datetime({ message: 'Fecha inválida' })
  .or(z.date())

// Enums
z.enum(['opcion1', 'opcion2', 'opcion3'])
```

2. Validaciones de Formato Español

```
// DNI/NIE
z.string()
  .regex(/^[0-9]{8}[A-Z]$/, 'DNI/NIE inválido (formato: 12345678A)')
  .toUpperCase()

// CIF
z.string()
  .regex(/^[A-Z][0-9]{7}[A-Z0-9]$/, 'CIF inválido (formato: A12345678)')
  .toUpperCase()

// Código Postal
z.string()
  .regex(/^\d{5}$/, 'El código postal debe tener 5 dígitos')

// Teléfono Internacional
z.string()
  .regex(/^[+]?[(]?[0-9]{1,4}[)]?[-\s.]?[(]?[0-9]{1,4}[)]?[-\s.]?[0-9]{1,9}$/, 'Telé-
fono inválido')
```

3. Validaciones de Relaciones

```
// Validación cruzada de fechas
contractCreateSchema.refine((data) => {
  const inicio = new Date(data.fechaInicio);
  const fin = new Date(data.fechaFin);
  return fin > inicio;
}, {
  message: 'La fecha de fin debe ser posterior a la fecha de inicio',
  path: ['fechaFin']
});
```

Manejo de Errores

Códigos de Estado HTTP

Código	Uso en INMOVA	Ejemplo
200	Operación exitosa	GET, PUT, DELETE exitoso
201	Recurso creado	POST exitoso
400	Validación fallida	Datos inválidos
401	No autenticado	Sesión expirada
403	Sin permisos	Operador intenta crear
404	No encontrado	Recurso inexistente
409	Conflicto	Email duplicado
500	Error del servidor	Error de Prisma

Tipos de Errores

1. Errores de Validación (400)

```
{
  "error": "Validación fallida",
  "message": "Los datos proporcionados no son válidos",
  "details": [
    {
      "path": ["email"],
      "message": "Email inválido"
    }
  ]
}
```

2. Errores de Autenticación (401)

```
{
  "error": "No autenticado",
  "message": "Debe iniciar sesión"
}
```

3. Errores de Autorización (403)

```
{
  "error": "Prohibido",
  "message": "No tiene permisos para crear proveedores"
}
```

4. Errores de Recurso (404)

```
{
  "error": "No encontrado",
  "message": "Proveedor no encontrado"
}
```

5. Errores de Conflicto (409)

```
{
  "error": "Proveedor duplicado",
  "message": "Ya existe un proveedor con este email"
}
```

Seguridad y Autorización

Niveles de Seguridad

1. Autenticación

```
const user = await requireAuth();
// Verifica sesión activa vía NextAuth
```

2. Autorización por Rol

```
const user = await requirePermission('create');
// Verifica que el rol del usuario permite la acción
```

Permisos por Rol:

Acción	Operador	Gestor	Administrador	Super Admin
Create	❌	✅	✅	✅
Read	✅	✅	✅	✅
Update	❌	✅	✅	✅
Delete	❌	❌	✅	✅

3. Aislamiento por Compañía

```
// Verificar que el recurso pertenece a la compañía del usuario
if (resource.companyId !== user.companyId) {
  return forbiddenResponse('No tiene acceso a este recurso');
}
```

4. Verificación de Relaciones

```
// Al crear un gasto, verificar que el edificio pertenece a la compañía
const building = await prisma.building.findUnique({
  where: { id: validatedData.buildingId },
});

if (!building || building.companyId !== user.companyId) {
  return forbiddenResponse('No tiene acceso a este edificio');
}
```

Prevención de Vulnerabilidades

✅ **SQL Injection**: Prevenido por Prisma ORM
✅ **XSS**: Sanitización de inputs con Zod
✅ **CSRF**: Protección de NextAuth
✅ **Path Traversal**: Validación de UUIDs
✅ **Mass Assignment**: Esquemas Zod explícitos
✅ **Information Disclosure**: Mensajes de error genéricos

Testing y Calidad

Checklist de Calidad para Cada Endpoint

✅ **Validación**
- [ ] Esquema Zod definido
- [ ] Todas las validaciones necesarias implementadas
- [ ] Mensajes de error descriptivos

✅ **Seguridad**
- [ ] Autenticación verificada
- [ ] Autorización por rol implementada
- [ ] Verificación de pertenencia a compañía
- [ ] Validación de relaciones

✅ **Manejo de Errores**
- [ ] Try/catch implementado
- [ ] Errores diferenciados por tipo
- [ ] Códigos HTTP correctos
- [ ] Logging de errores

✅ **Funcionalidad**
- [ ] Operaciones CRUD completas
- [ ] Paginación (si aplica)
- [ ] Filtros (si aplica)
- [ ] Inclusión de relaciones

✅ **Documentación**
- [ ] JSDoc comments
- [ ] Documentación de API actualizada
- [ ] Ejemplos de uso

Ejemplos de Testing

Test de Validación

```
// POST /api/providers con email inválido
const response = await fetch('/api/providers', {
  method: 'POST',
  body: JSON.stringify({
    nombre: 'Test',
    tipo: 'Fontanería',
    email: 'invalid-email',
    telefono: '+34612345678'
  })
});

expect(response.status).toBe(400);
const data = await response.json();
expect(data.error).toBe('Validación fallida');
```

Test de Autorización

```
// DELETE /api/providers/[id] sin permisos
const response = await fetch('/api/providers/123', {
  method: 'DELETE',
  headers: { 'Cookie': operatorSessionCookie }
});

expect(response.status).toBe(403);
```

Migración y Compatibilidad

Cambios Importantes

1. Estructura de Respuestas

Antes:

```
// Lista simple
[{ ... }, { ... }]

// Error simple
{ "error": "mensaje" }
```

Después:

```
// Lista con metadatos
{
  "data": [{ ... }, { ... }],
  "meta": {
    "total": 100,
    "limit": 20,
    "offset": 0
  }
}

// Error estructurado
{
  "error": "tipo",
  "message": "descripción",
  "details": {}
}
```

2. Códigos de Estado

Antes:
- Uso inconsistente de códigos
- Muchos 500 genéricos

Después:
- Códigos semánticos correctos
- Diferenciación clara de errores

3. Validación

Antes:

```
if (!nombre || !tipo) {
  return NextResponse.json({ error: 'Campos requeridos' }, { status: 400 });
}
```

Después:

```
const validatedData = providerCreateSchema.parse(body);
```

Compatibilidad con Frontend

Para mantener compatibilidad con el frontend existente:

1. **Listas sin paginación**: El frontend puede seguir usando la lista directamente

```
// Frontend puede acceder a .data o usar directamente si no tiene paginación
const items = response.data || response;
```

1. **Errores**: El frontend debe verificar `.error` y `.message`

```
if (response.error) {
  console.error(response.message);
  // Mostrar detalles si existen
  if (response.details) {
    console.error(response.details);
  }
}
```

Próximos Pasos

Recomendaciones a Corto Plazo

1. ✅ **Aplicar mejoras a otros endpoints**
   - Contracts
   - Payments
   - Maintenance
   - Tenants
   - Units
   - Buildings

2. ✅ **Implementar rate limiting**
   - Prevenir abuso de API
   - Usar middleware de Next.js

3. ✅ **Añadir caché**
   - Redis para datos frecuentes
   - Cache-Control headers

4. ✅ **Mejorar logging**
   - Estructurar logs por nivel
   - Integración con servicio de monitoreo

Recomendaciones a Largo Plazo

1. **API Versioning**
   - `/api/v1/providers`
   - Mantener compatibilidad

2. **Documentación Interactiva**
   - Swagger/OpenAPI
   - Postman collection

3. **Testing Automatizado**
   - Tests unitarios (Jest)
   - Tests de integración (Playwright)
   - Tests E2E

4. **Optimización de Performance**
   - Queries optimizadas
   - Eager loading vs lazy loading
   - Indexes de base de datos

Conclusión

Las mejoras implementadas en los endpoints de INMOVA proporcionan:

✅ **Robustez**: Validación completa y manejo de errores consistente
✅ **Seguridad**: Autenticación, autorización y verificaciones en múltiples niveles
✅ **Mantenibilidad**: Código limpio, documentado y siguiendo patrones consistentes
✅ **Escalabilidad**: Arquitectura preparada para crecimiento futuro
✅ **Usabilidad**: Respuestas claras y mensajes de error descriptivos

Estas mejoras establecen una base sólida para el desarrollo continuo de la plataforma INMOVA.

Autor: Backend Senior Developer
Fecha: Diciembre 2024
Versión: 1.0.0