Documentación de APIs Backend - INMOVA

Índice

Introducción

Esta documentación describe los endpoints de la API backend de INMOVA, una plataforma de gestión inmobiliaria. Todos los endpoints implementan:

- **Validación de datos con Zod**: Esquemas tipados y validados
- **Manejo robusto de errores**: Try/catch con logging completo
- **Códigos HTTP semánticos**: Uso correcto de 200, 201, 400, 401, 403, 404, 409, 500
- **Autenticación y autorización**: Control de permisos por rol
- **Logging estructurado**: Trazabilidad de todas las operaciones

---

Autenticación

Todos los endpoints requieren autenticación mediante **NextAuth.js**. Las credenciales se validan mediante:

```
import { getServerSession } from 'next-auth';
import { authOptions } from '@/lib/auth-options';
import { requireAuth, requirePermission } from '@/lib/permissions';
```

Métodos de Autenticación

1. **requireAuth()**: Verifica sesión activa
2. **requirePermission(action)**: Verifica permisos específicos (create, read, update, delete)

Roles de Usuario

- **super_admin**: Acceso total al sistema
- **administrador**: Gestión completa de su compañía

- **gestor**: Operaciones CRUD estándar
- **operador**: Solo lectura y operaciones básicas

Códigos de Estado HTTP

Código	Significado	Uso
200	OK	Operación exitosa (GET, PUT, DELETE)
201	Created	Recurso creado exitosamente (POST)
400	Bad Request	Error de validación de datos
401	Unauthorized	No autenticado
403	Forbidden	Sin permisos suficientes
404	Not Found	Recurso no encontrado
409	Conflict	Conflicto (ej: registro duplicado)
500	Internal Server Error	Error del servidor

Manejo de Errores

Estructura de Respuesta de Error

Todas las respuestas de error siguen este formato:

```
{
  "error": "Tipo de error",
  "message": "Descripción legible del error",
  "details": {} // Opcional: detalles adicionales
}
```

Errores de Validación (Zod)

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

Ejemplo de Manejo de Errores

```
try {
  const validatedData = schema.parse(body);
  // Lógica de negocio...
} catch (error: any) {
  logger.error('Error creating resource:', error);

  if (error instanceof z.ZodError) {
    return NextResponse.json(
      {
        error: 'Validación fallida',
        message: 'Los datos proporcionados no son válidos',
        details: error.errors,
      },
      { status: 400 }
    );
  }

  if (error.message?.includes('permiso')) {
    return NextResponse.json(
      { error: 'Prohibido', message: error.message },
      { status: 403 }
    );
  }

  return NextResponse.json(
    { error: 'Error interno del servidor', message: 'Error al crear recurso' },
    { status: 500 }
  );
}
```

Endpoints

Proveedores

GET /api/providers

Obtiene todos los proveedores de la compañía.

Parámetros de Query:
- `tipo` (string, opcional): Filtrar por tipo de proveedor
- `search` (string, opcional): Búsqueda en nombre, email o teléfono

Respuesta Exitosa (200):


```
[
  {
    "id": "uuid",
    "nombre": "Fontanería Rápida S.L.",
    "tipo": "Fontanería",
    "email": "contacto@fontaneria.com",
    "telefono": "+34612345678",
    "cif": "B12345678",
    "direccion": "Calle Principal 123",
    "ciudad": "Madrid",
    "codigoPostal": "28001",
    "calificacion": 4.5,
    "notas": "Proveedor confiable",
    "_count": {
      "maintenanceRequests": 15,
      "expenses": 32
    }
  }
]
```


POST /api/providers

Crea un nuevo proveedor.

Body:


```
{
  "nombre": "Fontanería Rápida S.L.",
  "tipo": "Fontanería",
  "email": "contacto@fontaneria.com",
  "telefono": "+34612345678",
  "cif": "B12345678",
  "direccion": "Calle Principal 123",
  "ciudad": "Madrid",
  "codigoPostal": "28001",
  "calificacion": 4.5,
  "notas": "Proveedor confiable"
}
```


Validaciones:
- `nombre` : Requerido, máx 200 caracteres
- `tipo` : Requerido, máx 100 caracteres
- `email` : Formato email válido, único
- `telefono` : Formato internacional válido
- `cif` : Formato CIF español (A12345678), único
- `codigoPostal` : 5 dígitos
- `calificacion` : 0-5

Respuesta Exitosa (201):
Retorna el proveedor creado con todos sus campos.

Errores:
- `400` : Validación fallida
- `401` : No autenticado
- `403` : Sin permisos
- `409` : Email o CIF duplicado

GET /api/providers/[id]

Obtiene un proveedor específico con sus relaciones.

Respuesta Exitosa (200):

```
{
  "id": "uuid",
  "nombre": "Fontanería Rápida S.L.",
  // ... demás campos ...
  "maintenanceRequests": [
    {
      "id": "uuid",
      "titulo": "Reparación fuga",
      "estado": "completado",
      "unit": {
        "numero": "101",
        "building": { "nombre": "Edificio Central" }
      }
    }
  ],
  "expenses": [
    {
      "id": "uuid",
      "concepto": "Reparación tubería",
      "monto": 350.00,
      "fecha": "2024-12-01T10:00:00Z"
    }
  ],
  "_count": {
    "maintenanceRequests": 15,
    "expenses": 32
  }
}
```

Errores:
- `401` : No autenticado
- `403` : Sin acceso al proveedor
- `404` : Proveedor no encontrado

PUT /api/providers/[id]

Actualiza un proveedor existente.

Body:
Cualquier campo del proveedor (todos opcionales).

Validaciones:
Iguales que POST, pero todos los campos son opcionales.

Respuesta Exitosa (200):
Retorna el proveedor actualizado.

Errores:
- `400` : Validación fallida
- `401` : No autenticado
- `403` : Sin permisos o sin acceso
- `404` : Proveedor no encontrado
- `409` : Email o CIF duplicado

DELETE /api/providers/[id]

Elimina un proveedor.

Restricciones:
- No se puede eliminar si tiene solicitudes de mantenimiento o gastos asociados

Respuesta Exitosa (200):

```
{
  "message": "Proveedor eliminado exitosamente"
}
```

Errores:
- `401` : No autenticado
- `403` : Sin permisos o sin acceso
- `404` : Proveedor no encontrado
- `409` : Tiene relaciones activas

Gastos

GET /api/expenses

Obtiene gastos con filtros y paginación.

Parámetros de Query:
- `buildingId` (uuid, opcional): Filtrar por edificio
- `unitId` (uuid, opcional): Filtrar por unidad
- `providerId` (uuid, opcional): Filtrar por proveedor
- `categoria` (string, opcional): Filtrar por categoría
- `fechaDesde` (ISO date, opcional): Fecha inicio
- `fechaHasta` (ISO date, opcional): Fecha fin
- `limit` (number, opcional): Número de resultados
- `offset` (number, opcional): Offset para paginación

Respuesta Exitosa (200):

```
{
  "data": [
    {
      "id": "uuid",
      "concepto": "Reparación tubería",
      "categoria": "mantenimiento",
      "monto": 350.00,
      "fecha": "2024-12-01T10:00:00Z",
      "facturaPdfPath": "/path/to/factura.pdf",
      "numeroFactura": "F-2024-001",
      "notas": "Reparación urgente",
      "building": {
        "id": "uuid",
        "nombre": "Edificio Central"
      },
      "unit": {
        "id": "uuid",
        "numero": "101"
      },
      "provider": {
        "id": "uuid",
        "nombre": "Fontanería Rápida"
      }
    }
  ],
  "meta": {
    "total": 100,
    "limit": 20,
    "offset": 0
  }
}
```

POST /api/expenses

Crea un nuevo gasto.

Body:

```
{
  "buildingId": "uuid",
  "unitId": "uuid", // opcional
  "providerId": "uuid", // opcional
  "concepto": "Reparación tubería",
  "categoria": "mantenimiento",
  "monto": 350.00,
  "fecha": "2024-12-01T10:00:00Z",
  "facturaPdfPath": "/path/to/factura.pdf",
  "numeroFactura": "F-2024-001",
  "notas": "Reparación urgente"
}
```

Categorías Válidas:

- `mantenimiento`
- `servicios`
- `impuestos`
- `seguros`
- `suministros`
- `personal`

- `marketing`
- `legal`
- `tecnologia`
- `otro`

Validaciones:
- `concepto`: Requerido, máx 500 caracteres
- `categoria`: Requerido, enum
- `monto`: Requerido, positivo, máx 10,000,000
- `fecha`: Requerida, formato ISO datetime
- `buildingId`, `unitId`, `providerId`: UUIDs válidos

Respuesta Exitosa (201):
Retorna el gasto creado con sus relaciones.

Errores:
- `400`: Validación fallida
- `401`: No autenticado
- `403`: Sin permisos o sin acceso
- `404`: Edificio/unidad/proveedor no encontrado

GET /api/expenses/[id]

Obtiene un gasto específico.

Respuesta Exitosa (200):
Retorna el gasto con todas sus relaciones.

PUT /api/expenses/[id]

Actualiza un gasto existente.

Body:
Cualquier campo del gasto (todos opcionales).

Respuesta Exitosa (200):
Retorna el gasto actualizado.

DELETE /api/expenses/[id]

Elimina un gasto.

Respuesta Exitosa (200):

```
{
  "message": "Gasto eliminado exitosamente"
}
```

Tareas

GET /api/tasks

Obtiene tareas con filtros y paginación.

Parámetros de Query:
- `estado` (string, opcional): pendiente | en_progreso | completado | cancelado
- `prioridad` (string, opcional): baja | media | alta | urgente
- `asignadoA` (uuid, opcional): Filtrar por usuario asignado
- `buildingId` (uuid, opcional): Filtrar por edificio
- `unitId` (uuid, opcional): Filtrar por unidad
- `limit` (number, opcional): Número de resultados
- `offset` (number, opcional): Offset para paginación

Respuesta Exitosa (200):


```
{
  "data": [
    {
      "id": "uuid",
      "titulo": "Reparar puerta",
      "descripcion": "La puerta del garaje no cierra bien",
      "estado": "pendiente",
      "prioridad": "alta",
      "fechaLimite": "2024-12-15T10:00:00Z",
      "fechaInicio": "2024-12-10T08:00:00Z",
      "notas": "Coordinar con el inquilino",
      "asignadoUser": {
        "id": "uuid",
        "name": "Juan Pérez",
        "email": "juan@inmova.com"
      },
      "creadorUser": {
        "id": "uuid",
        "name": "María García",
        "email": "maria@inmova.com"
      },
      "building": {
        "id": "uuid",
        "nombre": "Edificio Central"
      },
      "unit": {
        "id": "uuid",
        "numero": "101"
      }
    }
  ],
  "meta": {
    "total": 50,
    "limit": 20,
    "offset": 0
  }
}
```


POST /api/tasks

Crea una nueva tarea.

Body:

```
{
  "titulo": "Reparar puerta",
  "descripcion": "La puerta del garaje no cierra bien",
  "estado": "pendiente",
  "prioridad": "alta",
  "fechaLimite": "2024-12-15T10:00:00Z",
  "fechaInicio": "2024-12-10T08:00:00Z",
  "asignadoA": "uuid",
  "buildingId": "uuid",
  "unitId": "uuid",
  "notas": "Coordinar con el inquilino"
}
```

Validaciones:
- `titulo`: Requerido, máx 200 caracteres
- `estado`: Enum (pendiente, en_progreso, completado, cancelado)
- `prioridad`: Enum (baja, media, alta, urgente)
- `fechaLimite`, `fechaInicio`: Formato ISO datetime
- `asignadoA`: UUID válido, debe pertenecer a la compañía

Respuesta Exitosa (201):
Retorna la tarea creada con todas sus relaciones.

Errores:
- `400`: Validación fallida
- `401`: No autenticado
- `403`: Sin permisos o sin acceso
- `404`: Usuario/edificio/unidad no encontrado

GET /api/tasks/[id]

Obtiene una tarea específica.

Respuesta Exitosa (200):
Retorna la tarea con todas sus relaciones.

PUT /api/tasks/[id]

Actualiza una tarea existente.

Body:
Cualquier campo de la tarea (todos opcionales).

Respuesta Exitosa (200):
Retorna la tarea actualizada.

DELETE /api/tasks/[id]

Elimina una tarea.

Respuesta Exitosa (200):

```
{
  "message": "Tarea eliminada exitosamente"
}
```

Validación de Datos

Todos los endpoints utilizan **Zod** para validación de datos. Los esquemas se encuentran en `/lib/validations/index.ts`.

Esquemas Disponibles

- `providerCreateSchema` / `providerUpdateSchema`
- `expenseCreateSchema` / `expenseUpdateSchema`
- `taskCreateSchema` / `taskUpdateSchema`
- `buildingCreateSchema` / `buildingUpdateSchema`
- `unitCreateSchema` / `unitUpdateSchema`
- `tenantCreateSchema` / `tenantUpdateSchema`
- `contractCreateSchema` / `contractUpdateSchema`
- `paymentCreateSchema` / `paymentUpdateSchema`
- `maintenanceCreateSchema` / `maintenanceUpdateSchema`
- `documentCreateSchema` / `documentUpdateSchema`

Ejemplo de Uso

```
import { providerCreateSchema } from '@/lib/validations';
import { z } from 'zod';

try {
  const validatedData = providerCreateSchema.parse(body);
  // Datos validados y tipados
} catch (error) {
  if (error instanceof z.ZodError) {
    // Manejar errores de validación
    return NextResponse.json(
      {
        error: 'Validación fallida',
        message: 'Los datos proporcionados no son válidos',
        details: error.errors,
      },
      { status: 400 }
    );
  }
}
```

Mejores Prácticas

1. Siempre Usar Validación Zod

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

// ✅ BIEN
const validatedData = providerCreateSchema.parse(body);
```

2. Manejo Consistente de Errores

```
try {
  // Lógica de negocio
} catch (error: any) {
  logger.error('Error description:', error);

  // Zod errors
  if (error instanceof z.ZodError) {
    return NextResponse.json({ ... }, { status: 400 });
  }

  // Permission errors
  if (error.message?.includes('permiso')) {
    return NextResponse.json({ ... }, { status: 403 });
  }

  // Auth errors
  if (error.message === 'No autenticado') {
    return NextResponse.json({ ... }, { status: 401 });
  }

  // Generic errors
  return NextResponse.json({ ... }, { status: 500 });
}
```

3. Logging Estructurado

```
logger.info('Resource created', { userId: user.id, resourceId: resource.id });
logger.error('Error creating resource', error);
```

4. Verificar Permisos de Compañía

```
// Siempre verificar que el recurso pertenece a la compañía del usuario
if (resource.companyId !== user.companyId) {
  return NextResponse.json(
    { error: 'Prohibido', message: 'No tiene acceso a este recurso' },
    { status: 403 }
  );
}
```

5. Verificar Duplicados Antes de Crear

```
const existing = await prisma.provider.findFirst({
  where: {
    companyId: user.companyId,
    email: validatedData.email,
  },
});

if (existing) {
  return NextResponse.json(
    { error: 'Proveedor duplicado', message: 'Ya existe un proveedor con este email' }
,
    { status: 409 }
  );
}
```

6. Incluir Relaciones en Respuestas

```
const expense = await prisma.expense.create({
  data: validatedData,
  include: {
    building: { select: { nombre: true, id: true } },
    unit: { select: { numero: true, id: true } },
    provider: { select: { nombre: true, id: true } },
  },
});
```

7. Paginación para Listas Grandes

```
const take = limit ? parseInt(limit) : undefined;
const skip = offset ? parseInt(offset) : undefined;

const [data, total] = await Promise.all([
  prisma.model.findMany({ take, skip }),
  prisma.model.count(),
]);

return NextResponse.json({
  data,
  meta: { total, limit: take, offset: skip },
});
```

8. Códigos HTTP Semánticos

- **200**: Operación exitosa (GET, PUT, DELETE)
- **201**: Recurso creado (POST)
- **400**: Validación fallida
- **401**: No autenticado
- **403**: Sin permisos
- **404**: Recurso no encontrado
- **409**: Conflicto (duplicado)
- **500**: Error del servidor

Resumen de Mejoras Implementadas

✅ Validación de Datos

- Esquemas Zod completos para todos los recursos
- Validación de tipos, formatos y rangos
- Mensajes de error descriptivos

✅ Manejo de Errores

- Try/catch en todos los endpoints
- Errores diferenciados por tipo (Zod, Auth, Permissions, Generic)
- Logging estructurado de errores

✅ Códigos HTTP

- Uso semántico correcto de códigos de estado
- Respuestas consistentes entre endpoints

✅ Seguridad

- Verificación de autenticación en todos los endpoints
- Control de permisos por rol
- Validación de pertenencia a compañía
- Verificación de duplicados

✅ Funcionalidad

- Paginación en listados
- Filtros múltiples
- Inclusión de relaciones
- Contadores de recursos relacionados

✅ Documentación

- Comentarios JSDoc en todos los endpoints
- Documentación completa de APIs
- Ejemplos de uso

Fecha de última actualización: Diciembre 2024
Versión: 1.0.0
Autor: Backend Senior Developer Team - INMOVA