API Documentation - Task Management System

Descripción General

Esta API REST permite gestionar tareas (tasks) con operaciones completas de CRUD. Incluye funcionalidades de filtrado, paginación y manejo de estados.

Base URL: `https://localhost:7001` o `http://localhost:5001`

Estructura de Respuesta

Todas las respuestas siguen el formato `ApiResponse<T>`:

```
{
  "data": T,           // Datos solicitados o null en caso de error
  "statusCode": 200,   // Código de estado HTTP
  "message": "string", // Mensaje descriptivo
  "success": true      // true si statusCode está entre 200-299
}
```

📋 Endpoints de Tasks

1. **GET** `/api/tasks` - Obtener todas las tareas

Obtiene una lista paginada de tareas con filtros opcionales.

Parámetros de Query (opcionales):

Parámetro	Tipo	Descripción	Ejemplo
`isCompleted`	`boolean`	Filtrar por estado	`true`, `false`
`title`	`string`	Buscar en título	`"Estudiar"`
`createdFrom`	`datetime`	Desde fecha	`2025-09-01T00:00:00Z`
`createdTo`	`datetime`	Hasta fecha	`2025-09-30T23:59:59Z`
`pageNumber`	`int`	Número de página	`1` (default)
`pageSize`	`int`	Elementos por página	`10` (default)

Ejemplos de Uso:

Obtener todas las tareas:

```
GET /api/tasks
```

Obtener tareas completadas:

```
GET /api/tasks?isCompleted=true
```

Buscar tareas por título:

```
GET /api/tasks?title=estudiar
```

Filtrar por fecha y paginar:

```
GET /api/tasks?createdFrom=2025-09-01&createdTo=2025-09-30&pageNumber=1&pageSize=5
```

Respuesta Exitosa (200):


```
{
  "data": [
    {
      "id": 1,
      "title": "Estudiar para el examen",
      "description": "Revisar capítulos 1-5",
      "isCompleted": false,
      "createdDate": "2025-09-27T10:00:00Z",
      "completedDate": null
    },
    {
      "id": 2,
      "title": "Hacer ejercicio",
      "description": "Correr 30 minutos",
      "isCompleted": true,
      "createdDate": "2025-09-26T08:00:00Z",
      "completedDate": "2025-09-26T09:00:00Z"
    }
  ],
  "statusCode": 200,
  "message": "Tareas obtenidas exitosamente",
  "success": true
}
```


---

2. **GET** `/api/tasks/{id}` - Obtener tarea por ID

Obtiene una tarea específica por su ID.

Parámetros de Ruta:

Parámetro	Tipo	Descripción
`id`	`int`	ID de la tarea

Ejemplo de Uso:

```
GET /api/tasks/1
```

Respuesta Exitosa (200):

```
{
  "data": {
    "id": 1,
    "title": "Estudiar para el examen",
    "description": "Revisar capítulos 1-5",
    "isCompleted": false,
    "createdDate": "2025-09-27T10:00:00Z",
    "completedDate": null
  },
  "statusCode": 200,
  "message": "Tarea obtenida exitosamente",
  "success": true
}
```

Respuesta de Error (404):

```
{
  "data": null,
  "statusCode": 404,
  "message": "Tarea con ID 999 no fue encontrada",
  "success": false
}
```

---

3. **POST** `/api/tasks` - Crear nueva tarea

Crea una nueva tarea.

Cuerpo de la Solicitud:

```
{
  "title": "string",        // Requerido, máximo 200 caracteres
```

```
  "description": "string"   // Opcional, máximo 1000 caracteres
}
```

Ejemplo de Uso:

```
POST /api/tasks
Content-Type: application/json

{
  "title": "Completar proyecto",
  "description": "Finalizar la API de gestión de tareas"
}
```

Respuesta Exitosa (201):

```
{
  "data": {
    "id": 3,
    "message": "Tarea creada exitosamente"
  },
  "statusCode": 201,
  "message": "Tarea creada exitosamente",
  "success": true
}
```

Respuesta de Error de Validación (400):

```
{
  "statusCode": 400,
  "message": "Datos de entrada inválidos",
  "errors": [
    "El título es requerido",
    "El título no puede exceder 200 caracteres"
  ]
}
```

---

4. **PUT** `/api/tasks/{id}` - Actualizar tarea completa

Actualiza completamente una tarea existente.

Parámetros de Ruta:

Parámetro	Tipo	Descripción

Parámetro	Tipo	Descripción
`id`	`int`	ID de la tarea

Cuerpo de la Solicitud:

```
{
  "title": "string",        // Requerido, máximo 200 caracteres
  "description": "string",  // Opcional, máximo 1000 caracteres
  "isCompleted": boolean    // Estado de completado
}
```

Ejemplo de Uso:

```
PUT /api/tasks/1
Content-Type: application/json

{
  "title": "Estudiar para el examen final",
  "description": "Revisar todos los capítulos y hacer ejercicios",
  "isCompleted": false
}
```

Respuesta Exitosa (200):

```
{
  "data": {
    "id": 1,
    "title": "Estudiar para el examen final",
    "description": "Revisar todos los capítulos y hacer ejercicios",
    "isCompleted": false,
    "createdDate": "2025-09-27T10:00:00Z",
    "completedDate": null
  },
  "statusCode": 200,
  "message": "Tarea actualizada exitosamente",
  "success": true
}
```

---

5. **PATCH** `/api/tasks/{id}/complete` - Marcar tarea como completada

Marca una tarea específica como completada y establece la fecha de completado.

Parámetros de Ruta:

Parámetro	Tipo	Descripción
`id`	`int`	ID de la tarea

Ejemplo de Uso:

```
PATCH /api/tasks/1/complete
```

Respuesta Exitosa (200):

```
{
  "data": {
    "id": 1,
    "title": "Estudiar para el examen",
    "description": "Revisar capítulos 1-5",
    "isCompleted": true,
    "createdDate": "2025-09-27T10:00:00Z",
    "completedDate": "2025-09-27T15:30:00Z"
  },
  "statusCode": 200,
  "message": "Tarea completada exitosamente",
  "success": true
}
```

Respuesta de Error - Tarea ya completada (409):

```
{
  "data": null,
  "statusCode": 409,
  "message": "La tarea con ID 1 ya está completada",
  "success": false
}
```

---

6. **DELETE** `/api/tasks/{id}` - Eliminar tarea

Elimina permanentemente una tarea.

Parámetros de Ruta:

Parámetro	Tipo	Descripción
`id`	`int`	ID de la tarea

Ejemplo de Uso:

```
DELETE /api/tasks/1
```

Respuesta Exitosa (200):

```
{
  "data": true,
  "statusCode": 200,
  "message": "Tarea eliminada exitosamente",
  "success": true
}
```

Endpoints de Prueba

GET `/api/test/success` - Probar funcionamiento

Endpoint para verificar que la API está funcionando correctamente.

Respuesta (200):

```
{
  "message": "Todo funcionando correctamente",
  "timestamp": "2025-09-27T15:30:00Z"
}
```

GET `/api/test/exception/{type}` - Probar manejo de excepciones

Endpoints para probar el middleware de manejo de excepciones.

Tipos disponibles:

- `notfound` → 404 Not Found
- `validation` → 400 Bad Request
- `completed` → 409 Conflict
- `argument` → 400 Bad Request
- `generic` → 500 Internal Server Error

Ejemplo:

```
GET /api/test/exception/notfound
```

Respuesta (404):

```
{
  "data": null,
  "statusCode": 404,
  "message": "Tarea con ID 999 no fue encontrada",
  "success": false
}
```

📝 Códigos de Estado HTTP

Código	Descripción	Cuándo se usa
200	OK	Operación exitosa
201	Created	Recurso creado exitosamente
400	Bad Request	Datos inválidos o error de validación
404	Not Found	Recurso no encontrado
409	Conflict	Conflicto de estado (ej: tarea ya completada)
500	Internal Server Error	Error interno del servidor

🔧 Ejemplos con cURL

Crear una tarea:

```
curl -X POST "https://localhost:7001/api/tasks" \
  -H "Content-Type: application/json" \
  -d '{
    "title": "Mi nueva tarea",
    "description": "Descripción de la tarea"
  }'
```

Obtener tareas completadas:

```
curl -X GET "https://localhost:7001/api/tasks?isCompleted=true"
```

Actualizar una tarea:

```
curl -X PUT "https://localhost:7001/api/tasks/1" \
  -H "Content-Type: application/json" \
  -d '{
    "title": "Tarea actualizada",
```

```
    "description": "Nueva descripción",
    "isCompleted": false
  }'
```

Completar una tarea:

```
curl -X PATCH "https://localhost:7001/api/tasks/1/complete"
```

Eliminar una tarea:

```
curl -X DELETE "https://localhost:7001/api/tasks/1"
```

🚀 Cómo Empezar

1. **Ejecutar la aplicación:**

   ```
   dotnet run
   ```

2. **Abrir Swagger UI:**
   - Navegar a: `https://localhost:7001/swagger`
   - Interfaz interactiva para probar todos los endpoints

3. **Usar herramientas como:**
   - **Postman** - Colección de endpoints
   - **Thunder Client** (VS Code)
   - **cURL** - Línea de comandos
   - **Swagger UI** - Interfaz web incluida

🔍 Notas Importantes

- Todas las fechas están en formato **ISO 8601** UTC
- Los filtros son **case-insensitive**
- La paginación comienza en **página 1**
- El middleware maneja automáticamente las **excepciones**
- Las validaciones se realizan usando **Data Annotations**
- La API sigue el patrón **RESTful**