ACTIVIDAD II

GESTION DE TAREAS – API RESTFUL

CRISTIAN DAVID MEDINA HERNÁNDEZ | CÓDIGO: 90108

ELECTIVA II

PROFESOR: DIEGO FERNANDO ZÁRATE PINEDA

UNIVERSIDAD INCCA DE COLOMBIA

INGENIERÍA DE SISTEMAS

BOGOTÁ D.C.

20 ENE 2026

Índice

Introducción

Esta actividad tiene como objetivo extender la funcionalidad de la API RESTful desarrollada en la Actividad 1, añadiendo tres mejoras clave:

- Un endpoint para buscar tareas por palabra clave en título o descripción.
- Un endpoint para marcar una tarea como pendiente (revertir el estado de "completada").
- Validaciones de entrada para garantizar que el título de la tarea nunca esté vacío al crear o editar.

Todas las funcionalidades se implementan sin base de datos, manteniendo el almacenamiento en memoria mediante una lista estática, tal como se estableció en la Actividad 1.

Implementación

Los endpoints implementados son los siguientes:

- GET `/api/tareas/buscar`: recibe un parámetro de consulta `q` (por ejemplo: `?q=examen`). La API filtra y devuelve todas las tareas con estado 200 OK, las tareas cuyo título o descripción contengan dicha palabra, realizando una comparación insensible a mayúsculas y minúsculas. Si el parámetro `q`está vacío o no se proporciona, la API devuelve error 400. Bas Request con el mensaje: "El parámetro 'q' es requerido".
- PATCH `/api/tareas/{id}/pendiente`: permite cambiar el estado de una tarea de completada a pendiente. Esto es útil cuando una tarea marcada como finalizada debe reactivarse. El método busca la tarea por su `id`. Si no existe, devuelve 404 Not Found. Si existe, establece `IsCompleted = false` y devuelve la tarea actualizada con código 200 OK.

Para garantizar la integridad de los datos, se aplicaron Data Annotations en el modelo `TaskModel`:

- `[Required(ErrorMessage = "El título es obligatorio.")]`
- `[MinLength(1, ErrorMessage = "El título no puede estar vacío.")]`

Además, en los métodos `CreateTask` y `UpdateTask` del controlador, se agregó la validación automática:

```
if (!ModelState.IsValid)
        return BadRequest(ModelState);
```

Pruebas con Postman

A continuación se lista una serie de ejemplos correspondientes a posibles solicitudes y respuestas al utilizar la herramieta postman.

1. Crear tarea (POST /api/tareas)
   Cuerpo:

```
{
        "title": "Preparar informe final",
        "description": "Incluir gráficos y conclusiones"

}
```

   Respuesta esperada (201 Created):

```
{
        "id": 1, "title": "Preparar informe final",
        "description": "Incluir gráficos y conclusiones",
        "isCompleted": false

}
```

   Si falta el título (400 Bad Request):

```
{
        "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",
        "title": "One or more validation errors occurred.",
        "status": 400,
        "errors": {
                "Title": [
                        "El título es obligatorio.",
                        "El título no puede estar vacío."
                ]
        }
}
```

2. Editar tarea (PUT /api/tareas/1)
   Cuerpo:
   {
   "title": "Preparar informe final",
   "description": "Incluir gráficos y enviar al tutor"
   }
   Respuesta esperada (200 OK)
   Si “id” no existe (404 Not Found)
   Si falta el título (400 Bad Request):
   {
   "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",
   "title": "One or more validation errors occurred.",
   "status": 400,
   "errors": {
   "Title": [
   "El título es obligatorio.",
   "El título no puede estar vacío."
   ]
   }
   }

3. Buscar tareas (GET /api/tareas/buscar?q=informe)
   Cuerpo: vacío
   Respuesta esperada (200 OK):
   {
   "id": 1, "title": "Preparar informe final",
   "description": "Incluir gráficos y conclusiones",
   "isCompleted": false
   }
   Si no existe la palabra clave (200 OK):
   [] devuelve un json vacío
   Si falta parámetro `q`o está vacío (400 Bad Request):
   {
   "type": "https://tools.ietf.org/html/rfc9110#section-15.5.1",
   "title": "One or more validation errors occurred.",
   "status": 400,
   "errors": {
   "q": [
   "El parámetro ‘q’es requerido."

```
            ]
        }
}
```


4. Marcar una tarea como completada (PATCH /api/tareas/1/pendiente)
   Cuerpo: vacío
   Respuesta esperada (200 OK):

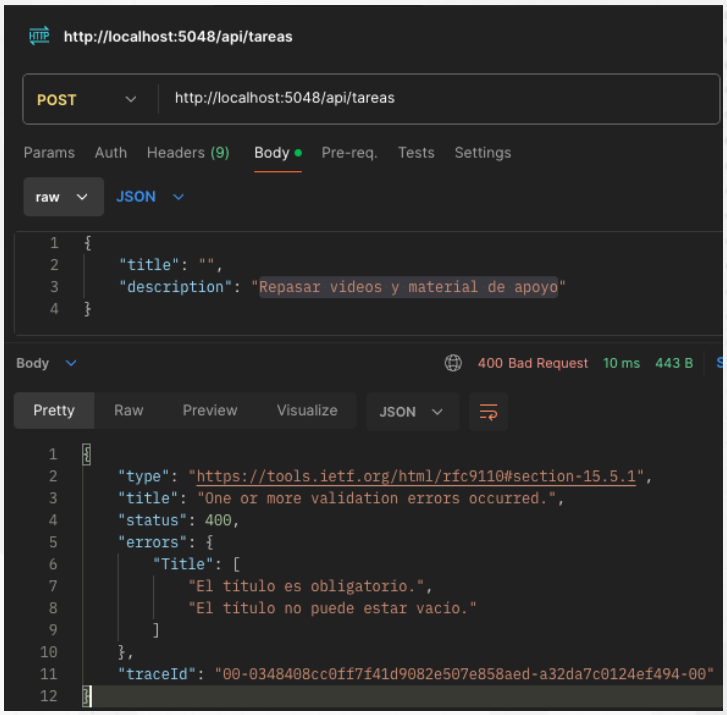

```
{
    "id": 1,
    "title": "Preparar informe final",
    "description": "Incluir gráficos y enviar al tutor",
    "isCompleted": false
}
```


   Si “id” no existe (404 Not Found)

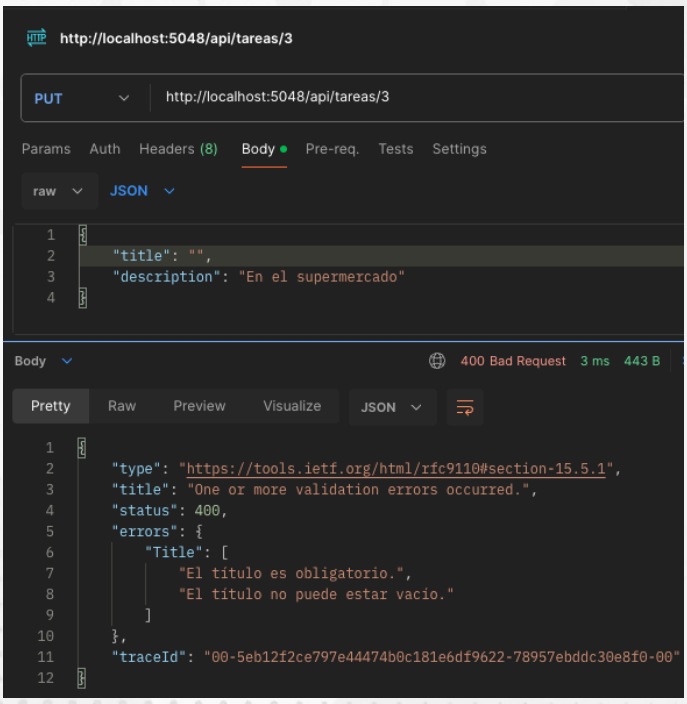
Se realizaron pruebas exhaustivas utilizando Postman para validar cada endpoint. Las pruebas incluyeron casos de éxito y manejo de errores (por ejemplo, enviar un título vacío o un ID inexistente). Todas las respuestas incluyeron los códigos de estado correctos y cuerpos JSON coherentes con el modelo `TaskModel`.

Las capturas de pantalla de las pruebas se incluyen como evidencia en la sección siguiente. Cada imagen muestra la URL, método HTTP, cuerpo de la solicitud (si aplica), código de estado y respuesta JSON.

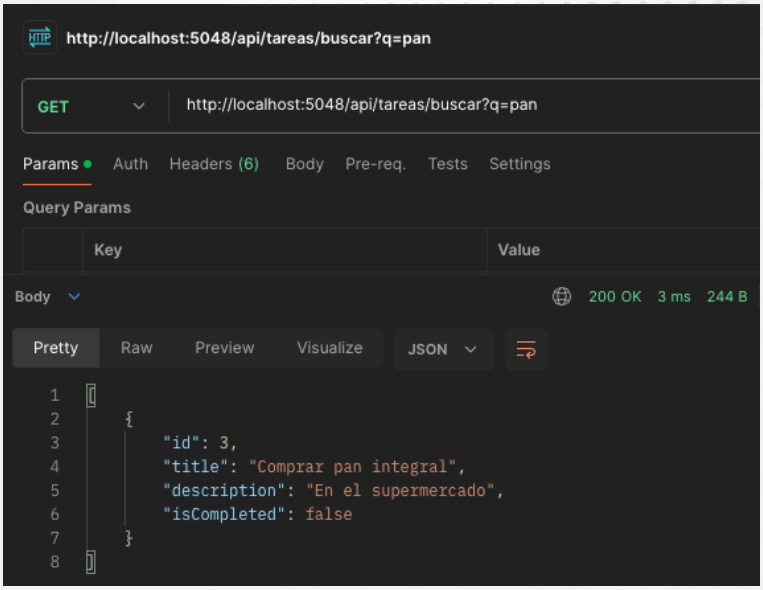
Capturas de Pantalla



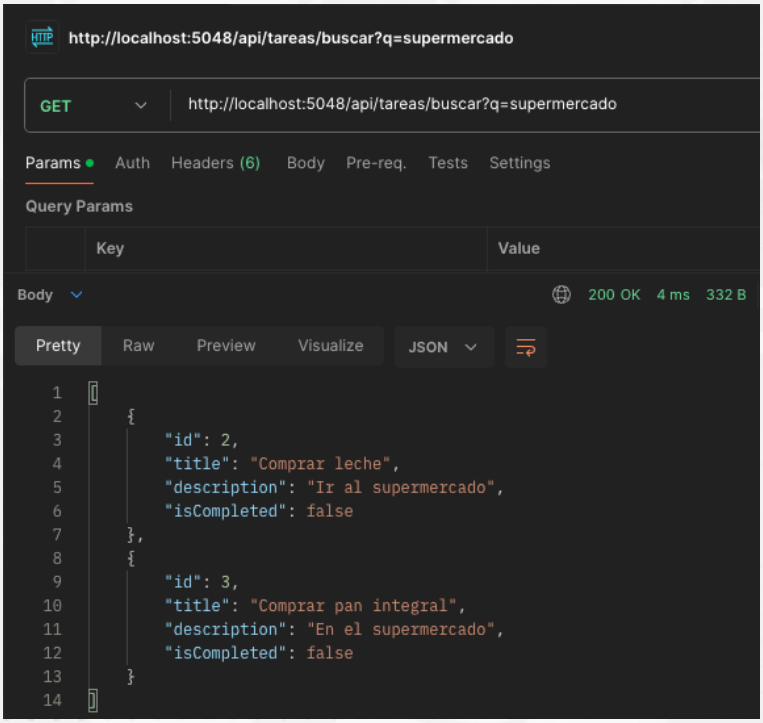
Captura 1 - Prueba POST: creación de tarea (título vacio)



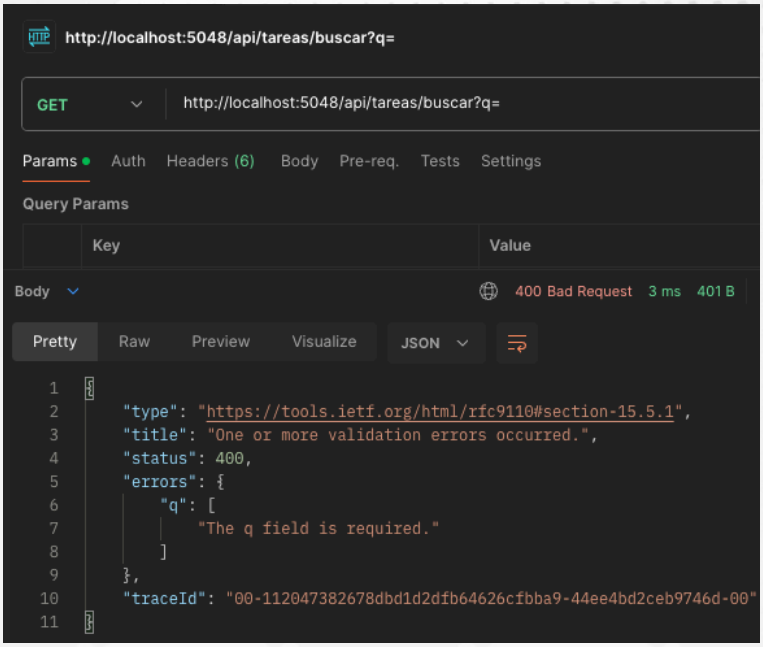
Captura 2 - Prueba PUT: edición de tarea (título vacío)



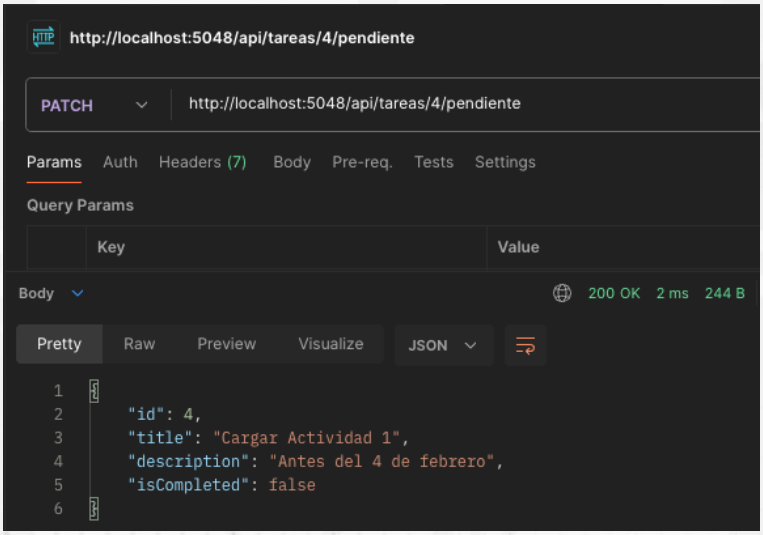
Captura 3 - Prueba GET: búsqueda de tarea por palabra clave



Captura 4 - Prueba GET: búsqueda de tarea por palabra clave



Captura 5 - Prueba GET: búsqueda de tarea (con parámetro q vacío)



Captura 6 - Prueba PATCH: marcar tarea como pendiente

Conclusión

La API ahora ofrece mayor flexibilidad y robustez gracias a la búsqueda textual, la reversión de estado y las validaciones automáticas. Estas mejoras se lograron sin alterar la arquitectura original ni introducir dependencias externas, manteniendo el diseño ligero y eficiente del proyecto inicial.

Apendice

- Repositorio público: https://github.com/cdmedinah-unincca/TaskApi
- README.md: (Actualiado) disponible en la raíz del repositorio
- Colección de Postman: exportada como `TaskApi.postman_collection_v0.2.json`