Gestión de recursos

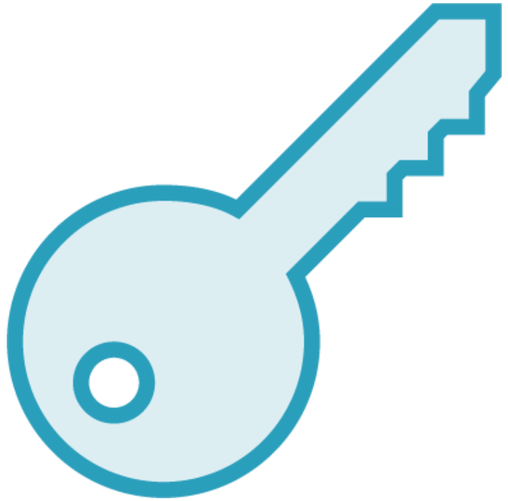
Resumen

- Seguridad e idempotencia de métodos
- Escenarios avanzados de creación de recursos
  - Padre-hijo
  - Recopilación de una sola vez
- PATCH vs. PUT
- Upserting

Resumen

- Soporte de OPTIONS
- Inspección de los formateadores de entrada
- Resumen de métodos HTTP por caso de uso

Seguridad de métodos e idempotencia de métodos



- Un método se considera seguro cuando no cambia la representación del recurso

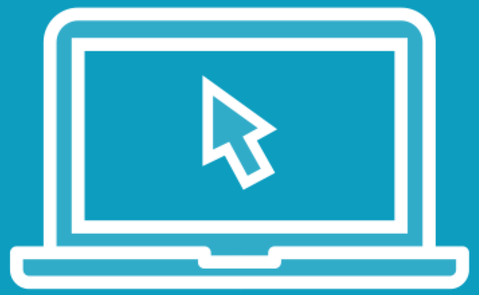
- Un método se considera idempotente cuando puede ser llamado varias veces con el mismo resultado

Seguridad e idempotencia de métodos

Método HTTP	¿Seguro?	¿Idempotente?
GET	si	si
OPTIONS	si	si
HEAD	si	si
POST	no	no
DELETE	no	si
PUT	no	si
PATCH	no	no

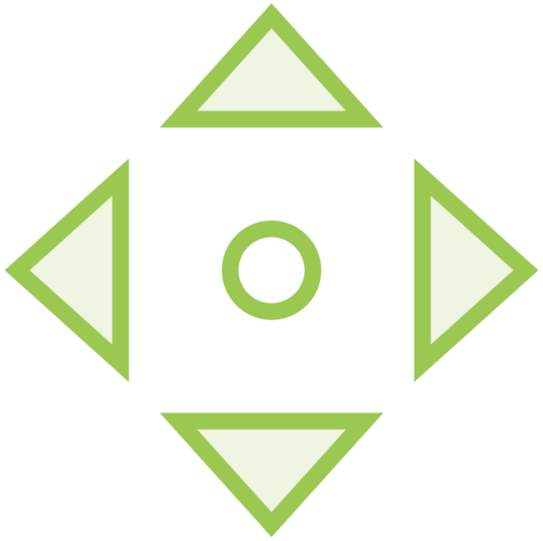
La seguridad e idempotencia ayudan a decidir qué método utilizar para cada caso de uso

Demo



Inspección y corrección de métodos POST

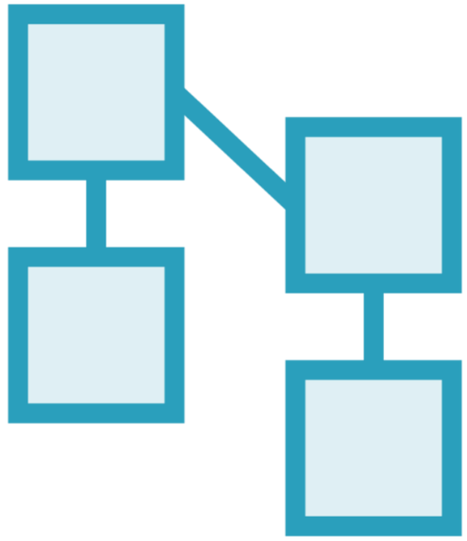
Ventajas de aplicar el atributo ApiController



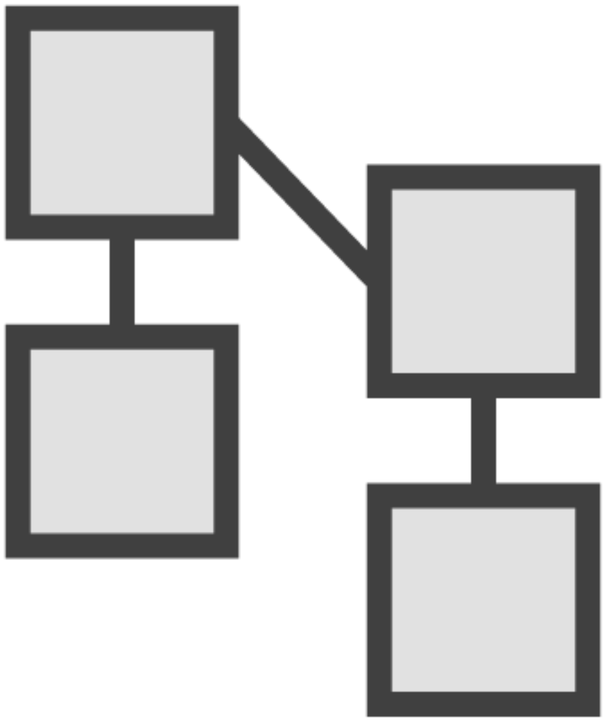
El enrutamiento basado en atributos es obligatorio

Se devuelve un objeto ProblemDetails en caso de errores

Se devuelve el código de estado 400 (solicitud incorrecta) en caso de entrada no válida



Las reglas de enlace de modelos se ajustan para adaptarse mejor a las API



[FromBody]
- Cuerpo de la petición

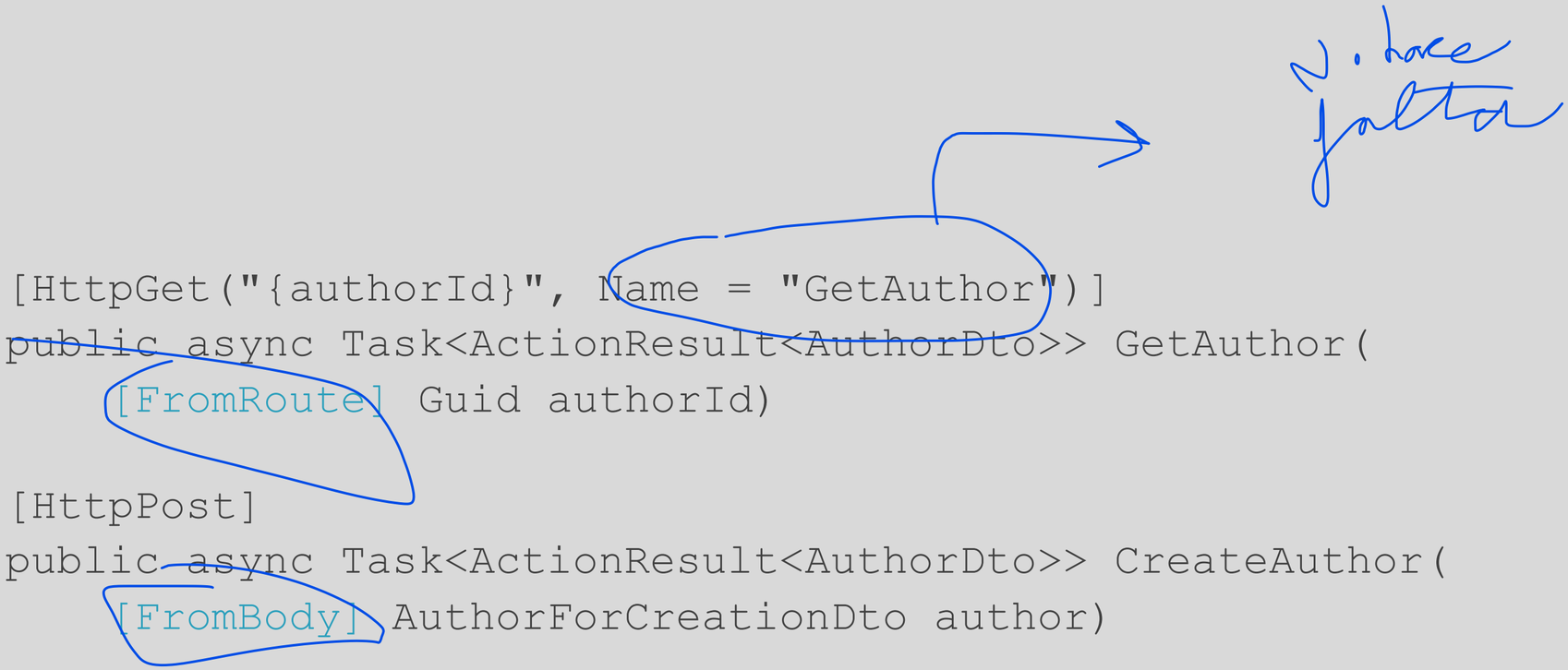
[FromForm]
- Datos del formulario en el cuerpo de la solicitud

[FromHeader]
- Cabecera de la solicitud

[FromQuery]
- Parámetros de la cadena de consulta

[FromRoute] la q hemos aprendido
- Datos de la ruta de la solicitud actual

[FromService]
- El servicio inyectado como parámetro de la acción



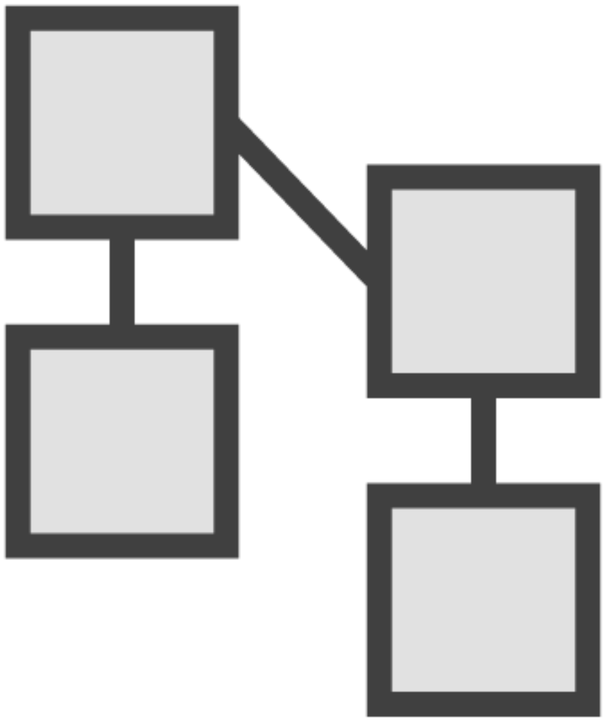
```
[HttpGet("{authorId}", Name = "GetAuthor")]
public async Task<ActionResult<AuthorDto>> GetAuthor(
    [FromRoute] Guid authorId)

[HttpPost]
public async Task<ActionResult<AuthorDto>> CreateAuthor(
    [FromBody] AuthorForCreationDto author)
```

Recibe un Id y devuelve un autor. Viene de la ruta

crea un autor y lo devuelve

Enlace de modelos con atributos de origen de enlace

Común para las API: `[FromBody]`, `[FromHeader]`, `[FromQuery]` y `[FromRoute]`



`[FromBody]`

- Inferido para tipos complejos

`[FromForm]`

- Inferido para parámetros de acción de tipo IFormFile y IFormFileCollection

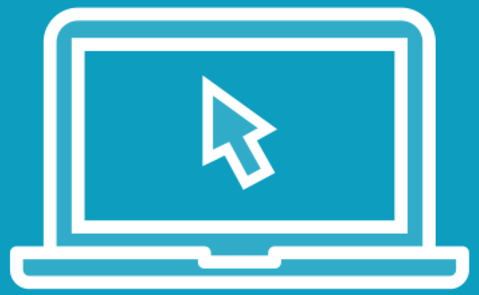
`[FromRoute]`

- Inferido para cualquier nombre de parámetro de acción que coincida con un parámetro de la plantilla de ruta

`[FromQuery]`

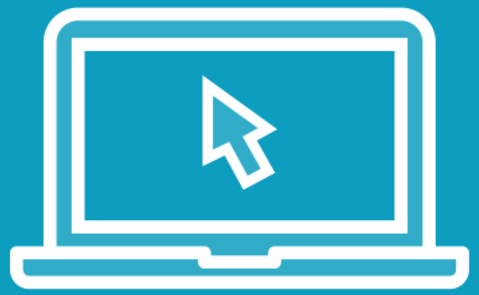
- Inferido para cualquier otro parámetro de acción

Demo



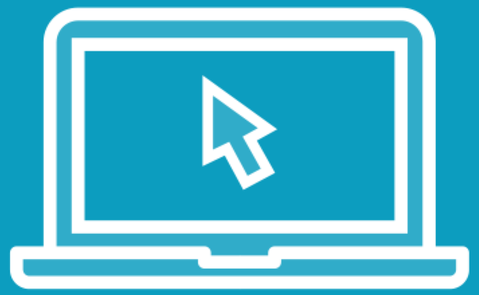
Creación de recursos hijos junto al recurso padre

Demo



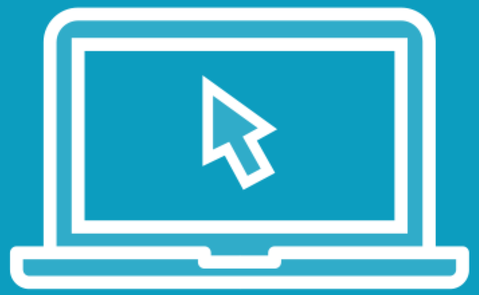
Crear una colección de recursos

Demo



Trabajar con arrays de claves y claves compuestas

Demo



Gestión de POST a un solo recurso

Actualizaciones completas (PUT) frente a actualizaciones parciales (PATCH)

- PUT se utiliza para realizar actualizaciones completas
  - Todos los campos del recurso se sobreescriben o se inicializan a su valor predeterminado
- PATCH se utiliza para realizar actualizaciones parciales
  - Permite enviar conjuntos de cambios mediante un JsonPatchDocument

Actualizaciones completas (PUT) frente a actualizaciones parciales (PATCH)

- HTTP PATCH se utiliza para actualizaciones parciales
  - El cuerpo de una solicitud de una actualización parcial se describe en el RFC 6902 (JSON Patch) https://tools.ietf.org/html/rfc6902
- Las peticiones PATCH deben enviarse con el tipo de medio "application/json-patch+json"

```
[
    {
        "op": "replace",
        "path": "/title",
        "value": “new title"
    }
    ,
    {
        "op": “remove",
        "path": "/description"
    }
]
```

- Array de operaciones
- Operación “replace”
- La propiedad “title” se actualiza con el valor “new title”
- Operación “remove”
- La propiedad “description” se inicializa a su valor predeterminado

Operaciones JSON Patch

Add

```
{"op": "add",
"path": "/a/b",
"value": "foo"}
```

Remove

```
{"op": "remove",
"path": "/a/b"}
```

Replace

```
{"op": "replace",
"path": "/a/b",
"value": "foo"}
```

Operaciones JSON Patch

Copy

```
{"op": "copy",
"from": "/a/b",
"path": "/a/c"}
```

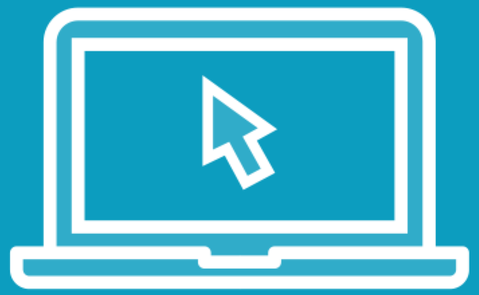
Move

```
{"op": "move",
"from": "a/b",
"path": "/a/c"}
```

Test

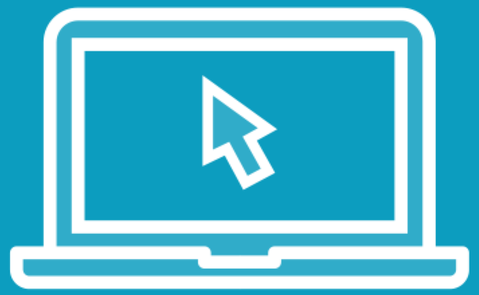
```
{"op": "test",
"path": "/a/b",
"value": "foo"}
```

Demo



Inspección de una acción del tipo PUT

Demo



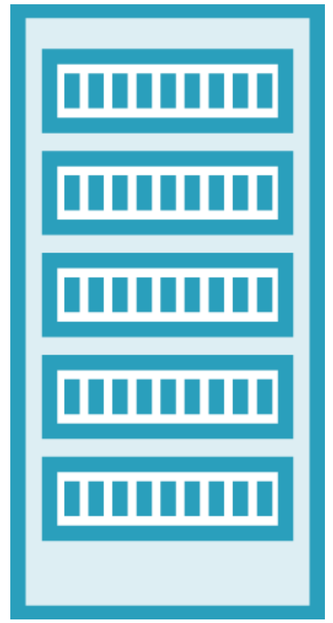
Soporte de actualizaciones parciales mediante PATCH

Uso de PUT o PATCH para crear recursos: Upserting

http://myapi/authors

http://myapi/authors/{guid}

http://myapi/authors/1



Upserting

- El servidor es responsable de la generación del URI

La solicitud PUT debe ir a un URI existente

Si no existe, se devuelve un 404

Se debe utilizar POST para la creación, ya que no podemos conocer el URI de antemano

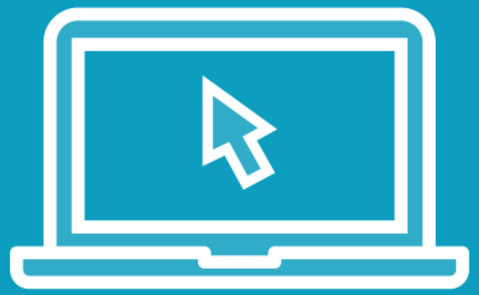
- El consumidor es responsable de la generación del URI

La solicitud PUT puede enviarse a un URI inexistente, ya que el consumidor puede crearlo

Si no existe, se crea el recurso

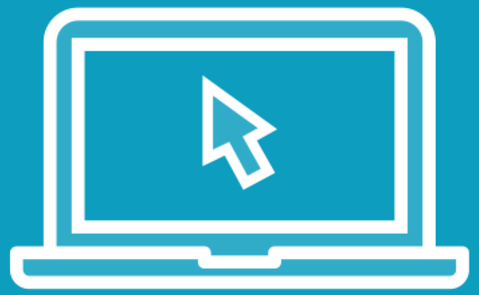
PUT puede utilizarse para la creación: upsert

Demo



Upserting con PUT

Demo



Upserting con PATCH

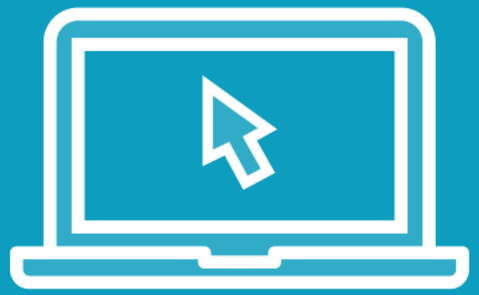
```
PUT http://host/api/authors/{authorId}/courses

DELETE http://host/api/authors
```

Consideraciones sobre acciones destructivas

Desde el punto de vista de REST, un recurso es sólo un recurso.

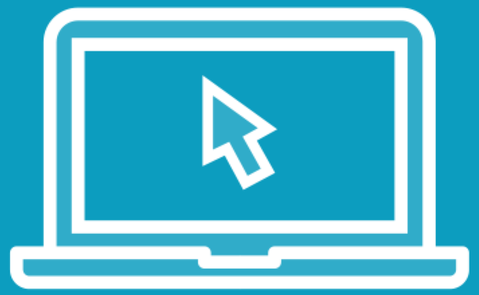
Se permite el uso de acciones destructivas, pero se desaconseja su uso.

Demo



Soporte de OPTIONS

Demo



Inspección de los formateadores de entrada

Revisión de métodos HTTP por caso de uso

Leer recursos

GET api/authors

200 [{author},{author}], 404

GET api/authors/{authorId}

200 {author}, 404

Borrar recursos

DELETE api/authors/{authorId}

204, 404

DELETE api/authors

204, 404

Rara vez se implementa

Revisión de métodos HTTP por caso de uso

Creación de recursos (servidor)

POST api/authors – {author}

201 {author}, 404

POST api/authors/{authorId} nunca

debería funcionar (404 or 409)

Crear un nuevo recurso para añadir una colección de una sola vez

POST api/authorcollections – {authorCollection}

201 {authorCollection}, 404

Creación de recursos (cliente)

PUT api/authors/{authorId} - {author}

201 {author}

PATCH api/authors/{authorId} - {JsonPatchDocument on author}

201 {author}

Revisión de métodos HTTP por caso de uso

Actualización de recursos (completa)

PUT api/authors/{authorId} – {author}

200 {author}, 204, 404

PUT api/authors – [{author}, {author}]

200 [{author}, {author}], 204, 404

Rara vez se implementa

Actualización de recursos (parcial)

PATCH api/authors/{authorId} - {JsonPatchDocument sobre author}

200 {author}, 204, 404

PATCH api/authors – {JsonPatchDocument sobre authors}

200 [{author}, {author}], 204, 404

Rara vez se implementa

Resumen

- Un método se considera seguro cuando no cambia la representación del recurso
- Un método se considera idempotente cuando puede ser llamado varias veces con el mismo resultado

Resumen

- Admitir la creación de una colección de recursos de una sola vez mediante la creación de un nuevo recurso
- Devolver un "405 Método no permitido" cuando no se permite el POST

Resumen

- **PUT se utiliza para actualizaciones completas**
  - Todos los campos de los recursos deberán actualizarse o inicializarse a sus valores predeterminados.
- **PATCH se utiliza para actualizaciones parciales**
  - El cuerpo de la solicitud es una lista de operaciones (un "conjunto de cambios") descritas por el estándar Json
  - Patch
  - Método recomendado

Resumen

- **Upserting consiste en crear un recurso con PUT o PATCH**
  - Es útil cuando el cliente puede decidir el URI del recurso.
- **A través de una petición OPTIONS un cliente puede saber lo que está permitido para un recurso determinado**

Resumen

- La compatibilidad con diferentes formatos de entrada se habilita a través de los formateadores de entrada

A continuación:
Validación de datos y notificación de errores de validación