Manipulating Resources

Kevin Dockx
Architect

@KevinDockx https://www.kevindockx.com

Coming Up

Method safety and method idempotency

Advanced resource creation scenarios

- Parent-child
- Collection in one go

PATCH vs. PUT

Upserting

Coming Up

Supporting OPTIONS

Inspecting input formatters

HTTP method overview by use case

Method Safety and Method Idempotency

A method is considered safe when it doesn't change the resource representation

A method is considered idempotent when it can be called multiple times with the same result

Method Safety and Method Idempotency

HTTP method	Safe?	Idempotent?
GET	Yes	Yes
OPTIONS	Yes	Yes
HEAD	Yes	Yes
POST	No	No
DELETE	No	Yes
PUT	No	Yes
PATCH	No	No

Method safety and idempotency help decide which method to use for which use case

Demo

Inspecting and fixing POST methods

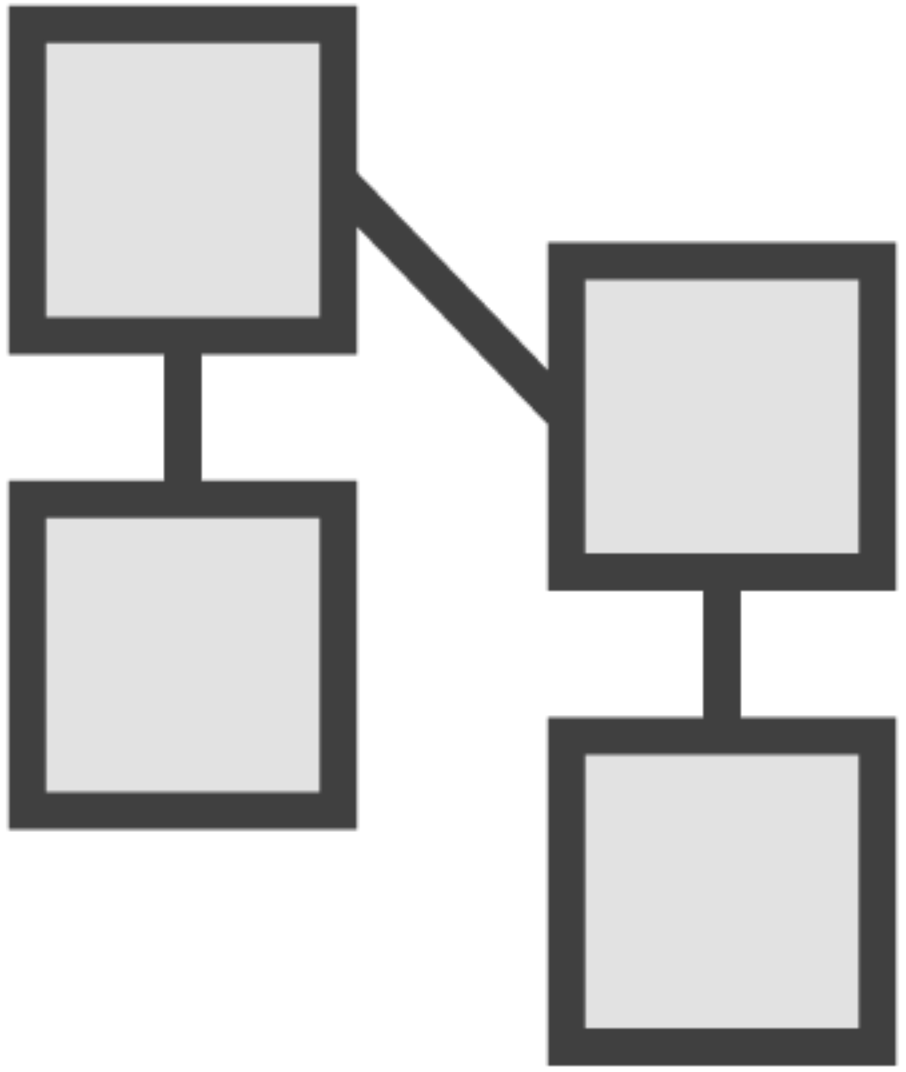
Advantages of Applying the ApiController Attribute

Attribute-based routing is obligatory

A `ProblemDetails` object is returned in case of mistakes

Status code 400 (Bad request) is returned on invalid input

Model binding rules are adjusted to better fit APIs



`[FromBody]`
- Request body

`[FromForm]`
- Form data in the request body

`[FromHeader]`
- Request header

`[FromQuery]`
- Query string parameters

`[FromRoute]`
- Route data from the current request

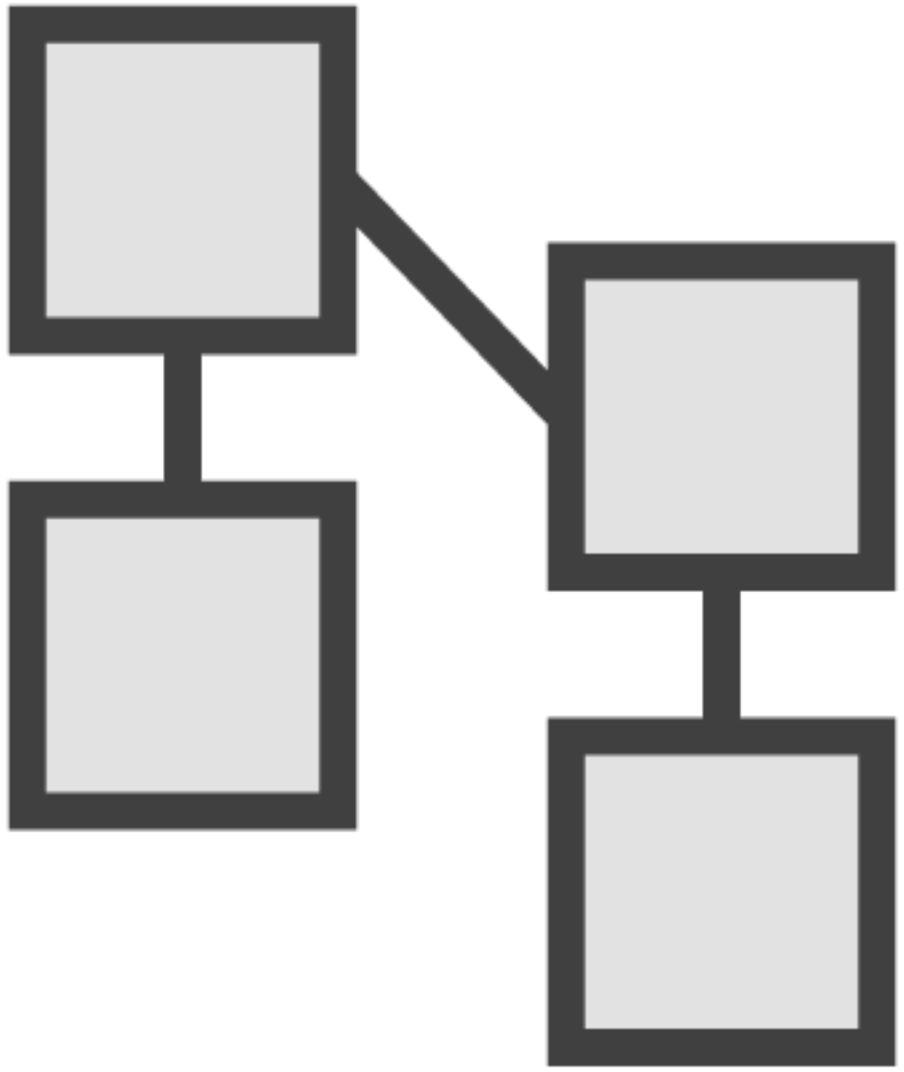
`[FromService]`
- The service injected as action parameter

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

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

Model Binding with Binding Source Attributes

Common for APIs: `[FromBody]`, `[FromHeader]`, `[FromQuery]` **and** `[FromRoute]`



`[FromBody]`

- Inferred for complex types

`[FromForm]`

- Inferred for action parameters of type `IFormFile` and `IFormFileCollection`

`[FromRoute]`

- Inferred for any action parameter name matching a parameter in the route template

`[FromQuery]`

- Inferred for any other action parameters

Demo

Creating child resources together with a parent resource

Demo

Creating a collection of resources

Demo

Working with array keys and composite keys

Demo

Handling POST to a single resource

Full Updates (PUT) vs. Partial Updates (PATCH)

PUT is for full updates

- All resource fields are either overwritten or set to their default values

PATCH is for partial updates

- Allows sending over change sets via `JsonPatchDocument`

Full Updates (PUT) vs. Partial Updates (PATCH)

HTTP PATCH is for partial updates

- The request body of a patch request is described by RFC 6902 (JSON Patch) https://tools.ietf.org/html/rfc6902

PATCH requests should be sent with media type “application/json-patch+json”

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

◀ array of operations

◀ “replace” operation

◀ “title” property gets value “new title”

◀ “remove” operation

◀ “description” property is removed (set to its default value)

JSON Patch Operations

Add

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

Remove

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

Replace

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

JSON Patch Operations

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"}
```

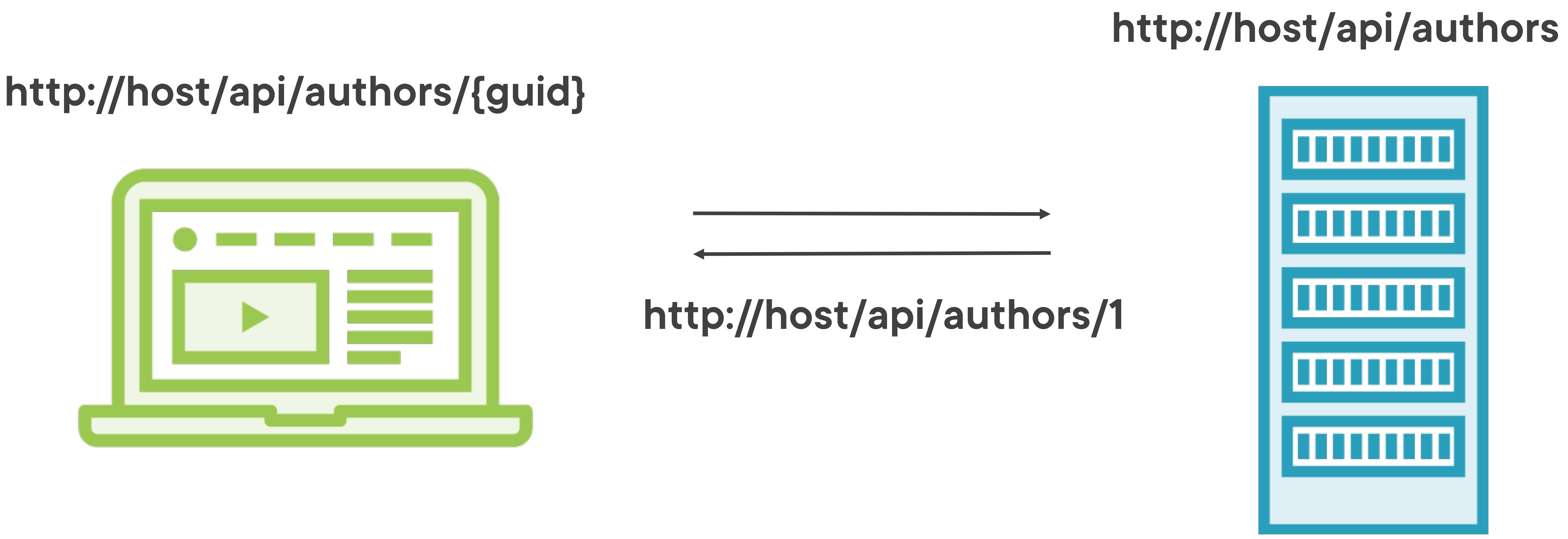
Demo

Inspecting a PUT action

Demo

Supporting partial updates with PATCH

Using PUT or PATCH for Creating Resources: Upserting



Upserting

Server is responsible for URI generation

PUT/PATCH request must go to an existing URI

If it doesn’t exist, a 404 is returned

POST must be used for creation as we cannot know the URI in advance

Consumer is responsible for URI generation

PUT/PATCH request can be sent to an unexisting URI, because the consumer is allowed to create it

If it doesn’t exist, the resource is created

PUT/PATCH can be used for creation: upserting

Demo

Upserting with PUT

Demo

Upserting with PATCH

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

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

Considering Destructive Actions

From REST's POV, a resource is just a resource
Destructive actions are allowed, but advised against

Demo

Supporting OPTIONS

Demo

Inspecting input formatters

HTTP Method Overview by Use Case

Reading resources

GET api/authors

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

GET api/authors/{authorId}

200 {author}, 404

Deleting resources

DELETE api/authors/{authorId}

204, 404

DELETE api/authors

204, 404

Rarely implemented

HTTP Method Overview by Use Case

Creating resources (server)

POST api/authors – {author}

201 {author}, 404

POST api/authors/{authorId} can never be successful (404 or 409)

Create a new resource for adding a collection in one go

POST api/authorcollections – {authorCollection}

201 {authorCollection}, 404

Creating resources (consumer)

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

201 {author}

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

201 {author}

HTTP Method Overview by Use Case

Updating resources (full)

PUT api/authors/{authorId} – {author}

200 {author}, 204, 404

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

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

Rarely implemented

Updating resources (partial)

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

200 {author}, 204, 404

PATCH api/authors – {JsonPatchDocument on authors}

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

Rarely implemented

Summary

A method is considered safe when it doesn't change the resource representation

A method is considered idempotent when it can be called multiple times with the same result

Summary

Support the creation of a collection of resources in one go by creating a new resource

Return a “405 Method not allowed” when POSTing is not allowed

Summary

PUT is for full updates

- All resource fields must be updated / set to their default values

PATCH is for partial updates

- Request body is a list of operations (a "change set") described by the Json Patch standard
- Preferred approach

Summary

Upserting is creating a resource with PUT or PATCH

- Useful when the consumer is allowed to decide on the resource URI

Through an OPTIONS request a consumer can learn what is allowed for a given resource

Summary

Support for different input formats is enabled via input formatters

Up Next:

Validating Data and Reporting Validation Errors