Implementing Advanced RESTful Concerns with ASP.NET Core 3

SUPPORTING PAGING FOR COLLECTION RESOURCES

Kevin Dockx
ARCHITECT

@KevinDockx https://www.kevindockx.com

Coming Up

Positioning this course

Tooling and framework versions

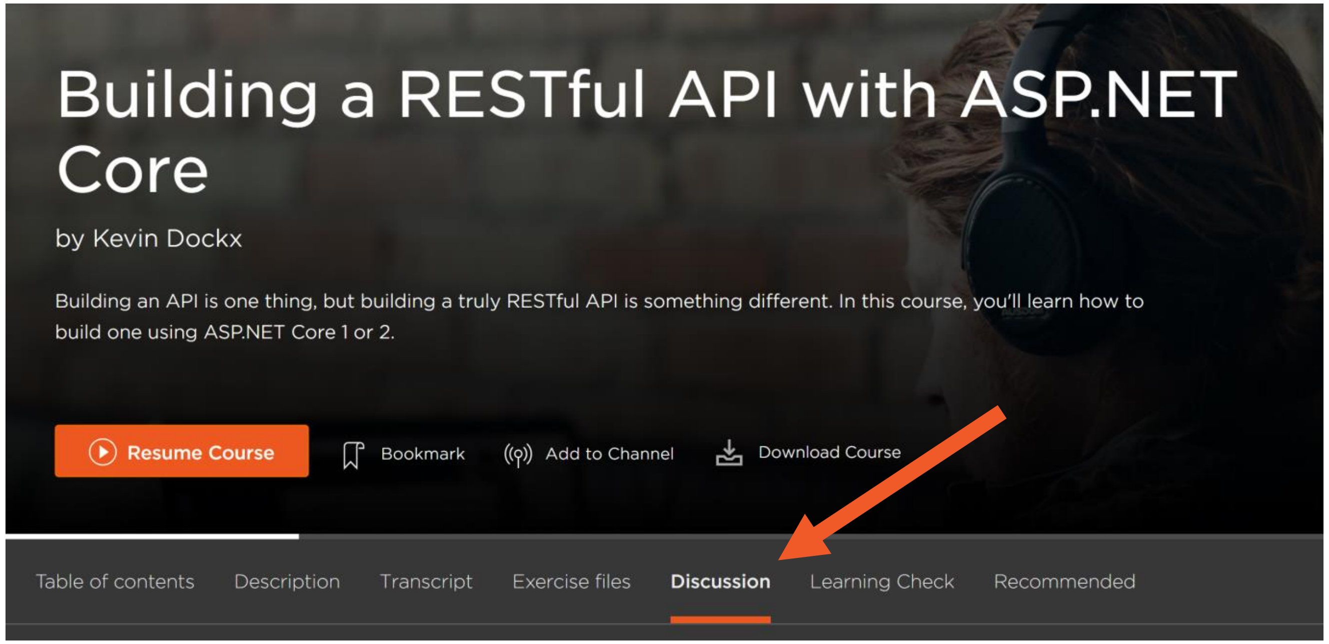
Recapitulating REST

Paging

- Paging through data right up to the data store
- Deferred execution
- Pagination metadata

Discussion tab on the course page

Twitter: @KevinDockx



Building a RESTful API with ASP.NET Core

by Kevin Dockx

Building an API is one thing, but building a truly RESTful API is something different. In this course, you'll learn how to build one using ASP.NET Core 1 or 2.

(course shown is one of my other courses, not this one)

Positioning this Course

Three focus points: REST, REST, and REST

Good knowledge of C#

Some knowledge of ASP.NET Core

ASP.NET Core Fundamentals (Scott Allen)

Building a RESTful API in ASP.NET Core 3

- Basic concerns like CRUD, RESTful contract design, content negotiation, data validation …

Implementing Advanced RESTful Concerns in ASP.NET Core 3

- The course you’re currently watching

Tooling

Visual Studio 2019
(version 16.3 or better)

Visual Studio Code

Visual Studio for Mac

JetBrains Rider,
Sublime…

Tooling

Postman

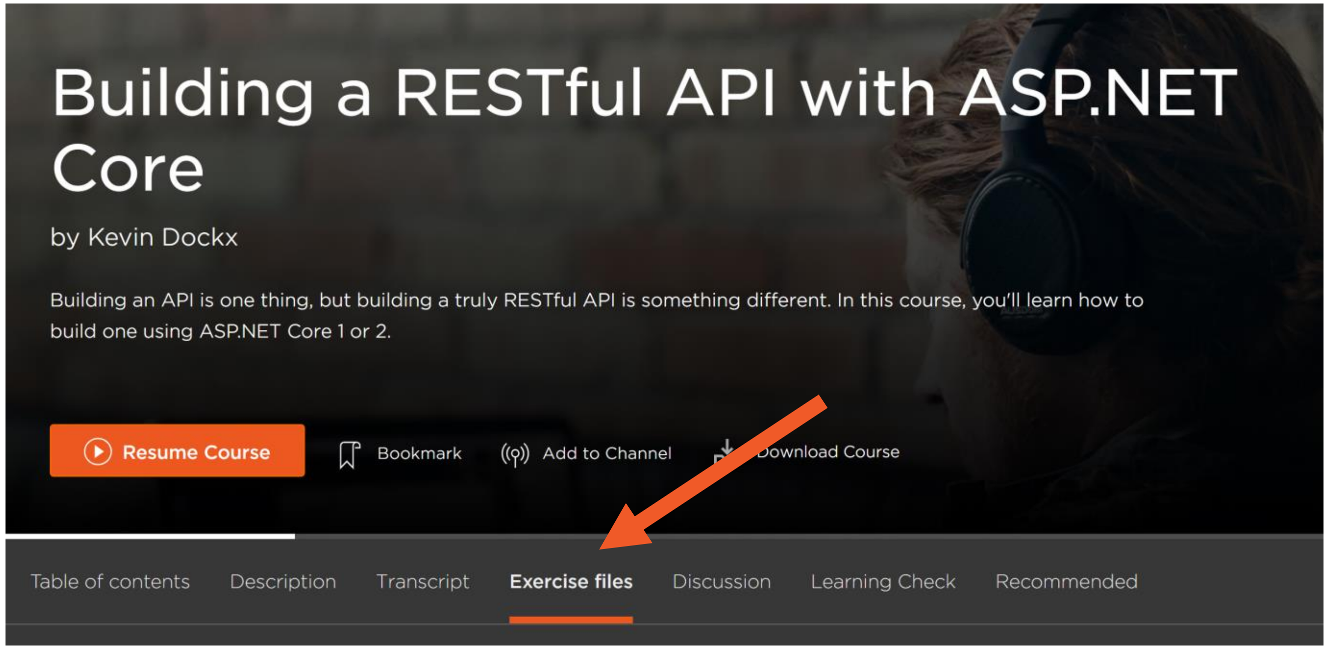
https://www.getpostman.com/

A browser of choice

Exercise files tab on the course page

Postman collection

Implementing_ Advanced_RESTful_ Concerns .postman_collection


Building a RESTful API with ASP.NET Core

by Kevin Dockx

Building an API is one thing, but building a truly RESTful API is something different. In this course, you'll learn how to build one using ASP.NET Core 1 or 2.

(course shown is one of my other courses, not this one)

Demo

Introducing the demo application

Functional Requirements and Constraints

Functional requirements

- Paging, sorting, data shaping

Constraints

- HATEOAS, caching, …

Correctly implementing functional requirements requires some knowledge of the constraints

Recapitulating REST

REST is defined by 6 constraints (one optional)

A constraint is a design decision that can have positive and negative impacts

Recapitulating REST

Uniform Interface

Client-Server

Statelessness

Layered System

Cacheable

Code on Demand (optional)

Recapitulating REST

Uniform Interface	Client-Server	Statelessness
Layered System	Cacheable	Code on Demand (optional)

Uniform Interface Constraint

API and consumers share one single, technical interface:

- URI (resource identifier)
- HTTP method
- Media type

The tighter this contract is, the more reliable and evolvable client and API become

Recapitulating REST

URI
(resource identifier)

HTTP method

Payload

Recapitulating REST

URI
(resource identifier)

HTTP method

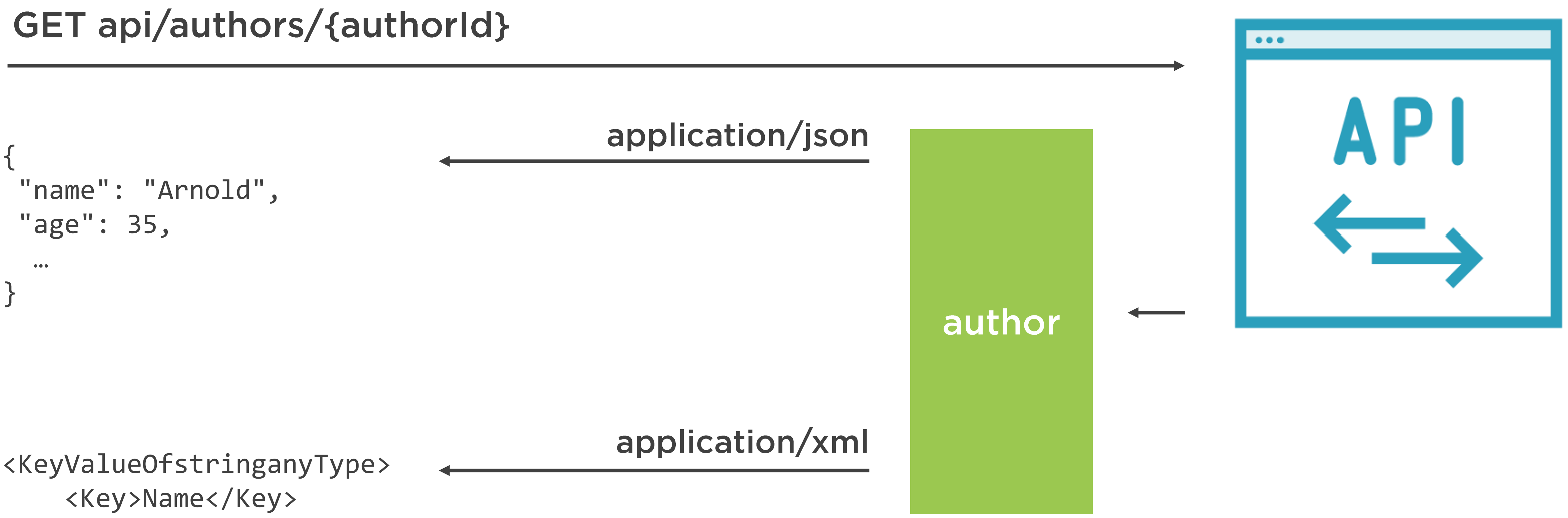
Payload

Identification of Resources

Individual resources are identified in requests using URIs

A resource is conceptually separate from its representation

Identification of Resources



```
{
 "name": "Arnold",
 "age": 35,
  …
}
```

```
<KeyValueOfstringanyType>
    <Key>Name</Key>
    <Value xmlns:d4p1="http://www.w3.org/2001/XMLSchema"
           i:type="d4p1:string">Arnold</Value>
</KeyValueOfstringanyType>
```

Recapitulating REST

URI
(resource identifier)

Identification of resources

HTTP method

Payload

Manipulation of Resources through Representations

Representation + metadata should be sufficient to modify or delete the resource

Recapitulating REST

URI
(resource identifier)

Identification of resources

HTTP method

Payload

Manipulation of resources through representations

Self-descriptive Message

Each message must include enough info to describe how to process the message

Self-descriptive Message

POST api/authors

```
{
 "name": "Arnold",
 "mainCategory": "Rum",
   …
}
```

Content-Type: application/json

Recapitulating REST

URI
(resource identifier)

Identification of resources

HTTP method

Payload

Manipulation of resources through representations

Self-descriptive message

Hypermedia as the Engine of Application State (HATEOAS)

Allows for a self-documenting API

Drives how to consume and use the API

Recapitulating REST

URI
(resource identifier)

Identification of resources

HTTP method

Payload

Manipulation of resources through representations

Self-descriptive message

HATEOAS

“A REST API should spend almost all of its descriptive effort in defining the media type(s) used for representing resources”

Roy Fielding - https://bit.ly/2Kmsung

Cacheable Constraint

Each response message must explicitly state if it can be cached or not

Paging through Collection Resources

Collection resources often grow quite large

- Implement paging on all of them

Paging helps avoid performance issues

Paging through Collection Resources

Parameters are passed through via the query string

- http://host/api/authors?pageNumber=1&pageSize=5

Page size should be limited

Page by default

Page all the way through to the underlying data store

Demo

Paging through collection resources

Deferred Execution

Query execution occurs sometime after the query is constructed

Deferred Execution

A query variable stores query commands, not results

- `IQueryable<T>`: creates an expression tree

Execution is deferred until the query is iterated over

- foreach loop
- `ToList()`, `ToArray()`, `ToDictionary()`
- Singleton queries

Returning Pagination Metadata

Should at least include links to the previous and next pages

Can include additional information: total count, amount of pages, …

```
{
  “results”: [ {author}, {author}, …],
  “metadata”: { “previousPage” : “/api/...”, …}
}
```

Pagination Metadata

Response body no longer matches the Accept header: this isn’t application/json, it’s a new media type

Breaks the self-descriptive message constraint: the consumer does not know how to interpret the response with content-type application/json

Returning Pagination Metadata

When requesting application/json, paging metadata isn't part of the resource representation

Use a custom header, like X-Pagination

Implementing a PagedList<T> Class

Custom `PagedList<T>` class

- CurrentPage, TotalPages, HasPrevious, HasNext

Class can be reused for all collection resources

Demo

Improving reuse with a PagedList<T> class

Demo

Returning pagination metadata

Summary

When correctly implemented, paging will improve performance

Pass page size and page number as query string parameters

- Limit page size
- Provide default values for page size and page number
- Metadata belongs in a custom header

Summary

Page all the way through to the underlying data store

- Deferred execution
- `Skip(), Take()`

Page by default

Return pagination metadata