

Building a RESTful API with ASP.NET Core

INTRODUCING REST



Kevin Dockx

ARCHITECT

@KevinDockx <https://www.kevindockx.com>



Coming Up



Course prerequisites & tooling

Introducing REST

Learning what the REST constraints are about

The Richardson Maturity Model

Positioning ASP.NET Core for building RESTful APIs



Course Prerequisites



Three focus points:
REST, REST and REST

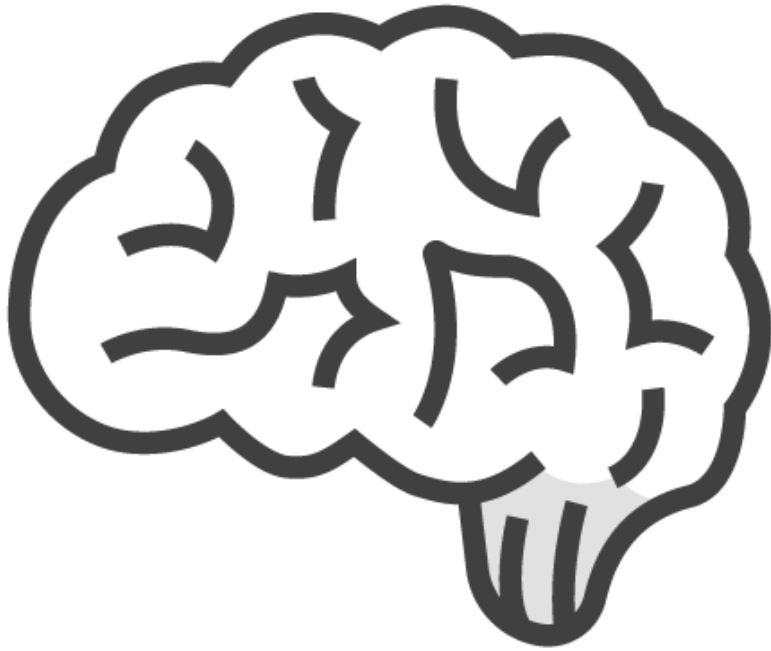


Good knowledge of
C#



Some knowledge of
ASP.NET Core

Course Prerequisites



ASP.NET Core Fundamentals (Scott Allen)

- <http://bit.ly/2gg9WSH>

Building Your First API with ASP.NET Core (yours truly)

- <http://bit.ly/2gmeTdO>



Tooling



Visual Studio 2015

ASP.NET Core 1
(update 3)

project.json

<http://bit.ly/2g0JDmh>

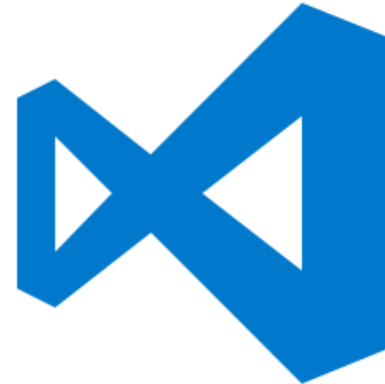


Visual Studio 2017

ASP.NET Core 1 & 2
(v15.3+)

.csproj/MSBuild

<http://bit.ly/2dSGoN5>

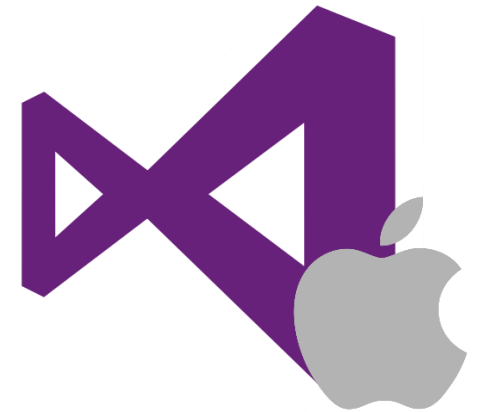


Visual Studio Code

ASP.NET Core 1 & 2

.csproj/MSBuild

<http://bit.ly/1J6QrU6>



Visual Studio for Mac

ASP.NET Core 1 & 2

.csproj/MSBuild

<http://bit.ly/2fXmQpH>



Choosing an IDE



Licences



Functionality &
maturity of the tooling

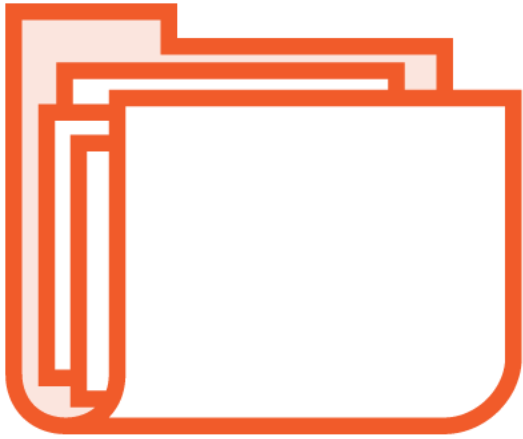


Personal preference

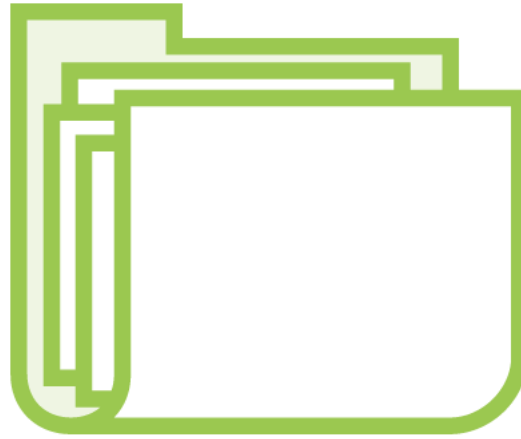
As far as the demo
code is concerned
there's a 95% overlap
between ASP.NET
Core 1.x and 2.0



Exercise Files



Based on
xproject/project.json,
for use in VS2015
(ASP.NET Core 1.0)



Based on
csproject/ms-build, for
use in VS2017
(ASP.NET Core 1.0)



Based on
csproject/ms-build, for
use in VS2017
(ASP.NET Core 2.0)



Choosing Between (ASP).NET Core 1 & 2



Larger API surface might require you to use a newer version

Different support lifecycles might require you to use an older version



Choosing Between (ASP).NET Core 1 & 2

LTS (long term support) releases

.NET Core 1.0 & 1.1

Supported for 3 years after the general availability date of a LTS release

Or 1 year after the general availability of a subsequent LTS release

Current releases

.NET Core 2.0

Supported within the same 3-year window as the parent LTS release

Supported for 3 months after the general availability of a subsequent Current release

And 1 year after the general availability of a subsequent LTS release



Tooling



Postman

<https://www.getpostman.com/>



A browser of choice

REST is...



Representational State Transfer is intended to evoke an image of how a well-designed web application behaves:

a network of web pages (a virtual state-machine)...

... where the user progresses through an application by selecting links (state transitions)...

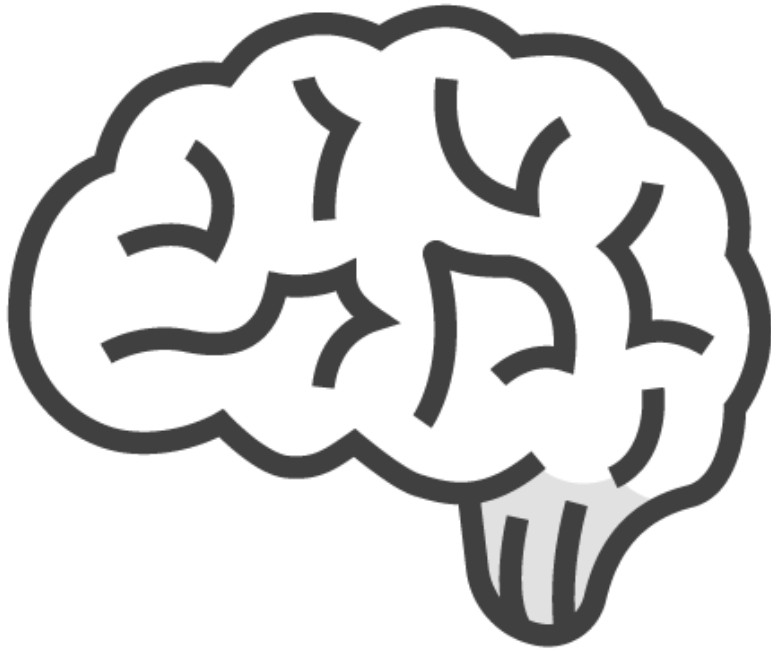
... resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use

Roy Fielding

<http://bit.ly/1rbtZik>



Introducing REST

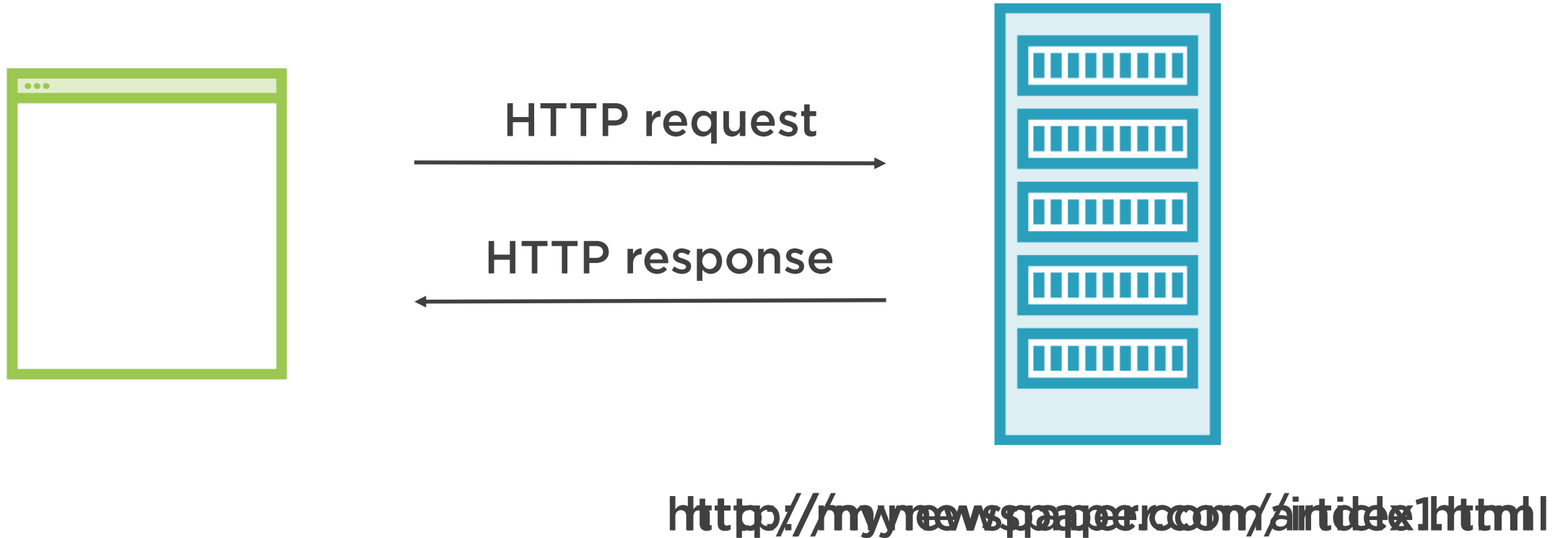


REST is an architectural style, not a standard - we use standards to implement it

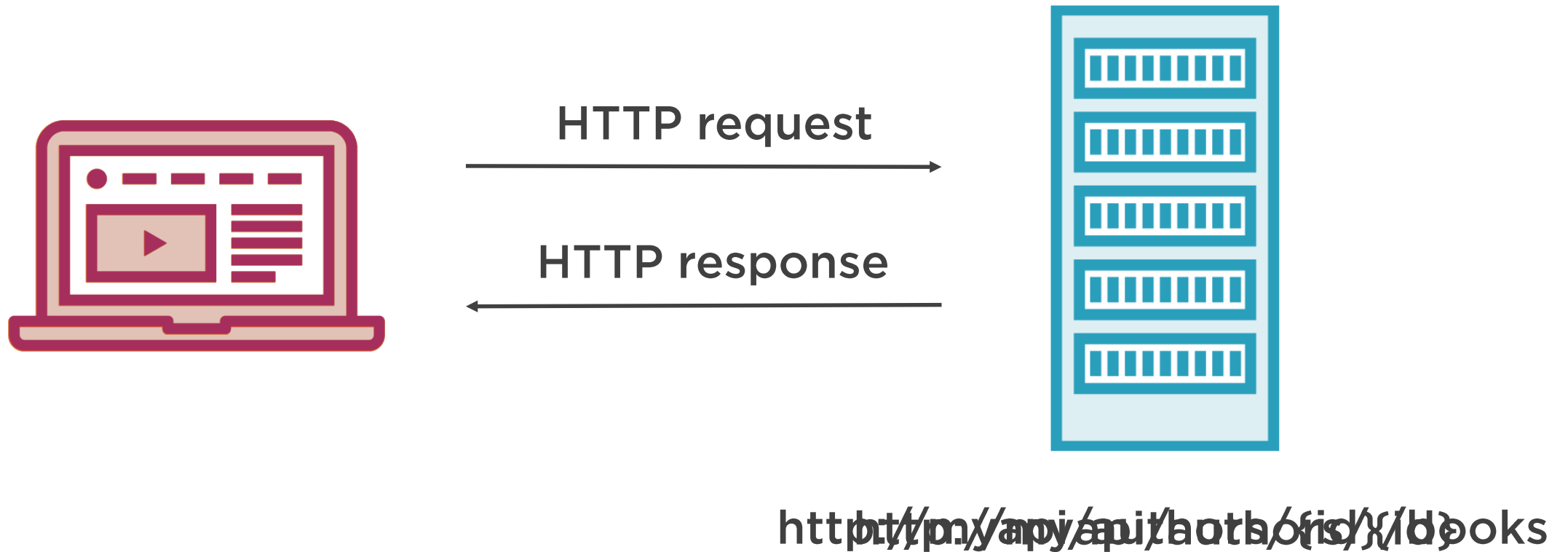
REST is protocol agnostic



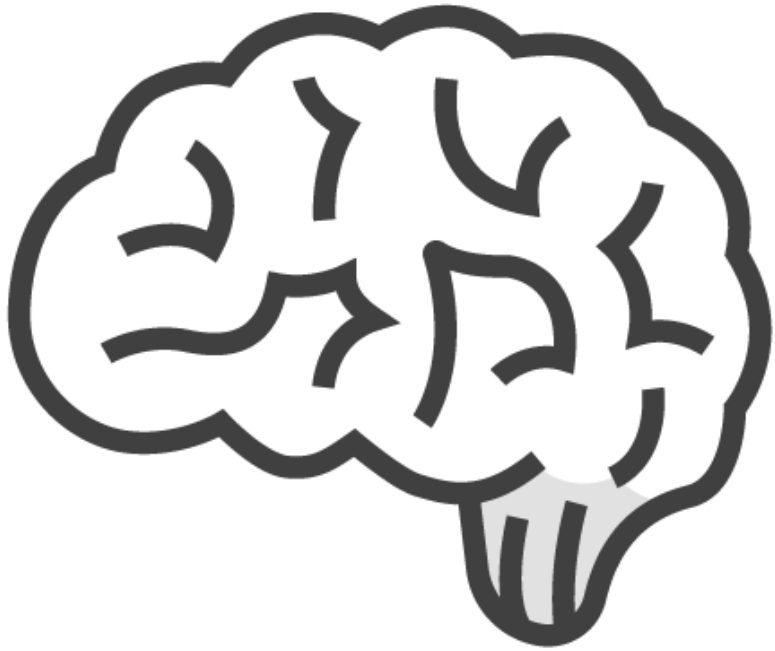
Representational State Transfer



Representational State Transfer



Learning what the REST Constraints are About



REST is defined by 6 constraints (one optional)

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

Learning what the REST Constraints are About

Client-Server

client and server
are separated

(client and server
can evolve
separately)

Statelessness

state is contained
within the request

Cacheable

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



Learning what the REST Constraints are About

Layered System

client cannot tell
what layer it's
connected to

Code on Demand (optional)

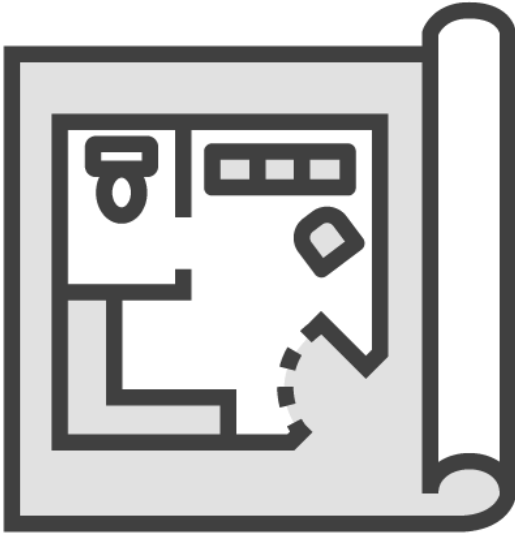
server can extend
client functionality

Uniform Interface

API and consumers
share one single,
technical interface:
URI, Method, Media
Type



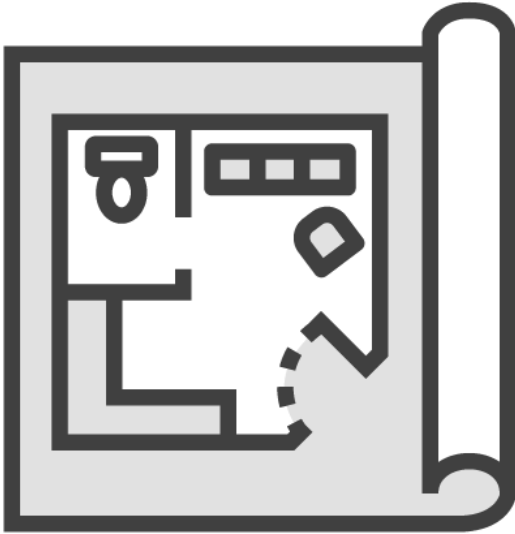
The Uniform Interface Subconstraints



Identification of resources

- A resource is conceptually separate from its representation
- Representation media types: application/json, application/xml, custom, ...

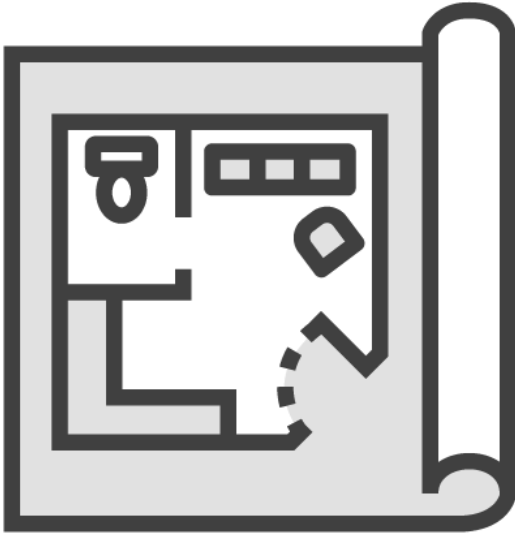
The Uniform Interface Subconstraints



Manipulation of resources through representations

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

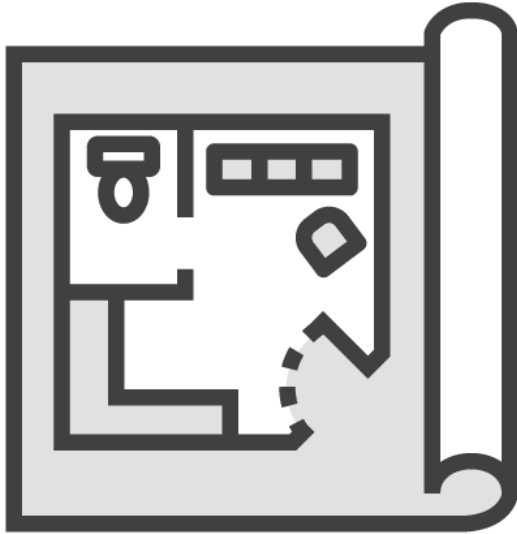
The Uniform Interface Subconstraints



Self-descriptive message

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

The Uniform Interface Subconstraints



Hypermedia as the Engine of Application State (HATEOAS)

- Hypermedia is a generalization of Hypertext (links)
- Drives how to consume and use the API
- Allows for a self-documenting API

A system is only considered RESTful when it adheres to all the required constraints

Most “RESTful” APIs aren’t really RESTful...

... but that doesn’t make them bad APIs, as long as you understand the potential trade-offs



The Richardson Maturity Model

Level 0 (The Swamp of POX)

HTTP protocol is used for remote interaction

... the rest of the protocol isn't used as it should be

RPC-style implementations (SOAP, often seen when using WCF)

POST (info on data)
<http://host/myapi>

POST (author to create)
<http://host/myapi>



The Richardson Maturity Model

Level 1 (Resources)

Each resource is mapped to a URI

HTTP methods aren't used as they
should be

Results in reduced complexity

POST

<http://host/api/authors>

POST

<http://host/api/authors/{id}>



The Richardson Maturity Model

Level 2 (Verbs)

Correct HTTP verbs are used

Correct status codes are used

Removes unnecessary variation

```
GET  
http://host/api/authors  
200 0k (authors)
```

```
POST (author representation)  
http://host/api/authors  
201 Created (author)
```



The Richardson Maturity Model

Level 3 (Hypermedia)

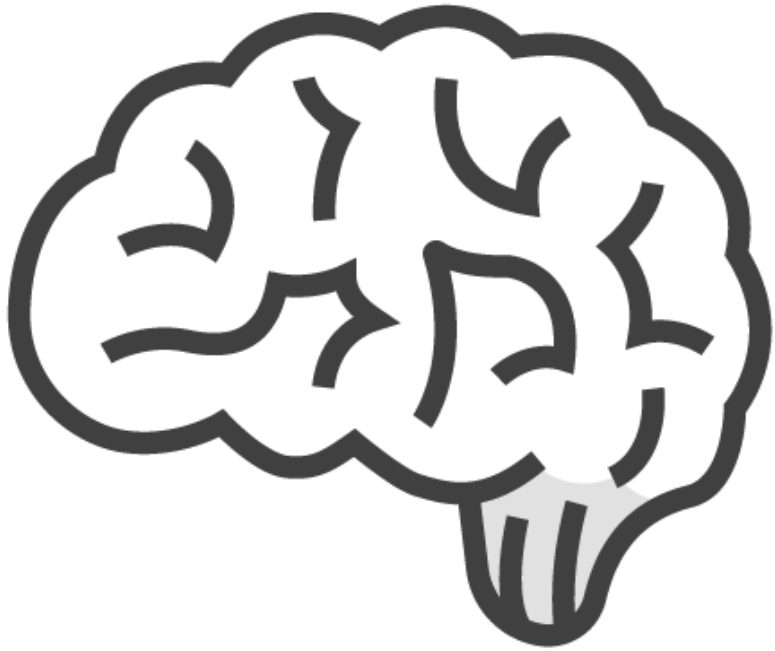
The API supports Hypermedia as the Engine of Application State (HATEOAS)

Introduces discoverability

```
GET
http://host/api/authors
200 0k (authors + links that
drive application state)
```



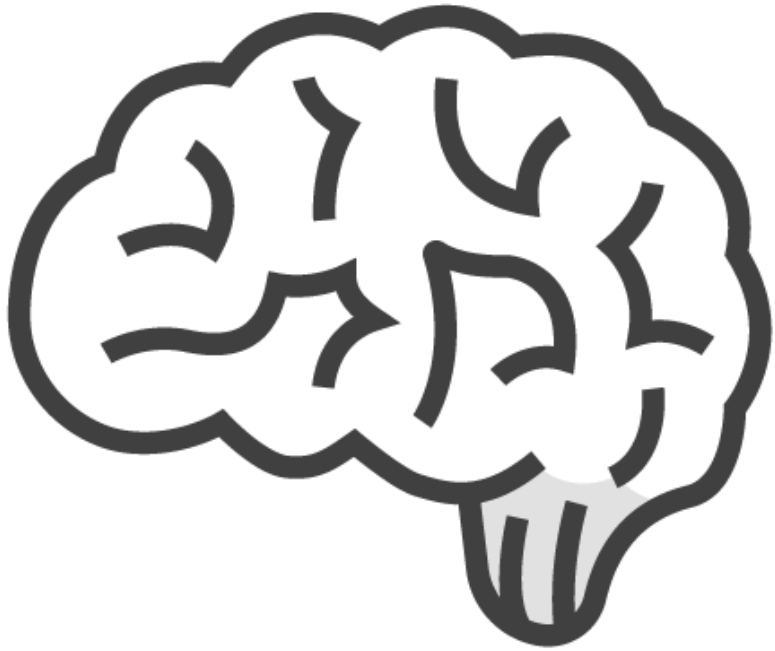
Positioning ASP.NET Core for Building RESTful APIs



Open-source, cross-platform framework for building modern internet connected applications



Positioning ASP.NET Core for Building RESTful APIs

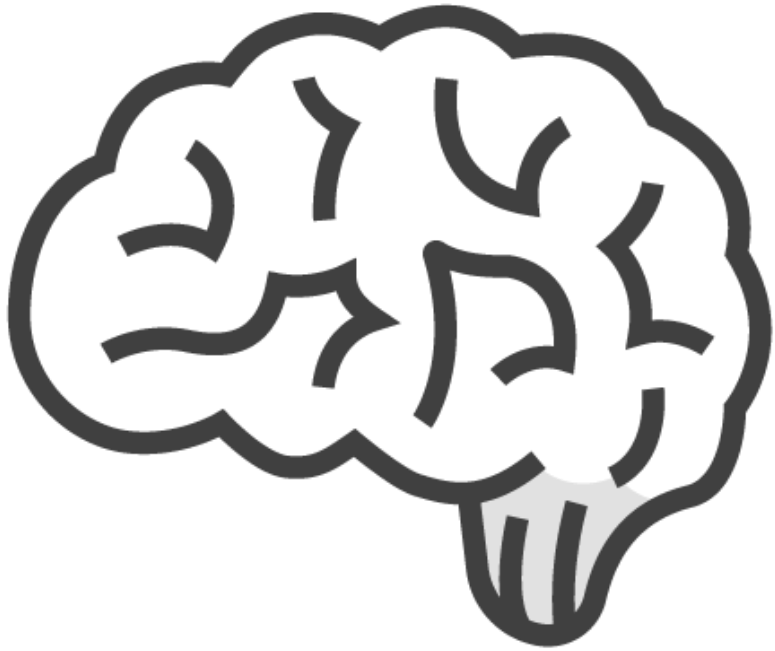


ASP.NET Core MVC middleware provides a framework for building APIs and web applications using the Model-View-Controller pattern

...but we don't get a RESTful API out of the box!



The Model-View-Controller Pattern

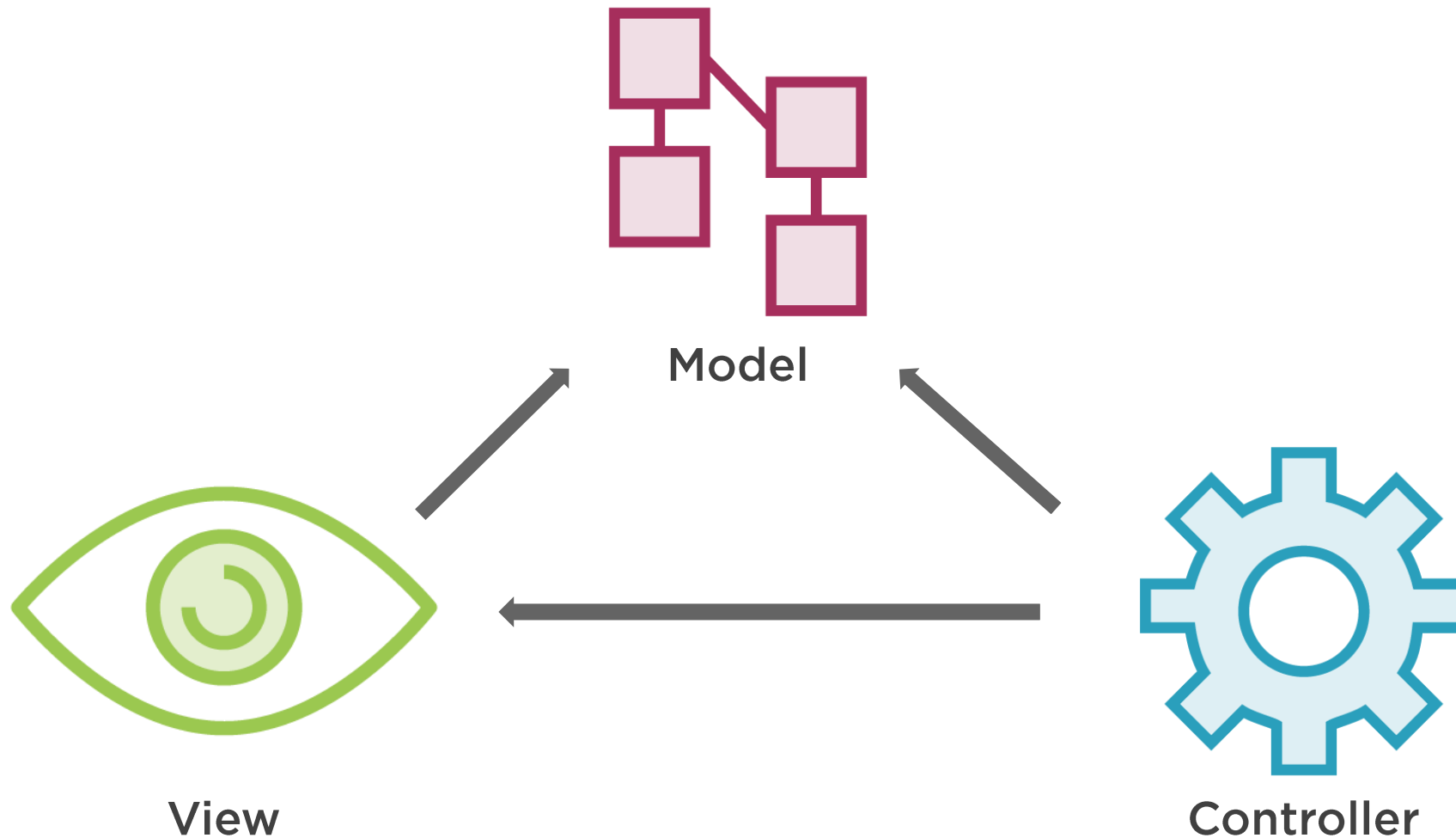


Architectural pattern for implementing user interfaces

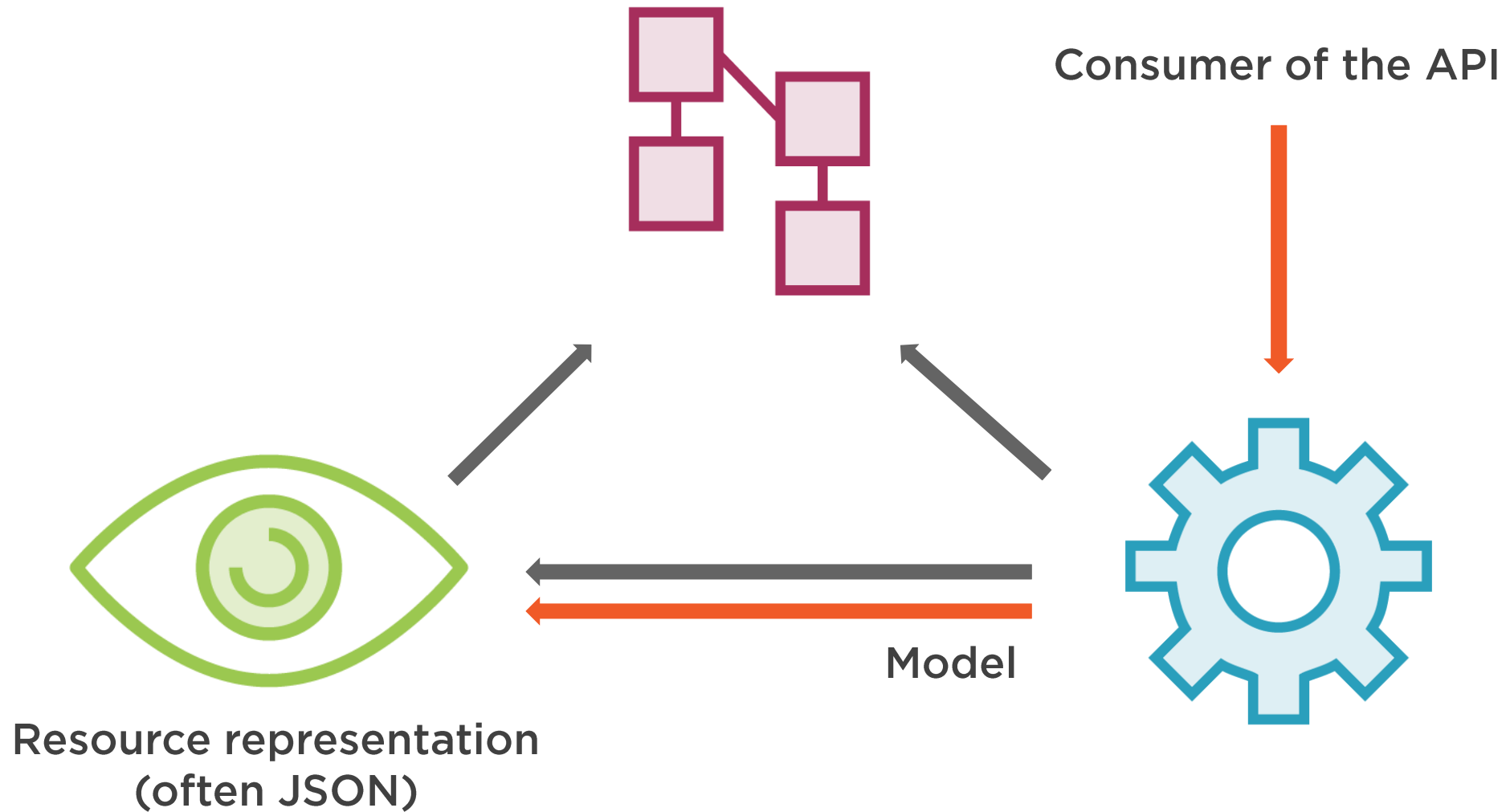
Encourages loose coupling and separation of concerns

Not the full application architecture!

The Model-View-Controller Pattern (API)



The Model-View-Controller Pattern (API)



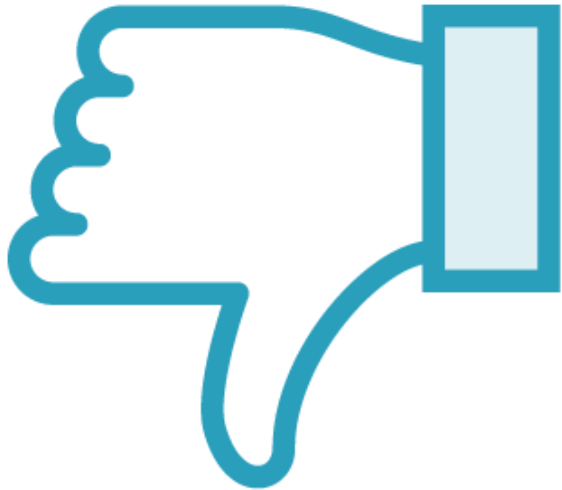
Demo



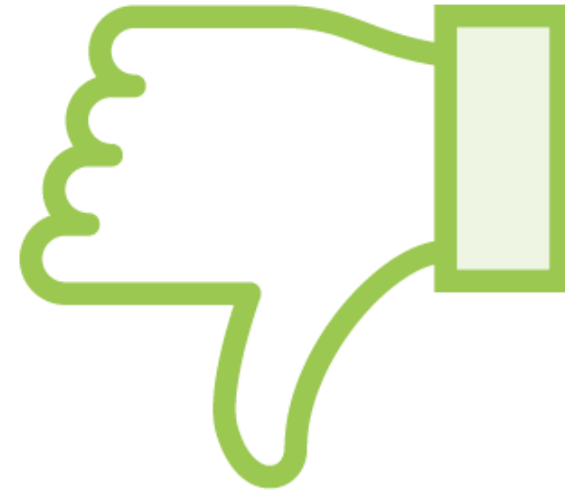
Inspecting the Starter Solution



ASP.NET Core 2 Metapackage



It can be hard to find the
functionality you need



Keeping track of version numbers
can be cumbersome (which might
lead to instabilities)

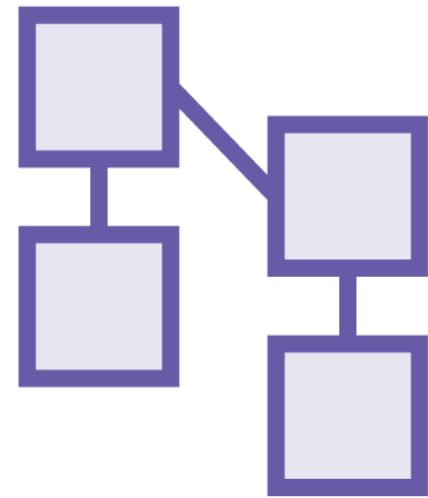
ASP.NET Core 2 Metapackage



All supported ASP.NET
Core packages



All supported Entity
Framework Core
packages



Internal and 3rd-party
dependencies used by
ASP.NET Core and
Entity Framework
Core

Summary



REST is an architectural style, evoking an image of how a well-designed web application should behave

Six constraints

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



Summary



The Richardson Maturity Model grades APIs by their RESTful maturity

ASP.NET Core is an open-source, cross-platform framework for building modern internet connected applications

The ASP.NET Core MVC middleware provides a framework for building APIs and web applications using the Model-View-Controller pattern

