

ASP.NET Core Web API

Deep Dive

Getting Started with REST



Kevin Dockx

Architect

@Kevindockx | www.kevindockx.com



Version Check



This version was created by using:

- ASP.NET Core 8.0
- .NET 8.0
- Visual Studio 2022



Version Check



This course is 100% applicable to:

- ASP.NET Core 8.0
- .NET 8.0



Version Check



New course versions are regularly released:

- <https://app.pluralsight.com/profile/author/kevin-dockx>



Coming Up



Course prerequisites, frameworks and tooling

Positioning ASP.NET Core MVC for building RESTful APIs

- Introducing the demo application
- Introducing Postman



Coming Up



Introducing REST

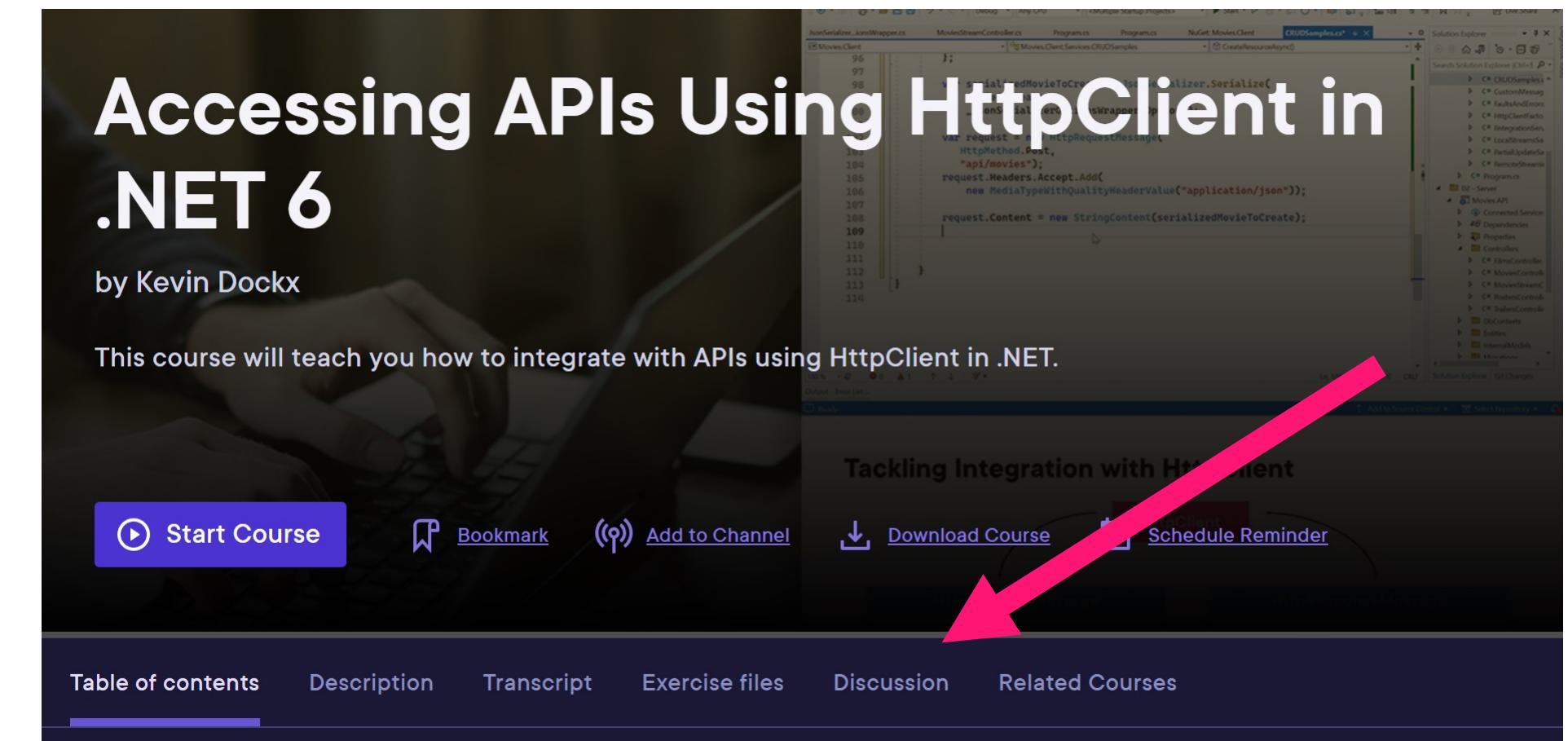
- Learning about the constraints

The Richardson maturity model



Discussion tab on the course page

X: **@KevinDockx**



Accessing APIs Using HttpClient in .NET 6

by Kevin Dockx

This course will teach you how to integrate with APIs using HttpClient in .NET.

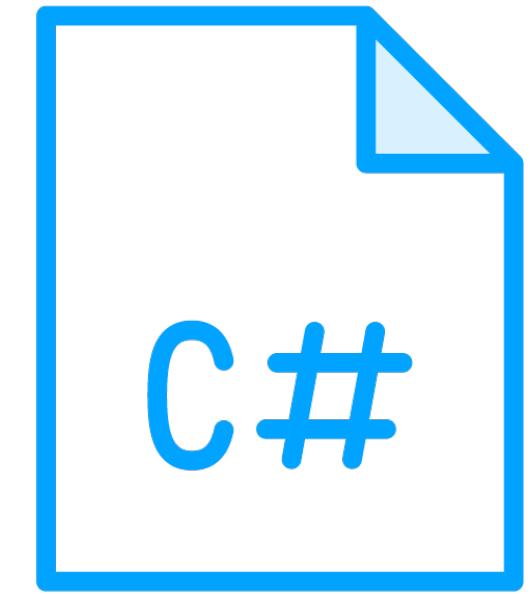
Start Course Bookmark Add to Channel Download Course Schedule Reminder

Table of contents Description Transcript Exercise files **Discussion** Related Courses

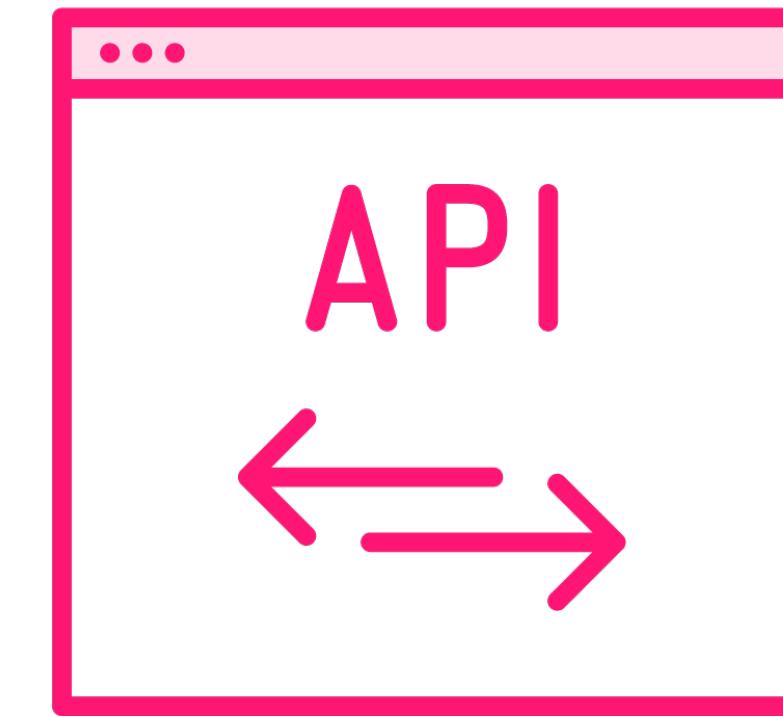
(course shown is one of my other courses, not this one)



Course Prerequisites



Good knowledge of C#



**Some knowledge of building
ASP.NET Core web APIs**



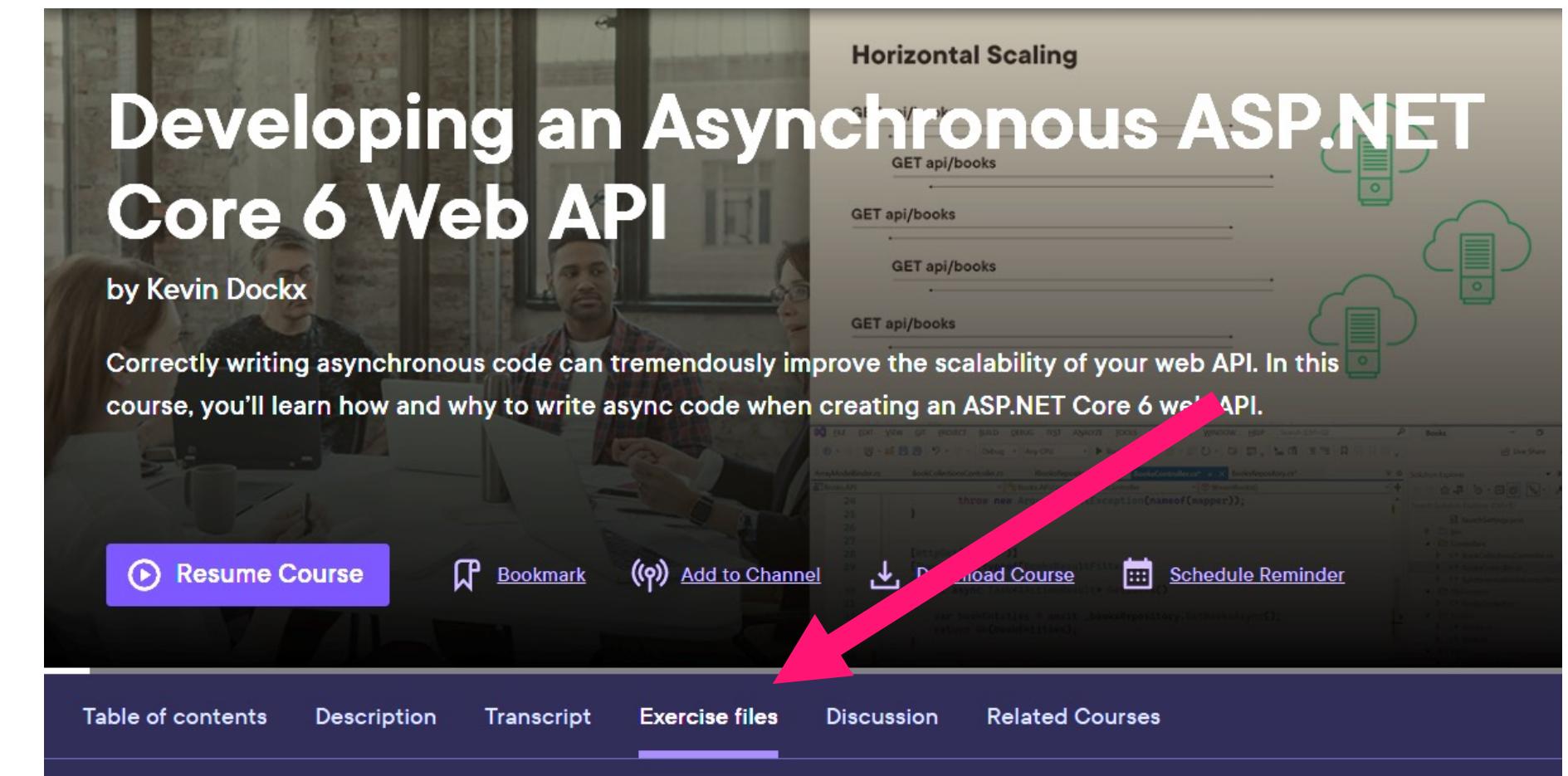
Course Prerequisites

While some important API concepts are repeated, the basics are not covered again in this course

- Have a look at the “Introducing the starter project” clip

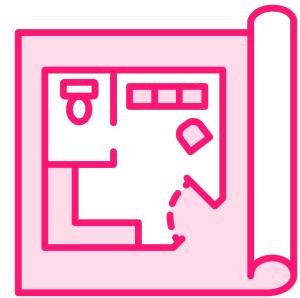


Exercise files tab on the course page

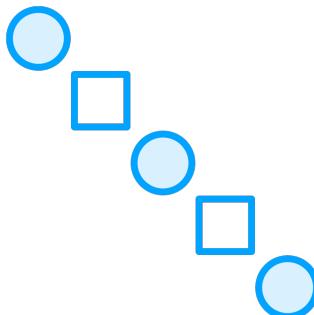


(course shown is one of my other courses, not this one)

Positioning ASP.NET Core MVC for Building RESTful APIs



Model-View-Controller is an architectural pattern for implementing user interfaces



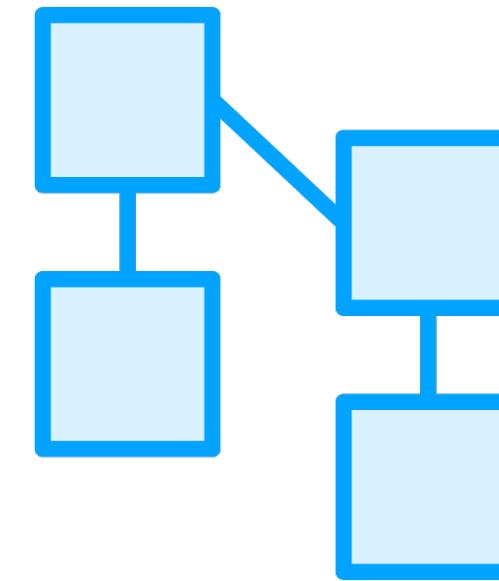
Encourages loose coupling and separation of concerns



... but it's not a full application architecture!



Clarifying the MVC Pattern



model

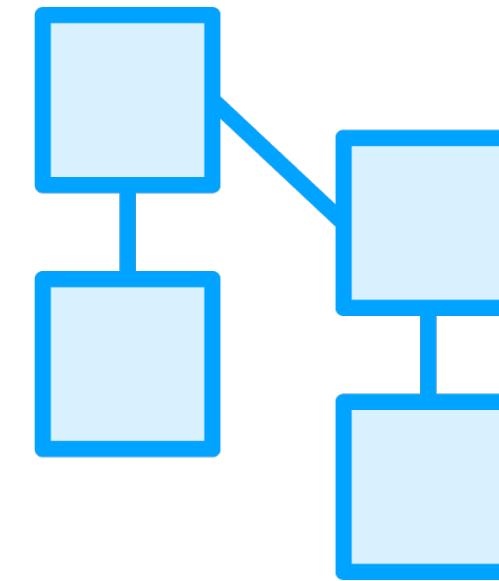


Model

Application data and/or business logic & rules



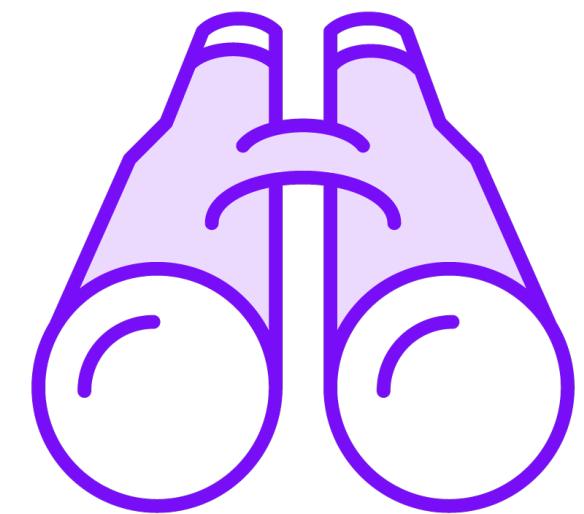
Clarifying the MVC Pattern



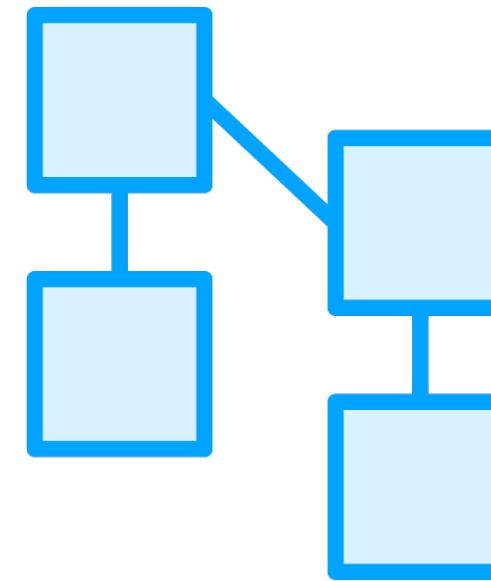
model



Clarifying the MVC Pattern



view



model

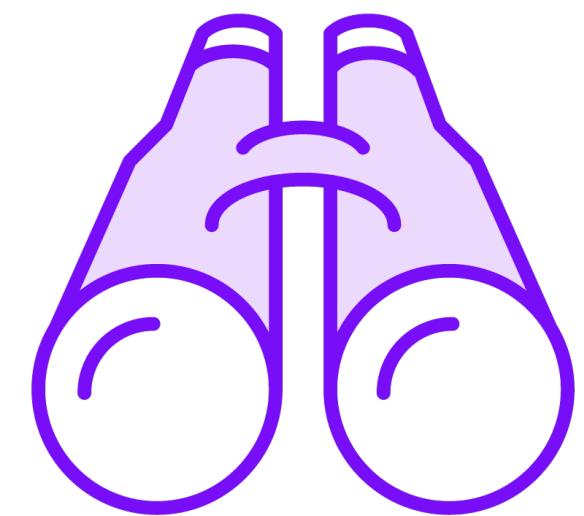


View

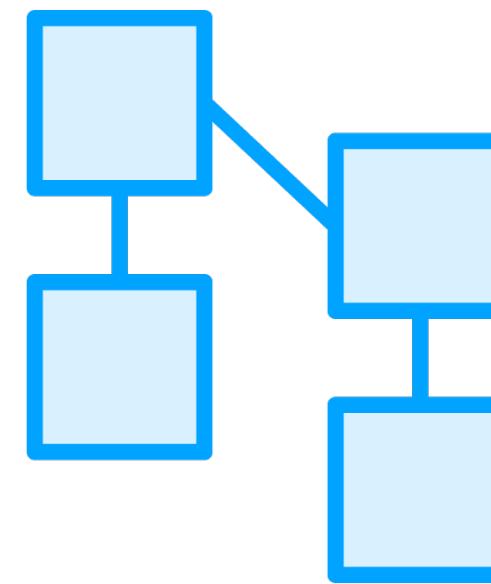
Representation of model data



Clarifying the MVC Pattern



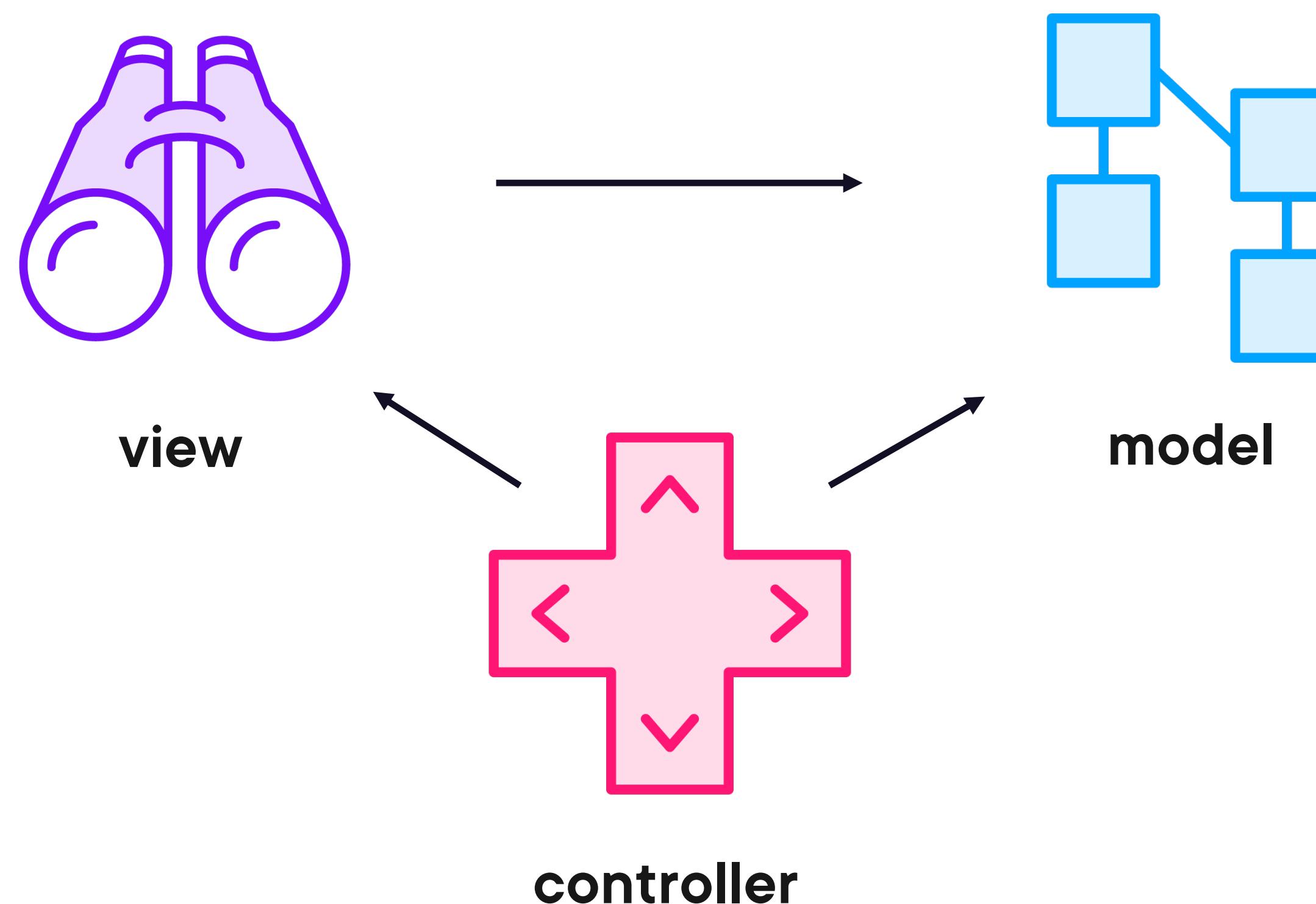
view



model



Clarifying the MVC Pattern

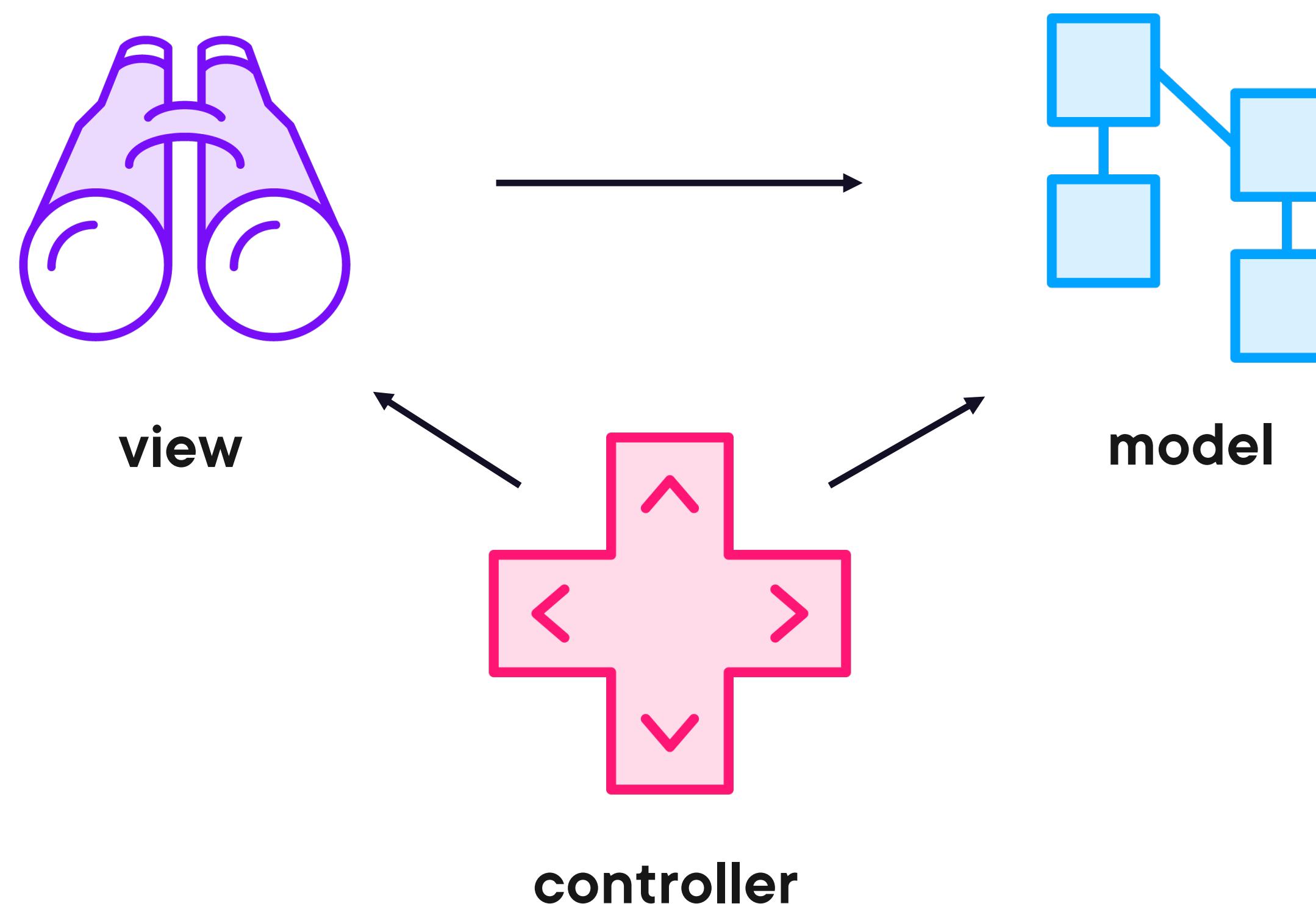


Controller

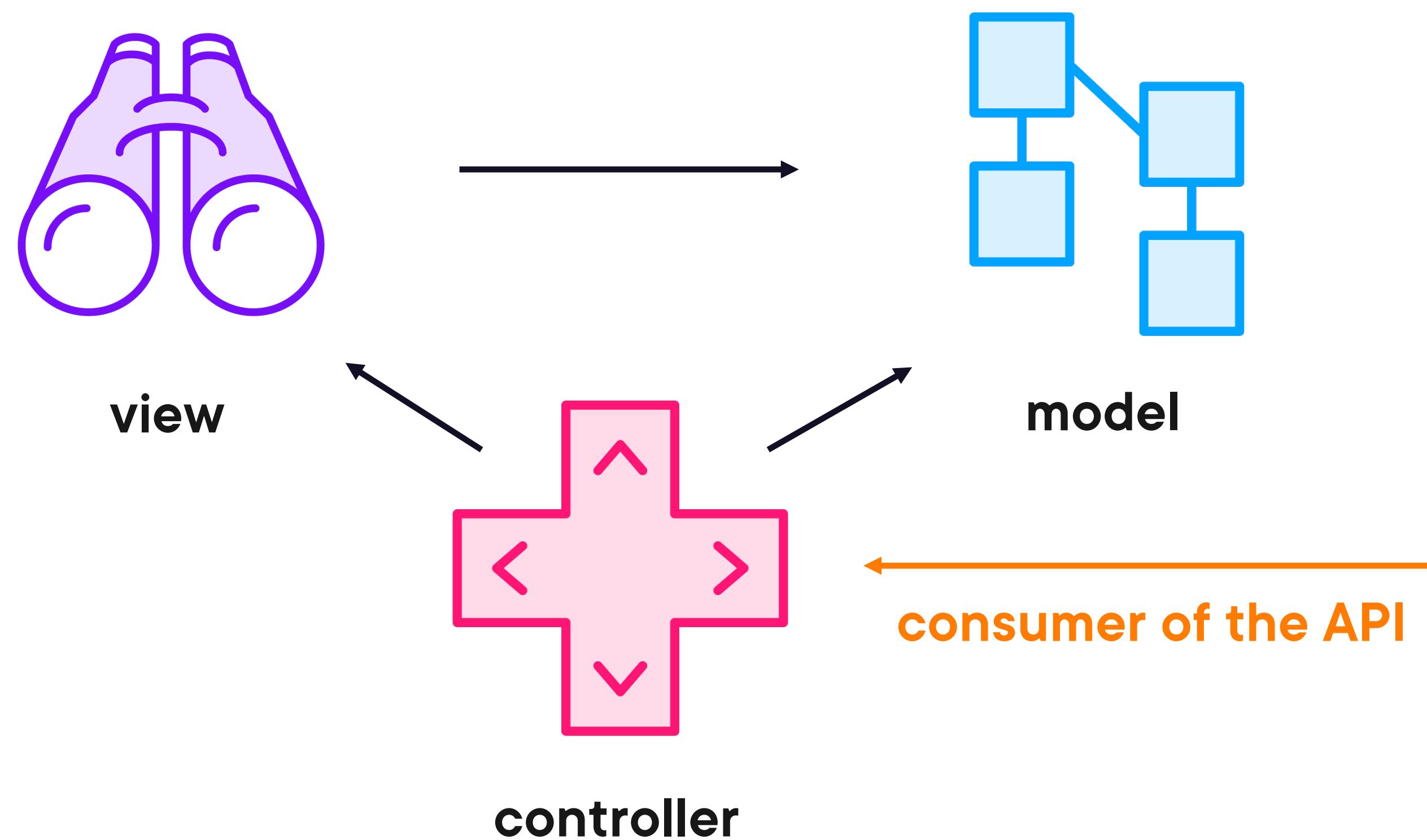
Enables accepting user input and translating that to model data the view can work with



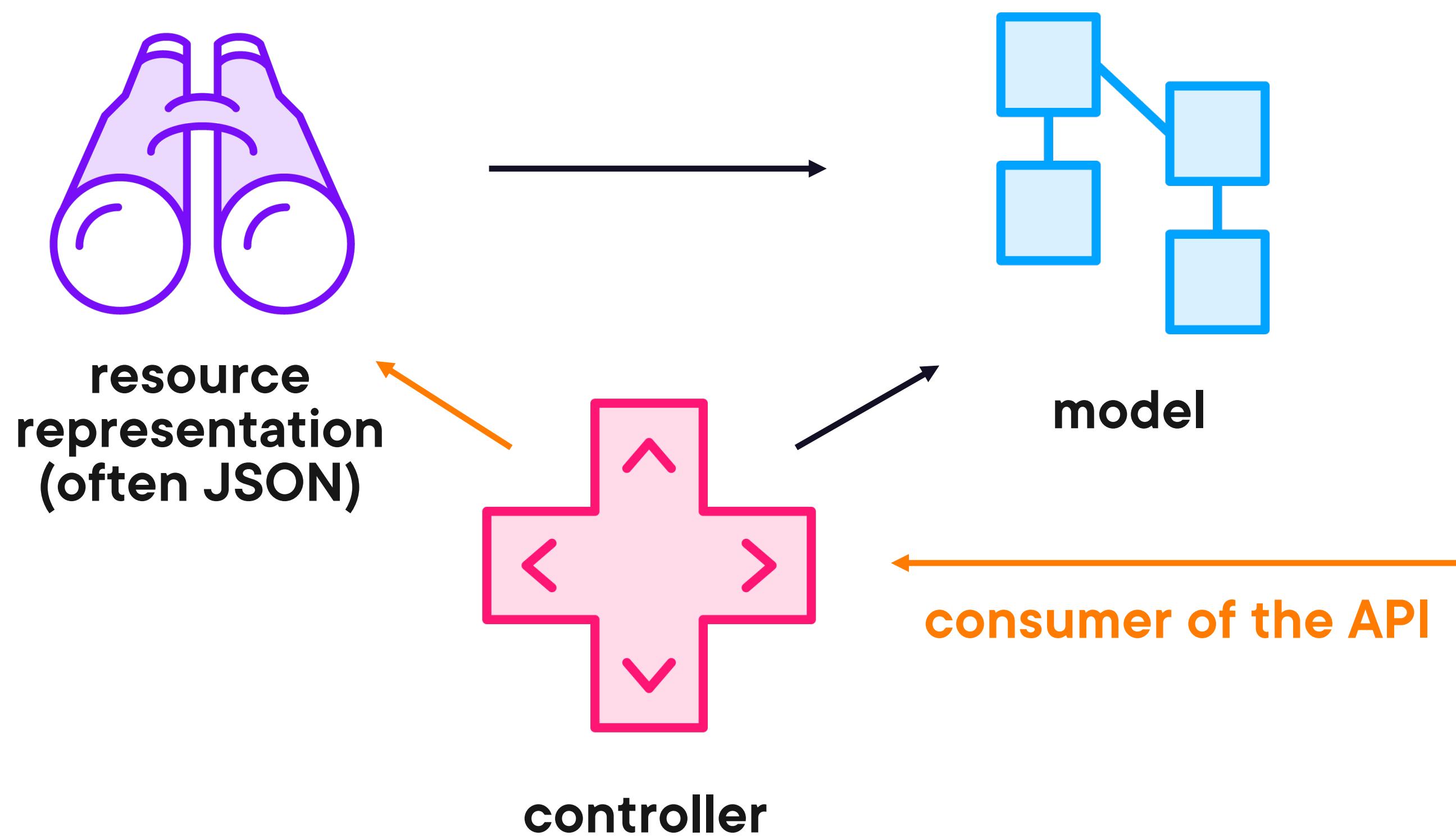
Clarifying the MVC Pattern



Clarifying the MVC Pattern



Clarifying the MVC Pattern



You don't get a RESTful API
out of the box just because
you use the ASP.NET Core
MVC pattern



Demo



Introducing the starter project



Demo



Using Postman and importing the collection with sample requests



Introducing REST

REST is... about HTTP?

REST is... about building APIs?

**REST is... about returning JSON from HTTP
request?**



REST



Representational State Transfer



Representational State Transfer

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



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) ...



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) ...

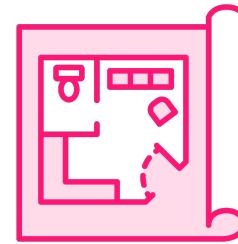


Representational State Transfer

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.



Introducing REST



REST is an architectural style



REST is **not a standard in its own right**



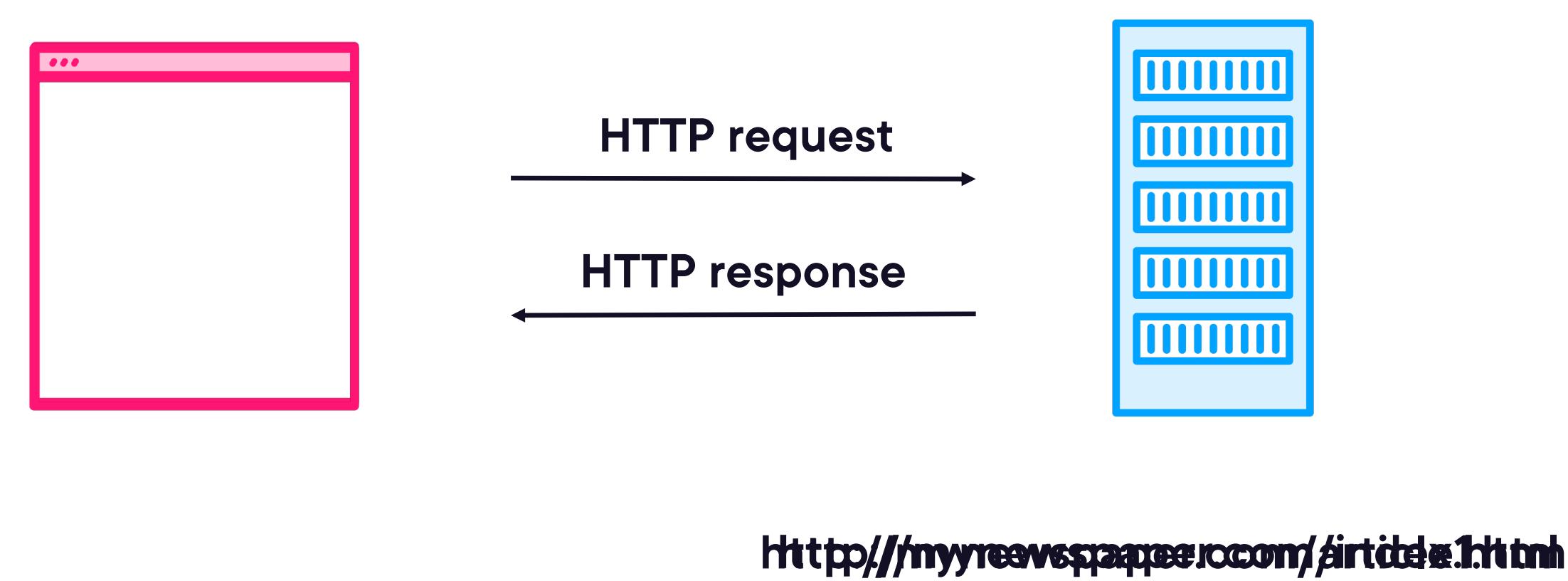
Standards are **used to implement the REST architectural style**



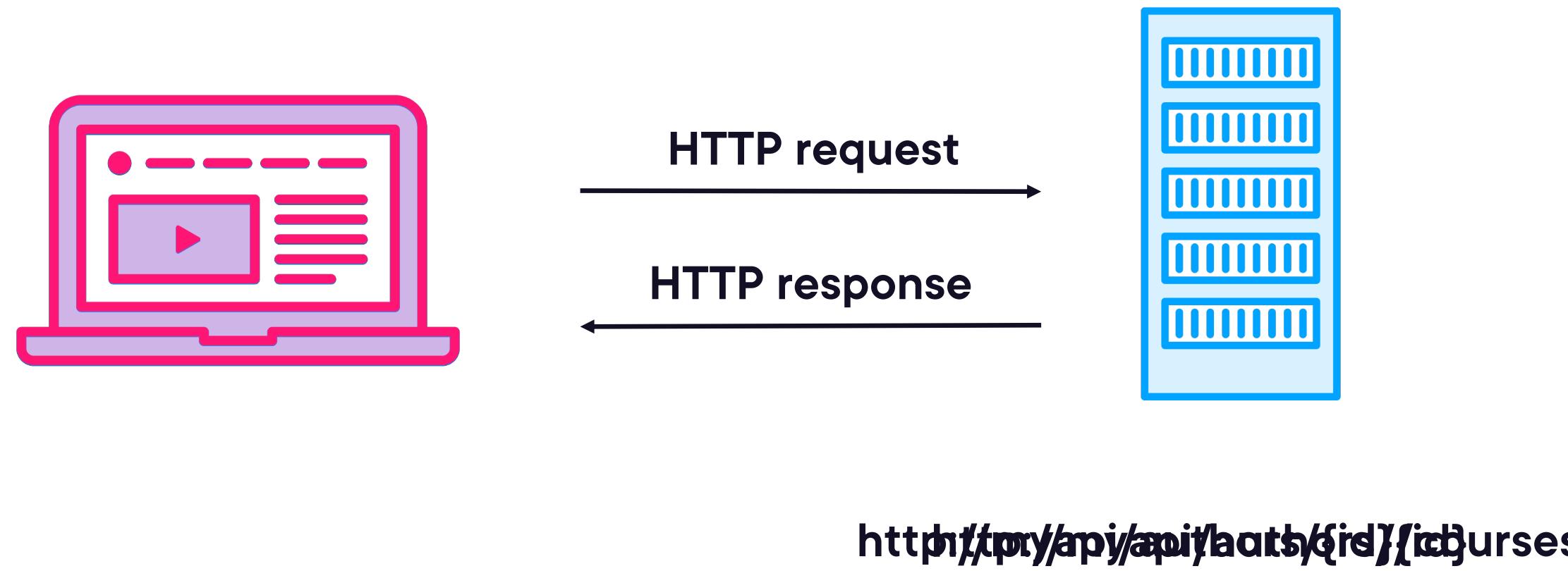
REST is, in principle, protocol agnostic



Introducing REST



Introducing REST



Learning what the REST Constraints Are About

REST is defined by 6 constraints

- 5 obligatory constraints
- 1 optional constraint



Constraint

A design decision that can have positive and negative impacts



Learning what the REST Constraints Are About

Uniform Interface

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



Identification of Resources

A resource is conceptually separate from its representation

Representation media types:

- application/json
- application/xml
- custom, ...



Manipulation of Resources through Representations

**Representation + metadata must be sufficient
to modify or delete the resource**



Self-descriptive Message

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



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



Learning what the REST Constraints Are About

Uniform Interface

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

Client-Server
client and server are separated (client and server can evolve separately)

Statelessness
state is contained within the request



Learning what the REST Constraints Are About

Layered System

client cannot tell what layer it's connected to

Cacheable

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

Code on Demand (optional)

server can extend client functionality



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

This model grades APIs by their
RESTful maturity



POST (info on data)

<http://host/api>

POST (author to create)

<http://host/api>

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/api/authors>

POST (author to create)

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

Level 1 (Resources)

**Each resource is mapped to a URI, HTTP methods aren't used as they should be
Results in reduced complexity**



GET

<http://host/api/authors>

200 Ok (authors)

POST (author representation)

<http://host/api/authors>

201 Created (author)

Level 2 (Verbs)

Correct HTTP verbs are used, correct status codes are used

Removes unnecessary variation



GET

<http://host/api/authors>

200 Ok (authors + links that drive application state)

Level 3 (Hypermedia)

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



The Richardson Maturity Model

Level 3 is a precondition for a RESTful API



Summary



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



Summary



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

Six constraints

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



Summary



The Richardson maturity model grades APIs by their RESTful maturity

- Level 3 is a precondition for RESTful APIs



Up Next:

Designing the Outer Facing Contract

