

Creating the API and Returning Resources



Kevin Dockx

Architect

@Kevindockx | www.kevindockx.com



Coming Up



Clarifying the MVC pattern

Returning resources

Interacting with an API

Content negotiation

Getting a file



Model-View-Controller

An architectural software pattern for implementing user interfaces



Clarifying the MVC Pattern

Very common pattern

- Exists in many languages, supported by many frameworks
- Used to build client-facing ASP.NET Core web applications



Clarifying the MVC Pattern



Improves separation of concerns



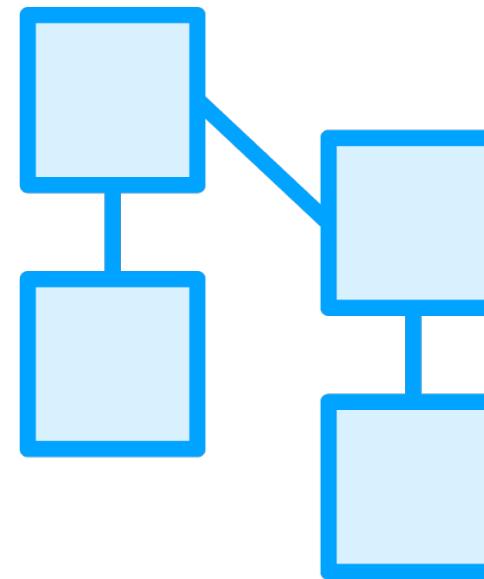
Improves testability



Promotes reuse



Clarifying the MVC Pattern



model

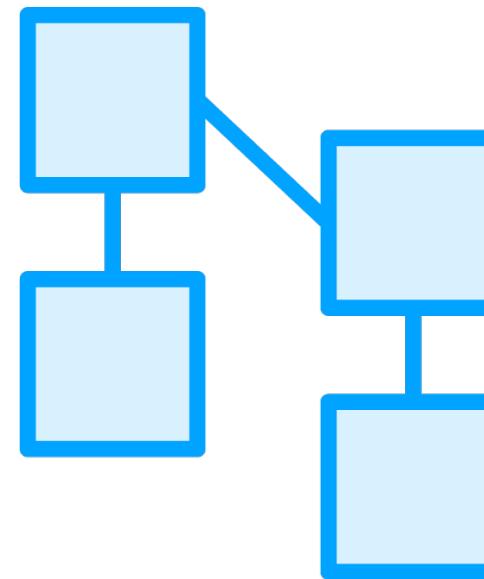


Model

Application data and/or business logic & rules



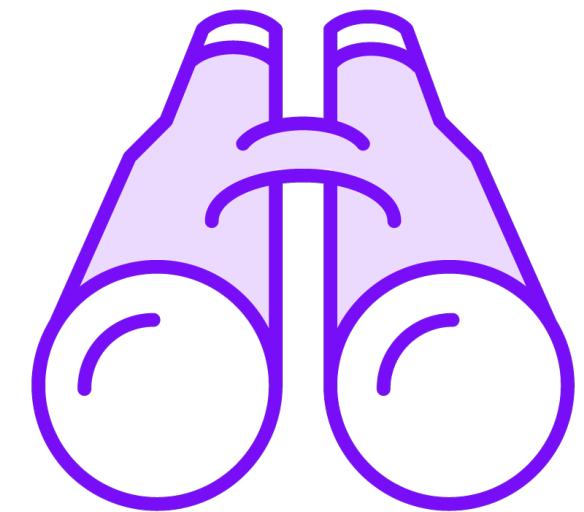
Clarifying the MVC Pattern



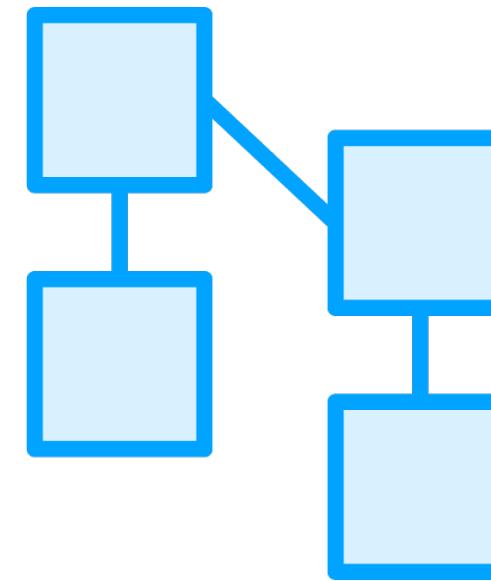
model



Clarifying the MVC Pattern



view



model

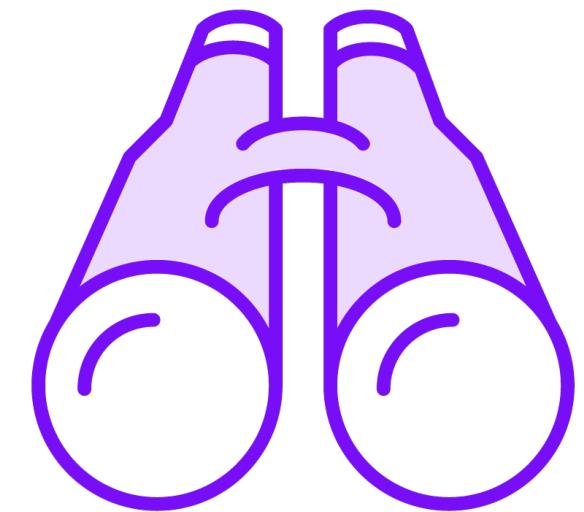


View

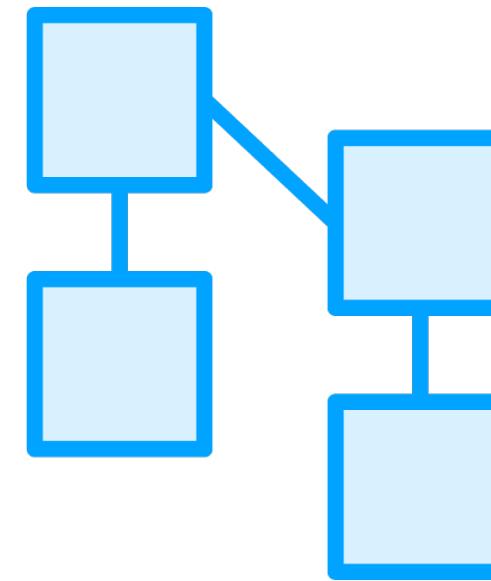
Representation of model data



Clarifying the MVC Pattern



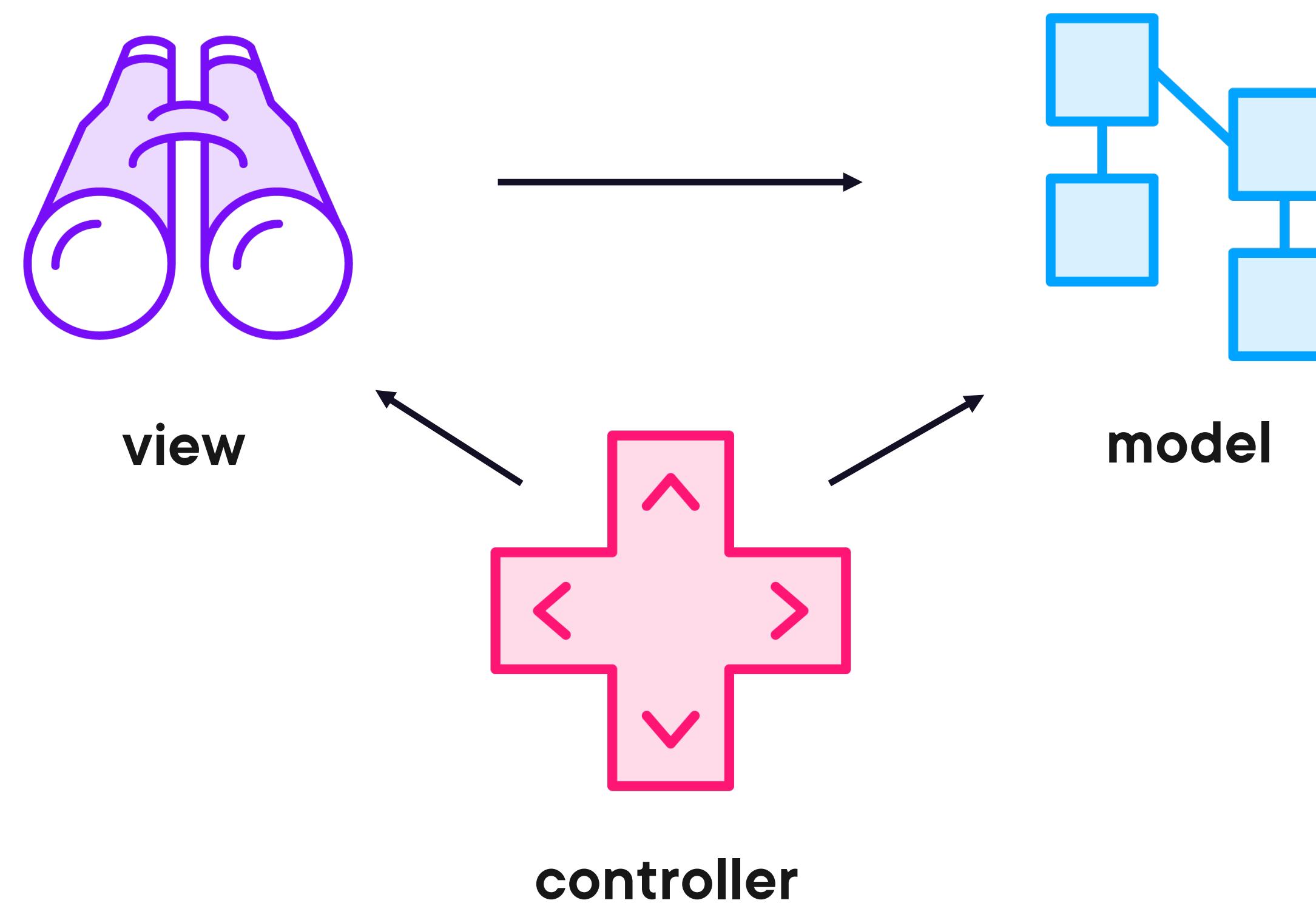
view



model



Clarifying the MVC Pattern

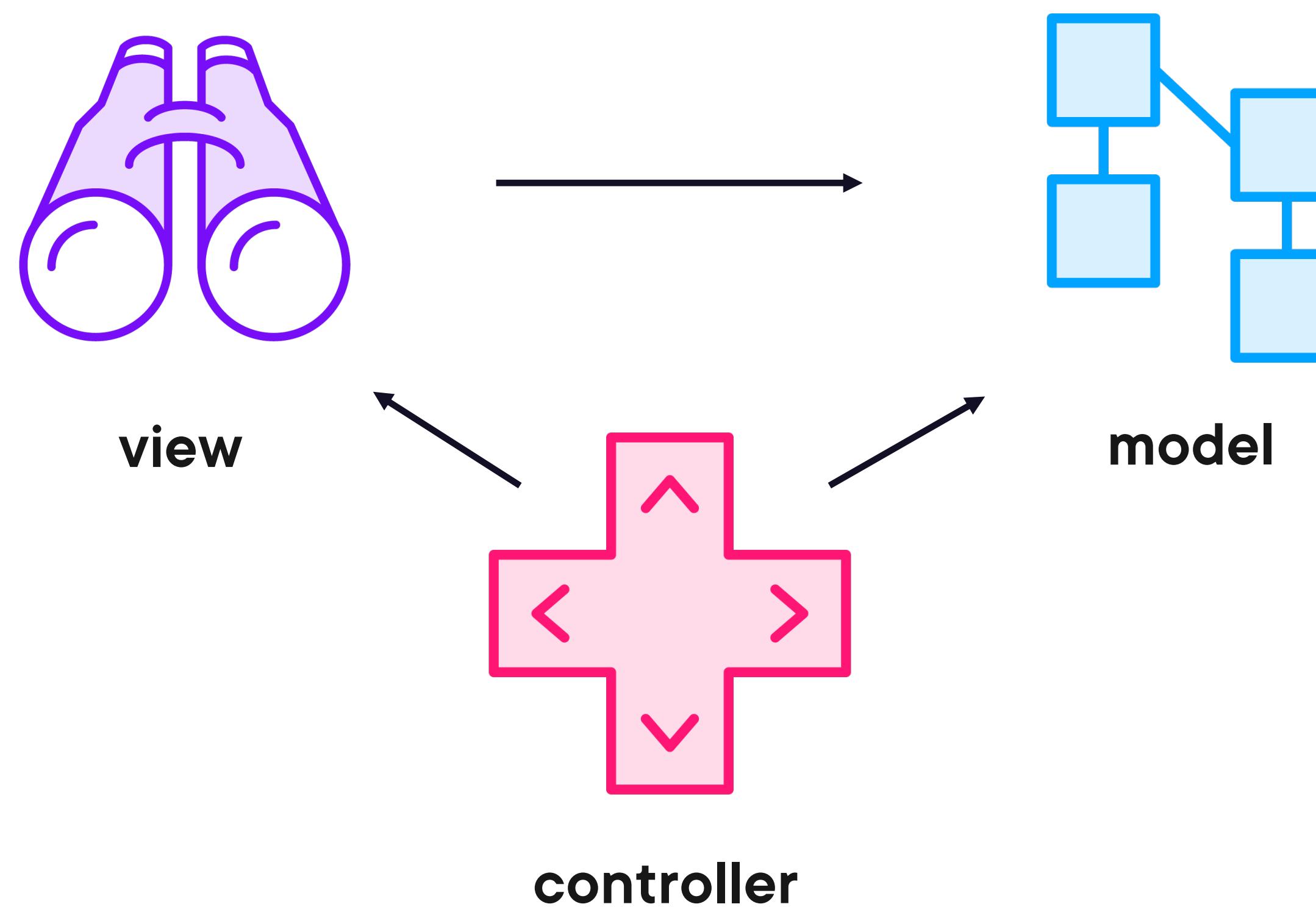


Controller

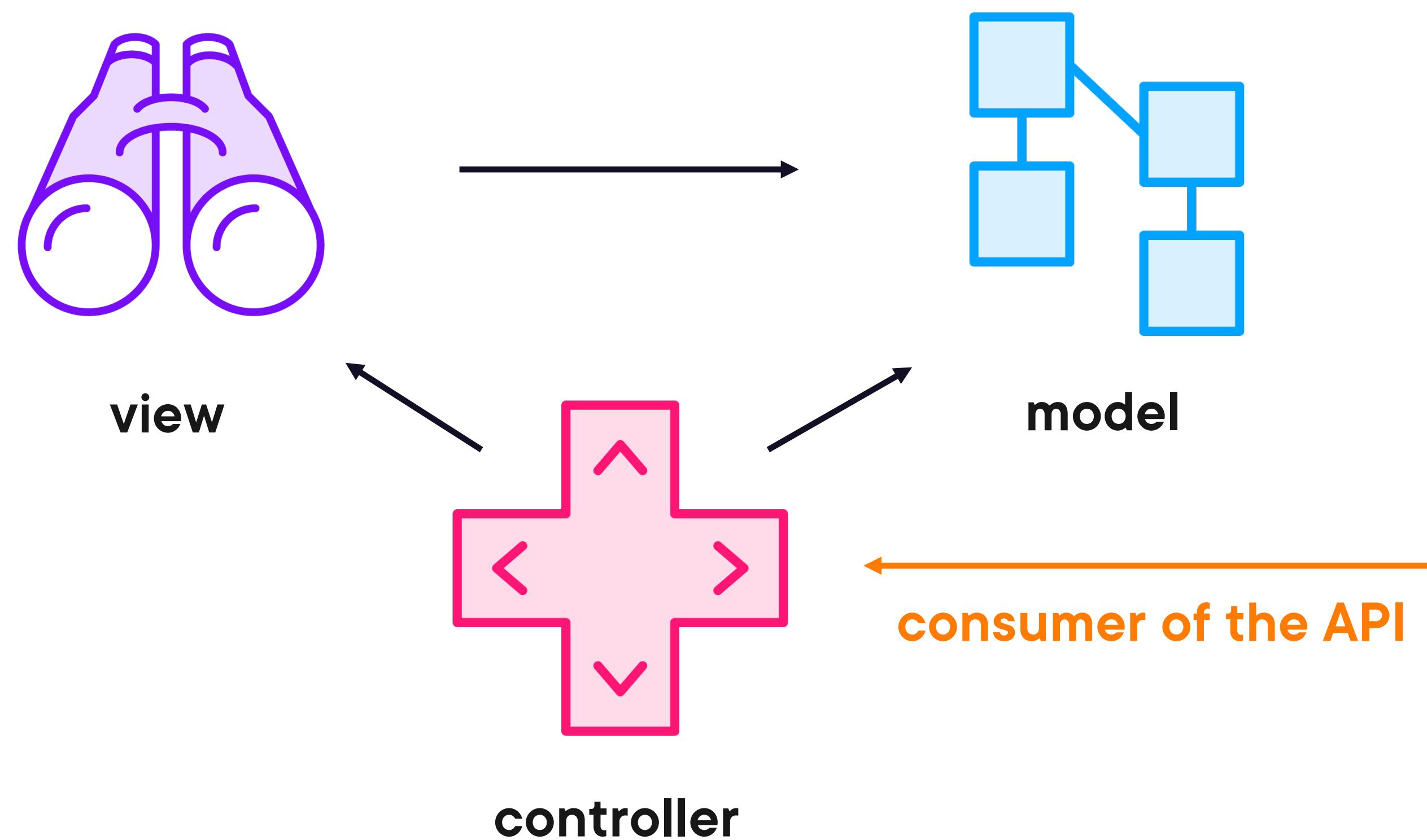
Enables accepting user input and translating that to model data the view can work with



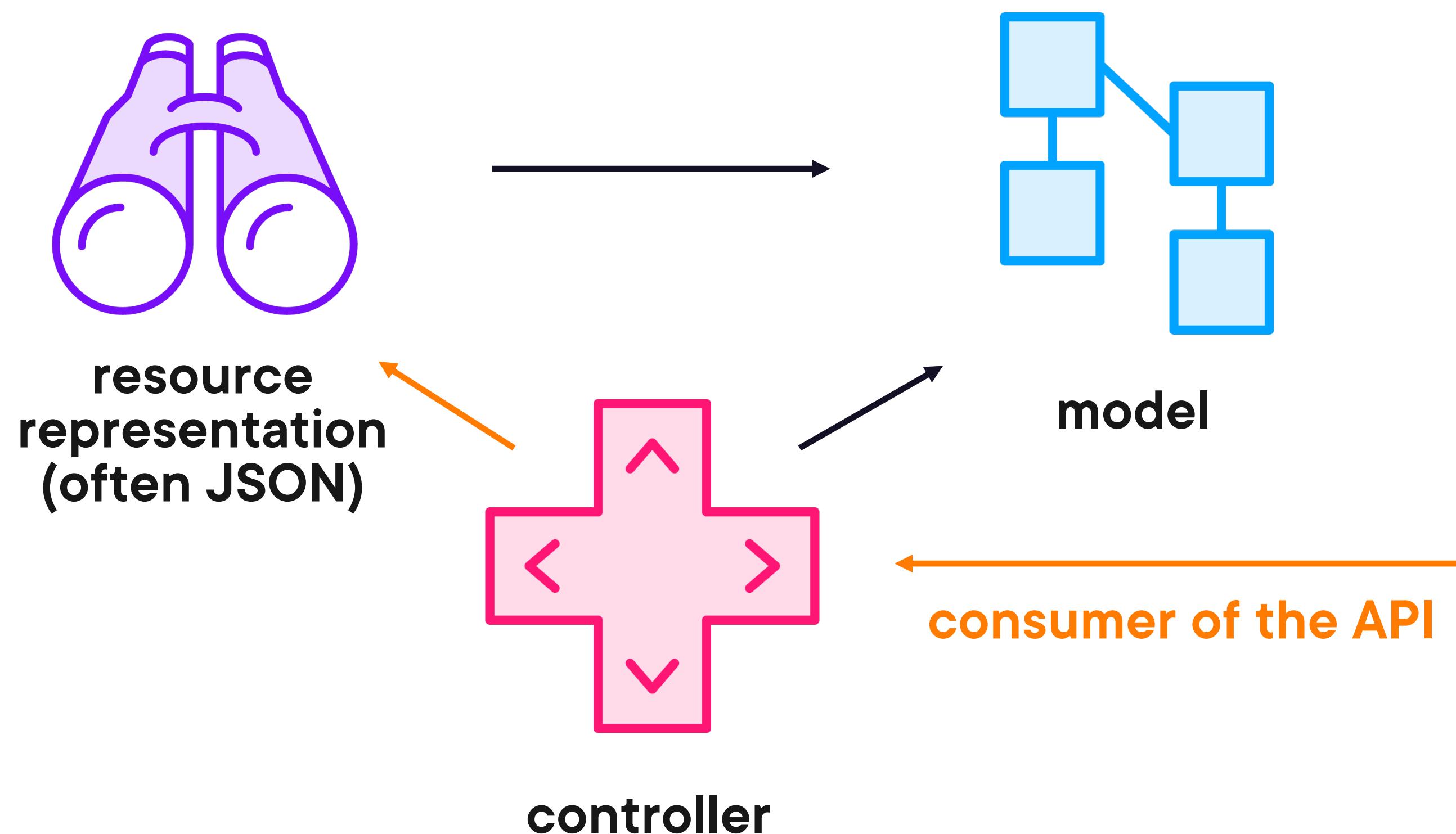
Clarifying the MVC Pattern



Clarifying the MVC Pattern



Clarifying the MVC Pattern



Demo



Registering API services on the container



Demo



Returning resources (part 1)



Routing

Routing matches a request URI to an action on a controller



```
app.UseRouting();  
  
app.UseAuthorization();  
  
app.UseEndpoints(endpoints => {  
    // map endpoints });
```

Learning About Routing

Middleware that runs in between selecting the endpoint and executing the selected endpoint can be injected



```
app.UseRouting();  
  
app.UseAuthorization();  
  
app.UseEndpoints(endpoints => {  
    // map endpoints });
```

Learning About Routing

Middleware that runs in between selecting the endpoint and executing the selected endpoint can be injected



```
app.UseRouting();  
app.UseAuthorization();  
app.UseEndpoints(endpoints => {  
    endpoints.MapControllers();});
```

Attribute-based Routing

**No conventions are applied
This is the preferred approach for APIs**



```
app.UseAuthorization();  
app.MapControllers();
```

Attribute-based Routing

Shortcut: call `MapControllers` on the `WebApplication` object directly



Attribute-based Routing

Use attributes at controller and action level:
`[Route], [HttpGet], ...`

Combined with a URI template, requests are matched to controller actions



Attribute-based Routing

HTTP Method	Attribute	Level	Sample URI
GET	HttpGet	Action	/api/cities /api/cities/1
POST	HttpPost	Action	/api/cities
PUT	HttpPut	Action	/api/cities/1
PATCH	HttpPatch	Action	/api/cities/1
DELETE	HttpDelete	Action	/api/cities/1
---	Route	Controller	---



Attribute-based Routing

For all common HTTP methods, a matching attribute exists

- [HttpGet], [HttpPost], [HttpPatch], ...



Attribute-based Routing

HTTP Method	Attribute	Level	Sample URI
GET	HttpGet	Action	/api/cities /api/cities/1
POST	HttpPost	Action	/api/cities
PUT	HttpPut	Action	/api/cities/1
PATCH	HttpPatch	Action	/api/cities/1
DELETE	HttpDelete	Action	/api/cities/1
---	Route	Controller	---



Attribute-based Routing

- [Route] doesn't map to an HTTP method
- Use it at controller level to provide a template that will prefix all templates defined at action level



Attribute-based Routing

HTTP Method	Attribute	Level	Sample URI
GET	HttpGet	Action	/api/cities /api/cities/1
POST	HttpPost	Action	/api/cities
PUT	HttpPut	Action	/api/cities/1
PATCH	HttpPatch	Action	/api/cities/1
DELETE	HttpDelete	Action	/api/cities/1
---	Route	Controller	---



Demo



Returning resources (part 2)



Demo



Using Postman



Demo



**Improving the architecture with
model classes**



One Application, Different Models

**The outer facing model (DTO) is different from
the entity model (which maps to your
datastore)**

- Will become apparent when we introduce Entity Framework Core



```
public class CityDto
{
    public int NumberOfPointsOfInterest { get; set; }
}
```

```
public class PersonDto
{
    public string FullName { get; set; }
}
```

One Application, Different Models

The outer facing model is different from the entity model
- E.g.: calculated fields on the outer facing model



```
public class CityDto
{
    public int NumberOfPointsOfInterest { get; set; }
}
```

```
public class PersonDto
{
    public string FullName { get; set; }
}
```

One Application, Different Models

The outer facing model is different from the entity model
- E.g.: calculated fields on the outer facing model



```
public class CityDto
{
    public int NumberOfPointsOfInterest { get; set; }
}
```

```
public class PersonDto
{
    public string FullName { get; set; }
}
```

One Application, Different Models

The outer facing model is different from the entity model
- E.g.: calculated fields on the outer facing model



```
// Entity
public class City
{
    public int Id { get; set; }
}
```

```
public class CityForCreationDto
{
    // no identifier
}
```

One Application, Different Models

The outer facing model is different from the entity model
- E.g.: identifiers on the entity model



```
// Entity
public class City
{
    public int Id { get; set; }
}
```

```
public class CityForCreationDto
{
    // no identifier
}
```

One Application, Different Models

The outer facing model is different from the entity model
- E.g.: identifiers on the entity model



The Importance of Status Codes

Status codes tell the consumer of the API

- Whether the request worked out as expected
- What is responsible for a failed request



The Importance of Status Codes

Common mistakes:

- Don't send back a 200 Ok when something's wrong
- Don't send back a 500 Internal Server Error when the client makes a mistake
- ...



The Importance of Status Codes

Level 100
Informational



The Importance of Status Codes

**Level 200
Success**

200 – OK

201 – Created

204 – No Content

**Level 300
Redirection**



The Importance of Status Codes

**Level 200
Success**

200 – OK
201 – Created
204 – No Content

**Level 400
Client mistake**

400 – Bad Request
401 – Unauthorized
403- Forbidden
404 – Not Found
409 - Conflict

**Level 500
Server mistake**

**500 – Internal
Server Error**



Demo



Returning correct status codes



Demo



Returning child resources



```
{  
  "type": "https://tools.ietf.org/html/rfc9110#section-15.5.5",  
  "title": "Not Found",  
  "status": 404,  
  "traceId": "00-79db742d484ac3c899c50e89429c4baf-bf024e4ea0ecafed-00"  
}
```

Describing Error Responses with Problem Details

Problem Details for HTTP APIs RFC



Demo



Manipulating ProblemDetails responses



Content Negotiation

The process of selecting the best representation for a given response when there are multiple representations available



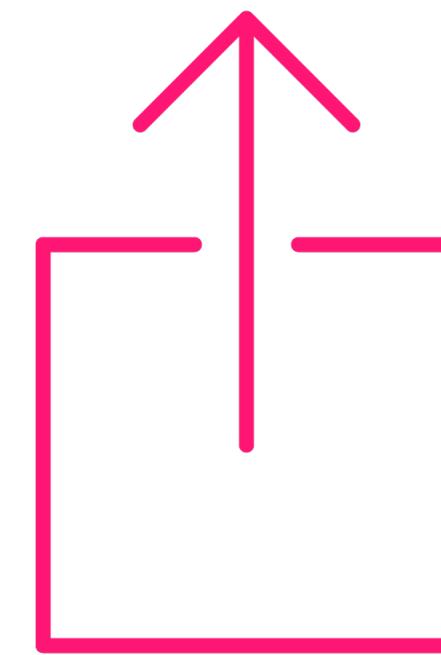
Formatters and Content Negotiation

The media type(s) is/are passed through via the Accept header of the request

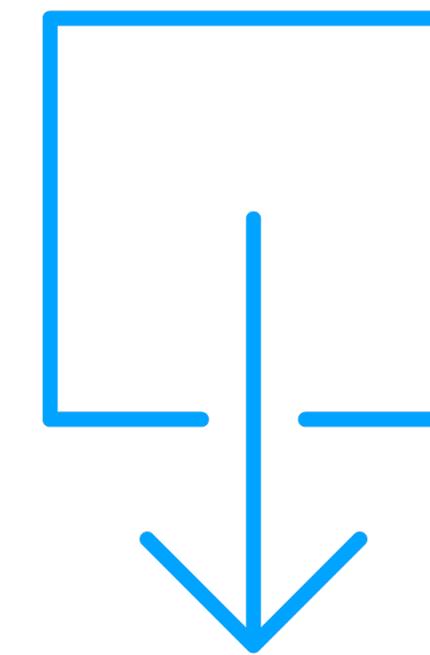
- application/json
- application/xml
- ...



Formatters and Content Negotiation



Output formatter
Deals with output
Media type: [Accept header](#)



Input formatter
Deals with input
Media type: [Content-Type header](#)



Formatters and Content Negotiation

Support is implemented by `ObjectResult`
– Action result methods derive from it



Demo



Formatters and content negotiation



Demo



Getting a file



Summary



Model-View-Controller

- **Model:** application data logic
- **View:** display data
- **Controller:** interaction between View and Model

The pattern improves reuse and testability



Summary



Routing matches a request URI to an action on a controller

- Attribute-based routing is advised for APIs



Summary



Content negotiation is the process of selecting the best representation for a given response when there are multiple representations available



Summary



Use the `File` method on `ControllerBase` to return files

- Think about setting the correct media type



Up Next:

Manipulating Resources and Validating Input

