Creating an API and Returning Resources

Kevin Dockx
Architect

@KevinDockx https://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

Model-View-Controller

An architectural software pattern for implementing user interfaces

Clarifying the MVC Pattern

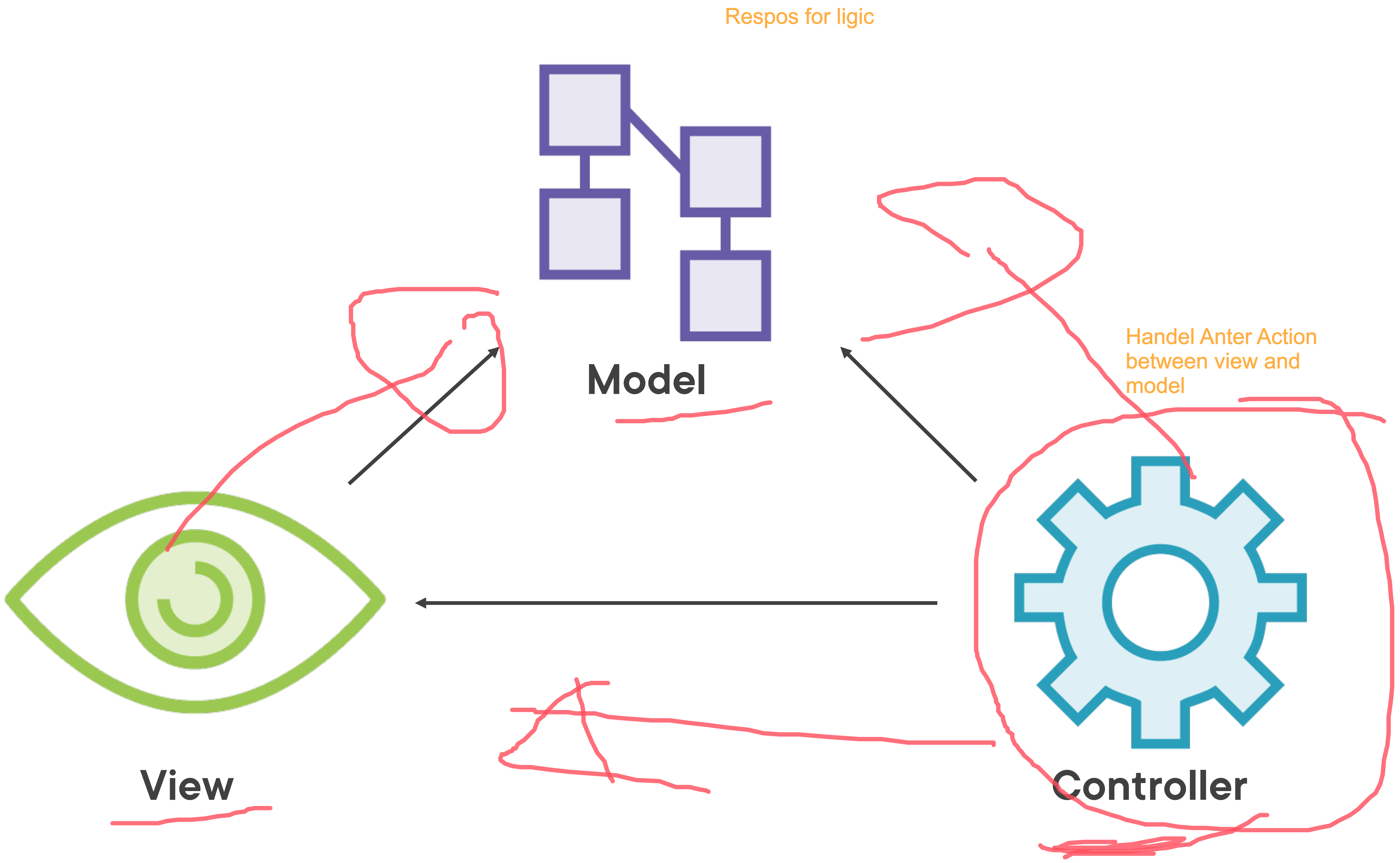
- ✓ **Loose coupling** اقتران فضفاض
- ✓ **Separation of concerns** فصل المخاوف
- ✓ **Testability**
- ✓ **Reusability**

Clarifying the MVC Pattern

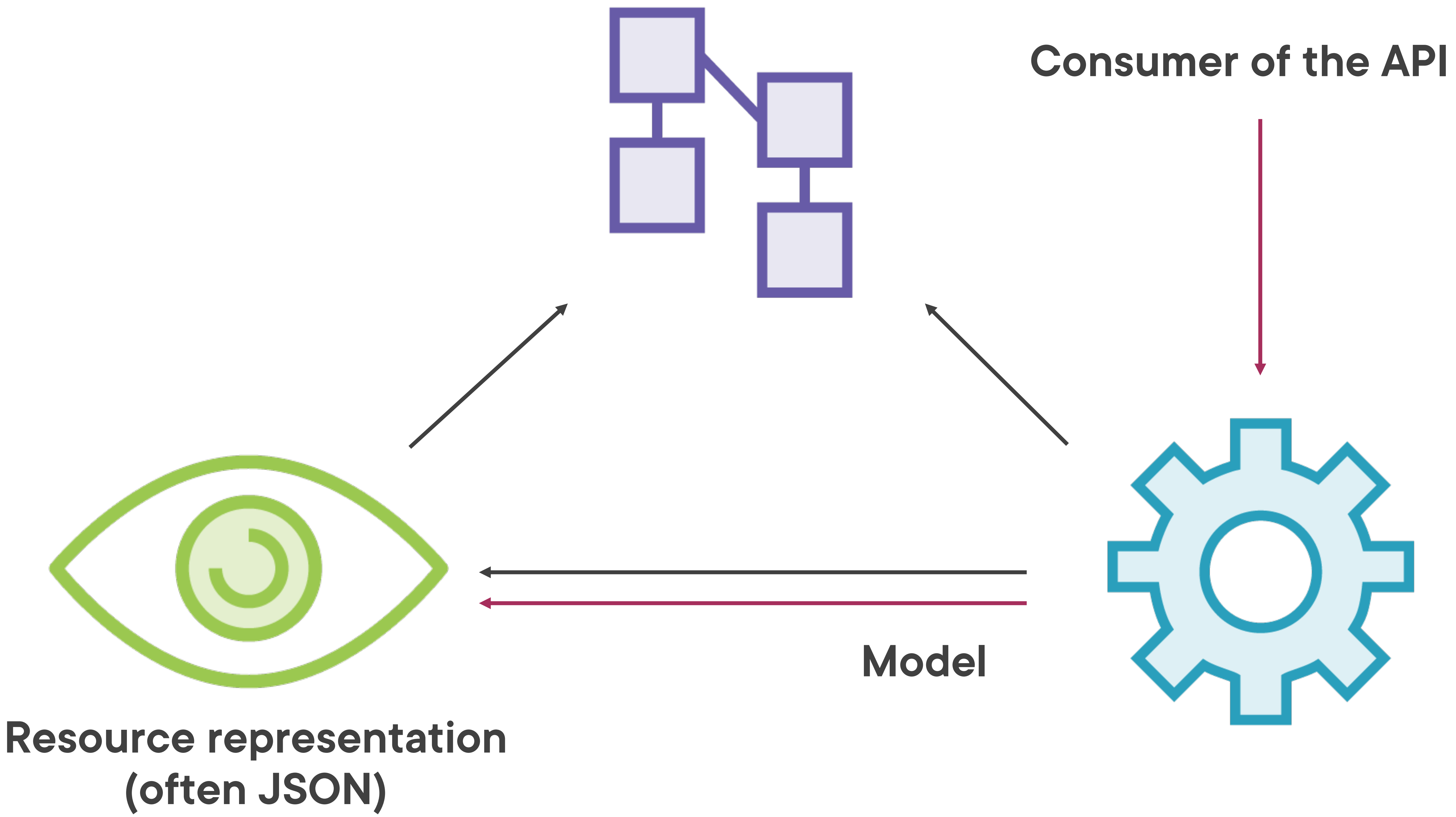
Not a full system and/or application architecture pattern!

- Typically lives near the presentation layer

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

Learning About Routing

app.UseRouting()

- Marks the position in the middleware pipeline where a routing decision is made

app.UseEndpoints()

- Marks the position in the middleware pipeline where the selected endpoint is executed ex

```
app.UseRouting();

app.UseAuthorization();        3midilwale

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

هذا هو النهج المفضل لواجهات برمجة التطبيقات

```
app.UseAuthorization();

app.MapControllers();
```

Attribute-based Routing

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

- Default in .NET 6
- Mixes request pipeline setup with route management

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

Ex

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	Level 400 Client mistake	Level 500 Server mistake
200 – OK	400 – Bad Request	500 – Internal Server Error
201 – Created	401 – Unauthorized	
204 – No Content	403- Forbidden	
	404 – Not Found	
	409 - Conflict	

Demo

Returning correct status codes

Demo

Returning child resources

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