Crear una API y devolver recursos

Resumen

- Aclaraciones sobre el patrón MVC
- Devolución de recursos
- Interacción con una API
- Negociación de contenidos
- Obtención de un archivo

Modelo-Vista-Controlador

Un patrón de arquitectura de software para la implementación de interfaces de usuario

Aclaraciones sobre el patrón MVC

Patrón muy común

- Existe en muchos lenguajes, con el apoyo de muchos frameworks
- Se utiliza para crear aplicaciones web ASP.NET Core orientadas al cliente

Modelo-Vista-Controlador

Un patrón de arquitectura de software para la implementación de interfaces de usuario

Aclaraciones sobre el patrón MVC

Acoplamiento débil

Separación de responsabilidades

Testabilidad

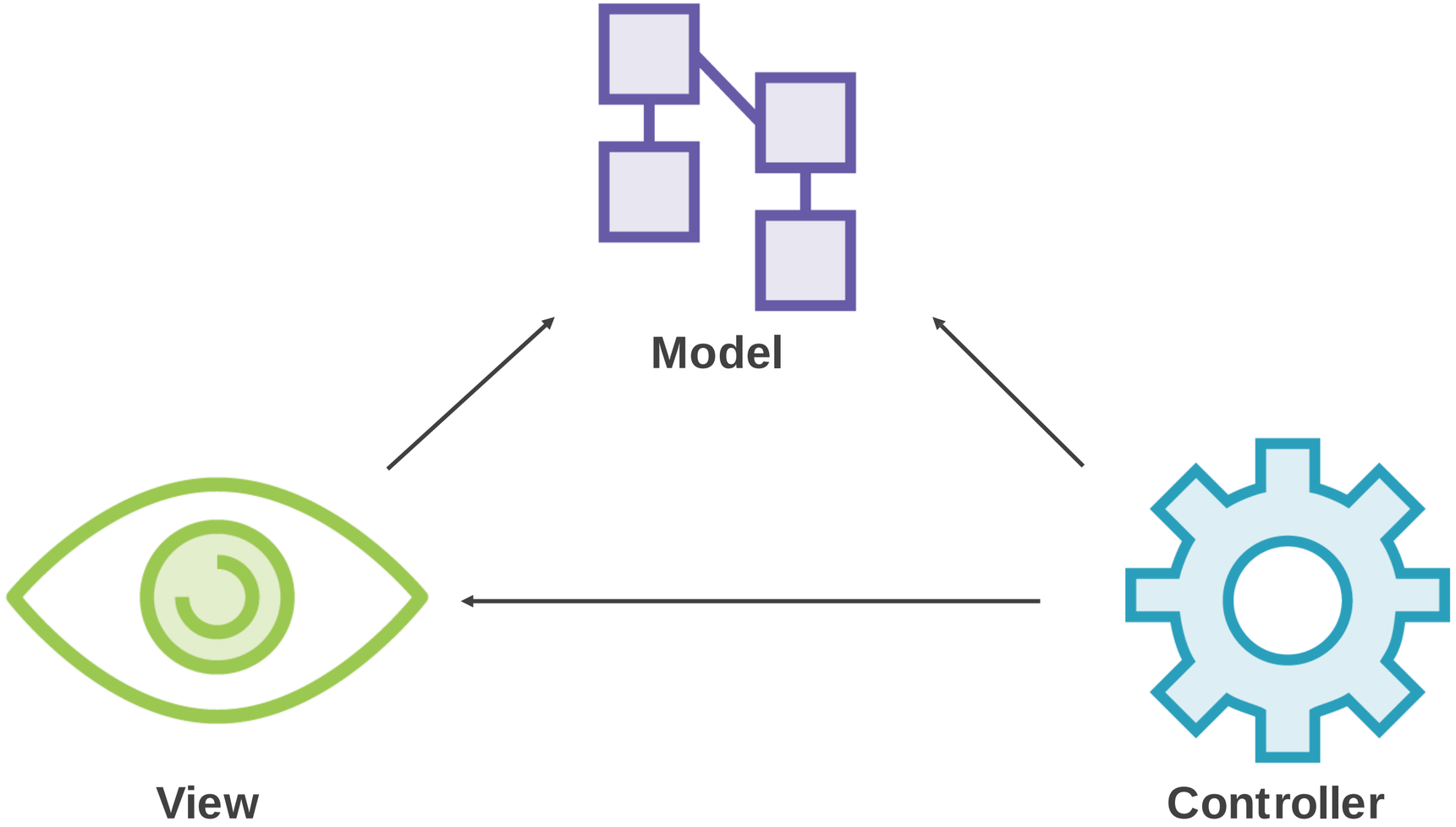
Reusabilidad

Aclaraciones sobre el patrón MVC

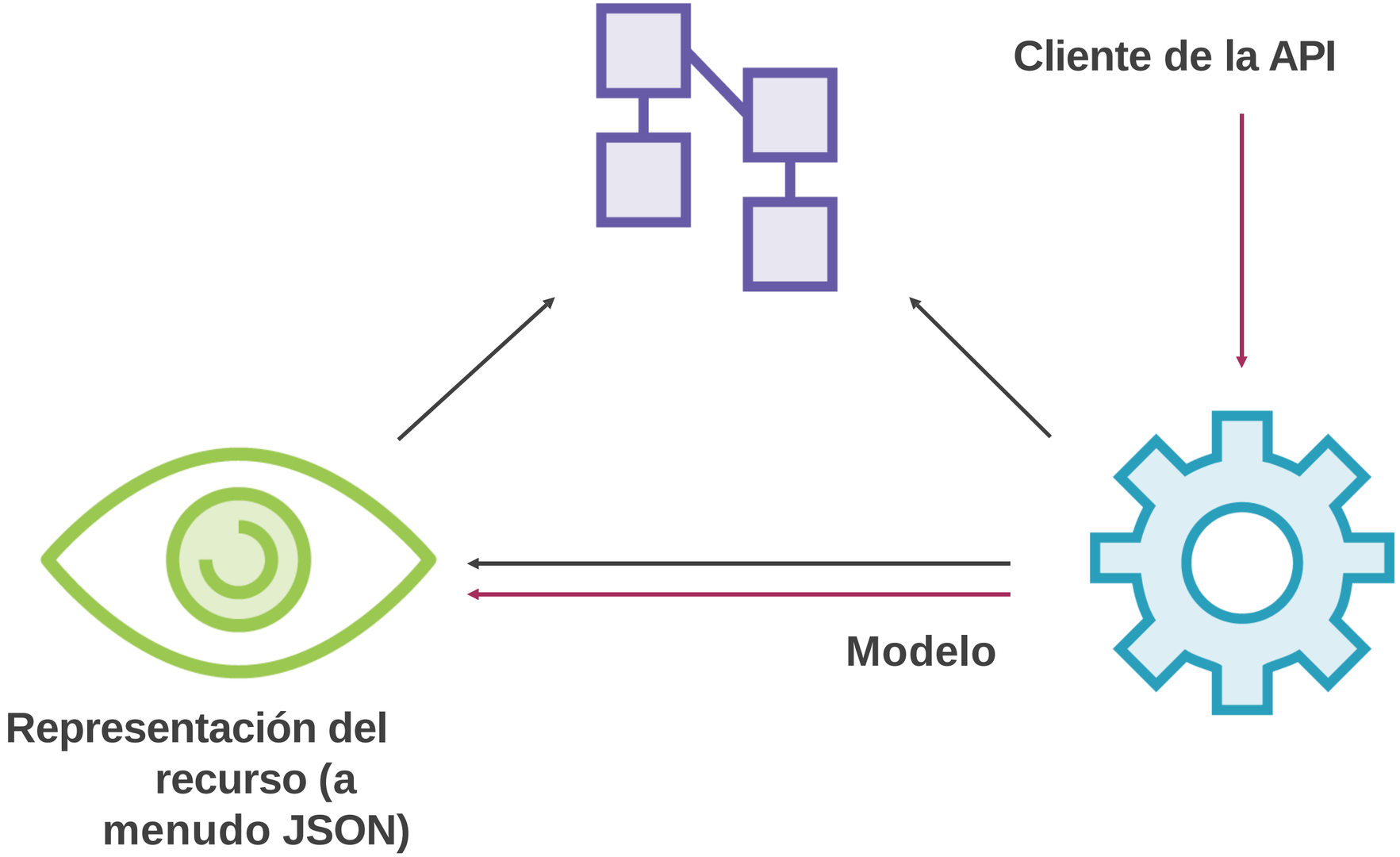
No es un patrón completo de arquitectura de sistemas y/o aplicaciones.

- Normalmente reside en la capa de presentación

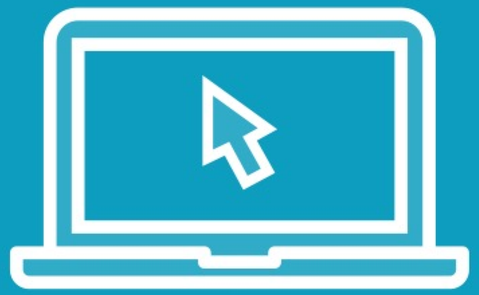
Aclaraciones sobre el patrón MVC



Aclaraciones sobre el patrón MVC

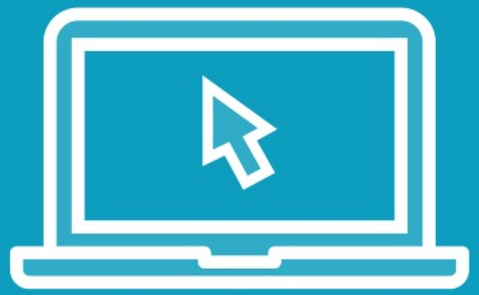


Demo



Registro de servicios API en el contenedor

Demo



Devolver recursos (parte 1)

Enrutamiento

El enrutamiento hace coincidir un URI de solicitud con una acción en un controlador

Aprendiendo sobre Enrutamiento

app.UseRouting()

- Marca la posición en la canalización del middleware donde se toma una decisión de enrutamiento

app.UseEndpoints()

- Marca la posición en el middleware pipeline donde se ejecuta el endpoint seleccionado

```
app.UseRouting();

app.UseAuthorization();

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

Aprendiendo sobre enrutamiento

El middleware que se ejecuta entre la selección del punto de conexión y la ejecución del punto de conexión seleccionado.

```
app.UseRouting();

app.UseAuthorization();

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

Aprendiendo sobre enrutamiento

El middleware que se ejecuta entre la selección del punto de conexión y la ejecución del punto de conexión seleccionado

```
app.UseRouting();

app.UseAuthorization();

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

Enrutamiento basado en atributos

No se aplican convenciones
Este es el enfoque preferido para APIs

```
app.UseAuthorization();

app.MapControllers();
```

Enrutamiento basado en atributos

Atajo: Llamar a `MapControllers` en el objeto `WebApplication` directamente

- Predeterminado en .NET 6
- Combina la configuración de las solicitudes con la gestión de las rutas

Enrutamiento basado en atributos

Utilizar atributos a nivel de controlador y de acción:

`[Route]`, `[HttpGet]`, …

En combinación con una plantilla URI, las solicitudes se ajustan a las acciones del controlador

Enrutamiento basado en atributos

Método HTTP	Atributo	Nivel	URI ejemplo
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	---

Enrutamiento basado en atributos

Para todos los métodos HTTP comunes, existe un atributo correspondiente

- `[HttpGet]`, `[HttpPost]`, `[HttpPatch]`, …

Enrutamiento basado en atributos

Método HTTP	Atributo	Nivel	URI ejemplo
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	---

Enrutamiento basado en atributos

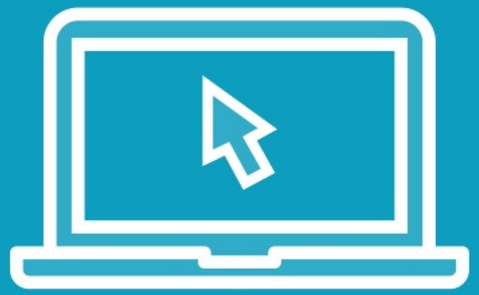
`[Route]` **no se asigna a un método HTTP**

- Utilízalo a nivel de controlador para proporcionar una plantilla que preceda a todas las plantillas definidas a nivel de acción

Enrutamiento basado en atributos

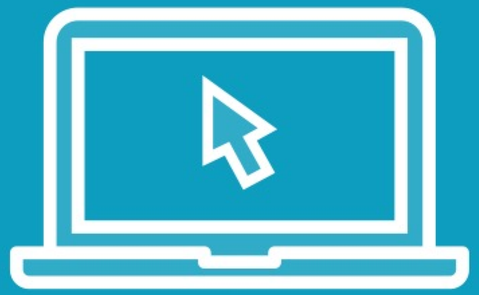
Método HTTP	Atributo	Nivel	URI ejemplo
GET	HttpGet	Action	/api/cities /api/cities/1
POST	HttpPost	Action	/api/cities
PUT	HttpPut	Action	/api/cities/1
PATCH	HttpPatch	Action	/api/cities/1
DELETE	HttpDelete	Action	/api/cities/1
---	Route	Controller	---

Demo



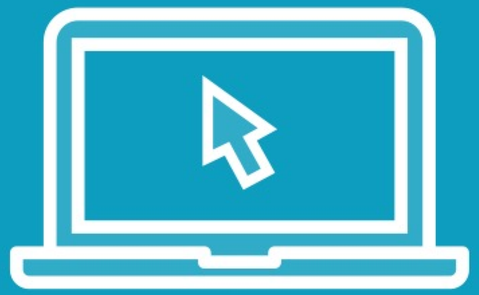
Devolver recursos (parte 2)

Demo



Utilizando Postman

Demo



Mejora de la arquitectura con clases modelo

Una aplicación, diferentes modelos

El modelo externo (DTO) es diferente del modelo de entidad (que se asigna al almacén de datos)

- Se hará evidente cuando introduzcamos Entity Framework Core

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

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

Una aplicación, diferentes modelos

El modelo externo es diferente del modelo de entidad

- Por ejemplo: campos calculados en el modelo externo

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

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

Una aplicación, diferentes modelos

El modelo externo es diferente del modelo de entidad

- Por ejemplo: campos calculados en el modelo externo

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

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

Una aplicación, diferentes modelos

El modelo externo es diferente del modelo de entidad

- Por ejemplo: campos calculados en el modelo externo

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

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

Una aplicación, diferentes modelos

El modelo externo es diferente del modelo de entidad

- Por ejemplo: campos calculados en el modelo externo

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

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

Una aplicación, diferentes modelos

El modelo externo es diferente del modelo de entidad

- Por ejemplo: campos calculados en el modelo externo

La importancia de los códigos de estado

Los códigos de estado indican al cliente de la API

- Si la solicitud ha funcionado como se esperaba
- Qué es lo que ha provocado el fallo de la solicitud

La importancia de los códigos de estado

Errores comunes:

- No devuelvas un 200 Ok cuando algo va mal
- No devuelva un 500 Internal Server Error cuando el cliente cometa un error
- …

La importancia de los códigos de estado

Nivel 200
Exito

200 – OK

201– Created

204 – No Content

Nivel 400
Error de cliente

400 – Bad Request

401– Unauthorized

403- Forbidden

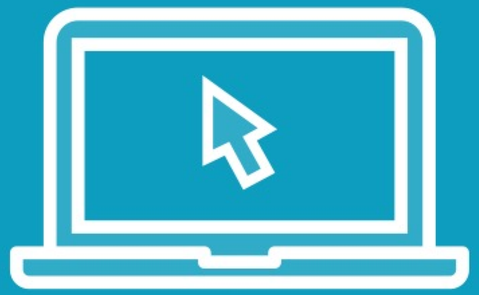
404 – Not Found

409 - Conflict

Nivel 500
Error de Servidor

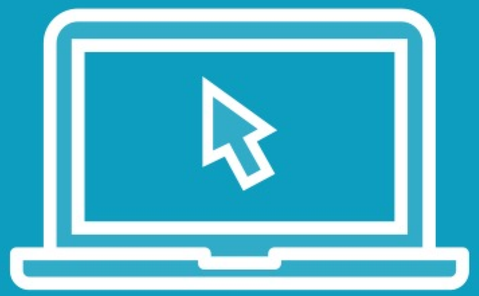
500 - Internal Server Error

Demo



Devolución de códigos de estado adecuados

Demo



Devolución de recursos hijos

Negociación de contenidos

El proceso de selección de la mejor representación para una respuesta dada cuando hay múltiples representaciones disponibles

Formateadores y Negociación de contenidos

El tipo(s) de medio se pasa(n) a través de la cabecera Accept de la petición

- application/json
- application/xml
- …

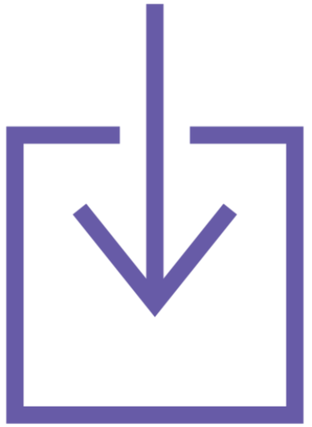
Formateadores y negociación de contenidos

Formateador de salida

Trata con la salida

Tipo de medios:

Cabecera Accept



Formateador de Entrada

Trata con la entrada

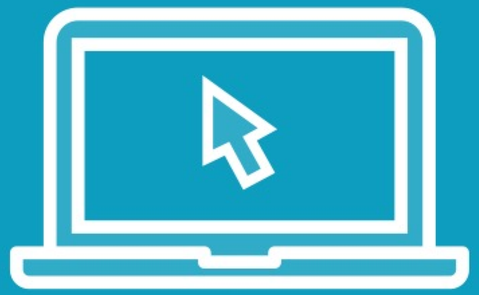
Tipo de Medios: Cabecera Content-Type

Formateadores y negociación de contenidos

El soporte se implementa mediante ObjectResult

- Los métodos de acción deben devolver este tipo

Demo



Formateadores y negociación de contenido

Resumen

Modelo-Vista-Controlador

- **Modelo**: Lógica de datos de la aplicación
- **Vista**: Mostrar datos
- **Controlador**: Interacción entre la vista y el modelo

Este patrón mejora la reutilización y la testabilidad

Resumen

El enrutamiento hace coincidir un URI de solicitud con una acción en un controlador

- Se aconseja el enrutamiento basado en atributos para las API

Resumen

La negociación del contenido es el proceso de selección de la mejor representación para una respuesta dada cuando hay múltiples representaciones disponibles

Resumen

Utiliza el método `File` de `ControllerBase` para devolver archivos

- Piensa en establecer el tipo de soporte correcto

A continuación:
Manipulación de recursos y validación de entradas