Capacitación MVC

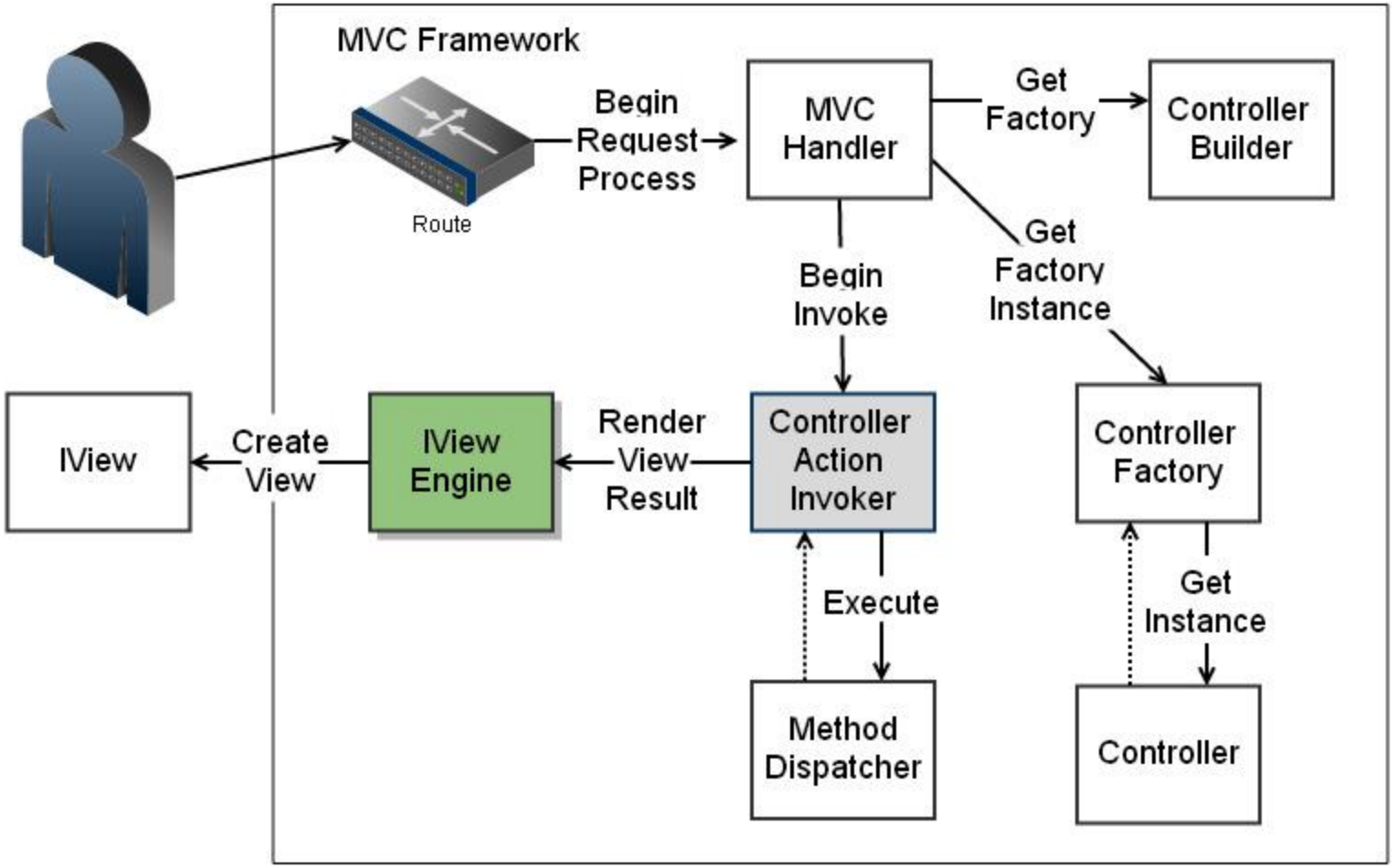
Día 3

Agenda - Día 3

- **MVC Pipeline**
  - Vista General
  - Puntos de Extensión
- **Action Filters**
- **Testing**
  - Controllers
  - Routes
- **Async Controllers**
- **Bundling & Minification**
- **Web API**

MVC Pipeline

Vista General



MVC Pipeline
Puntos de Extensión

- **Controller Factory**
  - ControllerBuilder.Current.SetControllerFactory
- **Validator Provider**
  - ModelValidatorProviders.Providers
- **Model Binders**
  - ModelBinders.Binders

Otros Puntos de extensión para tener en cuenta

- DependencyResolver
- Route Contraints
- IActionInvoker
- Route Handlers

Demo

“Puntos de extensión”

MVC Filters

Tipo de Filtros

- **Action Filters**
  - Estos filtros realizan operaciones tanto antes como luego de una llamada a una acción.
- **Result Filters**
  - Permiten realizar operaciones sobre el resultado de una ejecución.
- **Exception Filters**
  - Permiten agregar lógica de manejo y log de errores en caso de que la ejecución termine en una excepción.
- **Authorization Filters**
  - Son los primeros filtros a ser evaluados antes de cada llamada a una acción.

MVC Filters

Filter Scopes

- **Global Filters**
  - FilterConfig.RegisterGlobalFilters(…..);
- **Controller Filters**
  - Solo se invocan para un controller especifico (Todas sus acciones)
- **Action “Specific” Filters**
  - Solo se invocan para una acción determinada

Demo

"Filtros"

Práctica

“Auditoria con Filtros”

Testing

- ¿Para que testear? …
  - Es una inversión de tiempo a futuro
  - Fundamental en el refactory
  - Da para hablar mucho mas…
- ¿Que busco testeando **Rutas**?
  - Validar que llego a donde quiero ir…
- ¿Qué busco testeando **Controllers**?
  - Validar que a determinada entrada de datos se produce una determinada salida…

Demo

"Testing"

Async Controllers

- Son **Controllers** que permiten realizar acciones fuera del thread de la aplicación web.
- Se implementan heredando de **AsyncController** (MVC 4) y implementando la acción de forma asincrónica
- Debería usarlos cuando:
  - La acción realiza una acción tipo I/O (Ej. web services)
  - Accedo algún recurso externo
  - Puedo sacrificar **simplicidad** por **performance**

Demo

“Async Controllers”

Bundling & Minification

- **Bundling**
  - Combina archivos para reducir el numero de HTTP Requests necesarias para cargar una pagina

- **Minification**
  - Utiliza algoritmos de compresión para reducir el tamaño del archivo manteniendo el formato del archivo.

Demo

“Bundling & Minification”

ASP.NET Web API

Intro

- Framework para crear servicios HTTP usando el framework .NET.
- Modelo de programación similar a ASP.NET MVC (proyecto, controller, model, routing, scaffolding).

ASP.NET Web API

Características

- Los controllers heredan de ApiController.
- Los controllers devuelven datos y no vistas (como en ASP.NET MVC)
- Las actions mapean a verbos HTTP (GET, POST, PUT, DELETE) por convención de nombres.
- Content negotiation
- HttpClient
- Model validation
- Autenticación y autorización
- Otras:
  - Self host
  - Soporte de Odata
  - Help page

AS .N
Routing y actions (REST-style)

ut
n
r ontroller}/{id}",
d uteParameter.Optional }

ap ction:
nombres (Get, Post, Put, Delete)
Get], [HttpPost], [HttpPut], [HttpDelete])

RL

-
-
- */api/employees/blabla*

Posibles HTTP requests:

Método HTTP	URI path	Action	Parámetro
GET	api/employees	GetAll	
GET	api/employees/4	GetByID	4
DELETE	api/employees/4	Delete	4

ASP.NET Web API

Routing y actions (RPC-style)

```
routes.MapHttpRoute(
    name: "API Default",
    routeTemplate: "api/{controller}/{action}/{id}",
    defaults: new { id = RouteParameter.Optional }
);
```

Mapeo método HTTP -> action:

- Actions: nombre del método o [ActionName("nombre_action")]
- Atributos ([HttpGet], [HttpPost], [HttpPut], [HttpDelete])

URLs que matchean:

- /api/employees/GetAllEmployees
- /api/employees/GetByID/1
- /api/employees/Create

Usar [NonAction] para que un método público del controller no sea invocable por HTTP.

ASP.NET Web API

Exception handling

- HttpResponseException
- Exception filters
- Registración de exception filters
- HttpError

ASP.NET Web API

Exception handling - HttpResponseException

- La mayoría de las excepciones que no son manejadas por el controller son traducidas a un HTTP response con status 500 (Internal Server Error).

- **HttpResponseException** retorna un HTTP status code especificado en el constructor. Ej:

```
throw new HttpResponseException(HttpStatusCode.NotFound);
```

- Se puede controlar el contenido del response incluyendo el mensaje completo del response y pasándolo en la **HttpResponseException:**

```
var resp = new HttpResponseMessage(HttpStatusCode.NotFound)
{
    Content = new StringContent(string.Format("No employee with ID = {0}", id)),
    ReasonPhrase = "Employee ID Not Found"
}

throw new HttpResponseException(resp);
```

ASP.NET Web API

Exception handling – Exception filters

- Son un medio para customizar cómo Web API maneja las excepciones.
- Se ejecutan cuando un action lanza una exception que no es manejada y que NO es una **HttpResponseException**.
- Implementan **System.Web.Http.Filters.IExceptionFilter**.
- Lo más simple: heredar de **System.Web.Http.Filters.ExceptionFilterAttribute** y overridear el método **OnException**.
- Se registran de la forma similar que en MVC
  - Globalmente en **GlobalConfiguration.Configuration.Filters**
  - Por controller
  - Por action

ASP.NET Web API

Exception handling - HttpError (1)

- **HttpError** provee una forma consistente de retornar información de error en el body del response. Ej: retornar HTTP status code 404 (Not Found):

```
var message = string.Format("Product with id = {0} not found", id);
HttpError err = new HttpError(message);
return Request.CreateResponse(HttpStatusCode.NotFound, err);
```

- Cuando el producto no existe, el response HTTP contiene un **HttpError** en el body del response.

```
HTTP/1.1 404 Not Found
Content-Type: application/json; charset=utf-8
Date: Thu, 09 Aug 2012 23:27:18 GMT
Content-Length: 51

{
  "Message": "Product with id = 12 not found"
}
```

- El **HttpError** se serializa mediante el mismo proceso de content-negotiation y serialización que cualquier model fuertemente tipado.

- Se puede crear también con el extension method **CreateErrorResponse**:

```
var message = string.Format("Product with id = {0} not found", id);
return Request.CreateErrorResponse(HttpStatusCode.NotFound, message);
```

ASP.NET Web API

Exception handling – HttpError(2)

- Se puede pasar el model state a **CreateErrorResponse** para incluir en el response los errores de validación:

```
if (!ModelState.IsValid)
{
    return Request.CreateErrorResponse(HttpStatusCode.BadRequest, ModelState);
}
```

- Respuesta generada:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json; charset=utf-8
Content-Length: 320
{
  "Message": "The request is invalid.",
  "ModelState": {
    "item": [
      "Required property 'Name' not found in JSON. Path '', line 1, position 14."
    ],
    "item.Name": [
      "The Name field is required."
    ],
    "item.Price": [
      "The field Price must be between 0 and 999."
    ]
  }
}
```

ASP.NET Web API

Exception handling – HttpError (3)

- **HttpError** is una colección key-value (hereda de **Dictionary<string, object>**), por lo que se le pueden agregar entradas:

```
public HttpResponseMessage GetProduct(int id)
{
    Product item = repository.Get(id);

    if (item == null)
    {
        var message = string.Format("Product with id = {0} not found", id);
        var err = new HttpError(message);
        err["error_sub_code"] = 42;
        return Request.CreateErrorResponse(HttpStatusCode.NotFound, err);
    }
    else
    {
        return Request.CreateResponse(HttpStatusCode.OK, item);
    }
}
```

ASP.NET Web API

Serializaciòn a JSON

- Un *media-type formatter* es un objeto que puede:
  - Leer objetos CLR del cuerpo de un mensaje HTTP
  - Escribir objetos CLR en el cuerpo de un mensaje HTTP

- Built-in media-type formatters: JSON (Json.NET) y XML. Los clients pueden pedir JSON o XML en el header Accept del request HTTP.

- Se puede configurar JsonMediaTypeFormatter para usar DataContractJsonSerializer en vez de Json.NET.

```
var json = GlobalConfiguration.Configuration.Formatters.JsonFormatter;
json.UseDataContractJsonSerializer = true;
```

ASP.NET Web API

Serialización a JSON – Qué se serializa

- Por defecto, se serializan todas los atributos y propiedades públicos. Para omitir la serialización, usar [**JsonIgnore]**

```
public class Product
{
    public string Name { get; set; }
    [JsonIgnore]
    public int ProductCode { get; set; } // omitted
}
```

- Approach más "opt-in“ y “ WCF style”. [DataContract] y [DataMember]. Sólo se serializan las propiedades con [DataMember]

```
[DataContract]
public class Product
{
    [DataMember]
    public string Name { get; set; }

    public int ProductCode { get; set; }  // omitted by default
}
```

- Las propiedades read-only son serializadas por default.

ASP.NET Web API

Serialización a JSON – Cómo se serializa (1)

Fechas

- Por defecto, Json.NET escribe fechas en formato ISO 860.
  - 2012-07-27T18:51:45.53403Z // fecha UTC con sufijo "Z"
  - 2012-07-27T11:51:45.53403-07:00 // fecha local con offset del timezone

- Por defecto, Json.NET preserva el timezone. Para convertir todas las fechas a UTC:

```
jsonFormatter.SerializerSettings.DateTimeZoneHandling =
    Newtonsoft.Json.DateTimeZoneHandling.Utc;
```

- Para usar el Microsoft JSON date format ("/Date(*ticks*)/") en vez del ISO 8601:

```
jsonFormatter.SerializerSettings.DateFormatHandling =
    Newtonsoft.Json.DateFormatHandling.MicrosoftDateFormat;
```

Indentación

- Para escribir JSON indentado:

```
jsonFormatter.SerializerSettings.Formatting = Newtonsoft.Json.Formatting.Indented;
```

ASP.NET Web API

Serialización a JSON – Cómo se serializa (2)

- Para que las propiedades JSON sean escritas con camel -case:

```
jsonFormatter.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver();
```

- Un action puede devolver un objeto anónimo y serializarlo a JSON:

```
public object Get()
{
   return new {
      Name = "Alice",
      Age = 23,
      Pets = new List<string> { "Fido", "Polly", "Spot" }
   };
}
```

El body del response contendrá el siguiente JSON:

```
{"Name":"Alice","Age":23,"Pets":["Fido","Polly","Spot"]}
```

- Si Web API recibe un objeto JSON *loosely structured*, éste se puede deserializar a un Newtonsoft.Json.Linq.JObject (no permite model validation):

```
public void Post(JObject person)
{
   string name = person["Name"].ToString();
   int age = person["Age"].ToObject<int>();
}
```

ASP.NET Web API

Serialización a JSON – Cómo se serializa (3)

- Referencias circulares

```
jsonFormatter.SerializerSettings.PreserveReferencesHandling =
  Newtonsoft.Json.PreserveReferencesHandling.All;
```

- Ejemplo de resultado:

```
{"$id":"1","Name":"Sales","Manager":{"$id":"2","Name":"Alice","Departm
  ent":{"$ref":"1"}}}
```

ASP.NET Web API

Serialización a XML

- Similar a JSON.

- Algunas diferencias:
  - Por defecto, las propiedades readonly no se serializan. Si la propiedad tiene un *backing field* privado, se le puede poner **[DataMember]**.
  - Las fechas son escritas en formato ISO 8601. Ej: "2012-05-23T20:21:37.9116538Z".

- Se puede configurar serializadores por tipo. Ej.: se necesita usar XmlSerializer para compatibilidad hacia atrás. Se puede usar XmlSerializer para algunos tipos y DataContractSerializer para los demás.

```
var xmlFormatter = GlobalConfiguration.Configuration.Formatters.XmlFormatter;

// Use XmlSerializer for instances of type "Product":
xmlFormatter.SetSerializer<Product>(new XmlSerializer(typeof(Product)));
```

ASP.NET Web API
Configuración

Member	Description
DependencyResolver	Enables dependency injection for controllers. See Using the Web API Dependency Resolver .
Filters	Action filters.
Formatters	Media-type formatters.
IncludeErrorDetailPolicy	Specifies whether the server should include error details, such as exception messages and stack traces, in HTTP response messages. See IncludeErrorDetailPolicy.
Initializer	A function that performs final initialization of the **HttpConfiguration**.
MessageHandlers	HTTP message handlers.
ParameterBindingRules	A collection of rules for binding parameters on controller actions.
Properties	A generic property bag.
Routes	The collection of routes; see Routing in ASP.NET Web API.
Services	The collection of services. See Services.

¿Preguntas?

ARGENTINA
Clay 2954
Capital Federal (C1426DLD)
tel: 54+11+5299 5400

Belgrano 133 - Piso 2
Bahía Blanca,
Buenos Aires (B8000IJC)

San Martín 902 - 1º Piso - Oficina 6
Paraná,
Entre Ríos (E3100AAT)

Calle 48 N°1165
La Plata,
Buenos Aires (1900)
tel: 54+11+5299 5400

BRASIL
Cardoso de Melo 1470 – 8, Vila Olimpia
San Pablo (04548004)
tel: 55+11+3045 2193

URUGUAY
Roque Graseras 857
Montevideo (11300)
tel: 598+2+7117879

USA
12105 Sundance Ct.
Reston (20194)
tel:+703 842 9455

www.hexacta.com

hexacta