



Grupo Valentys

Curso:

Framework Gale – Sesión 1/3



Hola!, Yo soy **David!**

## CTO & CIO Grupo Valentys

Pasión por la arquitectura y sus abstracciones , si no es complejo no es divertido :P.

Mi Pasión: La innovación, el Emprendimiento y la tecnología.

Mi Sueño: Dejar de trabajar, para trabajar en lo que realmente me apasiona, la innovación y lograr cambiar al mundo con mis ideas en tecnología.



# En esta oportunidad veremos:

- Porqué Gale
- Principio RESTful
- Framework API
  - Modelo
  - Controlador
  - Servicios (Patrón Indirección)
  - Y la V??

# Porqué Gale

Valentys  
Material Design  
MVC  
AngularJS  
Mobile  
Experiencia  
Usabilidad  
Calidad

Soluciones  
Restful API  
DRY Web Components  
Desacoplamiento  
Ionic Escalabilidad

# Bueno antes de empezar.. ¿Qué es Gale?

- Gale es un conjunto de librerías desarrollado por Valentys Ltda. para acelerar el desarrollo de aplicaciones realizadas en AngularJS, por lo que brinda un punto inicial para el comienzo de cualquier proyecto asociado al área de I+D+i.
- El proyecto es Open Source (Apache License 2.0) y provee un grupo de componentes reutilizables que se añaden al grupo actual realizado por Material Design y usado como pilar de construcción para los desarrollos y lineamientos de nuestra empresa.



# Principio de Responsabilidad Única

- Una clase debe tener una y solo una única razón por la cual debe ser modificada.
- Cada clase debe ser responsable de realizar una actividad del sistema.

## Ventajas:

- Eliminación de clases monolíticas que aglutinen varias responsabilidades.
- Alta cohesión (una clase debe ser coherente y debe estar en lo posible relacionada con la clase).
- Bajo Acoplamiento (las clases no se encuentran directamente relacionadas entre ellas , lo que permite que al modificar una , no se tenga que modificar el resto).



# Patrón Controlador

- El patrón controlador es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado.
- Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, esto para aumentar la reutilización de código y a la vez tener un mayor control.

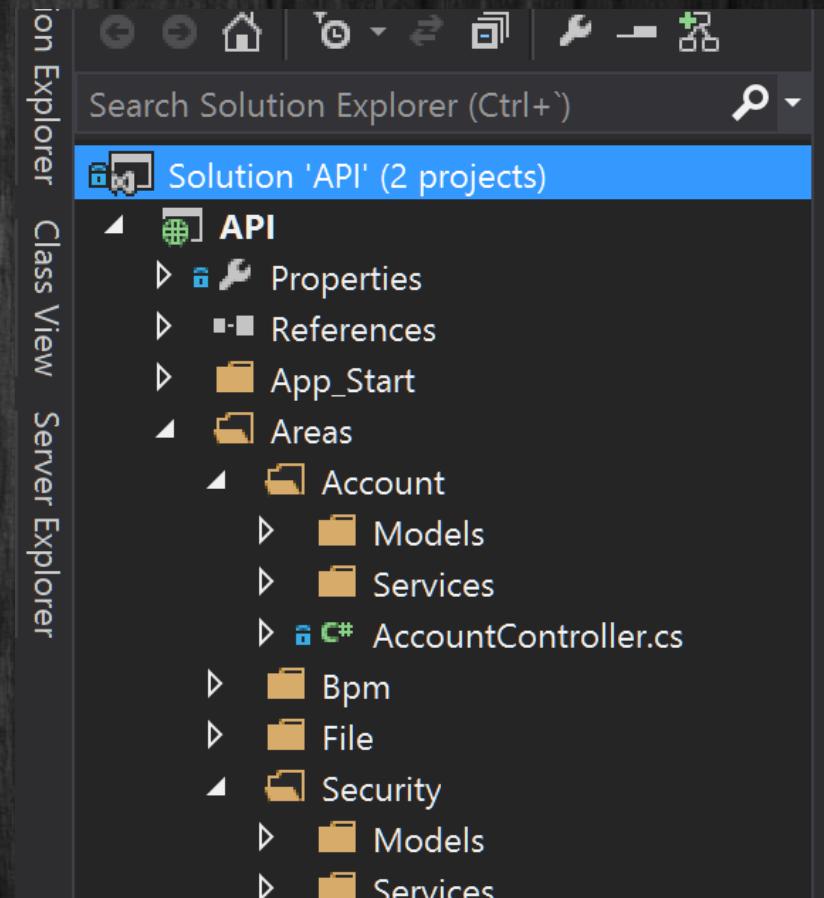
```
/// <summary>
/// Retrieve Target Account Information
/// </summary>
/// <param name="id">Account Token</param>
/// <returns></returns>
[HttpGet]
0 references | 0 authors | 0 changes
public IHttpActionResult Get(Guid id)
{
    return new Services.Me(id.ToString());
}

/// <summary>
/// Retrieve Current Account Information
/// </summary>
/// <returns></returns>
[HttpGet]
[Gale.REST.Http.Routing.Route("/Account/Me")]
0 references | 0 authors | 0 changes
public IHttpActionResult Current()
{
    return new Services.Me(this.User.PrimarySi
```



# Organización por características

- La organización de un proyecto por sus características es utilizada para la administración mas simple e iterativa de un desarrollo de gran tamaño , y permite una orientación mas simple de sus componentes debido al bajo acoplamiento entre sus partes (y posterior reutilización).
- En Gale tanto las interfaces graficas como la API RESTFul se encuentran modeladas por características , para un uso y mantenimiento mas fácil permitiendo separar el desarrollo de cada sección de forma simple y distributiva.



# Principio DRY y KISS

```
Web - node - 78x44
Valentys Ltda.
Contact: dmunoz@valentys.com

Web server: http://localhost:8000
Livereload: disabled
Base path: 'app'

Lifting Deployment Server...settings thing's up

nning "sync" task
nning "clean:dist" (clean) task
  0 paths cleaned.

nning "clean:post" (clean) task
  0 paths cleaned.

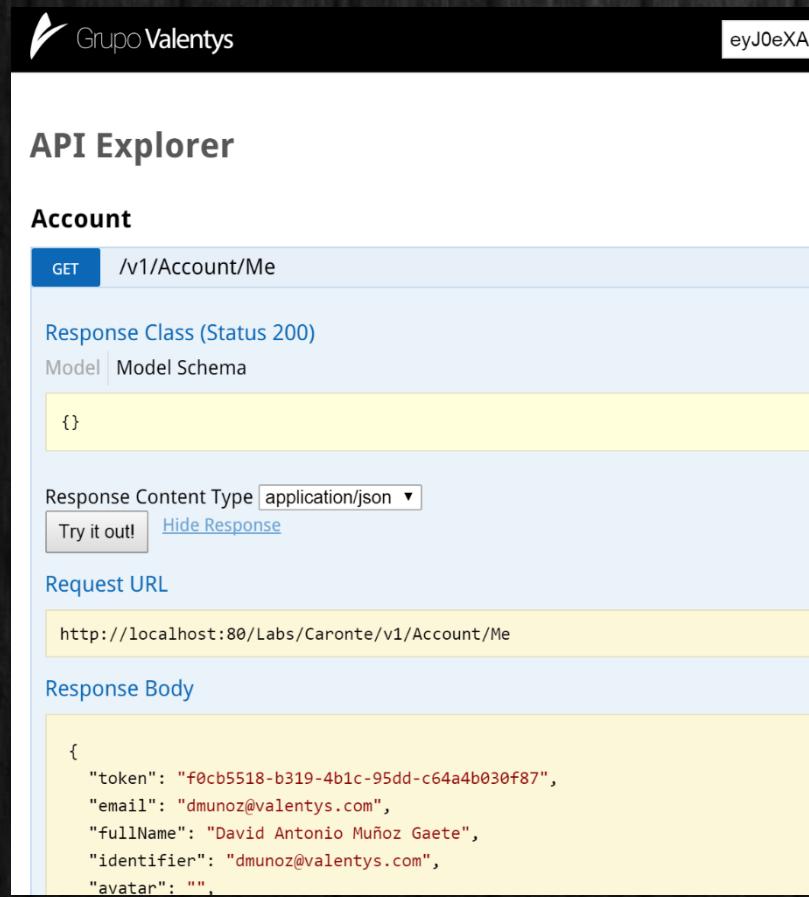
nning "injector:development" (injector) task
  ssing option `template`, using `dest` as template instead
  jecting css files (4 files)
  jecting js files (47 files)
  Nothing changed

nning "connect:development" (connect) task
  iting forever...
  arted connect web server on http://localhost:8000
```

- En Gale reducimos la cantidad de tareas repetitivas (*Don't Repeat Yourself*), utilizando herramientas disponibles en el mercado y mejorando las que existen , permitiendo eliminar tareas monótonas y engorrosas del día a día.
- El principio KISS (*Keep it simple, stupid!*) establece que la mayoría de sistemas funcionan mejor si se mantienen simples que si se hacen complejos; Por ello la simplicidad en Gale se mantiene como uno de los pilares fundamentales en la creación del framework.



# Documentación “Viva” y en línea



The screenshot shows the API Explorer interface for the Account endpoint. At the top, there's a header with the Grupo Valentys logo and a token field containing "eyJ0eXAi". Below the header, the title "API Explorer" is displayed, followed by the section "Account". Under "Account", there's a "GET /v1/Account/Me" operation. The "Response Class (Status 200)" section shows a JSON schema with an empty object "{}". The "Request URL" is listed as "http://localhost:80/Labs/Caronte/v1/Account/Me". The "Response Body" section displays a JSON response with the following data:

```
{  
  "token": "f0cb5518-b319-4b1c-95dd-c64a4b030f87",  
  "email": "dmunoz@valentys.com",  
  "fullName": "David Antonio Muñoz Gaete",  
  "identifier": "dmunoz@valentys.com",  
  "avatar": ""  
}
```

Swagger es un simple pero poderoso visualizador para la estructura y definición de tu API RESTful.

Con el ecosistema mas grande del planeta, miles de desarrolladores soportan Swagger en prácticamente la totalidad de los lenguajes de programación.

Con la habilitación de Swagger tendrás documentación interactiva, un SDK generador de clientes y descubrimiento, además de probar la funcionalidad de tu desarrollo en línea , sin utilizar herramientas de terceros.



# Gale no es algo nuevo... solo es algo mucho mejor



- Nosotros no inventamos la rueda, simplemente la hacemos mejor.
- Tomamos las mejores practicas y estándares basados en la experiencia y se unieron para mejorar el proceso , eficiencia y calidad de los desarrollos.
- Patrones como observer, fluent, subscriber, command, facade y decenas de otros se encuentran en los cimientos de Gale como una gran orquesta para quienes tienen la misión de producir software de una forma coherente y controlada.



# Principio RESTFull

Valentys  
Material Design  
Restful API  
Soluciones  
MVC  
Diseño DRY  
AngularJS Web Components  
Mobile  
Desacoplamiento  
Experiencia  
Usabilidad Ionic  
Calidad Escalabilidad

# Definición

- REST (Representational State Transfer), es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP.
- RESTful es la implementación de la arquitectura REST en servicios orientados a web.
- **4 Principios estrictos:**
  - Uso Correcto de URI's
  - Uso explícito de verbos HTTP (*GET, POST, PUT, DELETE, PATCH*) y códigos de estado
  - Nunca debe mantener algún estado en el servidor.
  - Debe responder tipos de medios de internet (Hipermédia – XML, JSON o ambos)



# Uso Correcto de URI's

- Las acciones directas sobre una ruta web , deben ser eliminada.

Ejemplo:    POST http://piik.in/v1/user/172522ec1028ab7/**edit**  
              PUT http://piik.in/v1/user/172522ec1028ab7

- No añadir métodos personalizados si los estándares no acompañan  
Para todo existe una solución estándar!

- Definición de URI en base a recursos (todo es un recurso), usuarios, roles, etc.
- Las URIs no deben implicar acciones y deben ser únicas
- Las URIs deben ser independientes de formato (el formato se define por content-type)



# Uso explícito de verbos

Para la manipulación de los recursos:

- **GET**: Para consultar y leer recursos
- **POST**: Para crear recursos
- **PUT**: Para editar recursos
- **DELETE**: Para eliminar recursos.
- **PATCH**: Para editar partes concretas de un recurso.

**GET /facturas**

-> Listado de facturas

**POST /facturas**

-> Crea una factura nueva

**PUT /facturas/123**

-> Edita la factura

**DELETE /facturas/123**

-> Elimina la factura

**PATCH /facturas/123**

información de la factura.



# Uso de Códigos de Estado

Al NO usar códigos de estado estandarizados nos exponemos a:

- No es REST ni es estándar.
- El cliente que acceda a este API debe conocer el funcionamiento especial y cómo tratar los errores de la misma, por lo que requiere un esfuerzo adicional importante para trabajar con una API con errores personalizados.
- Tenemos que preocuparnos por mantener nuestros propios códigos o mensajes de error, con todo lo que eso supone.
- Insostenible en el tiempo , debido a que se debe implementar una buena documentación que describa todos los tipos de errores.



# Uso de Códigos de Estado

## Códigos estándar mas usados:

Código HTTP	Operación	Descripción
200 OK	GET, PUT, DELETE Resource	No error, operation successful
201 Created	POST Resource was created	Successful creation of a resource.
204 No Content	GET, PUT, DELETE N/A	The request was processed successfully, but no response body is needed
400 Bad Request	GET, POST, PUT, DELETE	Malformed syntax or a bad query.
401 Unauthorized	GET, POST, PUT, DELETE	Action requires user authentication
403 Forbidden	GET, POST, PUT, DELETE	Authentication failure or invalid Application ID.
404 Not Found	GET, POST, PUT, DELETE	Resource not found.
408 Request Timeout	GET, POST	Request has timed out.
500 Server Error	GET, POST, PUT	Internal server error.
501 Not Implemented	POST, PUT, DELETE	Requested HTTP operation not supported.



# No mantiene estado

Una aplicación o cliente de servicio web REST debe incluir dentro del encabezado y del cuerpo HTTP de la petición todos los parámetros, contexto y datos que necesita el servidor para generar la respuesta.

En **Resumen**: Se debe Eliminar **TODO** componente de sesión o almacenamiento de variables específicas de usuario o de contexto en el lado del servidor o API!.

## Ventajas:

- El no mantener estado mejora el rendimiento de los servicios web y simplifica el diseño e implementación de los componentes del servidor, ya que la ausencia de estado en el servidor elimina la necesidad de sincronizar los datos de la sesión con una aplicación externa.



# Soporte a Tipos de Medios

La última restricción al momento de diseñar un servicio web REST tiene que ver con el formato de los datos que la aplicación y el servicio intercambian en las peticiones/respuestas.

Acá es donde realmente vale la pena mantener las cosas simples, legibles por humanos, y conectadas.

Tipo de formato Requerido	Encabezado Content-Type
JSON	application/json
XML	application/xml
XHTML	application/xhtml+xml

The screenshot shows a browser's developer tools Network tab with the XHR tab selected. At the top, it says "Response Content Type application/json". Below that is the URL "http://localhost:80/Labs/Caronte/v1/Account/Me". Under "Response Body", there is a JSON object:

```
{  
  "token": "f0cb5518-b319-4b1c-95dd-c64a4b030f87",  
  "email": "dmunoz@valentys.com",  
  "fullName": "David Antonio Muñoz Gaete",  
  "identifier": "dmunoz@valentys.com",  
  "avatar": "",  
  "lastConnection": "2015-08-26T19:12:06.713",  
  "roles": []  
}
```

At the bottom of the Network tab, it shows "Content-Length: 310" and "Content-Type: application/json; charset=utf-8".



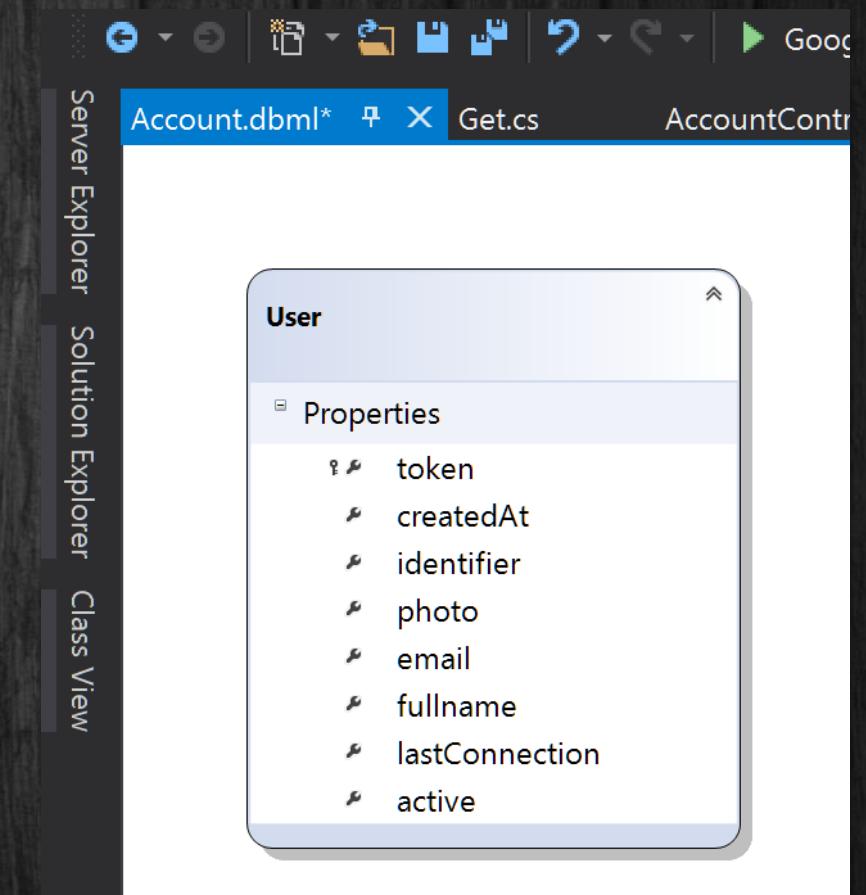
# Framework API

Valentys  
Material Design  
Restful API  
Soluciones API  
MVC  
DRY  
Diseño  
AngularJS  
Web Components  
Mobile  
Desacoplamiento  
Experiencia  
Ionic  
Calidad  
Usabilidad  
Escalabilidad

# Modelos

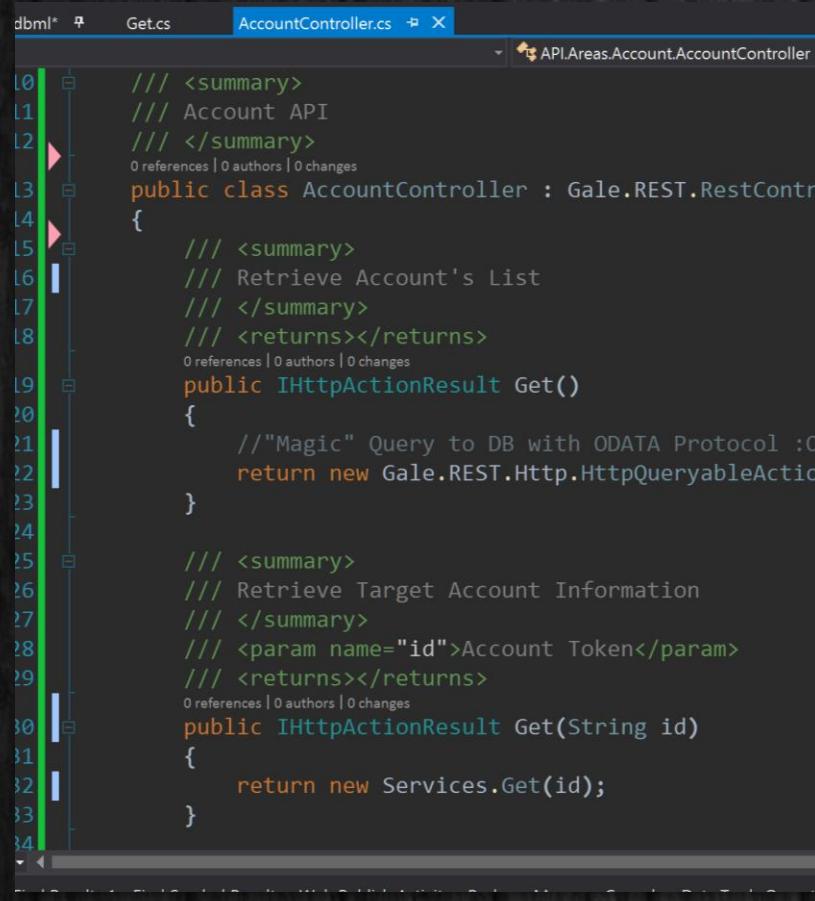
Un modelo contiene la representación de la información con la cual el sistema opera, y por ende queda definida como una entidad de negocio.

*En GALE, el modelo puede tener directa o indirecta relación con la base de datos y se construye a través de la herramienta de modelado grafico para “dbml”*



Video: Exponiendo un modelo con blueprint en GALE

# Controlador y el Patrón de indirección



```
dbml.cs Get.cs AccountController.cs API.Areas.Account.AccountController
L0  /// <summary>
L1  /// Account API
L2  /// </summary>
L3  public class AccountController : Gale.REST.RestController
L4  {
L5    /// <summary>
L6    /// Retrieve Account's List
L7    /// </summary>
L8    /// <returns></returns>
L9    public IHttpActionResult Get()
L10   {
L11     // "Magic" Query to DB with ODATA Protocol :0
L12     return new Gale.REST.Http.HttpQueryableActionResult();
L13   }
L14
L15   /// <summary>
L16   /// Retrieve Target Account Information
L17   /// </summary>
L18   /// <param name="id">Account Token</param>
L19   /// <returns></returns>
L20   public IHttpActionResult Get(string id)
L21   {
L22     return new Services.Get(id);
L23   }
L24 }
```

REST nos permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es increíblemente más simple y convencional que otras alternativas que se han usado en los últimos diez años como SOAP y XML-RPC.

El Controlador es el intermediario entre el usuario y la lógica de negocio definida (Patrón indirección), generando que la lógica puede ser re-implementada, tantas veces como sea necesario.



# Servicios de Negocio

Un servicio contiene la lógica de negocio con la cual el sistema o característica opera, y por ende queda definida como un componente de negocio.

En GALE, el servicio queda separado del modelo o controlador que expone el endpoint implementando la lógica de negocio que puede ser reutilizada.



## Video: Creando un servicio y exponiéndolo en una API



# Bueno y la V ??? (Vistas)

En GALE , las vistas no existen!, y es implementada en su totalidad en el cliente usando AngularJS + Material Design + GaleJS para desarrollos web o AngularJS + Ionic + GaleJS en desarrollos móviles Híbridos .

En Conclusión: No existen **vistas en el Servidor API.**

*(Exceptuando las plantillas razor usadas para correos o símiles).*



# Gracias por tu tiempo!

Súmate a la experiencia Valentys:



<http://www.valentys.com>



<https://www.facebook.com/GrupoValentys>



Grupo Valentys



@GrupoValentys



# Referencias

- <http://www.dosideas.com/noticias/java/314-introduccion-a-los-servicios-web-restful.html>
- <http://gale-docs.azurewebsites.net/>

