# **2 Marco teórico**

## **REST**

El término REST, acrónimo de *REpresentational State Transfer*,  fue introducido por primera vez por Roy Fielding (uno de los creadores de HTTP) en  su tesis doctoral para identificar un estilo arquitectónico basado en un conjunto de principios para el diseño de arquitecturas de software basadas en la red (Fielding 2000).

Posteriormente, el término fue extendido para describir un estilo para construir servicios web basados en los principios de REST, los servicios web construidos conforme a este estilo arquitectónico son llamados RESTful (Leonard Richardson, 2007)

REST no es un estándar, sino que es un estilo de arquitectura (de forma análoga, no hay un estándar cliente‐servidor, sino que hay un tipo de arquitectura cliente‐servidor). Sin ser un estándar, REST hace uso de estándares:

‐ HTTP [RFC 1945]: HyperText Transfer Protocol.

‐ URL [RFC 1738] (Uniform Resource Locator): Mecanismo de identificación de recursos.

‐ XML / HTML / PNG / etc.: Distintos formatos de representación de recursos.

‐ Tipos MIME: Como text/xml, text/html, image/png, etc.

Un concepto importante en REST es la existencia de recursos, que pueden ser accedidos utilizando un identificador global (URI). Para manipular estos recursos, los componentes de la red (clientes y servidores) se comunican a través de una interfaz estándar (HTTP) e intercambian representaciones de estos recursos (los archivos que se descargan y se envían).

Fielding define REST en base a un conjunto características o restricciones, los sistemas construidos en base a este estilo arquitectónico deben cumplir con estas características.

Las características que definen REST son las siguientes:

− Cliente – Servidor: Estilo de interacción basada en pull (bajo demanda).

− Sin estado: Cada petición desde un cliente hacia un servidor debe contener toda la información necesaria para comprender la petición, y no puede tomar provecho de ningún contexto almacenado en el servidor.

− Caché: Para mejorar la eficiencia de la red, las respuestas deben ser capaces de ser etiquetadas como cacheables o no cacheables.

− Interfaz uniforme: Todos los recursos son accesibles mediante un interfaz genérico (por ejemplo, HTTP GET, POST, PUT, DELETE).

− Recursos con nombres: El sistema se compone de recursos que son nombrados usando una URL.

− Representaciones de recursos interconectados: Las representaciones de los recursos están interconectadas usando URLs, por tanto un cliente está capacitado para progresar de un estado a otro.

− Componentes en capas: Intermediarios, tales como servidores proxy, servidores caché, gateways, etc., pueden ser introducidos entre los clientes y los recursos, para ofrecer rendimiento, seguridad.

A continuación se detallan cada una de estas características.

Cliente-Servidor

Consiste en la separación entre los 2 agentes básicos en un intercambio de información: el cliente y el servidor, ambos deben ser independientes entre sí, lo que permite lograr flexibilidad. Esta separación significa que, por ejemplo, los clientes no son conscientes de los datos almacenos, los cuales están dentro del servidor, con lo cual se mejora la portabilidad del código en el cliente. Por el lado de los servidores, estos no tienen en cuenta la interfaz del usuario o el estado del usuario, con lo cual los servidores pueden ser más simples y escalables. Clientes y servidores pueden ser desarrollados independientemente e incluso ser reemplazados mientras sus interfaces no sean alteradas.

Sin estado

Esto significa que el servidor no debe almacenar datos del cliente para mantener un estado del mismo, y que el estado para poder atender el pedido deben estar contenidos dentro del propio pedido, ya sea, como parámetros como parte de la URI del pedido, en el cuerpo o en el encabezado del pedido. Luego de que el servidor procesa un pedido, el estado resultante del procesamiento del recurso es comunicado al cliente a través del encabezado, del estado y del cuerpo de la respuesta. El cliente debe incluir toda la información necesaria para que el servidor complete el pedido.

Esta propiedad ayuda a la escalabilidad del sistema ya que el servidor no tiene que mantener un estado de los pedidos realizados por el cliente.

Cache

Esta técnica consiste en minimizar el uso de la red cuando sea posible, a través del reuso de interacciones previas.

Esto implica que el servidor debe ser capaz de especificar la manera en que las respuestas pueden ser o no cacheables. Una correcta administración de las opciones de cache evita que se realicen algunas de las interacciones innecesarias entre cliente y servidor permitiendo una mejora en la escalabilidad y el rendimiento.

Interface uniforme

Esta restricción define la interface entre el cliente y el servidor. Esto simplifica y desacopla la arquitectura, la cual permite a cada parte evolucionar independientemente. Las principales características para cumplir con una interface uniforme son:

Basado en recursos: recursos individuales son identificados en las peticiones usando URIs como identificadores de recursos. Los recursos están conceptualmente separados de las representaciones que se retornan a los clientes en una petición. Por ejemplo, ante un pedido, el servidor no envía su base de datos, en lugar de eso, entrega algún HTML, XML o JSON que representa algún registro de la base de datos expresado, por ejemplo, en inglés y codificado en UTF-8, dependiendo de los detalles de la solicitud y de la implementación del servidor.

Manipulación de los recursos a través de sus representaciones: Cuando un cliente maneja una representación de un recurso, incluyendo algún metadato adjunto, este tiene la suficiente información para modificar o borrar el recurso en el servidor, siempre que este tenga permisos para hacerlo.

Mensajes auto-descriptivos: cada mensaje debe incluye la información suficiente para describir como debe procesarse el mensaje. Por ejemplo, cual parser invocar puede ser especificado a través de un Internet media type. La respuesta también puede indicar explícitamente la habilidad de ser cacheable.

En resumen, según las restricciones mencionadas, cada servicio desarrollado en base a esta arquitectura será convertido en un recurso independiente el cual podrá ser accedido por una URI específica por medio de HTTP estándar sin mantener estado de la comunicación/uso de los mismos.

## **Swagger**

Swagger es un framework (especificación e implementación) completo para describir, producir, consumir y visualizar Servicios REST.

El objetivo de Swagger es que la documentación del sistema se actualice conforme se actualice el Servicio, de modo que la documentación de los métodos, parámetros y modelo esté integrada en el código del Servicio permitiendo que las APIs estén siempre sincronizadas.

Con la especificación de recursos declarativos de Swagger, los clientes pueden entender y consumir servicios sin conocer la implementación en el servidor o acceder al código del mismo.

El framework de interface de usuario permite tanto a los usuarios y desarrolladores interactuar con la API en un sandbox UI (entorno de prueba) dándole una visión de cómo la API responde a los parámetros y opciones.

Swagger está formado por 3 componentes:

- Servidor: Contiene la descripción de la API REST.

- Cliente: Utiliza la descripción de la API generada por el servidor.

- UI: Lee la descripción de la API desde el servidor y presenta esta como una página web y un sandbox interactivo para ejecutar las APIs que brinda nuestro servicio.

Servidor: El servidor expone un número de APIs a través de una URL (api-docs). Esta es una descripción en formato JSON de los recursos disponibles.

Cliente: El cliente es una aplicacion que desea usar las APIs que ofrece el servidor. El cliente recibe una URL que apunta a la api-docs y convierte este JSON en un objeto que se utiliza para llamar a la API REST. Los clientes están disponibles en un gran número de lenguajes de programación.

La interface (Swagger UI) cumple un doble propósito: como documentación y como una manera de permitir a los desarrolladores ejecutar las APIs REST sin tener que escribir código. Esta es completamente dinámica, dada una URL con la especificación de una API en formato JSON, la UI permite listar y ejecutar las APIs REST disponibles.

Swagger se compone de varios módulos, entre ellos:

- Swagger UI es una aplicación HTML (+JS+CSS) que permiten generar documentación de un API Swagger.

Es en esencia una consola web completamente HTML que permite listar operaciones y ejecutar operaciones.

- Swagger Core define las anotaciones Java y la lógica requerida para generar un cliente o servidor Swagger. Incluye ejemplos en Java, Scala, con Play2 framework.

- Swagger CodeGen ofrece un motor de plantillas para generar código cliente en diferentes lenguajes parseando la declaración de recursos.

Swagger Specification:

Desde el servidor se genera la documentación de la API y esta es publicada a través de una URL como un archivo en formato JSON o XML.

Esta documentación es conocida como Swagger specification (especificación de Swagger) y es consumida tanto por el cliente como por la Swagger UI (interface de usuario de Swagger).

Esta especificación está compuesta por dos archivos, que se describen a continuación:

- Listado de recursos (Resource listing): el cual lista las APIs que están disponibles en el servidor dando una breve descripción de ellas.

\* Descripción de la API (API Description): la misma contiene una descripción detallada de cada API contenida en el archivo de "Listado de recursos", incluyendo la descripción funcional (parámetros, nombres de funciones, valores retornados, entre otros) como también, una descripción entendible para el usuario de cómo utilizar la API.

La documentación de la API generada desde el servidor es un listado de recursos en formato JSON que describen los recursos Swagger que están disponibles, desde donde se pueden acceder, y como usarlos.

Si accedemos a la URL de la documentación de la API desde un navegador web podremos ver el listado antes mencionado.

Hay tres maneras de crear especificaciones Swagger, dependiendo de qué servidor se está utilizando:

\* Codegen (generador de código): esta es la manera tradicional de crear una especificación. El codegen de Swagger convierte las anotaciones en el código del servicio en la especificaciones.

\* Automáticamente: Algunos servidores, tales como swagger-node-express (https://npmjs.org/package/swagger-node-express) y swagger-play (https://github.com/wordnik/swagger-core/tree/master/modules/swagger-play2) permiten crear las APIs REST y la especificación de Swagger al mismo tiempo.

\* Manualmente: Se puede crear la especificación escribiendo el JSON a mano. Después de escribir la especificación a mano (o si se obtuvo la especificación desde otro servidor) se puede usar uno de los generadores de servidor para crear el código, tal como "node.js server generator" (https://github.com/wordnik/swagger-codegen/tree/master/samples/server-generator/node)

Tomando como ejemplo el recurso "service" de la API de Spamina con la operación "hire" y algunos parámetros requeridos y opcionales (el cual se utiliza para contratar un servicio en una empresa) el archivo Resource Listing quedaría de la siguiente formar:

{

"apiVersion": "1.0.0",

"swaggerVersion": "1.2",

"apis": [

{

"path": "/service.{format}",

"description": "Operaciones sobre servicios"

}

]

}

El cual especifica los siguientes atributos:

swaggerVersion: esta variable indica la versión de la especificación de Swagger, la que se está utilizando es la 1.2

basePath: Es la URL principal desde donde se expone la API

resourcePath: Es el path relativo del recurso que se quiere describir. Combinando el basePath más el resourcePath obtenemos la URI del recurso. Por ejemplo:

https://92.54.22.124/api/index.php/service

apis: Es la lista de APIs expuestas para este recurso.

Entre otros

Para poder generar este documento, Swagger nos ofrece una serie de anotaciones para documentar nuestra API, sus operaciones y parámetros, las cuales insertaremos en nuestro código. Este es un ejemplo en PHP:

/\*\*

\* @SWG\Resource(

\* apiVersion="1.0.0",

\* swaggerVersion="1.2",

\* basePath="https://92.54.22.124:443/api/index.php",

\* resourcePath="/service",

\* description="Operaciones sobre servicios",

\* produces="['application/json']"

\* )

\*/

Por otro lado, el archivo que define cada una de las APIs expuestas para cada recurso tendrá el siguiente formato:

{

"basePath": "https://92.54.22.124:443/api/index.php",

"swaggerVersion": "1.2",

"apiVersion": "1.0.0",

"resourcePath": "/service",

"apis": [

{

"path": "/service/hire",

"operations": [

{

"method": "POST",

"summary": "Contratar servicio",

"nickname": "hireService",

"parameters": [

{

"paramType": "header",

"name": "X-Emailsecurity-User",

"type": "string",

"required": true,

"description": "Usuario de autenticaci\u00f3n"

},

{

"paramType": "header",

"name": "X-Emailsecurity-Password",

"type": "string",

"required": true,

"description": "Contrase\u00f1a del usuario"

},

{

"paramType": "query",

"name": "clientId",

"type": "integer",

"required": true,

"description": "identificador de la empresa."

},

{

"paramType": "query",

"name": "service",

"type": "string",

"required": true,

"description": "Servicio a contratar. Valores posibles: firewall, archiving o mailbox."

},

{

"paramType": "query",

"name": "mailboxType",

"type": "string",

"required": false,

"description": "Tipo de mailbox a contratar. Utilizar solo si service = mailbox. Valores Posible: freemium, parla2, parla10 o parla30."

},

{

"paramType": "query",

"name": "activationCode",

"type": "string",

"required": false,

"description": "C\u00f3digo de activaci\u00f3n, formato xxxx-xxxx-xxxx-xxxx."

},

{

"paramType": "query",

"name": "licenses",

"type": "integer",

"required": false,

"description": "Cantidad de licencias asignadas."

},

{

"paramType": "query",

"name": "months",

"type": "integer",

"required": false,

"description": "Cantidad de meses de duraci\u00f3n de las licencias."

},

{

"paramType": "query",

"name": "demo",

"type": "boolean",

"required": false,

"description": "Indica si las licencias son demo o no. Valores posibles: true o false."

}

],

"responseMessages": [

{

"code": 461,

"message": "Fallo la autenticaci\u00f3n (usuario o contrase\u00f1a no v\u00e1lidos)."

},

{

"code": 462,

"message": "El usuario no existe"

},

{

"code": 463,

"message": "Autenticaci\u00f3n es obligatoria"

}

],

"notes": "Este recurso sirve para asignar un nuevo servicio a una empresa."

}

]

},

Donde, al igual que para el anterior documento, también se definen los parámetros "basePath", "swaggerVersion", "resourcePath".

Dentro del arreglo apis se definen la lista de APIs del recurso detallando:

- El path de la API

- Las operaciones

- los métodos sobre esas operaciones (GET, POST, PUT, DELETE, PATCH)

- los parámetros requeridos y opcionales

- los mensajes de respuesta posibles

entre otros.

Las anotaciones necesarias para generar estos documentos son:

/\*\*

\* @SWG\Api(

\* path="/service/hire",

\* @SWG\Operation(

\* method="POST",

\* summary="Contratar servicio",

\* notes="Este recurso sirve para asignar un nuevo servicio a una empresa.",

\* nickname="hireService",

\* @SWG\Parameter(

\* name="X-Emailsecurity-User",

\* description="Usuario de autenticación",

\* required=true,

\* type="string",

\* paramType="header"

\* ),

\* @SWG\Parameter(

\* name="X-Emailsecurity-Password",

\* description="Contraseña del usuario",

\* required=true,

\* type="string",

\* paramType="header"

\* ),

\* @SWG\Parameter(

\* name="clientId",

\* description="identificador de la empresa.",

\* required=true,

\* type="integer",

\* paramType="query"

\* ),

\* @SWG\Parameter(

\* name="service",

\* description="Servicio a contratar. Valores posibles: firewall, archiving o mailbox.",

\* required=true,

\* type="string",

\* paramType="query"

\* ),

\* @SWG\Parameter(

\* name="mailboxType",

\* description="Tipo de mailbox a contratar. Utilizar solo si service = mailbox. Valores Posible: freemium, parla2, parla10 o parla30.",

\* required=false,

\* type="string",

\* paramType="query"

\* ),

\* @SWG\Parameter(

\* name="activationCode",

\* description="Código de activación, formato xxxx-xxxx-xxxx-xxxx.",

\* required=false,

\* type="string",

\* paramType="query"

\* ),

\* @SWG\Parameter(

\* name="licenses",

\* description="Cantidad de licencias asignadas.",

\* required=false,

\* type="integer",

\* paramType="query"

\* ),

\* @SWG\Parameter(

\* name="months",

\* description="Cantidad de meses de duración de las licencias.",

\* required=false,

\* type="integer",

\* paramType="query"

\* ),

\* @SWG\Parameter(

\* name="demo",

\* description="Indica si las licencias son demo o no. Valores posibles: true o false.",

\* required=false,

\* type="boolean",

\* paramType="query"

\* ),

\* @SWG\ResponseMessage(code=461, message="Fallo la autenticación (usuario o contraseña no válidos)."),

\* @SWG\ResponseMessage(code=462, message="El usuario no existe"),

\* @SWG\ResponseMessage(code=463, message="Autenticación es obligatoria")

\* )

\* )

\*/

Los tipos de datos primitivos soportados por Swagger son los mismos que los soportados por el esquema JSON (http://json-schema.org/latest/json-schema-core.html#anchor8):

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombre** | **Tipo** | [**formato**](https://github.com/swagger-api/swagger-spec/blob/master/versions/1.2.md#dataTypeFormat) | **Comentarios** |
| integer | integer | int32 | signed 32 bits |
| long | integer | int64 | signed 64 bits |
| float | number | float |  |
| double | number | double |  |
| string | string |  |  |
| byte | string | byte |  |
| boolean | boolean |  |  |
| date | string | date |  |
| dateTime | string | date-time |  |

La siguiente tabla muestra los tipos de datos disponibles

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombre del campo** | **Tipo** | **Validity** | **Descripción** |
| type | string | Any | **Required (if**$ref**is not used).** The return type of the operation. The value MUST be one of the Primitives, array or a model's id. |
| $ref | string | Any | **Required (if**type**is not used).** The Model to be used. The value MUST be a model's id. |
| format | string | primitive | Fine-tuned primitive type definition. See Primitives for further information. The value MUST be one that is defined under Primitives, corresponding to the right primitive type. |
| defaultValue | *special* | primitive | The default value to be used for the field. The value type MUST conform with the primitive's type value. |
| enum | [string] | string | A fixed list of possible values. If this field is used in conjunction with thedefaultValue field, then the default value MUST be one of the values defined in the enum. |
| minimum | string | number,integer | The minimum valid value for the type, inclusive. If this field is used in conjunction with the defaultValue field, then the default value MUST be higher than or equal to this value. The value type is string and should represent the minimum numeric value. Note: This will change to a numeric value in the future. |
| maximum | string | number,integer | The maximum valid value for the type, inclusive. If this field is used in conjunction with the defaultValue field, then the default value MUST be lower than or equal to this value. The value type is string and should represent the maximum numeric value. Note: This will change to a numeric value in the future. |
| items | [Items Object](https://github.com/swagger-api/swagger-spec/blob/master/versions/1.2.md#434-items-object) | array | **Required.** The type definition of the values in the container. A container MUST NOT be nested in another container. |
| uniqueItems | boolean | array | A flag to note whether the container allows duplicate values or not. If the value is set to true, then the array acts as a set. |

Los campos que contiene el archivo Resource Listing son los siguiente:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Type** | **Description** |
| swaggerVersion | string | **Required.** Specifies the Swagger Specification version being used. It can be used by the Swagger UI and other clients to interpret the API listing. The value MUST be an existing Swagger specification version. |
| Currently, "1.0", "1.1", "1.2" are valid values. The field is a string type for possible non-numeric versions in the future (for example, "1.2a"). |
| apis | [ Resource Object ] | Required. Lists the resources to be described by this specification implementation. The array can have 0 or more elements. |
| apiVersion | string | Provides the version of the application API (not to be confused by thespecification version). |
| info | [Info Object](https://github.com/swagger-api/swagger-spec/blob/master/versions/1.2.md#513-info-object) | Provides metadata about the API. The metadata can be used by the clients if needed, and can be presented in the Swagger-UI for convenience. |
| authorizations | [Authorizations Object](https://github.com/swagger-api/swagger-spec/blob/master/versions/1.2.md#514-authorizations-object) | Provides information about the authorization schemes allowed on this API. |

Los campos contenidos en la “API Declaration” son los siguientes:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Type** | **Description** |
| swaggerVersion | string | **Required.** Specifies the Swagger Specification version being used. It can be used by the Swagger UI and other clients to interpret the API listing. The value MUST be an existing Swagger specification version. |
| Currently, "1.0", "1.1", "1.2" are valid values. |
| apiVersion | string | Provides the version of the application API (not to be confused by thespecification version). |
| basePath | string | Required. The root URL serving the API. This field is important because while it is common to have the Resource Listing and API Declarations on the server providing the APIs themselves, it is not a requirement. The API specifications can be served using static files and not generated by the API server itself, so the URL for serving the API cannot always be derived from the URL serving the API specification. The value SHOULD be in the format of a URL. |
| resourcePath | string | The relative path to the resource, from the basePath, which this API Specification describes. The value MUST precede with a forward slash ("/"). |
| apis | [API Object] | Required. A list of the APIs exposed on this resource. There MUST NOT be more than one API Object per path in the array. |
| models | [Models Object](https://github.com/swagger-api/swagger-spec/blob/master/versions/1.2.md#526-models-object) | A list of the models available to this resource. Note that these need to be exposed separately for each API Declaration. |
| produces | [string] | A list of MIME types the APIs on this resource can produce. This is global to all APIs but can be overridden on specific API calls. |
| consumes | [string] | A list of MIME types the APIs on this resource can consume. This is global to all APIs but can be overridden on specific API calls. |
| authorizations | [Authorizations Object](https://github.com/swagger-api/swagger-spec/blob/master/versions/1.2.md#5210-authorizations-object) | A list of authorizations schemes required for the operations listed in this API declaration. Individual operations may override this setting. If there are multiple authorization schemes described here, it means they're all applied. |

Cada una de las operaciones contenidas dentro del archivo “API Declaration” esta compuesto por los siguientes campos:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Type** | **Description** |
| method | string | **Required.** The HTTP method required to invoke this operation. The value MUST be one of the following values: "GET", "HEAD", "POST", "PUT","PATCH", "DELETE", "OPTIONS". The values MUST be in uppercase. |
| summary | string | A short summary of what the operation does. For maximum readability in the swagger-ui, this field SHOULD be less than 120 characters. |
| notes | string | A verbose explanation of the operation behavior. |
| nickname | string | **Required.** A unique id for the operation that can be used by tools reading the output for further and easier manipulation. For example, Swagger-Codegen will use the nickname as the method name of the operation in the client it generates. The value MUST be alphanumeric and may include underscores. Whitespace characters are not allowed. |
| authorizations | [Authorizations Object](https://github.com/swagger-api/swagger-spec/blob/master/versions/1.2.md#5210-authorizations-object) | A list of authorizations required to execute this operation. While not mandatory, if used, it overrides the value given at the API Declaration'sauthorizations. In order to completely remove API Declaration's authorizations completely, an empty object ({}) may be applied. |
| parameters | [Parameter Object] | Required. The inputs to the operation. If no parameters are needed, an empty array MUST be included. |
| responseMessages | [Response Message Object] | Lists the possible response statuses that can return from the operation. |
| produces | [string] | A list of MIME types this operation can produce. This is overrides the globalproduces definition at the root of the API Declaration. Each string value SHOULD represent a MIME type. |
| consumes | [string] | A list of MIME types this operation can consume. This is overrides the global consumes definition at the root of the API Declaration. Each stringvalue SHOULD represent a MIME type. |
| deprecated | string | Declares this operation to be deprecated. Usage of the declared operation should be refrained. Valid value MUST be either "true" or "false". *Note:*This field will change to type boolean in the future. |

Los parámetros soportados en las operaciones están definidos por los siguientes campos:

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Type** | **Description** |
| method | string | **Required.** The HTTP method required to invoke this operation. The value MUST be one of the following values: "GET", "HEAD", "POST", "PUT","PATCH", "DELETE", "OPTIONS". The values MUST be in uppercase. |
| summary | string | A short summary of what the operation does. For maximum readability in the swagger-ui, this field SHOULD be less than 120 characters. |
| notes | string | A verbose explanation of the operation behavior. |
| nickname | string | **Required.** A unique id for the operation that can be used by tools reading the output for further and easier manipulation. For example, Swagger-Codegen will use the nickname as the method name of the operation in the client it generates. The value MUST be alphanumeric and may include underscores. Whitespace characters are not allowed. |
| authorizations | [Authorizations Object](https://github.com/swagger-api/swagger-spec/blob/master/versions/1.2.md#5210-authorizations-object) | A list of authorizations required to execute this operation. While not mandatory, if used, it overrides the value given at the API Declaration'sauthorizations. In order to completely remove API Declaration's authorizations completely, an empty object ({}) may be applied. |
| parameters | [Parameter Object] | **Required.** The inputs to the operation. If no parameters are needed, an empty array MUST be included. |
| responseMessages | [Response Message Object] | Lists the possible response statuses that can return from the operation. |
| produces | [string] | A list of MIME types this operation can produce. This is overrides the globalproduces definition at the root of the API Declaration. Each string value SHOULD represent a MIME type. |
| consumes | [string] | A list of MIME types this operation can consume. This is overrides the global consumes definition at the root of the API Declaration. Each stringvalue SHOULD represent a MIME type. |
| deprecated | string | Declares this operation to be deprecated. Usage of the declared operation should be refrained. Valid value MUST be either "true" or "false". *Note:*This field will change to type boolean in the future. |

El detalle de las especificaciones para cada versión se encuentra en las URLs:

Specificación versión 1.2: <https://github.com/swagger-api/swagger-spec/blob/master/versions/1.2.md>

Specificacion versión 2.0: <https://github.com/wordnik/swagger-spec/blob/master/versions/2.0.md>

La lista de Anotaciones para la version de Swagger-php puede ser consultada a través del siguiente link:

http://zircote.com/swagger-php/annotations.html

Para la implementación en Spamina utilizamos la versión 1.2 que es la soportada por la versión de Swagger-PHP

Ventajas

- Permite tanto a humanos como a maquinas descubrir y entender las capacidades de los servicios sin acceder al código fuente, documentación o inspeccionar tráfico a través de la red

- Permite a los consumidores de los servicios entender e interactuar con los servicios remotos con una cantidad mínima de implementación.

- no requiere que se reescriba la API existente

- no requiere que se vincule algún software a algún servicio

- Permite describir servicios no propios, solamente deben ser descriptos a través de la especificación de Swagger.

- Swagger es una especificación y por lo tanto es independiente del lenguaje con el que están implementados los clientes y el servidor.

- La documentación está basada en el mismo código, cualquier cambio en el código es reflejado en la documentación, no se necesita mantener una documentación extra.

- Swagger ofrece implementaciones del lado del servidor en Scala, Java y HTML5

- Swagger permite generar clientes Scala, Java, Javascript, Ruby, PHP y ActionScrip 3.

- Swagger soporta JSON y XML.

- Swagger proporciona una UI para que desarrolladores y usuarios interactuar y probar el API

Desventajas

- la gran cantidad de anotaciones que requiere para la documentación puede ensuciar un poco el código de la API.

- Permite probar cada API individualmente a través del cliente que ofrece, pero no permite la automatización de estas pruebas a través de la generación de casos de prueba automáticos.

Limitaciones

Al momento de instanciar el framwork en la herramienta que estamos desarrollando en mi trabajo, nos encontramos con la limitación de que Swagger NO permite definir parámetros opcionales, con lo cual tuvimos que agregar algunas modificaciones para poder utilizar en los llamados a la API parámetros opcionales.

## **Rest-Assured**

Referencias bibliográficas

* (Fielding, 2000). Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine: Tesis doctoral.
* (Pautasso) libro "REST: From Research to Practice" de "E. Wilde and C. Pautasso".
* (Leonard Richardson, 2007). RESTful Web Services. O'Reilly Media.
* (Swagger webpage) - Swagger: A simple, open standard for describing REST APIs with JSON | Reverb for Developers - <https://helloreverb.com/developers/swagger>
* (Swagger spec.) The Swagger Specification - https://github.com/wordnik/swagger-spec