Article Eduard Lebedyuk · Nov 20, 2019 9m read

# Developing REST API with a spec-first approach

In this article, I would like to talk about the spec-first approach to REST API development.

While traditional code-first REST API development goes like this:

- Writing code
- REST-enabling it
- Documenting it (as a REST API)

Spec-first follows the same steps but reverse. We start with a spec, also doubling as documentation, generate a boilerplate REST app from that and finally write some business logic.

This is advantageous because:

- You always have relevant and useful documentation for external or frontend developers who want to use your REST API
- Specification created in OAS (Swagger) can be imported into a variety of tools allowing editing, client generation, API Management, Unit Testing and automation or simplification of many other tasks
- Improved API architecture. In code-first approach, API is developed method by method so a developer can easily lose track of the overall API architecture, however with the spec-first developer is forced to interact with an API from the position if API consumer which usually helps with designing cleaner API architecture
- Faster development as all boilerplate code is automatically generated you won't have to write it, all that's left is developing business logic.
- Faster feedback loops consumers can get a view of the API immediately and they can easier offer suggestions simply by modifying the spec

Let's develop our API in a spec-first approach!

## Plan

- 1. Develop spec in swagger
  - Docker
  - Locally
  - Online
- 2. Load spec into IRIS
  - API Management REST API
  - ^REST
  - Classes
- 3. What happened with our spec?
- 4. Implementation
- 5. Further development
- 6. Considerations
  - Special parameters
  - CORS
- 7. Load spec into IAM

# Develop specification

The first step is unsurprisingly writing the spec. InterSystems IRIS supports Open API Specification (OAS):

OpenAPI Specification (formerly Swagger Specification) is an API description format for REST APIs. An OpenAPI file allows you to describe your entire API, including:

- Available endpoints (/users) and operations on each endpoint (GET /users, POST /users)
- · Operation parameters Input and output for each operation
- Authentication methods
- Contact information, license, terms of use and other information.

API specifications can be written in YAML or JSON. The format is easy to learn and readable to both humans and machines. The complete OpenAPI Specification can be found on GitHub: <u>OpenAPI 3.0</u> <u>Specification</u>

- from Swagger docs.

We will use Swagger to write our API. There are several ways to use Swagger:

- Online
- Docker: docker run -d -p 8080:8080 swaggerapi/swagger-editor
- Local installation

After installing/running Swagger, you should see this window in a web browser:

H Swagger Editor. File + Edit + Generate Server + Generate Client +	
	<u>^</u>
2 - info:	
3 description: "This is a sample server Petstore server. You can find out more about Swagger at [http	
://swagger.io](http://swagger.io) or on [irc.freenode.net, #swagger](http://swagger.io/irc/). For	
this sample, you can use the api key `special-key` to test the authorization filters."	Swadder Petstore
	on agger i etetere
5 title: "Swagger Petstore"	[ Base URL: petstore.swagger.io/v2 ]
6 termsOfService: "http://swagger.io/terms/"	
	This is a sample server Petstore server, You can find out more about Swagger at http://swagger.jo or on inc.freenode.net. #swagger. For this
8 email: "apiteam@swagger.io"	sample, you can use the api key special-key to test the authorization filters.
10 name: "Apache 2.0"	Terms of service
11 unl: "http://www.apache.org/licenses/LICENSE-2.0.html"	Contractions development
12 nost: "petstore.swagger.10"	Contact the developer
13 DasePath: //v2	Apache 2.0
14 tdg5:	Find out more should Swagner
15' - Hand, pet	Time out more about onagget
10 description. Everything about your pets	
19 description: "Find out mona"	
10 unit "http://waggar.io"	
	Schemes
21 description: "Access to Petstore orders"	
	HTIPS V
23 description: "Operations about user"	
26 url: "http://swagger.io"	
	Det Everything about your Pets Eind out more: http://swagger.jo
	Find out more, http://www.second
	POST /pet Add a new pet to the store
34 - "pet"	PUT /pet Update an existing pet
35 summary: "Add a new pet to the store"	
36 description:	
37 operationid: addret	GET /pet/findByStatus Finds Pets by status
30 Consumes.	
40 - "annliation/yal"	
41 produces	
42	✓ Pot/±indByTags Finds Pets by tags
43 - "annlication/ison"	
44 parameters:	
45 - in: "body"	GET /pet/{petId} Find pet by ID
46 name: "body"	
47 description: "Pet object that needs to be added to the store"	
	POST /pet/{petId} Updates a pet in the store with form data
	Lool / Feedral observed observed observed and served and served and served observed observe

On the left side, you edit the API specification and on the right, you immediately see rendered API documentation/testing tool.

Let's load our first API spec into it (in <u>YAML</u>). It is a simple API with one GET request - returning random number in a specified range.

Math API Specification

Here's what it consists of.

Basic information about our API and used OAS version.

```
swagger: "2.0"
info:
   description: "Math"
   version: "1.0.0"
   title: "Math REST API"
```

Server host, protocol (http, https) and Web application names:

```
host: "localhost:52773"
basePath: "/math"
schemes:
    - http
```

Next we specify a path (so complete URL would be http://localhost:52773/math/random/:min/:max) and HTTP request method (get, post, put, delete):

```
paths:
   /random/{min}/{max}:
    get:
```

After that, we specify information about our request:

```
x-ISC CORS: true
summary: "Get random integer"
description: "Get random integer between min and max"
operationId: "getRandom"
produces:
- "application/json"
parameters:
- name: "min"
  in: "path"
  description: "Minimal Integer"
  required: true
  type: "integer"
  format: "int32"
- name: "max"
  in: "path"
  description: "Maximal Integer"
  required: true
  type: "integer"
  format: "int32"
responses:
  200:
    description: "OK"
```

In this part we define our request:

- Enable this path for CORS (more on that later)
- Provide summary and description
- operationId allows in-spec reference, also it's a generated method name in our implementation class
- produces response format (such as text, xml, json)
- parameters specify input parameters (be they in URL or body), in our case we specify 2 parameters range for our random number generator
- responses list possible responses form server

As you see this format is not particularly challenging, although there are many more features available, here's a <u>specification</u>.

Finally, let's export our definition as a JSON. Go To File Convert and save as JSON. The specification should look like this:

Math API Specification

## Load specification into IRIS

Now that we have our spec, we can generate boilerplate code for this REST API in InterSystems IRIS.

To move to this stage we'll need three things:

- REST Application name: package for our generated code (let's say math)
- OAS spec in a JSON format: we just created it in a previous step
- WEB Application name: a base path to access our REST API (/math in our case)

There are three ways to use our spec for code generation, they are essentially the same and just offer various ways to access the same functionality

- 1. Call ^%REST routine (Do ^%REST in an interactive terminal session), documentation.
- Call %REST class (Set sc = ##class(%REST.API).CreateApplication(applicationName, spec), noninteractive), <u>documentation</u>.
- 3. Use API Management REST API, documentation.

I think documentation adequately describes required steps so just choose one. I'll add two notes:

- In case (1) and (2) you can pass a dynamic object a filename or a URL
- In cases (2) and (3) you must make an additional call to create a WEB application: set sc =
   ##class(%SYS.REST).DeployApplication(restApp, webApp, authenticationType), so in our case set sc =
   ##class(%SYS.REST).DeployApplication("math", "/math"), get values for authenticationType argument from
   %sySecurity include file, relevant entries are \$\$\$Authe\*, so for unauthenticated access pass
   \$\$\$AutheUnauthenticated. If omitted, the parameter defaults to password authentication.

## What happened with our spec?

If you've created the app successfully, new math package should be created with three classes:

- Spec stores the specification as-is.
- Disp directly called when the REST service is invoked. It wraps REST handling and calls implementation methods.
- Impl holds the actual internal implementation of the REST service. You should edit only this class.

Documentation with more information about the classes.

#### Implementation

Initially our implementation class math.impl contains only one method, corresponding to our /random/{min}/{max} operation:

```
/// Get random integer between min and max<br/>/// The method arguments hold values for:<br/>/// min, Minimal Integer<br/>/// max, Maximal Integer<br/>ClassMethod getRandom(min As %Integer, max As %Integer) As %DynamicObject
{
    //(Place business logic here)
    //Do ..%SetStatusCode(<HTTP_status_code>)
    //Do ..%SetHeader(<name>,<value>)
    //Quit (Place response here) ; response may be a string, stream or dynamic object
}
```

Let's start with the trivial implementation:

```
ClassMethod getRandom(min As %Integer, max As %Integer) As %DynamicObject
{
    quit {"value":($random(max-min)+min)}
}
```

And finally we can call our REST API by opening this page in browser : http://localhost:52773/math/random/1/100

The output should be:

```
{
    "value": 45
}
```

Also in the Swagger editor pressing Try it out button and filling the request parameters would also send the same request:

#### Developing REST API with a spec-first approach Published on InterSystems Developer Community (https://community.intersystems.com)

💮 Swagger Editor. File + Edit + Generate Server + Generate Client +		
1 magger: "2.0" 2 info 3 decription: "Mab" 3 decription: "1.0 de	GET /random/{min}/{max} Get random integer	
s tills: Nuth BET #07" 6 host: localhost:5276" 7 bespecht: /wath	Get random integer between min and max	
a schnede: 9 - http: 10 paths: 11 //random/(min)/(max):	Parameters	Cancel
12 'get: 13 x-15C_CORS: true 14 summary: "Get random integer" 15 description: "Get random integer between min and max"	Name Description	
16 operationis" getBandom" 17 produces: 18 - "application/json" 19 parameters:	min * required Minimal Integer integer (\$int32) (path)	
20· - nær: "min" 21 in: "path" 22 description: "Minimal Integer" 23 reouined: true	max * required Maximal Integer Integer (Sint3)	
24 type: "integer" 25 format: "int32" 26 - name: "nam" 27 io: "path"	(path) 100	
28 description: "Maximal Integer" 29 required: true		01
30 type: 'integer' 31 format: "int2"	Execute	Clear
30         type: "integer"           31         format: "int2"           32:         response:           33:         200:           34         description: "0/"	Responses	Crear Response content type application/json
30 type: "integer" 31 format: "int22" 32 response: 33 description: "0k"	Execute Responses Curt	Clear Response content type application/json v
30 type: Tinteger 32 responsi 33 - 2001 34 description: "Out"	Execute Responses Curl curl -X GET "http://localhost:52776/math/random/1/100" -E "	Clear Response content type application/json v
30 type: "integer" 31 forunt: "int22 33 forunt: "int22 34 description: "0K"	Execute Responses Curl enrl -X GBT "http://locallost:52776/math/rendom/1/100" -H " RequestUBL	Clear Response content type application/json v
30 type: Tinteger 31 tespent: 32 tespent: 33 2000 34 description: "Out"	Execute Responses Curl eurl - X CET "http://localhost:52776/math/random/1/100" -E " Request URL http://localhost:52776/math/random/1/100 Server response	Lear Response content type application/json
39 type: Integer 31 trout: Int22 33 trout: Int22 34 description: OKT 34 description: OKT	Execute Responses Curl ourl -X CET "http://localhost:52776/math/readom/4/100" -R * RequestURL http://localhost:52776/math/readom/1/100 Server response Code Detais	Lear Response content type application/json v
30 type: Tinteger 31 type: Tinteger 32 type: 33 type: 34 description: "Out"	Execute Responses Curl curl curl curl curl curl curl ty://locallost:52776/math/random/1/100 thttp://locallost:52776/math/random/1/100 thttp://locallost:527	Lear Response content type application/json

Congratulations! Our first REST API created with a spec-first approach is now live!

## Further development

Of course, our API is not static and we need to add new paths and so on. With spec-first development, you start with modifying the specification, then updating the REST application (same calls as for creating the application) and finally writing the code. Note that spec updates are safe: your code is not affected, even if the path is removed from a spec, in implementation class the method would not be deleted.

## Considerations

More notes!

#### Special parameters

InterSystems added special parameters to swagger specification, here they are:

Name	Datatype	Default	Place	Description
x-ISC <u>D</u> ispatchParent	classname	%CSP.REST	info	Superclass for dispatch class.
x-ISC <u>C</u> ORS	boolean	false	operation	Flag to indicate that CORS requests for this endpoint /method combination should be supported.
x-ISC <u>R</u> equiredResource	array		operation	Comma- separated

			list of
			defined
			resources
			and their
			access
			modes (reso
			urce:mode)
			that are
			required for
			access to
			this endpoint
			of the REST
			service.
			Example: ["
			%Developm
			ent:USE"]
x-ISC <u>S</u> erviceMethod	string	operation	Name of the
			class
			method
			called on the
			back end to
			service this
			operation;
			default is
			operationId,
			which is
			normally
			suitable.

## CORS

There are three ways to enable CORS support.

- 1. On a route by route basis by specifying x-ISCCORS as true. That's what we have done in our Math REST API.
- 2. On per API basis by adding

Parameter HandleCorsRequest = 1;

and recompiling the class. It would also survive spec update.

3. (Recommended) On per API basis by implementing custom dispatcher superclass (it should extend %CSP.REST) and writing CORS processing logic there. To use this superclass add x-ISCDispatchParent to your specification.

## Load spec into IAM

Finally, let's add our spec into IAM so it would be published for other Developers.

If you have not started with IAM, check out this article. It also covers offering REST API via IAM so I'm not describing it here. You might want to modify spec host and basepath parameters so that they point to IAM, rather than the InterSystems IRIS instance.

Open the IAM Administrator portal and go to the Specs tab in the relevant workspace.

L Ir	I Management	Workspaces	Dev Portals	Vitals	Organization					í
88	Change Workspace									
DE	default	de	efault Dev	Portal	specs					
≣	<b>Dev Portal</b> Overview	n	ame		type	created_at		+ Ad	id Spec	
	Settings Developers	v	itals		spec	10/15/2019 - 10:10:00	Edit	Download	1	
	Pages Partials	fi	iles		spec	10/15/2019 - 10:10:00	Edit	Download	1	
	Specs	a	Idmin		spec	10/15/2019 - 10:10:00	Edit	Download	1	

Click the Add Spec button and input the name of the new API (math in our case). After creating new Spec in IAM click Edit and paste the spec code (JSON or YAML - it doesn't matter for IAM):

le Name	File Type	
unauthenticated / math	Spec	
1 swagger: "2.0" 2 info: 3 description: "Math"	Require Authentication     Files that require Authentication and	e only shown to logged-in developers.
<pre>4 version: "1.0.0" 5 title: "Math REST API" 6 host: "localhost:8000"</pre>	Files that are Unauthenticated will	be public.
7 basePath: "/math" 8 tags: 9 - name: "math"	Delete File	
description: "Math functions" 11 schemes: 12 - "http"		
13 paths: 14 /random/(min)/(max): 15 get:		
16         x-ISC_CORS: true           17         tags:		

Don't forget to click Update File.

Now our API is published for Developers. Open Developer Portal and click Documentation in the upper right corner. In addition to the three default APIs our new Math REST API should be available:

#### Developing REST API with a spec-first approach Published on InterSystems Developer Community (https://community.intersystems.com)

	ystems <sup>.</sup>					Documentation About	Guides Q
	API Catalog					Search	
	Vitals API Vitals API		Developer Portal Files API Developer Portal API		IAM-Admin A Swagger definition for the IAM	I Admin API	
	version: 1.0.0	nodes, cluster	version: 1.0	files	version: 13.1.0	Status, Services, Routes (3 more)	
[	Math REST API Math						
	version: 1.0.0	math					

#### Open it:

				Documentation	About Gui
GETTING STARTED Introduction RESOURCES • Math	[ Base (THL: localhost:8000/math ] Math Schemes http				
	math <sup>Math functions</sup> Get random integer		Request		
	GET /random/{min}/{max}		curl -X GET "h application/js	ttp://localhost:8000/math/random/1/100" -H "accep on"	
	Parameters	Cano	el Request URL	st:8000/math/random/1/100	
	min * Required integer (\$int64) (psth)	Minimal ID 1	Server Response 200 – Undocume	nted	
	max * Required integer (Sint64)	Maximal ID 100	Response body		
	(path)	Make API Call Clear	( "value": 11 )		Download

Now Developers can see the documentation for our new API and try it at the same place!

## Conclusion

InterSystems IRIS simplifies the development process for a REST API and the spec-first approach allows faster and easier REST API life cycle management. With this approach, you can use a variety of tools for a variety of related tasks, such as client generation, unit testing, API Management, and many others.

## Links

OpenAPI 3.0 Specification

- Creating REST Services
- Starting with IAM
- IAM Documentation

<u>#API</u> <u>#Best Practices</u> <u>#InterSystems API Manager (IAM)</u> <u>#REST API</u> <u>#InterSystems IRIS</u>

Source URL: https://community.intersystems.com/post/developing-rest-api-spec-first-approach