Article
Eduard Lebedyuk · Mar 14, 2018 10m read

REST Design and Development

Intro

For many in today's interoperability landscape, REST reigns supreme. With the overabundance of tools and approaches to REST API development, what tools do you choose and what do you need to plan for before writing any code?
This article focuses on design patterns and considerations that allow you to build highly robust, adaptive, and consistent REST APIs. Viable approaches to challenges of CORS support and authentication management will be discussed, along with various tips and tricks and best tools for all stages of REST API development. Learn about the open-source REST APIs available for InterSystems IRIS Data Platform and how they tackle the challenge of ever-increasing API complexity.
The article is a write-up for a recent webinar on the same topic.

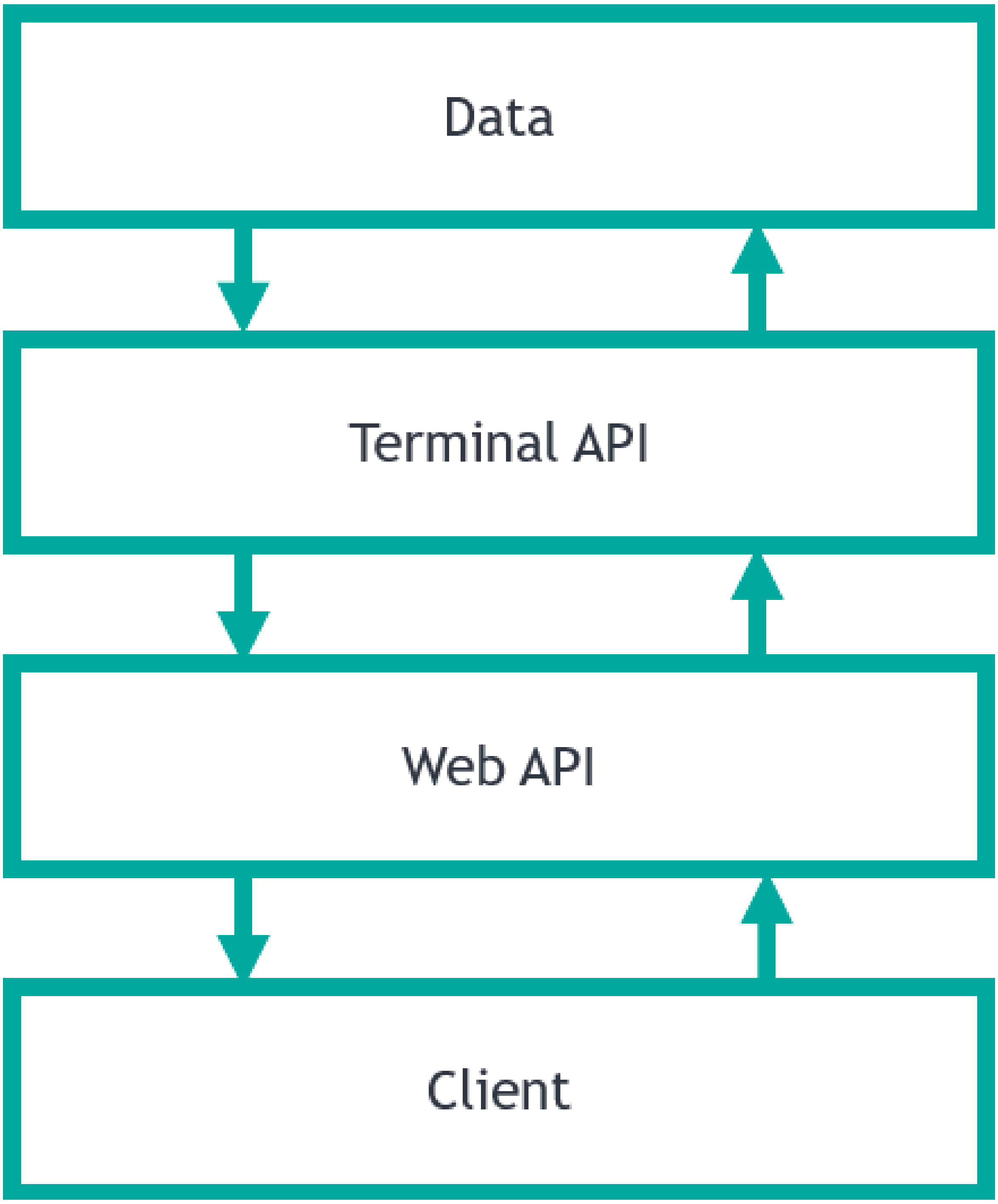
2020 Note

A year and a half passed since I wrote this article. I think it's still relevant but I'd like to showcase some exciting new features that make developing REST APIs easier than ever:
- API Management - manage API lifecycle.
- %JSON.Adaptor - simplify JSON<->Object conversion.
- Spec-first REST API development - effective way to develop REST API.

Contents

REST API architecture

I hope that the reader is familiar with the REST API and the basics of its implementation using InterSystems technologies. Many articles, webinars, and training courses are devoted to this topic. In any case, the first thing I would like to start with is a common high-level architecture of your application, which includes the REST API. It can look like this:



Here we can see the main layers of the application:

Data

Persistent data, such as classes.

Terminal API

This is a layer that manages the data. All interaction with data should be done through this API. Methods of this API do not write to the device, they only return one of:

- Object
- Status or exception

Web API

Converts an incoming request into a form understood by terminal API and calls it. The result returned by the terminal API is converted into client-readable form (usually JSON) returned to the client. Web API does not interact with data directly. I use the term Web API and not the REST API because Web API can be implemented using a different architecture, for example, based on the WebSocket protocol, which is also supported by the InterSystems platform.

Client

Usually written in JavaScript (but not necessarily) this is the application that is available to the end user. This level should not interact directly with the database, implement business logic or store the state of the application. At this level, only the simplest business logic is usually available: interfaces, preliminary input validation, simple data operations (sorting, grouping, aggregation), etc.

This separation removes complexity from your application, facilitates its debugging. This approach is called a multitier architecture.

Brokers

Brokers are classes which contain your REST API logic. They:

- Are subclasses of %CSP.REST
- Contain routes which match URLs to methods in your code

For example, we have this route:

<Route Url="/path/:param1/:param2" Method="GET" Call="Package.Class:ClassMethod" />

If the client sends GET request to /path/10/20 the ClassMethod of Package.Class would be called with arguments 10 and 20.

Brokers separation

«Physical» separation

Since there can be many routes in a REST API, one broker would soon be overloaded with methods. To prevent this, divide the brokers into several classes as follows:

Where:

- Abstract broker converts requests, checks CORS requests and performs other technical tasks unrelated to REST API logic.
- Brokers 1..N contain methods that process requests by calling terminal API methods and returning terminal API responses to the client.

«Logical» separation and versioning

Usually, brokers contain routes matching URLs to called methods, but they also contain Maps which pass requests from one broker to another. Here's how that works:

<Map Prefix="/form" Forward="Destination.Broker"/>

In this example, all requests starting from /form are passed to Destination.Broker. It makes broker separation

possible.
Brokers should be separated by:
- Version
- Domain

Broker parameters

Here are the recomended parameters for REST brokers:

```
Parameter CONTENTTYPE = {..#CONTENTTYPEJSON};
Parameter CHARSET = "UTF-8";
Parameter UseSession As BOOLEAN = 1;
```

Where

- CONTENTTYPE — adds information about response being JSON.
- CHARSET — converts the response into UTF8.
- UseSession — use sessions to track users, more on that in Authentication.

Since all brokers are inherited from one abstract broker, it is sufficient to specify these parameters once in the abstract broker (remember, that only parameters of the primary superclass are inherited).

Code generalization

One of the problems that often arise when developing a REST API is copy-pasting code from one broker to another, which begets a large number of routes, and copied code and is a bad practice. To avoid this, I recommend using code generation with the %Dictionary package which contains all meta information that you may need for method generators. An example of a large scale use of code generation is RESTForms library. Finally, the use of code generation allows you to achieve greater uniformity of your REST API.

CORS

Cross-origin resource sharing (CORS) is a mechanism that allows restricted resources (e.g. fonts) on a web page to be requested from another domain outside the domain from which the first resource was served. To enable access to your REST API using CORS, add the following parameter to the broker:

```
Parameter HandleCorsRequest = 1;
```

This would allow access to REST API by pages from any other domain. This is unsafe and therefore you must also override the OnHandleCorsRequest method, its signature:

```
ClassMethod OnHandleCorsRequest(pUrl As %String) As %Status {}
```

This method should check request origin and allow access only from whitelisted origins. Here's a method to get origin:

```
ClassMethod GetOrigins() As %String
{
    set url = %request.GetCgiEnv("HTTP_REFERER")
    return $p(url,"/",1,3) // get http(s)://origin.com:port
}
```

Authentication

In order for one user to consume one license and work within one session, you need to configure your REST and CSP applications in the System Management Portal so that the following conditions are met:

1. All brokers effectively have Parameter UseSession = 1;
2. All web applications allow only authenticated (i.e. password) access.
3. All web applications have reasonable Session timeout (i.e. 900, 3600).
4. All web applications have the same GroupById value.
5. All web applications have the same cookie path.

Development Tools

For debugging, you can use external tools like:

- REST client (Postman) - is a primary debugging tool for a REST API developer. It allows saving and documenting requests, prettification of JSON responses, sending requests into several environments, and provides many other tools for a developer.
- Intercepting proxy server (Fiddler) - allows a developer to intercept, display, edit and duplicate requests.
- Packet analyzer (Wireshark) - helps in the cases of broken packets, encoding problems, analysis of the remote headless system and other situations when the above-mentioned tools are not enough.

I strongly recommend against using command line tools such as curl and wget.
Also, I have written a series of articles (Part 1 - external tools, Part 2 - internal tools) about various approaches to REST API (and not only REST API) debugging.

Examples

Some examples of working REST APIs.

InterSystems IRIS provides several REST APIs in %API package

- Atelier
- API management
- InterSystems: DocDB
- InterSystems: BI
- InterSystems: Text Analytics
- InterSystems: UIMA

Additionally documentation has a tutorial on REST.
There are several courses on REST on Learning.InterSystems.com.

And there are some Open-sourced REST APIs available on GitHub.

InterSystems IRIS: BI - MDX2JSON

MDX2JSON — REST API for MDX2JSON transformation. Also provides metainformation about Cubes, Dashboards, Widgets and other elements. Available since 2014.1+. Client is a web-based renderer for InterSystems IRIS: BI Dashboards built with AngularJS and highcharts. Here's how it looks:

InterSystems IRIS: Interoperability Workflow

Workflow is a module that allows users to participate in the execution of automated business processes. Workflow REST API exposes users' tasks. Available since 2014.1+. Web client provides visualization:

RESTForms

RESTForms facilitates the creation of new REST APIs by providing a robust self-discovery generic REST API solution. Available since 2016.1+. I have written about RESTForms in detail: part 1, part 2.

Some routes available in RESTForms:

Method	URL	Descriptions
GET	form/info	List all RESTForms enabled classes
GET	form/info/all	Get metainformation about all available classes
GET	form/info/:class	Get metainformation about one available classes
GET	form/object/:class/:id	Get object by id and class
GET	form/object/:class/:id/:property	Get object property by id, class and property name
POST	form/object/:class	Create object
PUT	form/object/:class/:id	Update object via dynamic object
PUT	form/object/:class	Update object via classic object
DELETE	form/object/:class/:id	Delete object
GET	form/objects/:class/:query	Execute SQL query

This project actively uses code generalization (using code generation and %Dictionary package).
Clients are available on Angular and React, and they look like this:

JSON

JavaScript Object Notation is text serialization format often used in REST API. JSON support in InterSystems products was available since version 2009.2, but have seen considerable improvements in version 2016.2. Documentation has a whole book about JSON.

Conclusions

- REST is one of the most popular and widely accepted API technologies
- If you are providing a REST API, you can choose among several client technologies, including but not limited to:
    - JavaScript (AngularJS, React, ExtJS, ...)
    - Mobile apps (Cordova and similar technologies or native apps)
- InterSystems technologies allow you to create REST APIs with ease

Links

- Documentation
- REST Webinar
- Debugging web: part 1, part 2
- RESTForms: part 1, part 2
- Community tutorial

#API #Best Practices #Open Source #REST API #Caché #InterSystems IRIS

Source URL:https://community.intersystems.com/post/rest-design-and-development