CPPServer

Dev-platform for high performance JSON microservices and modern Apps with C++

Contact

https://cppserver.com

Phone:
+58 (424) 2686639

Email:
cppserver@martincordova.com

Created by:
Martín Córdova y Asociados, C.A.
Caracas - Venezuela

JSON response structure

Describes the general structure of the JSON response generated by ANY microservice, including cases of success, data validation error or system error. The general structure pretends to serve well front-end clients (JavaScript) as well as other processes, like web server running Java or NodeJS and invoking microservices when using CPPServer as a service hub.

Objective

To make it easy for CPPServer clients to find out if the invocation was a success or not, and if it's not then to make it easy to discern the type of error, either a system error (database error) or a parameter input validation that failed, whatever the case, there will be enough information in the response for the client to proceed, like highlight a form's field to show a validation error, or present some specific business-related message to the user. The general structure cannot be changed, but additional fields are accepted to serve the specific client software needs. The supplied CPPServer web frontend framework depends on this JSON response specification.

Whatever the client, GUI or a process, the JSON returned by CPPServer must be complete and clear enough for both, without ambiguity and adaptable to very specific business purposes.

General structure

```
{
    "status": "OK|ERROR|INVALID",
    "description" : "",
    "custom-field": "",
    "validation": {
        "id": "",
        "description": ""
    },
    "data": [{}],
    "data-composite": {
      "data1": [{}],
      "data2": [{}]
    }
}
```

status definition:

OK	No error, proceed to read “obj.data” array or any other data structure.
ERROR	A system error, typically it will be a database error, or a backend error if a microservice is consuming a remote service. obj.description will contain a general error message; the specific details of the error are stored in the CPPServer logs (stderr.log) at the server-side. There may be an arbitrary number of additional fields in the same top-level object related to the error status, so that a front-end may have enough data to present more specific messages to the user, or a process consuming the microservice may take action based on these fields.
INVALID	An input parameter (GET/POST) failed the validation requirements (data type, not null, some custom rule), obj.validation.id will contain the form’s field ID, and obj.validation.description the message to be displayed, also happens if the execution of the microservice was denied by the security layer because the user does not belong to the authorized roles, obj.validation.id will be “_dialog_” and the description field will contain the placeholder “$err.accessdenied” that should be translated by the frontend to the proper localized message.

HTTP status code will be 200 regardless of the JSON status field value.

JavaScript response handler

This is a basic generic code template to manage the response of the CPPServer after the Ajax request returned, it could be the result of a GET or a POST HTTP command:

```
//inpsect server response
function responseHandler(jsonResp) {
    if (jsonResp!=null) {
        var resp = JSON.parse(jsonResp);
        if (resp.status=="OK") {
            //proceed to bind the form's widgets to the data arrays
        } else if (resp.status=="INVALID") {
            //resp.validation.id resp.validation.description
            //contain the necessary data for the web framework
        } else if (resp.status="ERROR") {
            // resp.description and any other custom-field in the resp object
        } else {
            alert("Invalid status code in JSON response");
        }
    }
}
```

Typically, this function would be called (because it was passed as a callback) by the Ajax request handler, another generic function of the front-end web framework, and the JSON response will be passed to this function to be parsed using JSON.parse().

JSON Examples and considerations

If the status is OK then there is no need to include the rest of the fields except for the data, the smaller the response the better.

```
{
    "status": "OK",
    "description": "",
    "data": [{
        "customerid": "VAFFE",
        "orderid": 10399,
        "orderdate": "1995-01-31",
        "total": 1765.59
    }]
}
```

```
{
    "status": "OK",
    "data": {
        "customer": [{
            "customerid": "ALFKI",
            "contactname": "Maria Anders",
            "companyname": "Alfreds Futterkiste",
            "phone": "030-0074321",
            "country": "Germany"
        }],
        "orders": [{
            "customerid": "ALFKI",
            "orderid": 10692,
            "orderdate": "1995-11-03",
            "total": 878
        }, {
            "customerid": "ALFKI",
            "orderid": 10702,
            "orderdate": "1995-11-13",
            "total": 330
        }, {
            "customerid": "ALFKI",
            "orderid": 10835,
            "orderdate": "1996-02-15",
            "total": 845.79999999999995
        }, {
            "customerid": "ALFKI",
            "orderid": 10952,
            "orderdate": "1996-04-15",
            "total": 471.19999999999999
        }, {
            "customerid": "ALFKI",
            "orderid": 11011,
            "orderdate": "1996-05-09",
            "total": 933.5
        }]
    }
}
```


The example above shows a multi-array response, much like master-detail but there is no need for the arrays to be nested in this case, just

one after the other, with easy access from JavaScript once the JSON has been parsed, much like obj.data.customer and obj.data.orders, each element of the array is an object, a very easy to navigate JSON data structure that is also very convenient for feeding advanced GUI widgets like grids, charts and forms. By the way, one line of C++ code from CPPServer's dblib module generates this JSON output in microseconds (if the database is fast, as it should be).

```
{
    "status": "INVALID",
    "validation": {
        "id": "custid",
        "description": "Este valor DEBE ser ingresado"
    }
}
```

If a request parameter, either from the query string or from the form's fields, fails to comply with the input validation rules, a message like the one above will be returned, it's not required to include the other fields, only those relevant to the validation error, the basic fields are the ID of the form's control corresponding to the parameter and the text to be displayed to the user. This is also useful for other processes calling microservices, this way they can properly identify why the request payload they are sending to the microservice is failing.

```
{
    "status": "ERROR",
    "description" : "Database system error"
}
```

In case of a server-side database I/O error, a microservice will return this JSON, no technical details will be provided because that information gets stored in the CPPServer log file STDERR.log. It is possible to add more fields if necessary, to tailor it to more complex functional requirements.

Changelog for this document:

2021-12-27: Creation