Article
Rubens Silva · Oct 8, 2019 2m read

Open Exchange

Frontier: An abstraction layer for rapid REST development: Part 5 - Errors

Hello again and welcome to the next tutorial on this series: **Part 5 - Errors.** Here we are going to learn how *Frontier* handles unexpected errors and how we can force them.

1. Core concepts
    - Getting started
    - Creating a simple request
    - Query parameters
    - Aliasing query parameters
    - Changing output format
    - Rest query parameters
    - Inferring object instances
    - Using literal notation
    - Seamlessly mixing instances with literals
    - Returning streams
2. Handling payloads
    - How it works
    - Making it useful
    - Unmarshalling payloads into instances
    - Using the unmarshaller to EDIT an existing object
3. Using the SQL API
    - Creating a simple dynamic query
    - Overwriting the default container property
    - Using cached queries
    - Passing parameters to queries
4. Sharing data across router methods
5. Errors
6. Managing errors with Reporters

5. Errors

Most of abnormal errors will be captured and handled by the engine automatically, just like the example below:

```
ClassMethod ForceError() As %DynamicObject
{
  // This is undefined so it's expected to throw.
  return whoops
}
```

would return this:

```
{
  "error": {
    "internalCode": "",
```

```
    "message": "Erro Caché: <UNDEFINED>zForceError+1^Frontier.Tutorial.Router.1 *whoo
ps"
  },
  "responseCode": 500
}
```

However there can have situations where some business rule must force an exception to happen, this can be accomplished by using the contextual method ThrowException, which takes at least one argument, the error message itself:

```
ClassMethod ForceError() As %DynamicObject
{
  return %frontier.ThrowException("Something happened.")
}
```

Which would return:

```
{
  "error": {
    "internalCode": 5001,
    "message": "Something happened."
  },
  "responseCode": 500
}
```

ThrowException always use status code 5001 (GeneralError) and also assume the HTTP Status **500 Internal Error**, but you can at least change the HTTP status by providing the second argument using one of the HTTP status parameters defined on %CSP.REST.

```
ClassMethod ForceError() As %DynamicObject
{
  return %frontier.ThrowException("Something happened.", ..#HTTP404NOTFOUND)
}
```

This will return:

```
{
  "error": {
    "internalCode": 5001,
    "message": "Something happened."
  },
  "responseCode": 404
}
```

Notice that the responseCode is now 404 instead of 500. But not only that, the browser will also receive a 404 http status.

> NOTE: Remember that statuses (status codes) must always be thrown instead of returned, otherwise *Frontier* will consider the value a response to be serialized, which is not ideal since a %Status is composed

by special characters.

In the next tutorial we'll see how to set up reporters to notify about impeding errors.

#JSON #REST API #Caché #InterSystems IRIS
Check the related application on InterSystems Open Exchange