Coding Challenge: Calculator Service

1. Introduction

As part of our hiring process, we ask candidates to develop some kind of minimal application in order to show off their skills & abilities. As such, this document serves as technical specification of a (simple) HTTP/REST calculator service, describing requirements, use cases, and additional details which should be taken into account by candidates.

1.1. Goals & Objectives

Implement an HTTP/REST-based 'Calculator Service' capable of some basic arithmetic operations, like add, subtract, square, etc. along with a history service keeping track of requests sharing a common an identifier.

1.2. General Requirements

1. This project must be implemented in C#, targeting Net 4+ or .Net Core.
2. CalculatorService must use an HTTP FrameWork, like ASP.MVC, ASP/WebAPI, NancyFX or ServiceStack.
3. CalculatorService should produce diagnostics/execution logs as rotating (ie. daily) files, including enough information to be of use while diagnosing service issues.
4. Building and deploying the solution or applications should be as easy as possible.
5. External dependencies (like databases, non-managed components, etc.) should be avoided as much as possible, in order to make reviewing, deploying and testing easier.
    - Obviously, this does not apply to .Net managed libraries and NuGets: Feel free to use whatever makes your task easier from within the .Net ecosystem.
6. With regard to the HTTP/REST API, at least JSON encoding must be supported as input/output encoding. Additional encodings are welcome.
7. Implementing unit, integration or any other form of automated testing, while not a must, will be a convenient addition.
8. The source code must be published in a public Github repository.
9. This source code must have a README file with all the related information about the application:
    - How to build the application.
    - How to deploy the application.
    - How to run the resulting application.
    - Environment requirements and/or details related to using/reviewing the application.
    - Etc.
10. Other information or documentation, in addition to that contained within README file, will be a plus. Including:
    - Code comments and notes which may help us understand you'r reasoning and decision taking while developing.
    - General notes, documents or diagrams, describing internal implementation details you may think are of importance during review.

All in all, this is an exercise designed in order to allow you to demonstrate your potential and skills. As such, we encourage you to try to provide us with clean, well organized code, easy to review, showing off your talent. Also, even if this project is an small one, which may no directly benefit from it, do not hesitate on implementing coding patterns and/or using development techniques which may help illustrating your experience, and let us know about your reasoning using code comments or any other way.

1.3. Architecture & Components

This should be the main components of the delivered application:

- **CalculatorService.Server**: The main component of the application, and the one actually implementing the 'calculator service' HTTP/REST interface & business logic.
- **CalculatorService.Client**: A demonstration console/command-line client, capable of performing requests to the main HTTP service from CLI.

2. High-Level Use Cases

Description of the high-level use cases, which CalculatorService should implement.

2.1. UC-CALC-ADD: Addition of two or more numeric operands

An user wants us to compute the addition of two or more operands. In order to do so, a request against CalculatorService.Server will be made, expecting to get back the computed result as response.

2.1.1. Actors

- User: The end user interacting with the application.
- Client: The HTTP/REST client making the request on behalf of a user.
- Server: CalculatorService's server component, in charge of computing the actual arithmetic operation.

2.1.2. Pre-conditions

1. Communication between client and server happens over an HTTP/REST interface exposed by *Server*.

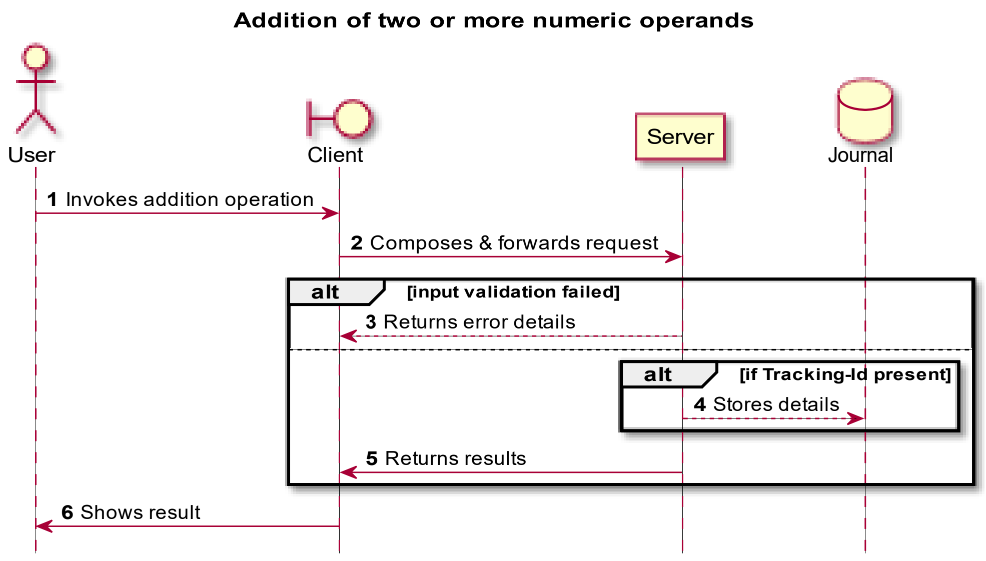
2.1.3. Main scenario

1. The *user* invokes an addition operation by using the *client*.
2. The *client* formats an HTTP/REST request and submits it to *server* for processing, waiting no more than 5 seconds for a reply.
   - The request should include at least two numeric operands to add.
   - The *client* may submit, as part of the request, an 'X-Evi-Tracking-Id' header, with a *user*-defined value.
3. The *server* validates input arguments/operands.
   1. The *server* returns an invalid request/arguments response, if/when input validation fails.
4. The *server* computes the requested arithmetic addition of received operands.
   - If a 'Tracking-Id' was provided, the *server* should store this request's details inside it's journal, indexed by the given Id.
5. The *server* returns the results of the the requested operation back to *client*.

2.1.4. Post-conditions

1. If valid 'X-Evi-Tracking-Id' was provided, this request's details has been stored at the *server*'s internal request journal.
   - Care should be taken of concurrent modifications (by two or more clients) of such 'request journal'

2.1.5. Sequence diagram



2.2. UC-CALC-SUB: Subtraction of two or more numeric operands

An user wants us to compute the subtraction of two or more operands. In order to do so, a request against CalculatorService.Server will be made, expecting to get back the computed result as response.

2.2.1. Actors

- User: The end user interacting with the application.
- Client: The HTTP/REST client making the request on behalf of a user.
- Server: CalculatorService's server component, in charge of computing the actual arithmetic operation.

2.2.2. Pre-conditions

1. Communication between client and server happens over an HTTP/REST interface exposed by *Server*.

2.2.3. Main scenario

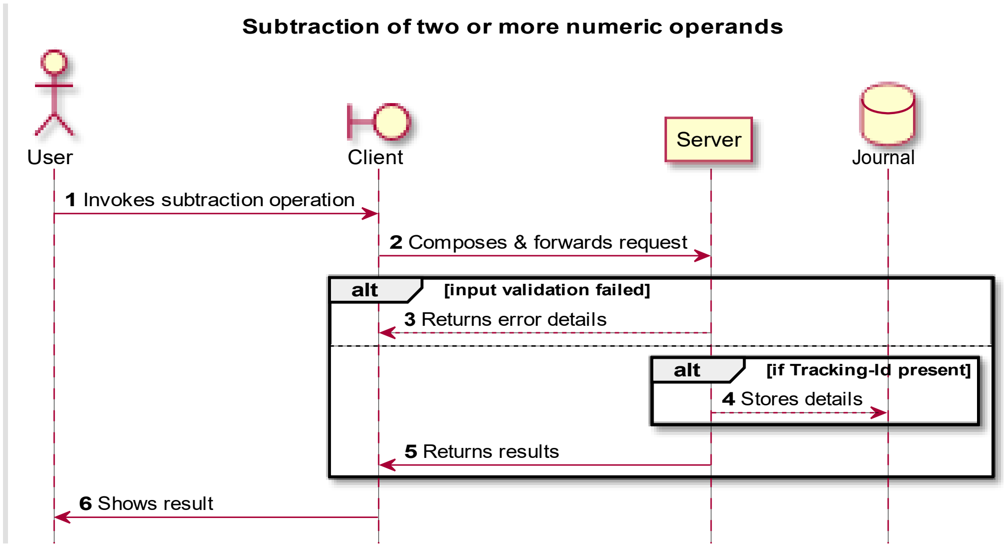
1. The *user* invokes an subtraction operation by using the *client*.

2. The *client* formats an HTTP/REST request and submits it to *server* for processing, waiting no more than 5 seconds for a reply.
   - The request should include at least two numeric operands to subtract from each other.
   - The *client* may submit, as part of the request, an 'X-Evi-Tracking-Id' header, with a *user*-defined value.
3. The *server* validates input arguments/operands.
   1. The *server* returns an invalid request/arguments response, if/when input validation fails.
4. The *server* computes the requested arithmetic subtraction against received operands.
   - If a 'Tracking-Id' was provided, the *server* should store this request's details inside it's journal, indexed by the given Id.
5. The *server* returns the results of the the requested operation back to *client*.

2.2.4. Post-conditions

1. If valid 'X-Evi-Tracking-Id' was provided, this request's details has been stored at the *server*'s internal request journal.
   - Care should be taken of concurrent modifications (by two or more clients) of such 'request journal'

2.2.5. Sequence diagram


Subtraction of two or more numeric operands
User
Client
Server
Journal
1 Invokes subtraction operation
2 Composes & forwards request
alt [input validation failed]
3 Returns error details
alt [if Tracking-Id present]
4 Stores details
5 Returns results
6 Shows result

2.3. UC-CALC-MUL: Multiply of two or more numeric operands

An user wants us to compute the multiplication of two or more operands. In order to do so, a request against CalculatorService.Server will be made, expecting to get back the computed result as response.

2.3.1. Actors

- User: The end user interacting with the application.
- Client: The HTTP/REST client making the request on behalf of a user.
- Server: CalculatorService's server component, in charge of computing the actual arithmetic operation.

2.3.2. Pre-conditions

1. Communication between client and server happens over an HTTP/REST interface exposed by *Server*.

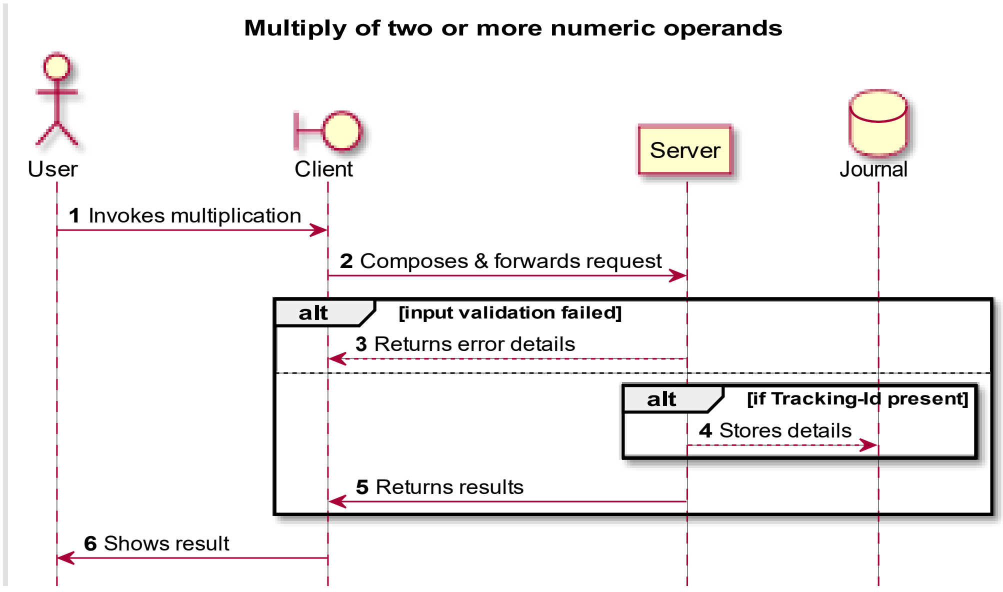
2.3.3. Main scenario

1. The *user* invokes an multiply operation by using the *client*.
2. The *client* formats an HTTP/REST request and submits it to *server* for processing, waiting no more than 5 seconds for a reply.
   - The request should include at least two numeric operands to multiply.
   - The *client* may submit, as part of the request, an 'X-Evi-Tracking-Id' header, with a *user*-defined value.
3. The *server* validates input arguments/operands.
   1. The *server* returns an invalid request/arguments response, if/when input validation fails.
4. The *server* computes the requested arithmetic multiplication of received operands.
   - If a 'Tracking-Id' was provided, the *server* should store this request's details inside it's journal, indexed by the given Id.
5. The *server* returns the results of the the requested operation back to *client*.

2.3.4. Post-conditions

1. If valid 'X-Evi-Tracking-Id' was provided, this request's details has been stored at the *server*'s internal request journal.
   - Care should be taken of concurrent modifications (by two or more clients) of such 'request journal'

2.3.5. Sequence diagram


Multiply of two or more numeric operands
User
Client
Server
Journal
1 Invokes multiplication
2 Composes & forwards request
alt [input validation failed]
3 Returns error details
alt [if Tracking-Id present]
4 Stores details
5 Returns results
6 Shows result

2.4. UC-CALC-DIV: Division of two or more numeric operands

An user wants us to compute the division of two or more operands. In order to do so, a request against CalculatorService.Server will be made, expecting to get back the computed result as response.

2.4.1. Actors

- User: The end user interacting with the application.
- Client: The HTTP/REST client making the request on behalf of a user.
- Server: CalculatorService's server component, in charge of computing the actual arithmetic operation.

2.4.2. Pre-conditions

1. Communication between client and server happens over an HTTP/REST interface exposed by *Server*.

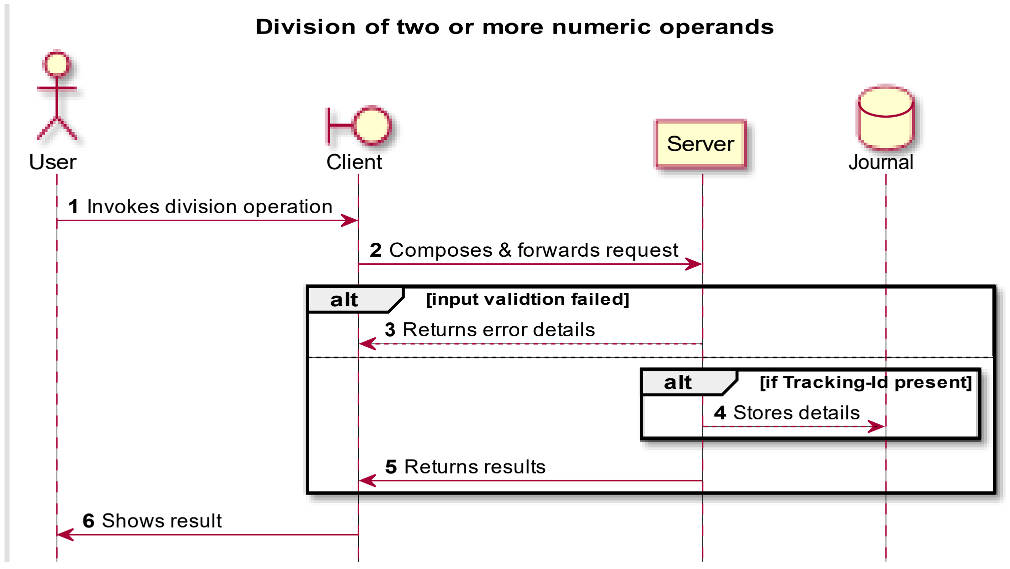
2.4.3. Main scenario

1. The *user* invokes a division operation by using the *client*.
2. The *client* formats an HTTP/REST request and submits it to *server* for processing, waiting no more than 5 seconds for a reply.
    - The request should include at least two numeric operands to multiply.
    - The *client* may submit, as part of the request, an 'X-Evi-Tracking-Id' header, with a *user*-defined value.
3. The *server* validates input arguments/operands.
    1. The *server* returns an invalid request/arguments response, if/when input validation fails.
4. The *server* computes the requested arithmetic division of received operands.
    - If a 'Tracking-Id' was provided, the *server* should store this request's details inside it's journal, indexed by the given Id.
5. The *server* returns the results of the the requested operation back to *client*.

2.4.4. Post-conditions

1. If valid 'X-Evi-Tracking-Id' was provided, this request's details has been stored at the *server*'s internal request journal.
    - Care should be taken of concurrent modifications (by two or more clients) of such 'request journal'

2.4.5. Sequence diagram


Division of two or more numeric operands
User
Client
Server
Journal
1 Invokes division operation
2 Composes & forwards request
alt [input validtion failed]
3 Returns error details
alt [if Tracking-Id present]
4 Stores details
5 Returns results
6 Shows result

2.5. UC-CALC-SQRT: Square root of a numeric operand

An user wants us to compute the squre root of an operand. In order to do so, a request against CalculatorService.Server will be made, expecting to get back the computed result as response.

2.5.1. Actors

- User: The end user interacting with the application.
- Client: The HTTP/REST client making the request on behalf of a user.
- Server: CalculatorService’s server component, in charge of computing the actual arithmetic operation.

2.5.2. Pre-conditions

1. Communication between client and server happens over an HTTP/REST interface exposed by *Server*.

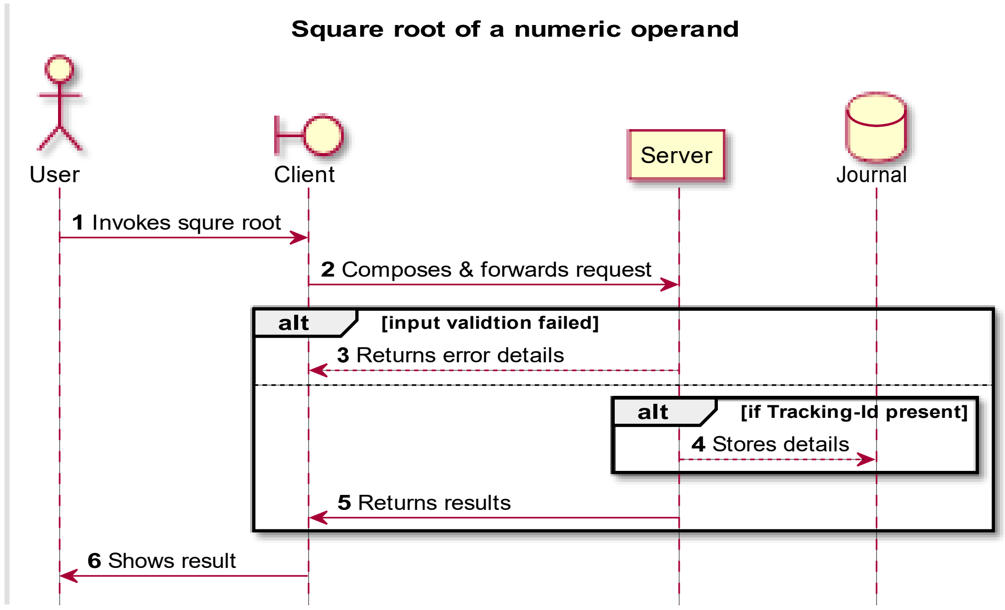
2.5.3. Main scenario

1. The *user* invokes a square root operation by using the *client*.
2. The *client* formats an HTTP/REST request and submits it to *server* for processing, waiting no more than 5 seconds for a reply.
    - The request should include exactly one numeric operand.
    - The *client* may submit, as part of the request, an 'X-Evi-Tracking-Id’ header, with a *user*-defined value.
3. The *server* validates input arguments/operands.
    1. The *server* returns an invalid request/arguments response, if/when input validation fails.
4. The *server* computes the requested arithmetic square root of received operand.
    - If a 'Tracking-Id’ was provided, the *server* should store this request’s details inside it’s journal, indexed by the given Id.
5. The *server* returns the results of the the requested operation back to *client*.

2.5.4. Post-conditions

1. If valid 'X-Evi-Tracking-Id’ was provided, this request’s details has been stored at the *server*'s internal request journal.
    - Care should be taken of concurrent modifications (by two or more clients) of such 'request journal’

2.5.5. Sequence diagram



2.6. UC-JOURNAL-QUERY: Query journal entries by Tracking-Id

An user wants to review it's history of requested operations (issued with a common Tracking-Id) since the last application restart. In order to do so, a request against CalculatorService.Server will be made specifying the Tracking-Id value, and a list of all operations processed with such Id since the last application restart should be returned.

2.6.1. Actors

- User: The end user interacting with the application.
- Client: The HTTP/REST client making the request on behalf of a user.
- Server: CalculatorService's server component, in charge of computing the actual arithmetic operation.

2.6.2. Pre-conditions

1. Communication between client and server happens over an HTTP/REST interface exposed by *Server*.
2. Details regarding all *user* requests providing a 'X-Evi-Tracking-Id' header, are available at *server*'s internal journal store.
    - This journal store does not need to be a durable one. Only requests made since the last application restart are expected to be available.

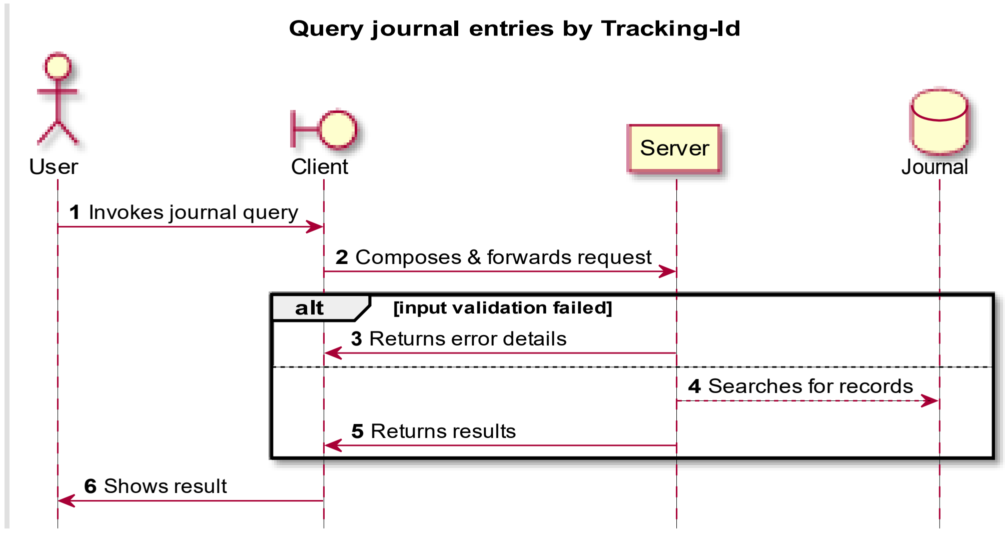
2.6.3. Main scenario

1. The *user* invokes a get/query journal operation by using the *client*.
2. The *client* formats an HTTP/REST request and submits it to *server* for processing, waiting no more than 5 seconds for a reply.
    - The request should include a 'Tracking-Id' to use when querying journal.
3. The *server* validates input arguments/operands.
    1. The *server* returns an invalid request/arguments response, if/when input validation fails.
4. The *server* searches journal for all records pertaining to 'Tracking-Id'.
5. The *server* returns the results of the previous search back to *client*.

2.6.4. Post-conditions

N/A

2.6.5. Sequence diagram



3. REST/API description

External Application Programming Interface (in HTTP/REST format), which should be exposed by CalculatorService.Server, and which is to be consumed by CalculatorService.Client.

3.1. Security & Authentication

No authentication, authorization or encryption requirements has been defined for this application.

3.2. Message formatting & serialization

CalculatorService API uses industry standard HTTP/1.1 as it's underlying data communications transport, along with JSON as it's message formatting/serialization. As such any reference message/parameter/property here (not explicitly stated otherwise) should be assumed to be sent/received by using the aforementioned notation. However, in addition to JSON, the server may optionally, support reception of input parameters as part of the HTTP Request, in URL-encoded form.

3.3. POST /calculator/add

Add two or more operands and retrieve the result. This method serves UC-CALC-ADD external interface.

3.3.1. Request Details

Parameters:

- Header:
  - X-Evi-Tracking-Id: An internal Tracking-Id.
    - Optional: This field is not required. If omitted, the operation shall not be persisted.
- Body:
  - Addends: Numeric list. Represents the input of the arithmetic operation.

Example:

```
POST /calculator/add HTTP/1.1
Accept: application/json
Content-Type: application/json
Content-Length: XXX
X-Evi-Tracking-Id: xxxx

{
  "Addends" : [3,3,2]
}
```

3.3.2. Response details

The response should indicate the operation result.

3.3.2.1. 200 - Success

The request has been processed successfully, and a result was received.

Properties:

- Body:
  - Sum: Numeric value. Represent the arithmetic operation result.

Example:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ...

{
  "Sum" : 8
}
```

3.3.2.2. 400 - Bad Request

An invalid request has been received. This may mean the HTTP requests and/or the HTTP body may contains some errors which should be fixed. More information.

3.3.2.3. 500 - Internal Error

An unexpected error condition was triggered which made impossible to fulfill the request. Please try again or contact support. More information.

3.4. POST /calculator/sub

Sub two or more operands and retrieve the result. This method serves UC-CALC-SUB external interface.

3.4.1. Request Details

Parameters

- Header:
  - X-Evi-Tracking-Id: An internal Tracking-Id.
    - Optional: This field is not required. If omitted, the operation shall not be persisted.
- Body:
  - Minuend: Number. Represents the minuend of the arithmetic subtraction.
  - Subtrahend: Number. Represents the subtrahend of the arithmetic subtraction.

Example

```
POST /calculator/sub HTTP/1.1
Accept: application/json
Content-Type: application/json
Content-Length: XXX
X-Evi-Tracking-Id: xxxx

{
  "Minuend" : 3,
  "Subtrahend": -7
}
```

3.4.2. Response details

The response should indicate the operation result.

3.4.2.1. 200 - Success

The request has been processed successfully, and a result was received.

Properties:

- Body:
  - Difference: Numeric value. Represent the arithmetic operation result.

Example:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ...

{
  "Difference" : -4
}
```

3.4.2.2. 400 - Bad Request

An invalid request has been received. This may mean the HTTP requests and/or the HTTP body may contains some errors which should be fixed. More information.

3.4.2.3. 500 - Internal Error

An unexpected error condition was triggered which made impossible to fulfill the request. Please try again or contact support. More information.

3.5. POST /calculator/mult

Multiply two or more operands and retrieve the result. This method serves UC-CALC-MUL external interface.

3.5.1. Request Details

Parameters:

- Header:
  - X-Evi-Tracking-Id: An internal Tracking-Id.
    - Optional: This field is not required. If omitted, the operation shall not be persisted.
- Body:
  - Factors: Numeric list. Represents the inputs of a arithmetic operation.

Example:

```
POST /calculator/mult HTTP/1.1
Accept: application/json
Content-Type: application/json
Content-Length: XXX
X-Evi-Tracking-Id: xxxx

{
  "Factors" : [8,3,2]
}
```

3.5.2. Response details

The response should indicate the operation result.

3.5.2.1. 200 - Success

The request has been processed successfully, and a result was received.

Properties:

- Body:
  - Product: Numeric value. Represent the arithmetic operation result.

Example:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ...

{
  "Product" : 48
}
```

3.5.2.2. 400 - Bad Request

An invalid request has been received. This may mean the HTTP requests and/or the HTTP body may contains some errors which should be fixed. More information.

3.5.2.3. 500 - Internal Error

An unexpected error condition was triggered which made impossible to fulfill the request. Please try again or contact support. More information.

3.6. POST /calculator/div

Divide two or more operands and retrieve the result. This method serves UC-CALC-DIV external interface.

3.6.1. Request Details

Parameters:

- Header:
  - X-Evi-Tracking-Id: An internal Tracking-Id.
    - Optional: This field is not required. If omitted, the operation shall not be persisted.
- Body:
  - Dividend: Number. Represents the dividend of the arithmetic division.
  - Divisor: Number. Represents the divisor of the arithmetic division.

Example:

```
POST /calculator/div HTTP/1.1
Accept: application/json
Content-Type: application/json
Content-Length: XXX
X-Evi-Tracking-Id: xxxx

{
  "Dividend" : 11,
  "Divisor": 2
}
```

3.6.2. Response details

The response should indicate the operation result.

3.6.2.1. 200 - Success

The request has been processed successfully, and a result was received.

Properties:

- Body:
  - Quotient: Numeric value. Represent the arithmetic operation result.
  - Remainder: Numeric value. The remained of the arithmetic division.

Example:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ...

{
  "Quotient" : 2,
  "Remainder": 1
}
```

3.6.2.2. 400 - Bad Request

An invalid request has been received. This may mean the HTTP requests and/or the HTTP body may contains some errors which should be fixed. More information.

3.6.2.3. 500 - Internal Error

An unexpected error condition was triggered which made impossible to fulfill the request. Please try again or contact support. More information.

3.7. POST /calculator/sqrt

Square root two or more operands and retrieve the result. This method serves UC-CALC-SQRT external interface.

3.7.1. Request Details

Parameters:

- Header:
  - X-Evi-Tracking-Id: An internal Tracking-Id.
    - Optional: This field is not required. If omitted, the operation shall not be persisted.

- Body:
  - Number: Number. Represents the input of the square root mathematical function.

Example:

```
POST /sqrt HTTP/1.1
Accept: application/json
Content-Type: application/json
Content-Length: XXX
X-Evi-Tracking-Id: xxxx

{
  "Number" : 16
}
```

3.7.2. Response details

The response should indicate the operation result.

3.7.2.1. 200 - Success

The request has been processed successfully, and a result was received.

Properties:

- Body:
    - Square: Number. Represent the arithmetic operation result.

Example:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ...

{
  "Square" : 4
}
```

3.8. POST /journal/query

Request all operations for a Tracking-Id since the last application restart. This method serves UC-CALC-QUERY external interface.

3.8.1. Request Details

Parameters:

- Body:
    - Id: The Tracking-Id for which journal should be queried against.

Example:

```
POST /journal/query HTTP/1.1
Accept: application/json
Content-Type: application/json
Content-Length: XXX

{
  "Id": "xxx"
}
```

3.8.2. Response details

The response should retrieve all the requested operations for the Tracking-Id.

3.8.2.1. 200 - Success

The request has been processed successfully, and a result was received.

Properties:

- Body:
    - Operations: List of all the operations performed with the specified Tracking-Id.

Example:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: ...

{
  "Operations" : [
     { "Operation": "Sum", "Calculation": "2 + 2 + 2 = 6", "Date": "2016-12-01T12:13:14Z" },
     { "Operation": "Mul", "Calculation": "3 * 3 * 3 = 27", "Date": "2016-12-01T12:13:15Z" },
     ...
   ]
}
```

3.8.2.2. 400 - Bad Request

An invalid request has been received. This may mean the HTTP requests and/or the HTTP body may contains some errors which should be fixed. More information.

3.8.2.3. 500 - Internal Error

An unexpected error condition was triggered which made impossible to fulfill the request. Please try again or contact support. More information.

3.9. Generic/Common Error Responses

In this section you will find a description of some generic error responses applicable to requests to RAC REST/API.

3.9.1. 400 - Bad Request

An invalid request has been received. This may mean the HTTP requests and/or the HTTP body may contains some errors which should be fixed.

3.9.1.1. Attributes

- Body:
  - ErrorCode: Mnemonic assigned to this specific error condition.
  - ErrorStatus: Numeric (HTTP-derived) error status corresponding to this request.
  - ErrorMessage: Textual description of the error, in a (hopefully) human understandable way.

NOTE: Requests not compliant with HTTP/1.1 specification, along with other special failure conditions may receive a 400 (or similar 4XX) status error, with missing 'body' part. This is expected and such cases should be handled by consumer by manually inspecting the request and/or consulting support.

3.9.1.2. Example

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Content-Length: ...

{
   "ErrorCode" : "InternalError",
   "ErrorStatus": 400,
   "ErrorMessage": "Unable to process request: ..."
}
```

3.9.2. 500 - Internal Error

An unexpected error condition was triggered which made impossible to fulfill the request. Please try again or contact support.

3.9.2.1. Attributes

- Body:
  - ErrorCode: Mnemonic assigned to this specific error condition.
  - ErrorStatus: Numeric (HTTP-derived) error status corresponding to this request.
  - ErrorMessage: Textual description of the error, in a (hopefully) human understandable way.

NOTE: Requests not compliant with HTTP/1.1 specification, along with other special failure conditions may receive a 400 (or similar 4XX) status error, with missing 'body' part. This is expected and such cases should be handled by consumer by manually inspecting the request and/or consulting support.

3.9.2.2. Example

```
HTTP/1.1 500 Internal Error
Content-Type: application/json
Content-Length: ...

{
   "ErrorCode" : "InternalError",
   "ErrorStatus": 500,
   "ErrorMessage": "An unexpected error condition was triggered which made impossible to fulfill the request. Please try aga
}
```

Comments

Add a comment