Low Level Design for Expression Evaluator API

1. General Information

Purpose and Scope

The document includes a solution on implementation of Expression Evaluator Microservice to address expression creation, validation, evaluation and holds customer specific data in cache memory.

The solution is described in such detail that is possible to understand the business value, functionality and make high level effort estimation.

Any part of the proposed design may subject to change implementation.

Input

This document is prepared based on the README.md of below link.

https://github.com/leapwise/expression-evaluator

Revision Information

Revision	Date	Signature	Comments
V1	13.11.2023	Irem AKTAS	Initial Document has been created

Use Case Diagram

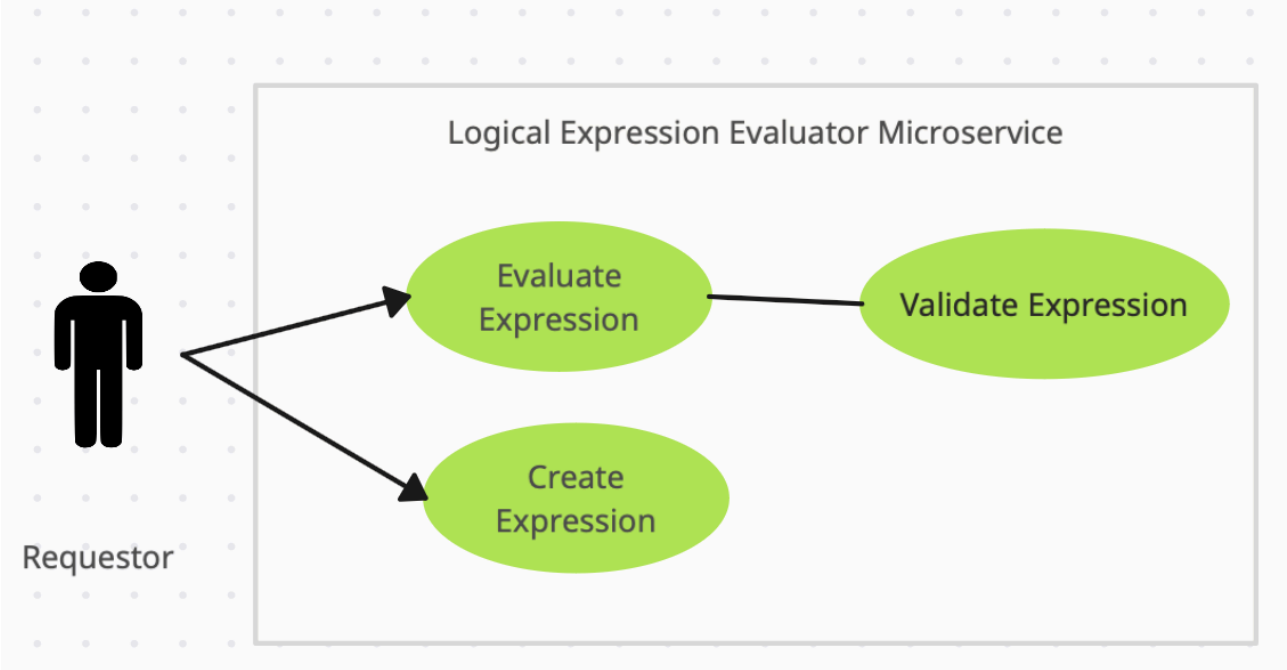

Figure 1 Expression Evaluator Use Case Diagram

Patent Aspects

The solution does not include any patentable part.

2. Technical Solution

This solution proposes the design of Expression Evaluator Microservice which exposes expression creation and evaluate expression API to apply dynamic search mechanism for related data.

The base API path should be **/expressionEvaluator/v1/expression** and **/expressionEvaluator/v1/evaluate**

Expression evaluator responsible for logical expression creation, validation and execution on given data object.

Domain Model contains Customer and Exception Model. A view of the domain model can be seen in the following logical View – Domain Model Diagram.

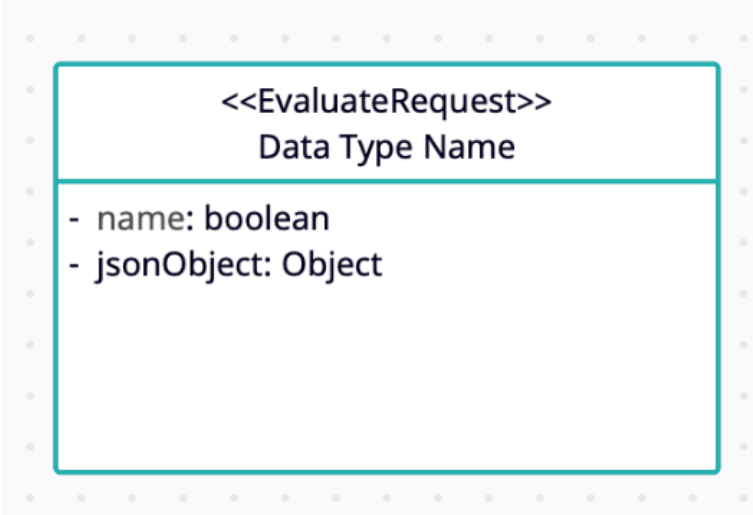

Figure 2 Expression Evaluator Logical View, Evaluate API Request Model

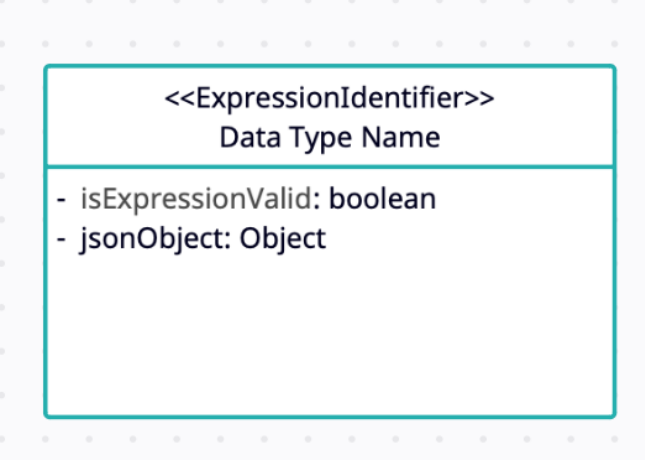

Figure 3 Expression Evaluator Logical View, Evaluate API Response Model

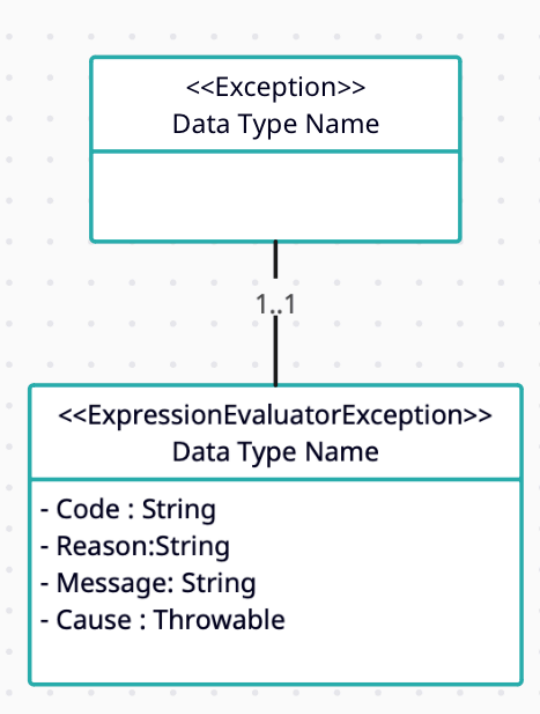

Figure 4 Expression Evaluator Logical View, Domain Model – Exception

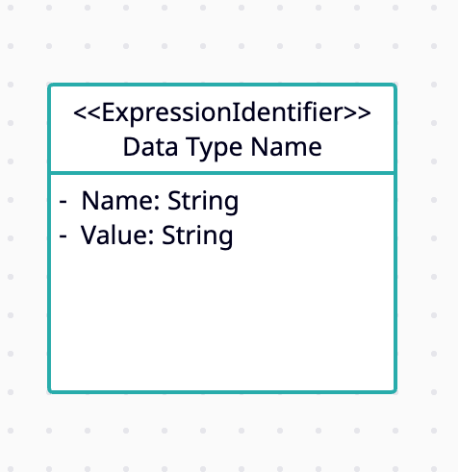

Figure 5 Expression Identifier Logical View, Domain Model

Expression Evaluator micro service exposes /expression and /evaluate APIs. When both APIs are triggered, first IDM (Identity Access Manager for Authentication) should be triggered to ensure that the requestor has right to trigger the API.

In case of the requestor is valid and **/expression** API is triggered, the API validates the request content before creating such as name should be provided. If the given expression name is unique then expression record is created.

When the requestor is valid and **/evaluate** API is triggered, the API validates the expression and assesses the expression with given data object in the request.

Identifiers are kept in cache to decrease latency.

System should throw below **Exceptions** :

When **/evaluate** API is triggered, If there is no such identifier exists in the system as provided in the request name, then logical expression evaluator throws such as **NoExpressionIdentifierFound** with status code 500

When **both APIs** are triggered, If the requester has not valid credentials to execute /evaluate API, micro service should throw **UnauthorizedUser** with status code 401.

When **/evaluate** API is triggered, If the provided expression is not a valid logical expression, the micro service should throw **ExpressionNotValid** with status code 500

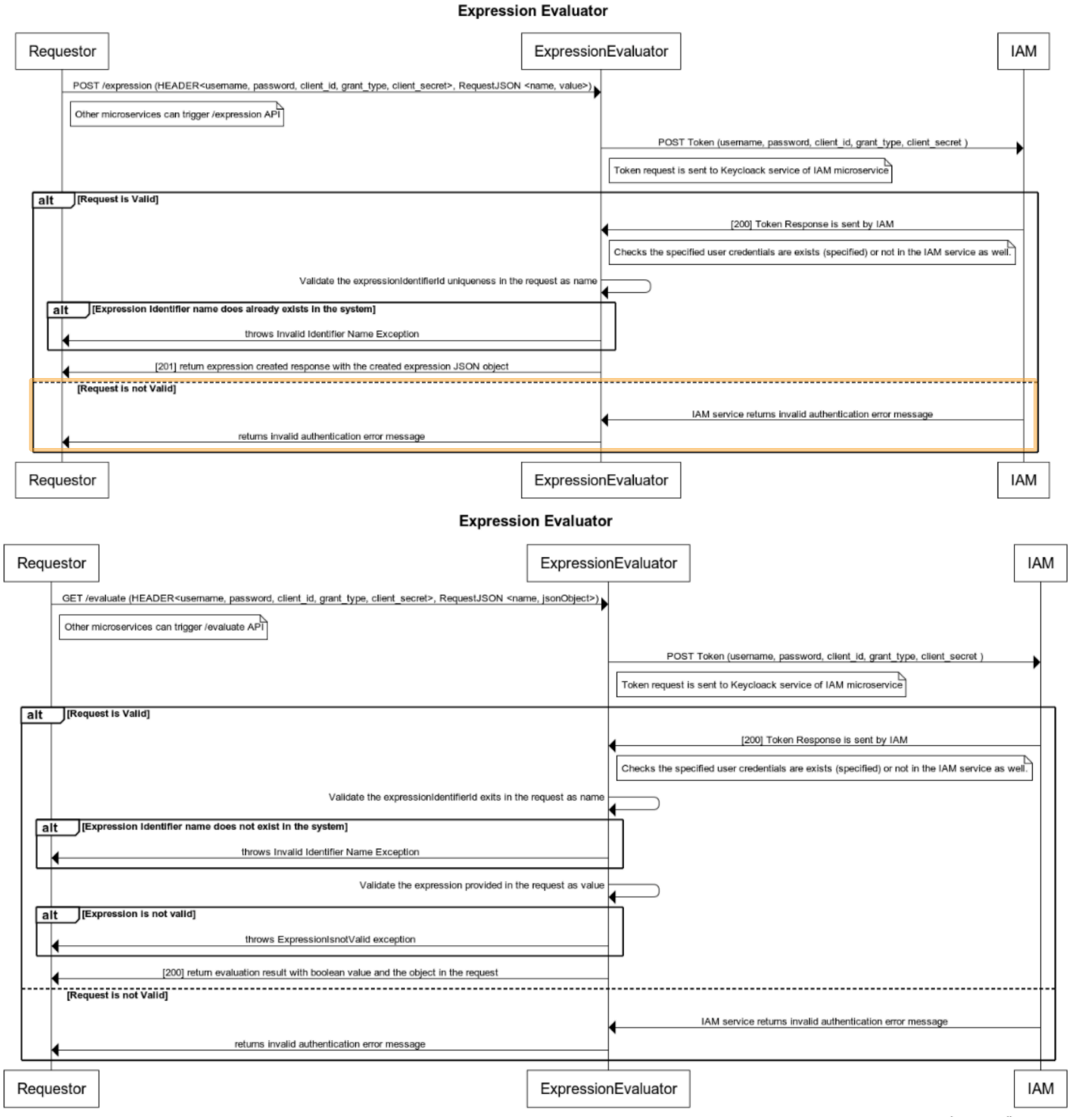

Figure 6,7 Expression Evaluator Business Flow – Sequence Diagram of /expression and /evaluate APIs

Please note : IAM service will be stub for current implementation. In Next sprints it will be replaced with the real IAM service.

3. Sample Expression Evaluator Request of /expression API


```
{
   "name": "Complex logical expression",
   "value": "(firstName == \"JOHN\" && salary < 100) OR (address != null && address.city ==
\"Washington\")"
}
```


3. Sample Expression Evaluator Response of /expression API


```
{
   "Status": 201,
   "Message": "Expression Successfully Created",
   "Data": {
      "id": 1,
      "name": "Complex logical expression",
      "value": "(firstName == \"JOHN\" && salary < 100) OR (address != null && address.city ==
\"Washington\")"
   }
}
```


3. Sample Expression Evaluator Request of /evaluate API

Request Returning False:


```
{
   "name": "Complex logical expression",
   "customer": {
      "firstName": "JOHN",
      "lastName": "DOE",
      "address": {
         "city": "Chicago",
         "zipCode": 1234,
         "street": "56th",
         "houseNumber": 2345
      },
      "salary": 99,
      "type": "BUSINESS"
   }
}
```

Request Returning True:


```
{
  "name": "Complex logical expression",
  "customer": {
    "firstName": "JOHN",
    "lastName": "DOE",
    "address": {
      "city": "Washington",
      "zipCode": 1234,
      "street": "56th",
      "houseNumber": 2345
    },
    "salary": 99,
    "type": "BUSINESS"
  }
}
```


4. Sample Expression Evaluator Response of /evaluate API

Response Returning False:


```
{
  "isExpressionValid": false,
  "expressionResult": {
    "firstName": "JOHN",
    "lastName": "DOE",
    "address": {
      "city": "Chicago",
      "zipCode": 1234,
      "street": "56th",
      "houseNumber": 2345
    },
    "salary": 99,
    "type": "BUSINESS"
  }
}
```


Response Returning True:


```
{
  "isExpressionValid": true,
  "expressionResult": {
    "firstName": "JOHN",
    "lastName": "DOE",
    "address": {
      "city": "Washington",
      "zipCode": 1234,
      "street": "56th",
      "houseNumber": 2345
    },
    "salary": 99,
    "type": "BUSINESS"
  }
}
```

5. Exposed APIs

https: {host}:{port}/leapwise/expressionEvaluator/v1/expression

https: {host}:{port}/leapwise/expressionEvaluator/v1/evaluate