Case study for tree manipulation primitives in Jolie.

In the following pages we present a case study for the implementation of a strong Jolie’s tree manipulation primitives.

# In memory data aggregation and manipulation

The GDPR regulation is pushing the boundaries of privacy well over the traditional regulatory framework for a more comprehensive approach that envisage new ways to handle personal data in a secure and privacy coherent way.

One of the key requirement of GDPR is that aggregated data must exist only within a specific workflow and not be persistent Jolie is being strongly orchestration oriented language provides a strong framework design aggregation workflows. Yet Jolie is not able to express with the same ease aggregation of data, and requires always to deposit on a persistent medium. This represents a strong limitation of the language that propose itself as a middleware process integration tool.

With a tree manipulation primitives the Jolie architect will be able to design data access API where the data will not come from persistent support but in memory data aggregation from different sources.

# **Example architecture**

The picture below shows the a typical example of data aggregation for client management

Application

Connector

Third Party

API

mService

mService

mService

Data API

In memory

data aggregation

We can identify four actors in this solution

1. A microservice that is connected that provides access to a DB ( e-commerce )
2. A microservices that provide access to a application such an ERP
3. A third party API , typically HTTP REST (Identity provider such as Auth0)
4. A orchestrator microservice

Typically the customer information are distributed over the first three actors, in a traditional approach any new aggregated view of the original data will need to be stored on several table in an aggregation DB (highlight in red)

The presence of powerful three manipulation will allow to create runtime representation of aggregated data bypassing the need of an aggregation DB.

This approach present the following advantages

1. Aggregated data exist only during a specific invocation of an API
2. New aggregated “views” can be developed only through orchestration
3. The access to specific aggregated data can be provided by controlling the access to API

# Primitives and its uses in the scenario

1. Match: Many API give some kind selection parameters but they may not be sufficient for the data aggregation requirement; by using this primitive the architect will be able to extend the selectivity capabilities of the original API
2. Group : Any data require the aggregation like sum of all purchases for a specific period
3. Lookup: Let us assume that we need to provide the payment status for all the order , the order information are normally contained in the e-commerce DB where invoicing and payment information are often provided by the ERP, so it is necessary to merge both set of information on resulting resulting tree structure
4. Project: If we need to hide or modify specific part of the original tree to guarantee the privacy of specific e-commerce transaction by removing or masking specific data.

# Example of behavior for the creation of a aggregated data

User logon data from the API

match specific period

Filtered Tree

Project to transform the tree into service request messages

microservice e-commerce

microservice to the ERP

Data from e-commerce

ERP Data

Combine tree

Lookup to merge the two set of data into a single document

Project to mask some data

Masked tree

The diagram above shows an example behavior that uses the primitive at runtime to model a specific data that exist only in memory

Syntax Example

Let us start with the JSON format of a single logon information

[users:[ {

"email": "test@gmail.com",

"email\_verified": true,

"name": "Test Person",

"given\_name": " Test Person ",

"family\_name": "Person",

"picture": "",

"gender": "male",

"locale": "it",

"updated\_at": "2018-03-28T13:29:54.238Z",

"user\_id": "",

"nickname": "TestPerson",

"identities": [

{

"provider": "google-oauth2",

"user\_id": "Test",

"connection": "google-oauth2",

"isSocial": true

}

],

"created\_at": "2018-03-28T10:46:29.083Z",

"user\_metadata": {

"cap": "07100",

"gender": "male",

"birth\_year": "1988",

"given\_name": "N.G."

},

"app\_metadata": {

"consenso\_allargato\_c\_accepted\_at": "2018-03-28T10:56:32.651554+00:00",

"privacy\_policy\_accepted\_at": "2018-03-28T10:56:32.651528+00:00",

"consenso\_allargato\_b\_accepted\_at": "2018-03-28T10:56:32.651554+00:00",

"consenso\_allargato\_a\_accepted\_at": "2018-03-28T10:56:32.651554+00:00"

},

"last\_ip": "79.7.208.209",

"last\_login": "2018-03-28T13:29:54.238Z",

"logins\_count": 22,

"blocked\_for": [],

"guardian\_authenticators": []

},

{

"email": "test1@gmail.com",

"email\_verified": true,

"name": "Test Person1",

"given\_name": " Test Person1 ",

"family\_name": "Person1",

"picture": "",

"gender": "male",

"locale": "it",

"updated\_at": "2018-03-28T13:29:54.238Z",

"user\_id": "",

"nickname": "TestPerson1",

"identities": [

{

"provider": "google-oauth2",

"user\_id": "Test",

"connection": "google-oauth2",

"isSocial": true

}

],

"created\_at": "2018-03-28T10:46:29.083Z",

"user\_metadata": {

"cap": "07100",

"gender": "male",

"birth\_year": "1988",

"given\_name": "N.G."

},

"app\_metadata": {

"consenso\_allargato\_c\_accepted\_at": "2018-03-25T10:56:32.651554+00:00",

"privacy\_policy\_accepted\_at": "2018-03-25T10:56:32.651528+00:00",

"consenso\_allargato\_b\_accepted\_at": "2018-03-25T10:56:32.651554+00:00",

"consenso\_allargato\_a\_accepted\_at": "2018-03-25T10:56:32.651554+00:00"

},

"last\_ip": "79.7.208.209",

"last\_login": "2018-03-27T13:29:54.238Z",

"logins\_count": 22,

"blocked\_for": [],

"guardian\_authenticators": []

},

]

Now maybe we would like to filter the record by a specific date

*requestFind<<responseApi.users*

*requestFind.filter=”****{$match:{"last\_login":"2018-03-28\*/"}}***

*find@TreeManiputlation(requestFind)(responseFind)*

responseFind will contain only one record for [test@gmail.com](mailto:test@gmail.com)

now we want to create the request message for both the ERP and eCommerce microservices by using a project operation that may look like this for the eCommerce

***requestProject << responseFind***

***requestProject.project = {‘customer.email\_customer’: ‘$users.email’ , ‘customer .transaction\_date’: ‘$lusers,last\_login’}***

***project@TreeManiputlation(requestProject)( responseProject)***

responseProject will look like

**{customers:[{email\_customer: “**[**test@gmail.com**](mailto:test@gmail.com)**” , transaction\_date: “2018-03-28T13:29:54.238Z” }]}**

Now we may need to merge via a lookup the information coming from both the called microservices

***responseERP = [invoices:[{numberOrder:’45673’,numberInvoice:”31543234”,paid: true ,exipired:false } invoices:[{numberOrder:’45672’,numberInvoice:”31543231”,paid: false ,exipired:true }]***

***responseEcommerce = [purchases: [{number\_order:’45673’: deliverd :true , deliver\_address : “ cherry st 34”}, {number\_order:’45672’: deliverd :true , deliver\_address : “ cherry st 34”}]***

**requestLookup.localCollection<< responseERP**

**requestLookup.foreignCollection<< responseEcommerce**

**requestLookup.lookUp = { localField: “invoices.numberOrder” , foreingField: ” purchases.number\_order” as: “paymentStatus”}**

***lookup@TreeManiputlation(requestLookup)( responseLookup)***

the resulting tree should look like

***{ paymentStatus :[ { number\_order:’45673’: deliverd :true , deliver\_address : “ cherry st 34” , numberInvoice:”31543234”,paid: true , exipired:false }, { number\_order:’45672’: deliverd :true , deliver\_address : “ cherry st 34” , numberInvoice:” 31543231”,paid: false , exipired:false },]}***

Let us imagine that we need to hide (mask) some data

***requestUpdate << responseLookup***

***requestUpdate..instructions{ $unset : ”paymentStatus.deliver\_address”}]***

***update@TreeManiputlation (requestUpdate)(responseUpdate)***

resulting in

***{paymentStatus :[ { number\_order:’45673’: deliverd :true , numberInvoice:”31543234”,paid: true , exipired:false }, { number\_order:’45672’: deliverd :true , numberInvoice:” 31543231”,paid: false , exipired:false },]}***