Intercom
Integrating People, Business, and Technology

WSO2

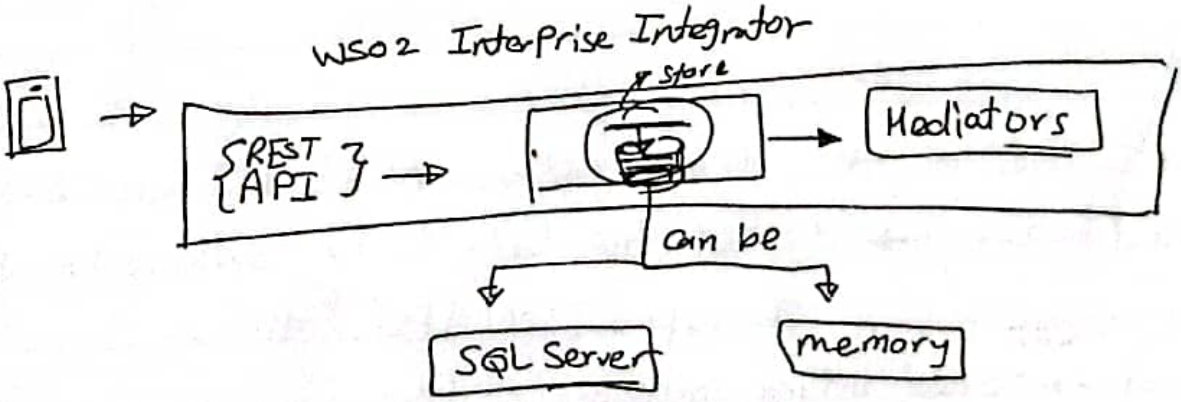
1. Components
   - Micro Integrator → Component of the WSO2 as (JsonToXML, ...)
   - StreamIntegrator → as Connecting to ETL
   - Integration Studio → IDE
   - Connectors → as plugin defined to do some actions (as file connector)
   - monitoring
   - Choreo

~~1. HTTP~~

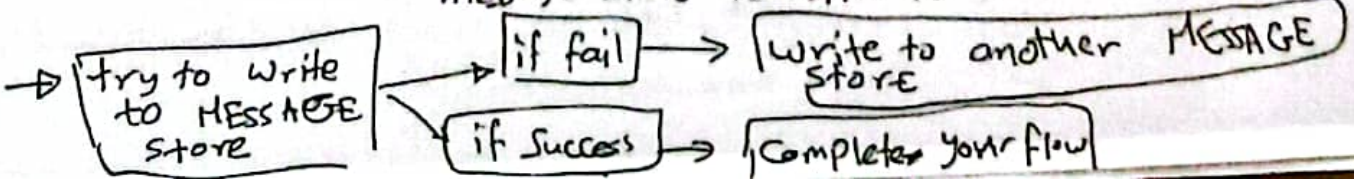
1. HTTP EndPoint → to define external REST API (GET, POST, PUT...)
2. Payload Factory → to set the body to be static value may be of type (JSON, TEXT, XML)
3. Respond → return response of the flow to the client (used at the end of the flow)
4. Call → used with (HTTP EndPoint) to call external endPoint
   - NOTE → HTTP EndPoint → defining the external endpoint
   - Call → used to call the defined endPoint

⑤ Log → used to log any thing in the flow and log type is →
- Simple
- Header
- Full

⑥ MEssage Store → used to store a message to be processed later with something/someone that will pull/get this message and process this message
- as an example when we want to make async request → as an example put message on MQ and a Consumer pull message from MQ and process it
- NOTE → store Can (be DB, InMemory, -)



NOTE very Important Note
- when we try to write to message store and unfortuently it fails/Cant write to the message Store, we Can define another message Store to write to it



Intercom
Integrating People, Business, and Technology

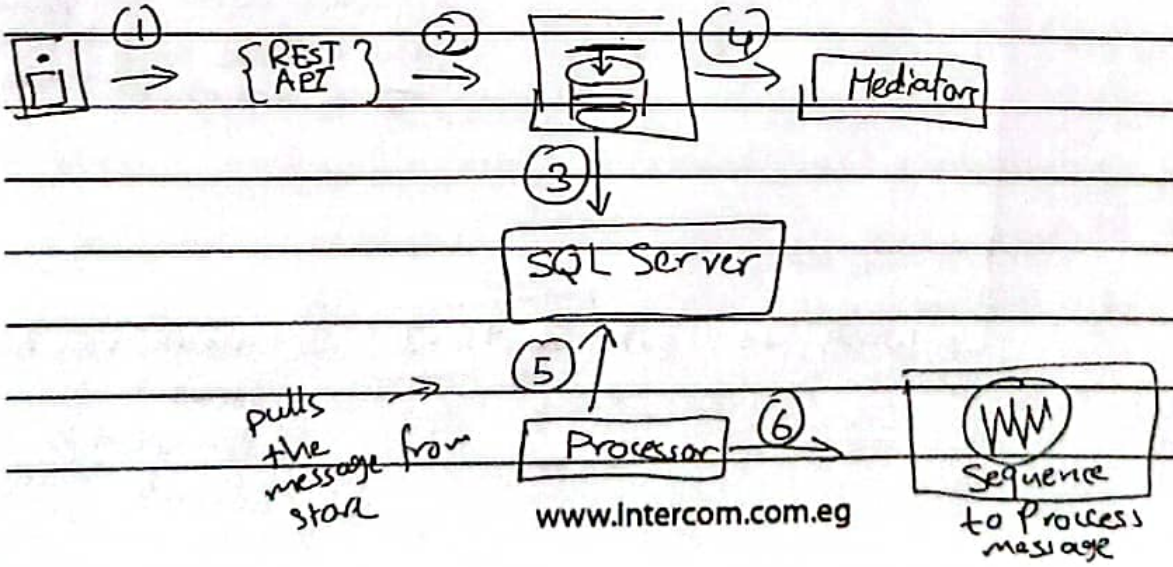
25 YEARS

→ and this property called | EnableProducerGrauted | Delivery = [ ]

true → when you define another message store to store failed saved ones → need to give it a name for message store to store in it failed ones

false → you will not define another message store to store failed ones and message will be last

(7) MESSAGE Processor → Process the message from the message store then send it to sequence or endPoint



→ Properties

↳ ① message Processor type: Message SAmpling Processor

② message Store: name of message store to pull from it

③ Sequence: Sequence name to send the message to this sequence to Process it

④ Sampling interval: pulls message every specific (Millis)

~~⑤ Cron Expression~~

Notes

↳ When we open the ~~to~~ Create scheduled task we will see tab called Task Implementation Properties

↳ this must be filled with specific attribute key-value

↳ ① injectTo → must hold Sequence

② message → must hold non-empty value

↳ XML

↳ LITERAL

③ sequenceName → must Contain the sequence name to handle the ~~Getting~~ each time of ~~Set~~ scheduled task

↳ when we set Count=-1 → means run each time, with Specific interval

Intercom
Integrating People, Business, and Technology

25 YEARS

(9) Property / property Group : used to set in the flow Values that we will use it later

How to

get_Property("□") → to get Value

Json-eval($.□) → retrive Value From Json object

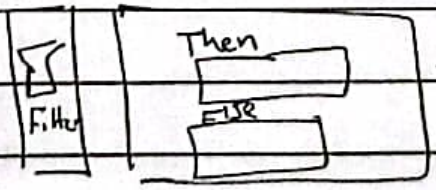
$url → to retrieve query or path param variable

$trp → to retrieve specific header from request

(10) Switch → like switch cases in any (java, Python, ---)

↳ Case □
Case □
:
default

(11) Filter → like if/else statement in any programming



→ when we want to check on value on request body JSON

{ id: 1
name: "yousef" }

$.name

(6)

(12) PayloadFactory → used to prepare (text, JSON, XML) data before the next mediator (as an example calling external api)

Properties

① payload Format (Inline, registry)
- Inline → we will write it by hand
- registry → stored into registry

② Media Type: (XML, JSON, TEXT)

③ payload → Data

④ Arguments → as ~~a placeholder~~ when we write a payload we can write the values from predefined values (ctx, axis2, trp, arg, Array)

⑤ Template Type (default)
- we write values from args be $[]

example
```
{
"id": $1,
"name": $2
}
```
- $1 : } can be from property, JSON, XML
- $2 : }

Freemarker
- from request body
- input JSON to a

```
{
"id" : ${payload.id},
"name": ${ctx.name},
"header": ${arg.arg1}
}
```
- arg1 → first argument in the defined argument

Argument
firstArg: yousef

(12) PayloadFactory → used to prepare (text, JSON, XML) data before the next mediator (as an example calling external api)

Properties

① payload Format (Inline, registry)
- Inline: we will write it by hand
- registry: stored into registry

② Media Type: (XML, JSON, TEXT)

③ payload → Data

④ Arguments → as a placeholder when we write a payload we can write the values from predefined values (ctx, axis2, trp, arg, Array)

⑤ Template Type (default)

we write values from args be $□

example
```
{
"id": $1,
"name":$2
}
```

$1 : } Can be from property, JSON, XML
$2 : }

Free marker

```
{
"id" : ${payload.id},
"name": ${ctx.name},
"header" : ${arg.arg1}
}
```

payload: from request body
ctx: input JSON to node
arg: first argument in the defined argument

Argument
firstArg: yousef

(13) Enrich → Can update/add/remove attribute in payload of request or payload output from mediator

(14) DBlookup → Select 1 only from table and store result of the selection int defined Properties

① select one row only
② means that you are not allowed to use (update, delete, insert

example → select id, name, age from employees

mapping values

from db → id
to property → emp_id

→ name
→ emp_name

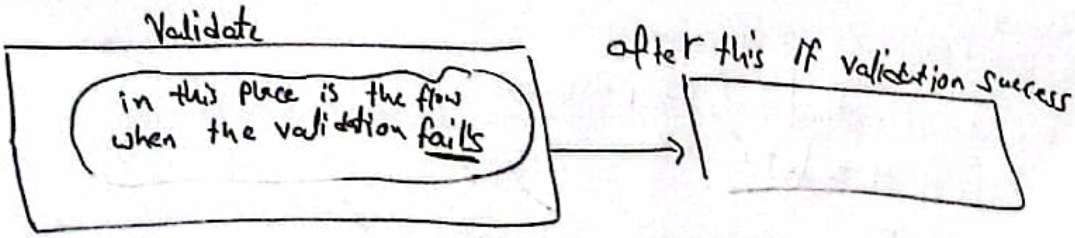
(15) validate → Validate (request/respone/any object) in (JSON, XML) aganist its defined schema

↳ properties

↳ ① source → source data to validate

② schema → We will store it into folder in our project called local-entries and this file contains the data of the schema

③ If we want to see the exception it will be stored int ERROR_CODE, ERROR_MESSAGE



16 ~~Caching~~

16 Cache → 1) When we try to check if value exist in Cache we use [FINDER] ~~to~~ Property

When we want to store the value of the response into cache we must use [Collector]

FINDER

COLLECTOR

1) Cache Type : FINDER

2) Cache Timeout : ☐ seconds
time that values must be exist in Cache and after that it will be removed from Cache

3) max bytes : ☐
maximum value of bytes to store for each value in Cache

4) Cache protocol method : GET, POST, ...
define HTTP Methods to Cache its values

5) Headers To Exclude:
as an example if we want to exclude tokens (each request it is changed)

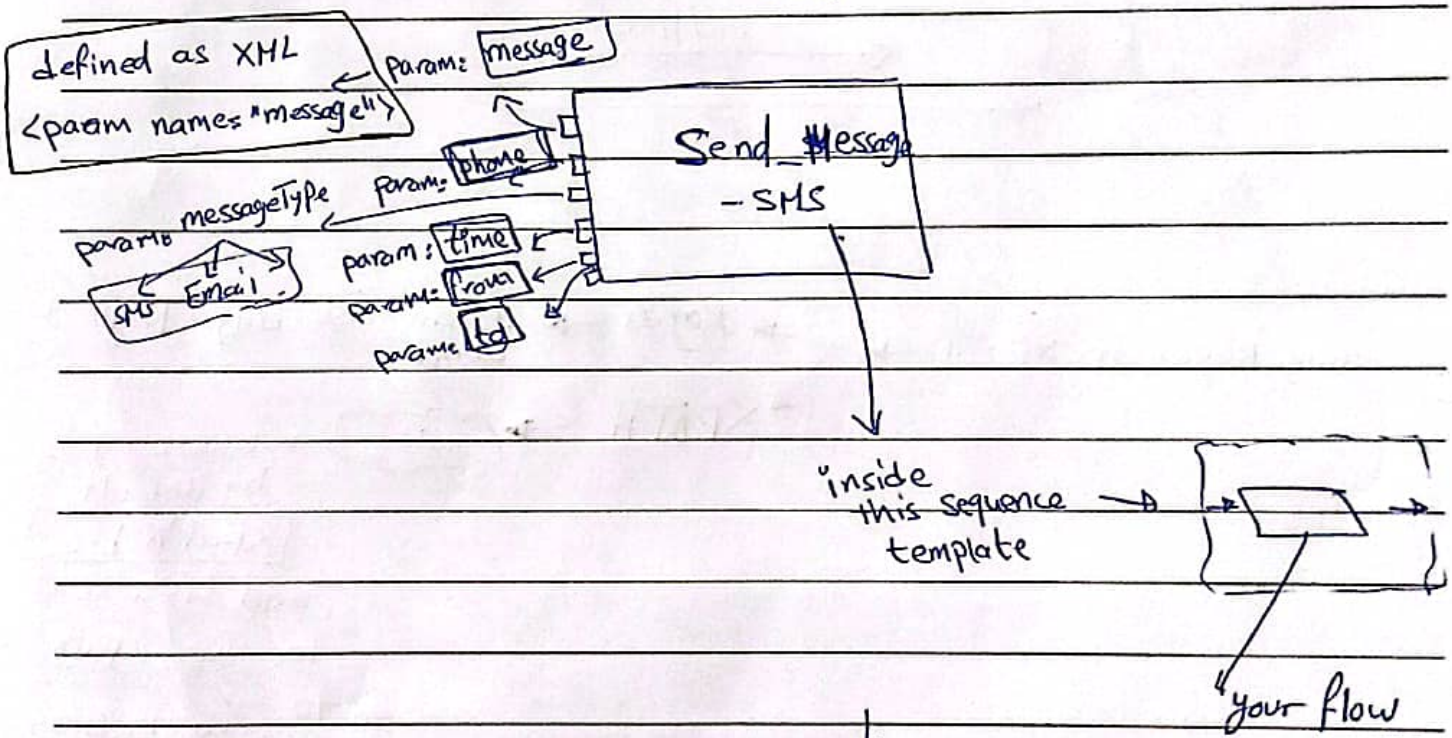
6) Response Codes: [-*]
we will store any HttpCode into hash

(17) Sequence-Template → this act as a function in any programing Languge (repeated code) you will put it into a function and this function will be called (the function may has a Params (send to it)

↓

Example

1



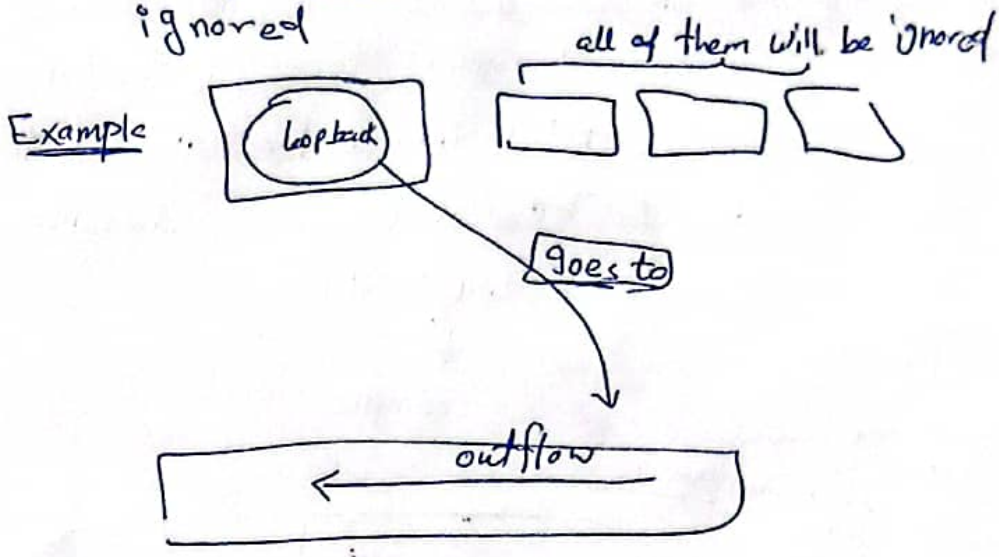
↓

2 → How to Call this Sequence_Template

↳ Called by using (Call_Template)

this mediator takes the name of your sequence-Template and add your Parameter that will be passed to it

(18) Loop back → it goes back from Current Sequence to the outSequence of your API and all thing after ~~Cur~~ it will be ignored



(19) using Regex or XPath →
- Regex → u can use any Regex u want
- XPATH → NOTE → WSO2 uses by default Xpath 1.0 and if u want to use Xpath 2.0 go to /config/deployment.toml and write this line

```
Synpase.enable_xpath_dom_failover= true
```

(19) Clone Mediator → used when you want to make Process on the same message ~~multiple~~ on different/multiple ways

asyn/sync

Example → if you have {id:1, name:"x"} and you want to Process this message by calling 2 backend Service → POST /students
→ POST /Persons

NoTE → we must use aggregate medistor with clone medistor

1. we must define the Aggregate ID to be the same in clone & aggregate mediator
2. define number of output that we will aggregate
3. Aggregate Expression → the most important one this defines which data that we collect at each response from each target

(17) Clone Mediator → used when you want to make process on the same message ~~multiple~~ on different/multiple ways

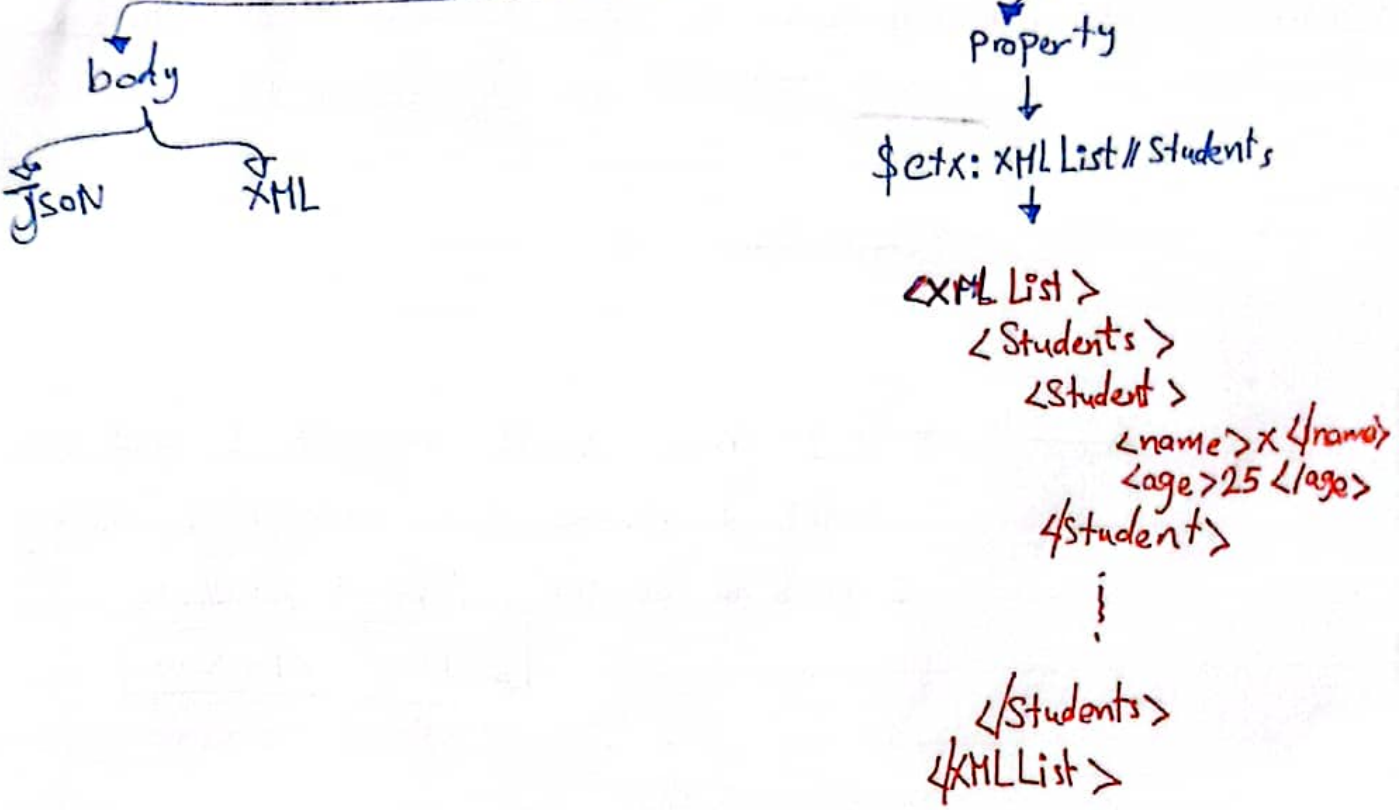
asyn/sync

Example → if you have {id:1, name:"x"} and you want to process this message by calling 2 backend service → POST /students
→ POST /persons

NOTE → we must use aggregate mediator with clone mediator

1. we must define the Aggregate ID to be the same in clone & aggregate mediator
2. define number of output that we will aggregate
3. Aggregate Expression → the most important one this defines which data that we collect at each response from each target

(20) ForEach Mediator → normal for loop in any language and it takes source to iterat over it can be



---

(21) iterate Mediator → iterates over List of values as (for loop) and can process the list ~~and~~ in async ~~to~~ with Aggregate mediator (its a must) not an option

→ Example: → Case: → when you have a list of ~~objects~~ ids of students and you want to call backend end point /students/{id} to get each ~~it~~ student, then you can use iterate

↓

i know that u asking your self then here i can use forEach instead but the golden feature in iterate is async process and aggregation, very Faster