A Basic Pattern for Service Development in a WescaleIT Container

# Basic Requirements

Building application by using stateless service APIs for SAP Fiori (or actually any environment based on REST services) we need to have a clean understanding of the handlingFo of transaction and transactional consistency. And we want to make sure, that within a development using a WescaleIT container these requirements are fulfilled by following a certain methdology.

First: any service requests from a client must be an own transaction as seen from the perspective of a database. Every service call is a function, which is executed atomically. There needs to be either auto-commit or auto-rollback on failure for the service and we need to guarantee this.

Every service has to be able to be repeated on failure, i.e. in a more sophisticated language we may want to say that every service is “idempotent” in a mathematical sense.

Services address Resources and a resource is described by a URL Within a business transaction, which is not the same as a database transaction, we need to be able to manage state for intermediate states, for the final state of a transaction and also for the immutable state of the final outcome. You can follow this by establishing a pattern, which starts from a form to filled out as step 1, a final commit or submit action, but also then as an archive for historic states.

For every entity we will have a certain attribute structure, which defines the object. But to represent the different transactional states, the object needs to go thru a transition with database entities with slightly different primary key structures.

Let us assume that we want to talk about an entity or object, which may have attribute A, B, C.

ENTITY

(PRIMARY KEY) || A | B | C |

But in the various states of a transaction the entity will need slightly different key structures, which we best represent in different similar tables, which have the same attribute structure.

The REST services for filling out a form, for submitting a form to create the final state for this but also for accessing historic data are slightly different, but the can give us the framework for actually doing it and to establish the methodology in the container. for every object.

Since all services must be idempotent and stateless, i.e. free of side effects, this has some implications, which we need to establish a certain set of patterns.

Every transaction will start with filling out a form, then submitting the form and then archiving it.

## Fill Out a Form

Filling out a form including error handling etc until it is final, requires that the data, which represent the content of the form are owned by a user or some kind of “case”. It is not public yet and the “case number” or the user ID is part of the primary key. Let us assume we take a general case number. Then the primary key looks like this:

CASE ID | OBJECT ID || A | B | C |

It is the content of the form not more. Typically the database to put that in is a row-store database.

With respect to MySQL there is the need for a seeparate schema to hold these tables. You can distribute this over MySQL instances easily via a hash over the case ID for example (which may include the user ID).

The REST operations on this are GET and PUT operations of a record. Locking on the object can and should be optimistic and last-one wins. Version numbers for optimistic lock can be achieved via auto-increment columns.

## Submit The Form (Make it Final)

Submitting a form needs to be idempotent to. I.e. it must be repeatable on failure. For this you do not want actions which for example just add some values. What you want is a document to be updated or inserted, which represents the action, such that we always, if the action was completed, to be repeated on failure, but is also not executed twice.

The key structure then is somehow

OBJECT ID || A | B | C |

and the operation is a POST operation (without parameters!).

Since in many SAP objects the primary key of an object is only created within the create operation, we need to have a key mapping being established between the form data and the final data. ENQUEUE operations at SAP may fail, and so we need to be able to redo. ANd it may fail because the system may not be available.

## Put The Form To An Archive

As soon as the object becomes final, it makes sense to already write the data in a second tables, which does not just represent the normalized object as you would do it for an object to be updated, but also in a format, where you take a snapshot, which is immutable. To become immutable, the object needs a version number and if a new version is created by an update, we also create a new record for this. This is helpful since now you can create historic snapshots. If you do this for all object including master data you do not have to copy master data into transaction documents, which else you have to do for historic data to make sense.

The key structure then is

OBJECT ID | VERSION# || A | B | C |

All there tables are best represented within different schema inside a database. (In MySQL schema and database are anyhow the same, but the general concept would be schema).

For historic data a columnar representation can play its benefits, since it will automatically compress equal strings in different entities and you usually have a 10 time performance improvement for reading in columnar representations.

Version Numbers in MySQL can be created via auto-increment columns for example.

We can take this pattern and use it for basically every object, which we may meet.

## Attachments

There is one thing, which we may want to add. Regarding attachments, which anyhow are immutable objects, it is actually helpful to directly go the the historic data model (since they are usually immutable anyhow) and then use the a link to this attribute in the transaction tables instead of the value.