Article
Istvan Hahn · Oct 20, 2016 6m read

Creating an Ensemble MicroService using RESTful Web Services

This is a beginner ' s guide to the design of a " MicroService " implemented in Ensemble. " MicroService " is a popular phrase these days which has a broad interpretation. My interpretation is: " MicroService " is a " NoSQL Service " . A what? The answer is in the article.

We learnt what the difference is between SQL and NoSQL databases. For me the difference is nearly the same between a SOA Web Service and a " MicroService " . I am going to explain it through an example.

Please note, that although this is a beginner ' s guide, I assume deep technology knowledge on data modelling, RESTful services and Ensemble.

Flight booking – the example

My example is very simple. I would like to book a flight for my family. I use www.skyscanner.hu, but it can be any booking site. So what do I do? On the landing page I enter the flight whish: I need ticket from Budapest to Prague and return, flying out on 29th October returning on 1st November, direct flight, on economy class for four persons. After submitting the search form, I get back the available options. I select one and enter the passenger details, discount program and finally pay by PayPal. Straight forward.

In a more detailed version: what is happening in the background.

n Start booking

n Enter select criteria

n Produce a list of matching flights

n Select a flight

n Block tickets temporarily

n Enter passenger details

n Calculate airfare based on list price applying some discount program

n Switching to PayPal and pay

n Finalizing the booking, releasing the temporary block

When the task is to develop a back end using SOAP Web Services, the most common architecture would be like:

n There will be a SOAP service supporting the booking. Let us say it is called FlightBookingService.

n The service will have methods like

o Search – based on the query parameters (origin, target, fly out, fly in date, class, number of passengers) returns a list of flight options.

o Block – block tickets for a particular flight. It is a temporary block for a certain time. If the block time-out expires,

the blocking released automatically.

o Allocate ticket – allocate tickets to particular passenger by assigning name to the blocked ticket.

o Calculate price – returns the full price based on the normal tariff and applying the discount program.

o Finalize booking – take the payment confirmation and make the blocked tickets valid. Optionally check the payment confirmation with the e-commerce software.

Even on the cover it is a service oriented approach, in fact the design pattern follows the monolithic programming principles. A single service is built to provide all methods, each method has its own complex input structure to process, each structure describes all potential variations and combinations of recognized data contents. Internally there is a complex data model (relational or object) describing the data structures and relations. Obviously data which is mandatory in the model must come from the method input parameter structure. But that is also a problem if more data comes with a parameter than the model is prepared of. For example if an airlines introduces new service (like pets on board) both the data model and the service interface need to be changed. What do we have at the end? A rigid, hardly scalable, complex service. Of course it is still much better than having even the presentation in the same layer… But still: is that really what we wanted?

OK. Any better option? Yes. Here comes the MicroService implemented as a RESTful Web Service. Why not MicroService implemented as a SOAP Web Service? Because SOAP freezes the input and output structures by the WSDL/XSD structure definitions. Why not monolithic service implemented as a RESTful Web Service? Because even if the input and output structures are flexible, the internal data model is rigid. So we need flexibility defining the structures accepted, stored, emitted by the service. Is that it? Not yet. There is also a pressure to make the definition of a service as close to a business object as possible and minimize the required data set describing an object to the absolute smallest.

Now I think I am able to share my definition of a MicroService. A MicroService manages basic business object type or resource class. The minimum set of data required by the service is defined but extensible without any limitation. The minimum set is determined by the functional requirement of the service. Or from the apposite point of view: if a data is not required by any function to perform, than it goes to an extension. Inside the service any extension to the core dataset is accepted, stored, and emitted but not managed.

Before we take a look how our example can be architected, let me explain the name “ NoSQL Service ” . In the data modelling for many years SQL (relational modelling) was (and still is) dominant. Relational databases were using rigid data structures. As an answer to it the NoSQL concept was introduced. NoSQL had the “ invention ” of schemeles data. Like NoSQL unchained the database engines from managing data driven by dictionary, MicroService unchains services from managing complex data structures. So MicroService is the NoSQL of service architecting.

How would then our example architected? First of all we need to decompose the monolithic service into resource classes. We are going to have:

n BookingSubject – something bookable. In our case it will be a seat on a flight. Each resource has a unique identifier. Like airlines/ date/ flight number/ row number/ seat. E.g. OK/2016-10-29/361/23/D. The attribute of the resource can be a status (free, temporarily reserved, reserved) and a booking reference. Anything else is just extension. Like origin, destination, departure time, unit price, class, pets on board, smoking, catering options.

n Booking – reservation of BookingSubjects. A collection of bookable subjects. The booking has an expiration date/ time to handle temporary blocking. Other than that it has a unique ID and the list of booked items.

n Person – a passenger. Optional. Our example in one of the previous sessions was a “ person ” service. What makes a person to a passenger? She/ He has a ticket. When optionally the airlines maintains a person registry, the ticket could refer to a registry entry.

n Ticket – an authorization to use a flight. The ticket is the unique reference to bookable subject and the person to whom it is reserved. Because the ticket does not really care about the persons details, the ticket structure must have an unambiguous identification of a ticket owner. It could be just a name (mostly unique on a flight) or a passport number (but not both). But if for whatever reason the airlines maintains a person registry, the ticket could

refer to a registry entry.

And some more. How does this model replace the original service?

n Search – get a collection of *BookingSubject* matching the search criteria. The search takes both the core and the extension data and match to the advanced search criteria. The returned collection contains the ID of the matched resources.

n Block – create a new *Booking*. Set the expiration to *floor(now+15minutes,departure time-50minutes)*. Add the selected *BookingSubjects* to it. Update the individual *BookingSubjects* status to *reserved*.

n Allocate ticket – create a new *Ticket*. Make ticket referring to *Booking*. Add a person info.

n Finalize booking – change the *Booking* expiration to *never*.

What if we need to create a hotel booking system? What needs to be changed? What can be reused? I leave the answers on you.

Implementation patterns

I think the previous articles in the series gave quite a useful set of implementation patterns when you are developing a MicroService for Ensemble as a beginner.

n We discussed how to create a service.

n How and why to enable CORS .

n Consume a service as a client.

n How to pass service parameters to a service.

n We also learnt handling exceptions.

n As well as to design a service API.

I think we are almost done. There is one missing piece from the puzzle. The " bonus track " .

So I say for the last time: Stay tuned, I ' ll be back soon with further reading on Ensemble RESTful web services. The next is " Applying MicroServices Architecture to Ensemble " .

#Beginner #REST API #Ensemble