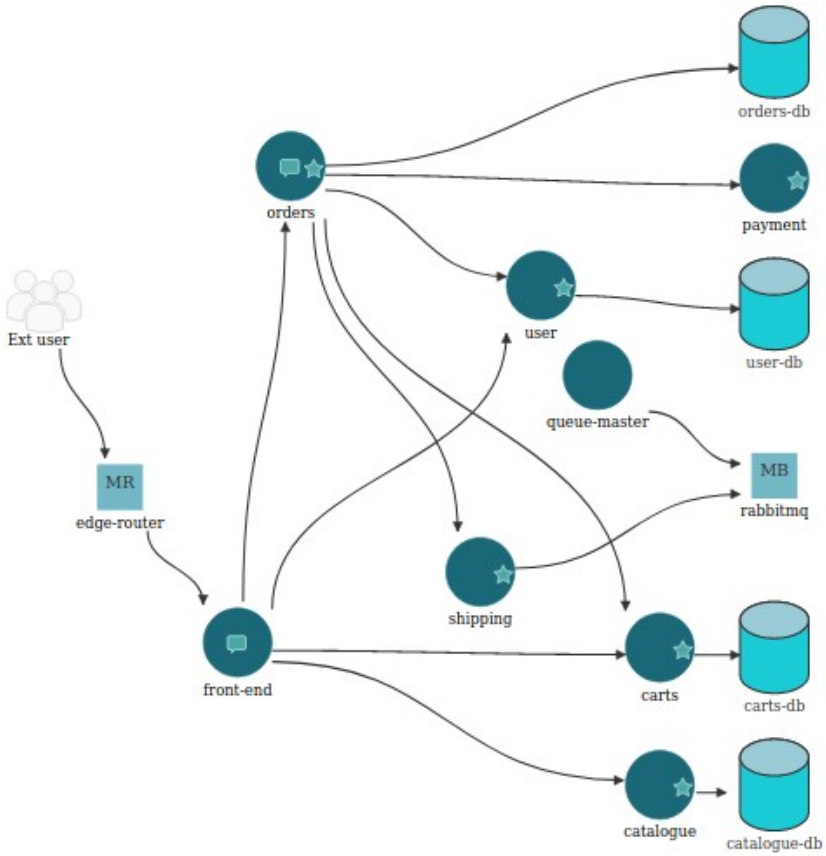


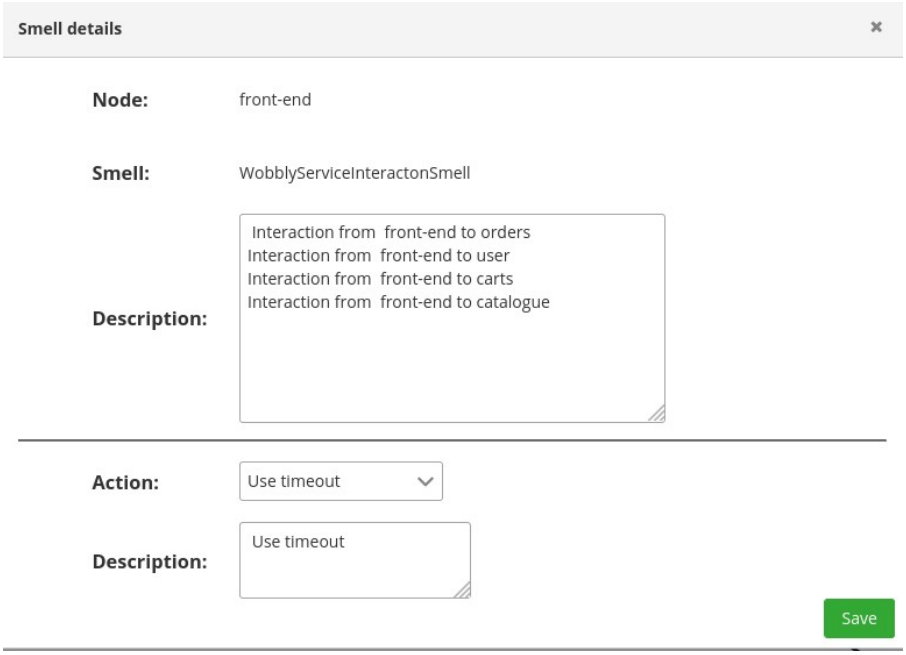
Architectural scheme of SockShop which presents 8 smells of two main types:

-Wobbly service interaction smells: they go against the isolation of failure design principle according to which failures within a microservices application must be isolated to the only microservice that fails. On the contrary, if an interaction between two services m1 and m2 is "wobbly", the failure of m1 will propagate in m2.

-Endpoint-based service interaction smells: they undermine the horizontal scalability design principle according to which, microservices that enhance an application should be scalable horizontally. In practice this leads to the possibility of adding / removing replicas of individual services according to their usefulness. If,there is a EBSI smell instead, the service would invoke a specific instance of a certain resource, eliminating the possibility of scaling that resource.

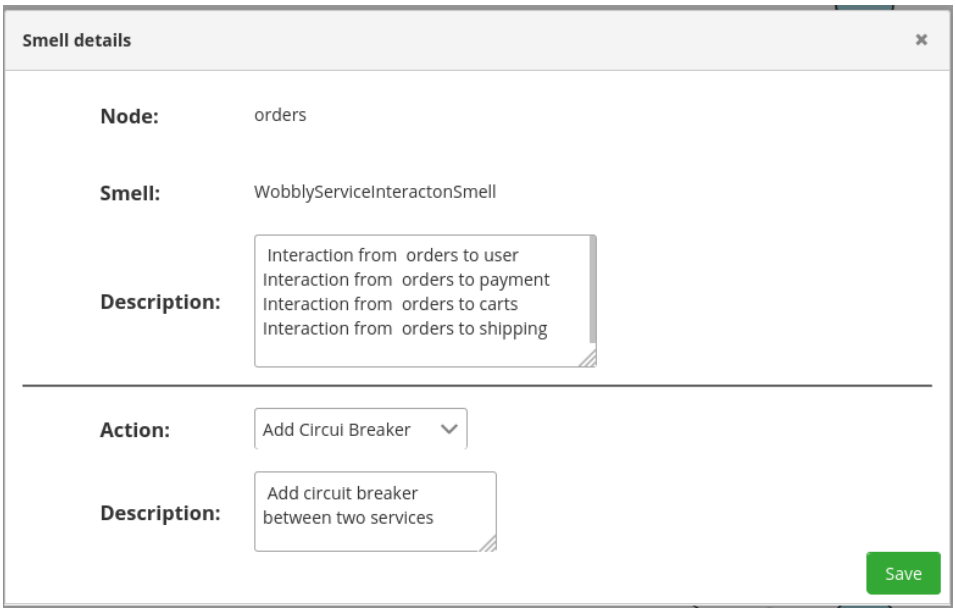
List of refactoring performed:

1)



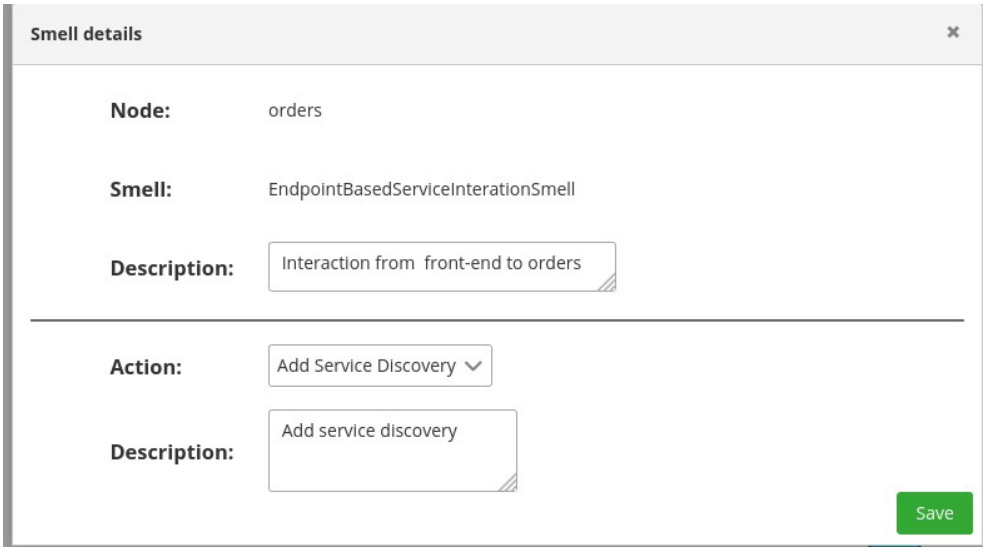
The first smell we deal with, is the "wobbly" interaction between the "front-end" node and all the service nodes with which it interacts, which are: "orders", "user", "carts", "catalog ".
The refactoring that I decided to adopt to solve this smell is the use of the timeout on each of the "front-end" interactions such a way as not to block any resources for an indefinite time in the absence of a response from one of the nodes with which it interacts.

2)



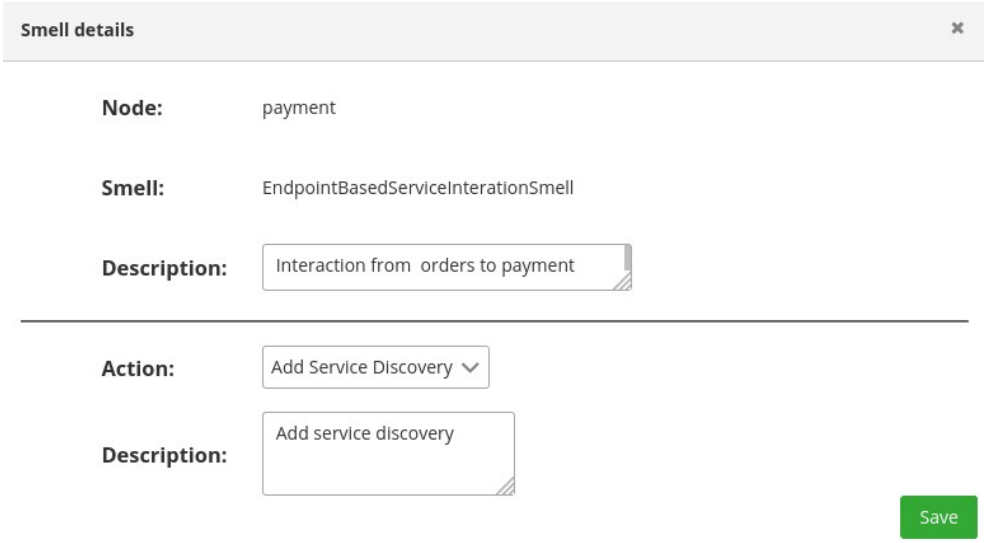
This refactoring always concerns a "Wobbly service interaction" smell, this time, however, between the "order" node and all the service nodes with which it interacts, namely: "user", "payment", "carts", "shipping". This time to solve the smell I decided to adopt a circuit breaker for each of these interactions in order to interrupt the requests in case there is a failure of one of the concerned services.

3)



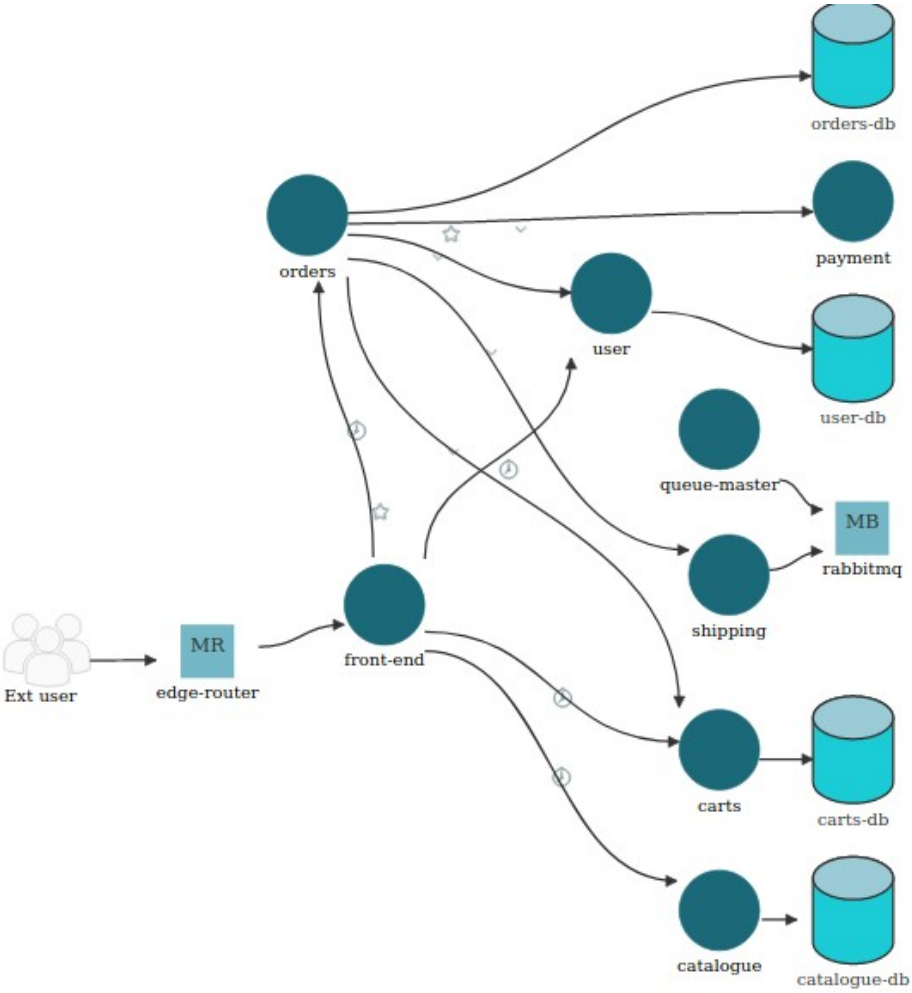
Then I moved on to Endpoint-based service interaction smells. The first one to which I decided to apply a refacroring is the one that interested the interaction between the "front-end" and "orders" nodes. I applied a service discovery to this interaction to ensure that the "orders" service can scale horizontally and then add / remove replicas if needed.

4)

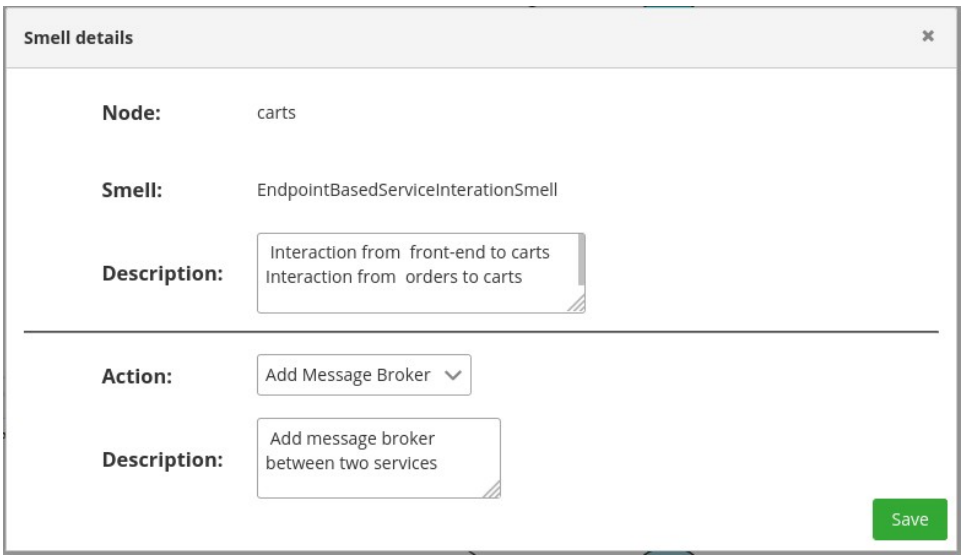


In the same way I solved the EBSI smell concerning the interaction between the "orders" and "payment" service.

SockShop current architectural scheme:



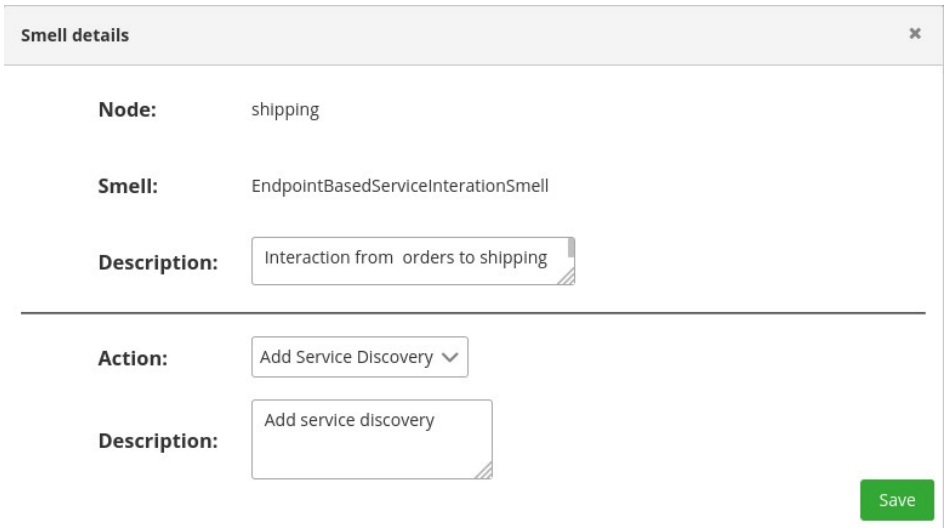
5)



In this case we have to deal with an EBSI smell that concerns the interaction from "front-end" to "carts" and that from "orders" to "carts". The refactoring involved the addition of a single Message Broker for the management of front-end requests and orders so as to also embrace the principle of reusability of resources. The addition of a message queue will facilitate the switching

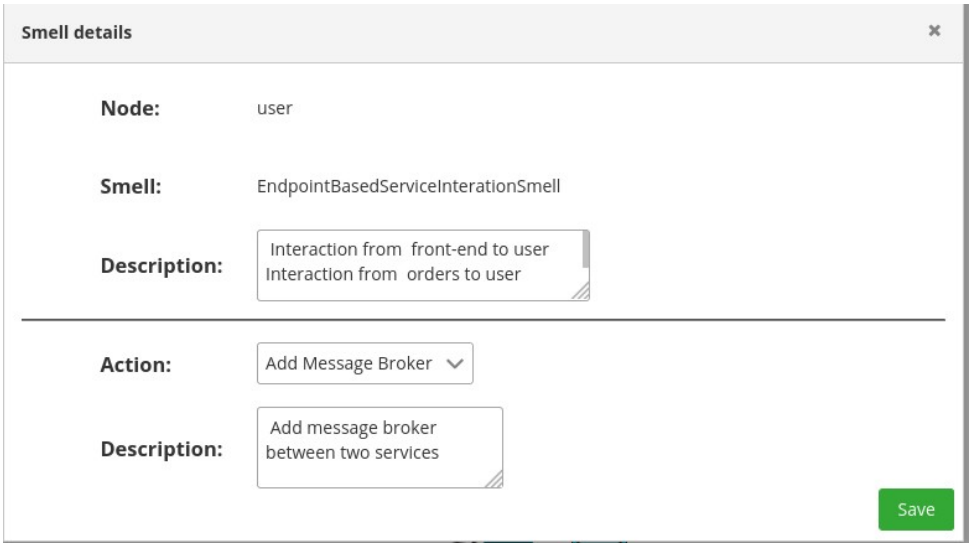
of requests between the various instances of the "carts" service in the event that it will have to scale horizontally.

6)



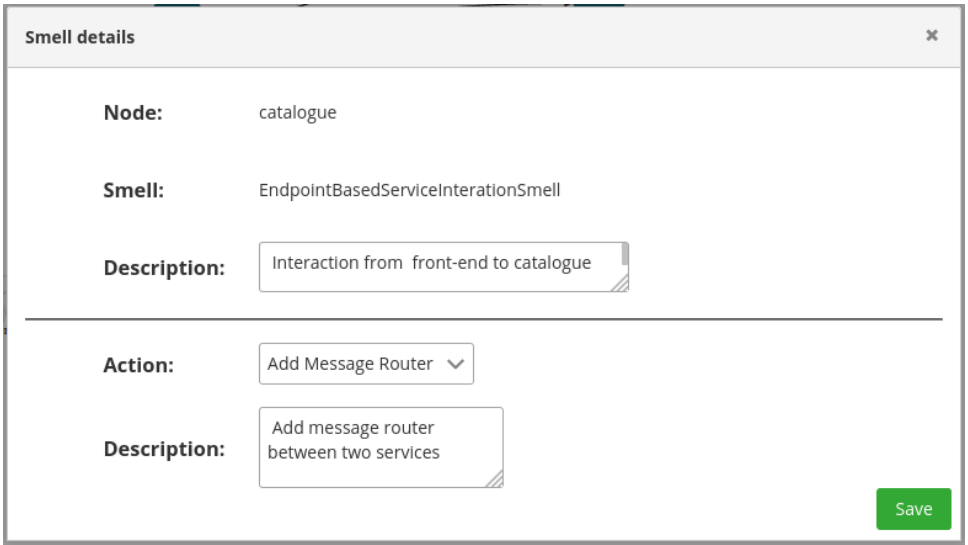
Also in this case of EBSI smell, the refactoring applied was the addition of a service discovery.

7)



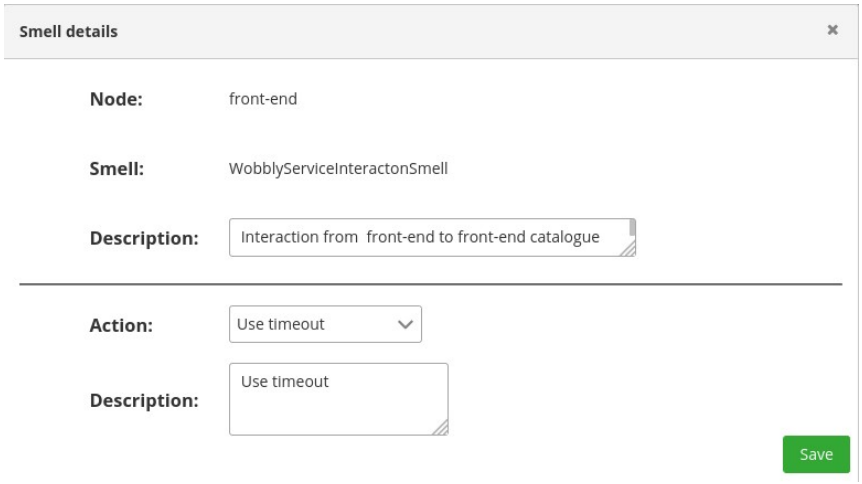
As in the previous case in which two different services required an EBSI with a third service this refactoring involved the addition of a single Message broker for switching requests .

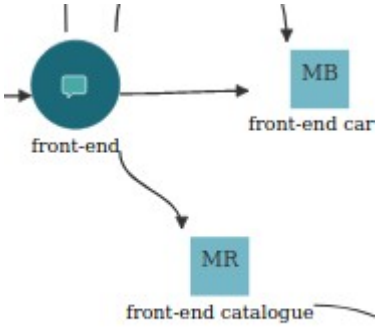
8)



The last refactoring went to correct the last EBSI smell, which involved the interaction between the "front-end" node and the "catalog" node. This time I chose to add a Message router. It eliminates the EBSI problem which forced the front-end to invoke a specific instance of the catalog, providing the possibility to scale the resources of the “catalog” node thanks to the load balancing of the requests received.

The addition of the message router gave birth to a new wobbly service interaction smell that I solved by adding a timeout as shown in the figure below.





The Last SockShop architectural scheme: