DP2-1920-GI-02: Sprint 4 June 1, 2020

**PROFILING AND OPTIMIZATION**

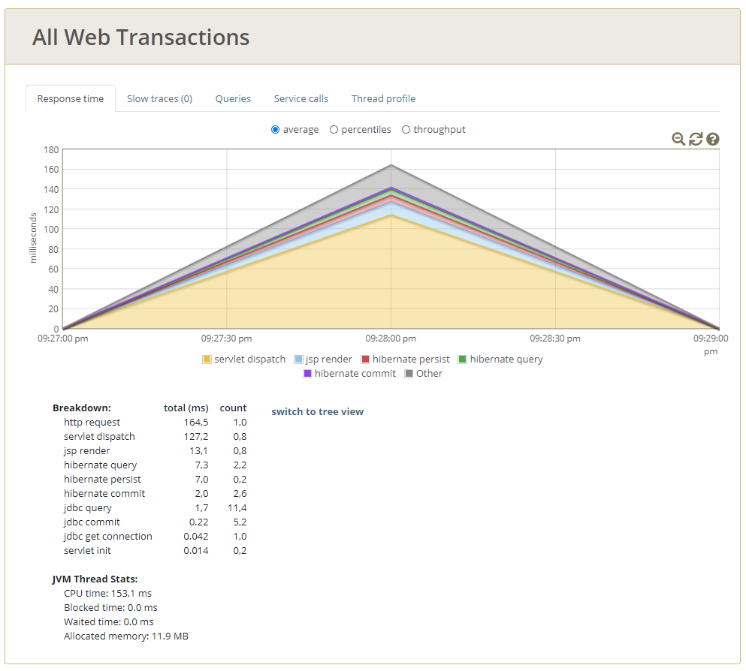
In part I of this document, we perform four different profilings using glowroot and analyze the inefficiencies detected. In part II we optimize the application by applying refactorizations based on three of those profilings.

**PART I: PROFILING**

**Profiling 1**

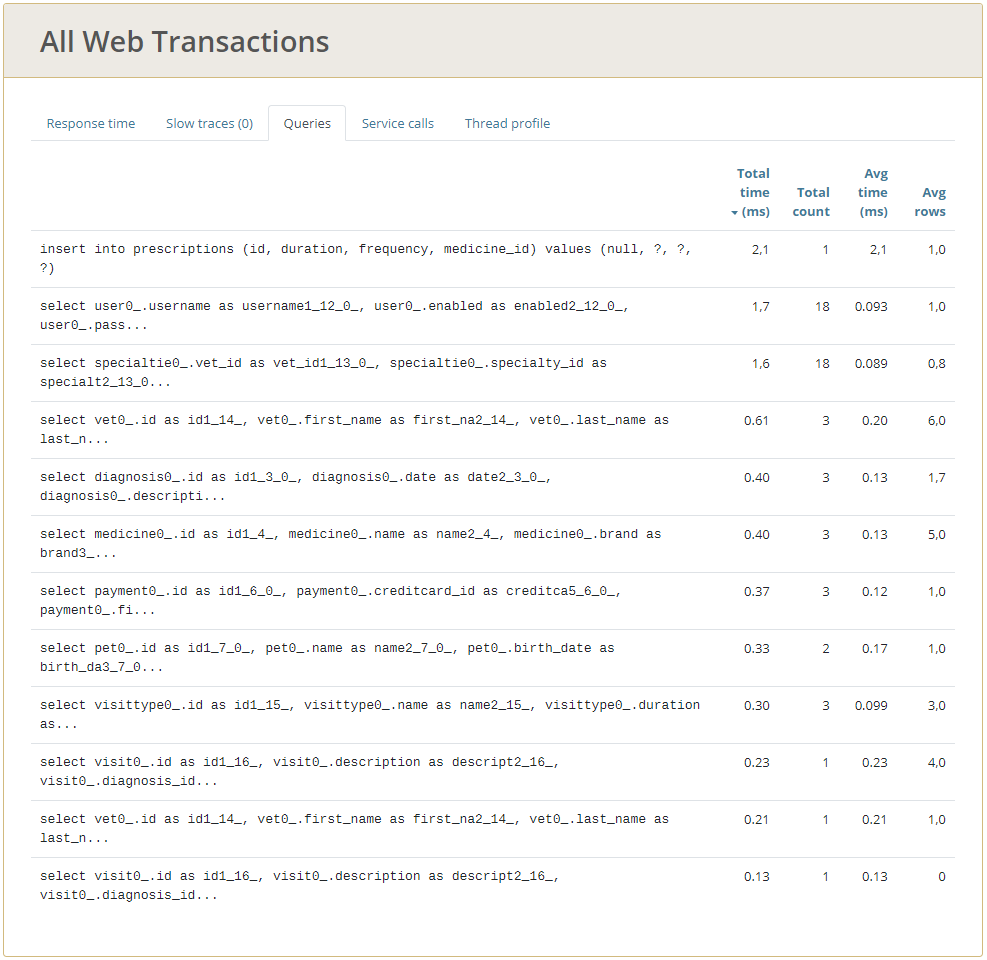
In the performance report, we identified the user story US-14 (a vet adds a new prescription to an existing diagnosis) as the bottleneck of the whole application. Therefore, we selected it for a more thorough analysis.

Using glowroot, we can visualize the proportion each stage of the process adds to the total loading time:



As can be seen in the above figure, we began to simulate the scenario at 9:27:00 pm. The peak of 160 milliseconds, one minute later, reflects the time the application took to respond to our requests.

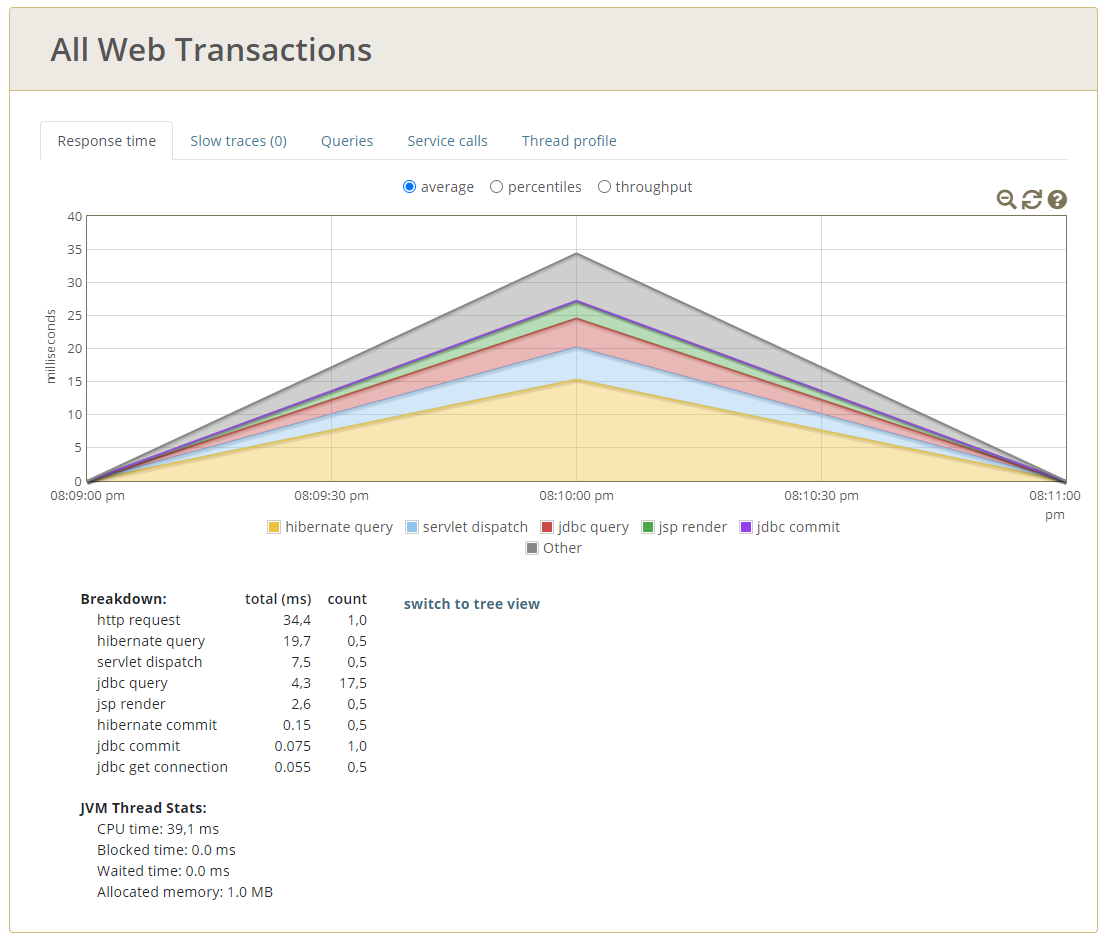
Compared to the response time of other scenarios, including the ones in this document, this delay is extraordinarily long. In order to find out its origin, we investigate the queries performed to the database:



As can be seen in the figure above, a total of 12 queries are made to the database. We assume that the cause of this unusually high number is the complex nature of the user story and the fact that it involves many different objects: A prescription is added by a vet to a diagnosis, which in turn is associated to a visit made by an owner for one of his or her pets.

**Profiling 2**

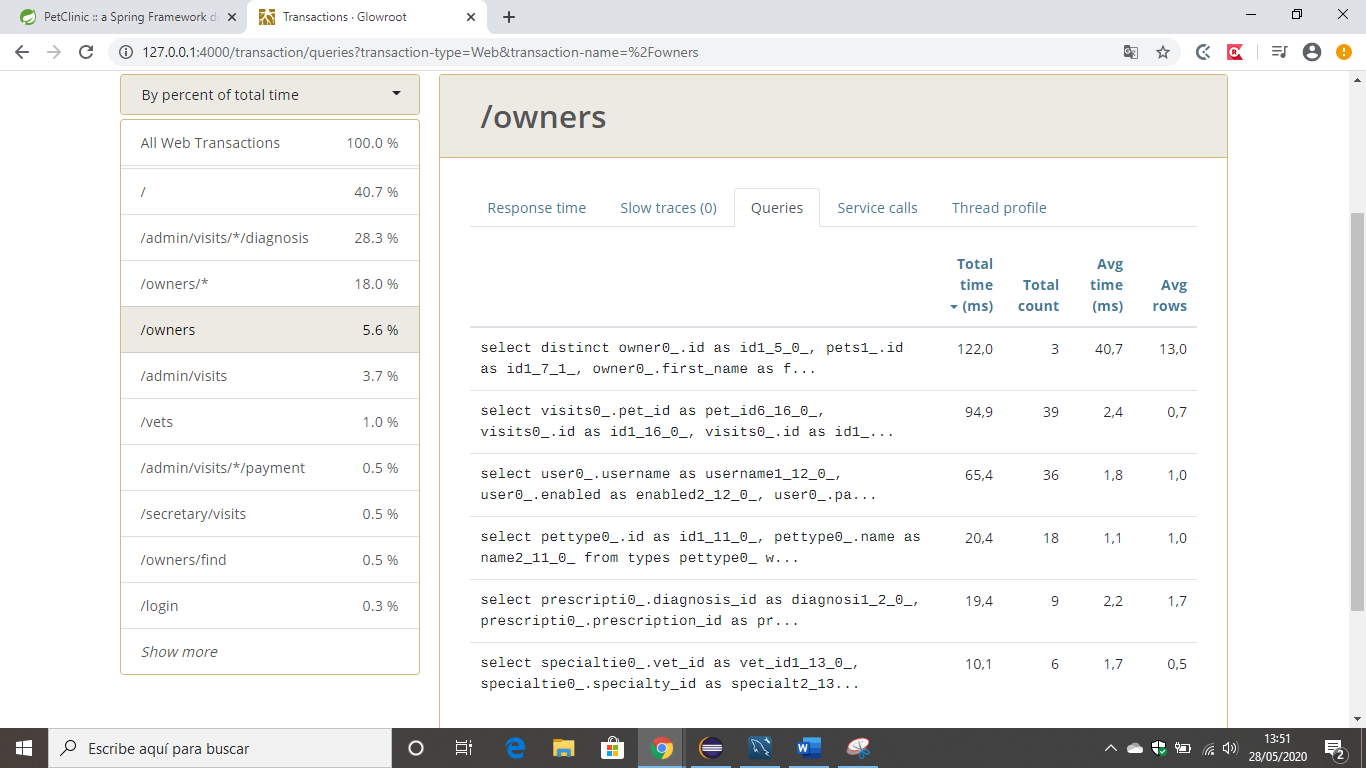
This profiling is based on a functionality that we adapted from the template project and that does not have any corresponding user story in our requirement catalogue: An admin can show a list of all the owners registered in the system.



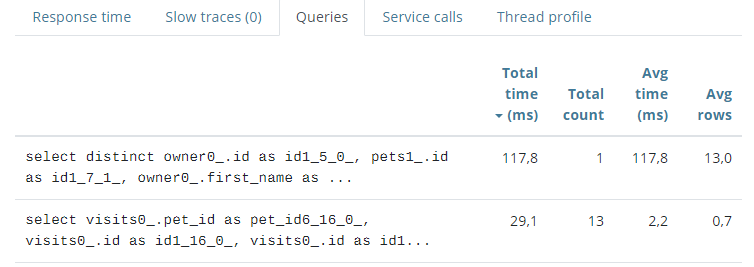
A N+1 Query problem has been detected: When the view of all owners (/owners) is loaded, all the owners and pets of each one appear. It has been detected that, in that view, for each pet that appears the visits of each one are loaded.

In our example data we have 13 pets associated with different owners, so for each pet that we have included in our database, 13 queries are made that return the visits of each pet has.

The queries that are made can be seen with:



When that view is loaded once:



When that view is loaded twice :

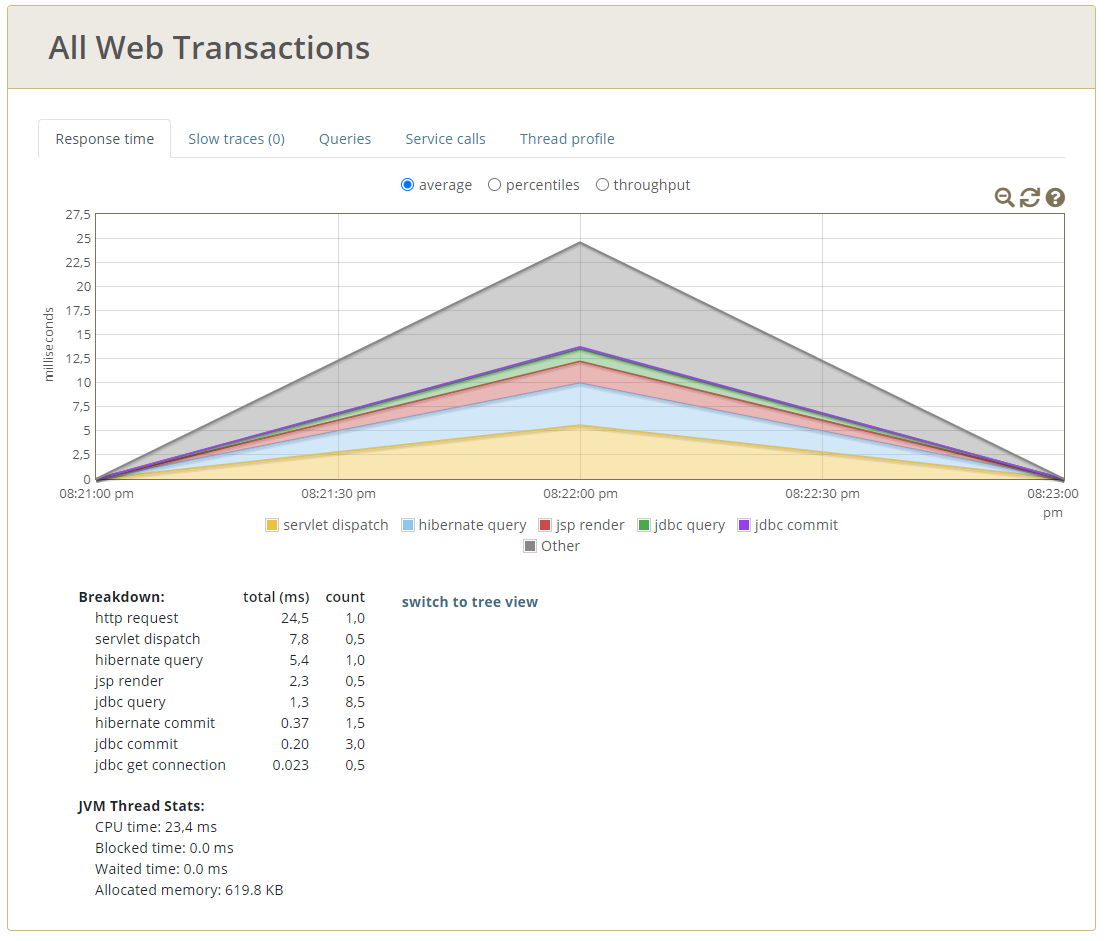


When that view is loaded three times:

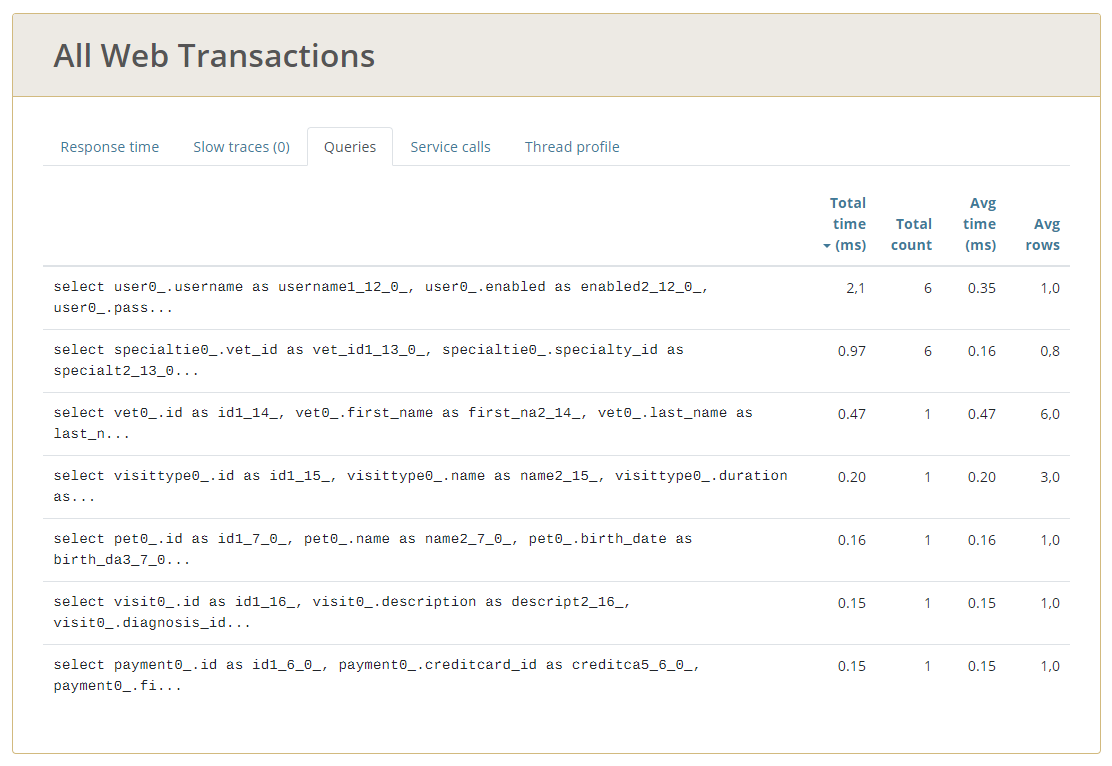


**Profiling 3**

When accessing the view dp2.com/vet/visits/8 as a vet, as specified in the user story US-7:



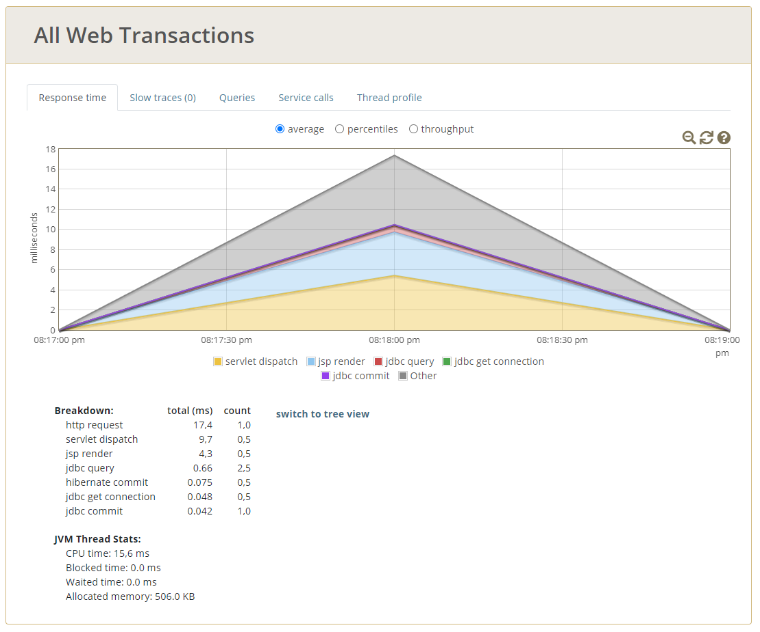
Seven queries are made to the database:



We consider this number of queries to be unnecessary and suggest using a cache in order to optimize the performance.

**Profiling 4**

When accessing the view dp2.com/owners/1 as an admin:



5 queries are made to the database, even though the data could be stored in a cache:

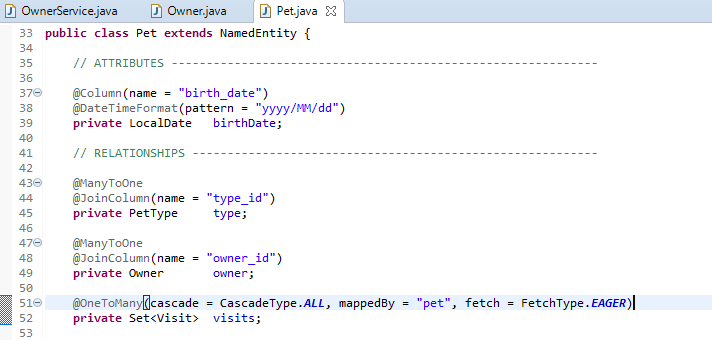


**PART II: OPTIMIZATION BY REFACTORING**

**Refactoring based on profiling 2**

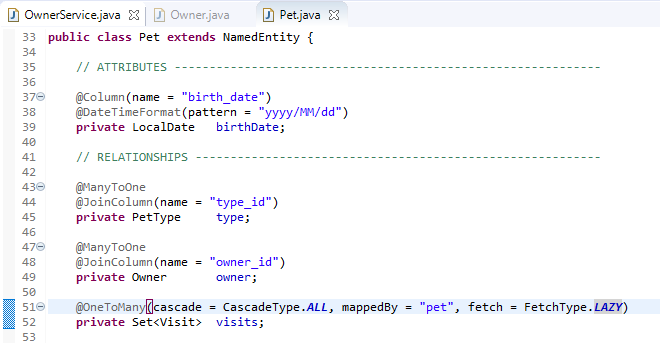
Problem:

It has been detected in the model that the relationship of pet with visits was of type **.EAGER**, which means that whenever a pet is loaded, it’s visits are loaded.



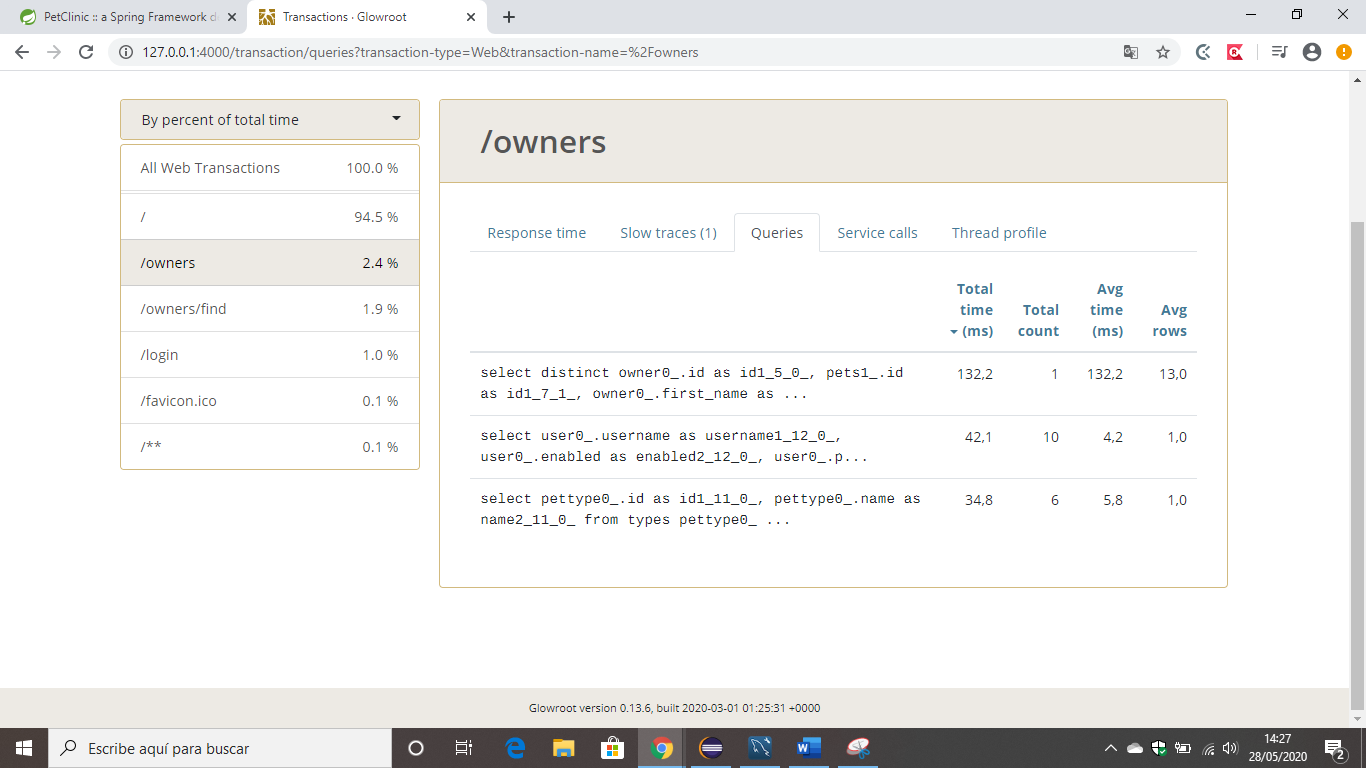
Solution:

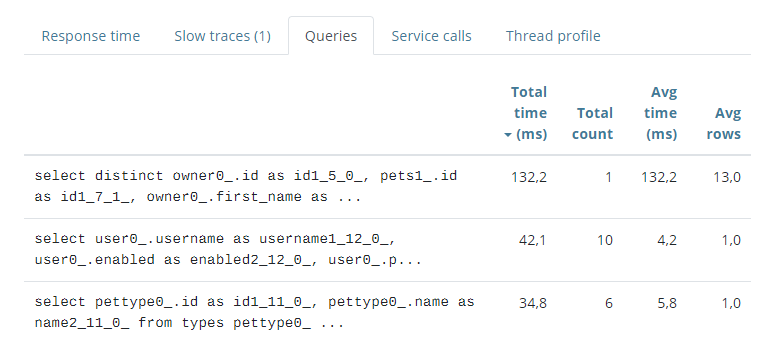
It has been changed and not the relationship pet-visits has been set to type .LAZY so that ii only loads when necessary (since visits is something we don’t need in the /owners view we are talking about).



Effects:

In this way, now in Glowroot you can see that those N Querys that were made for each pet in our database have disappeared.





**Refactoring based on profiling 3**

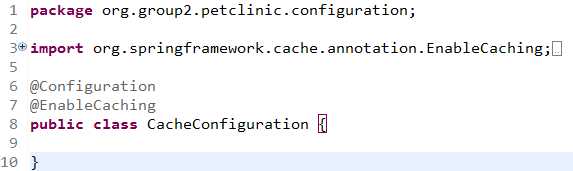
Problem:

As discussed previously, queries are made to the database that could be avoided by using caches.

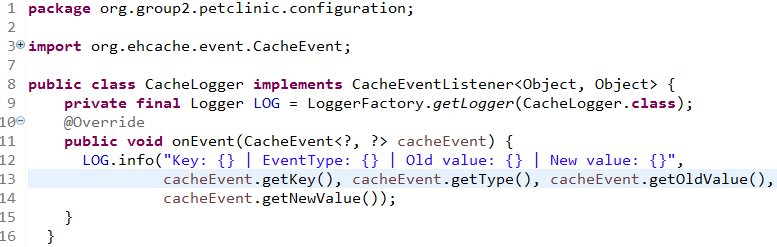
Solution:

We added a cache for findVisitById.

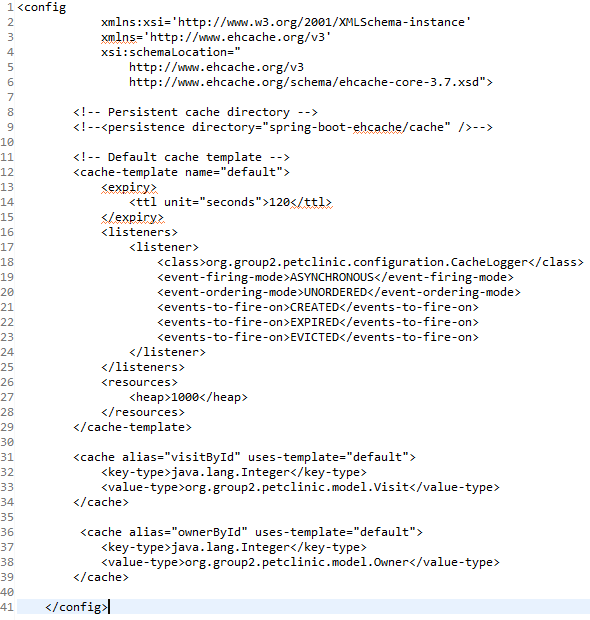
First, we added the cache configuration as explained in the video on EV:



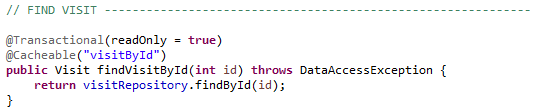
We added a cache logger:

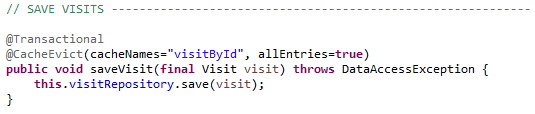


We added the ehcache3 template:



We added the necessary annotations:





Effects:

Now, 4 queries are made to the database when the view dp2.com/vet/visits/8 is loaded, while previously it was 7. With the cache, we were able to avoid 3 queries.



**Refactoring based on profiling 4**

Problem:

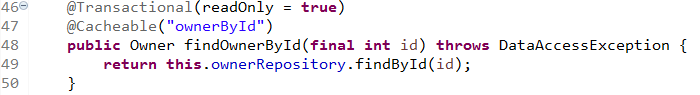
As discussed previously, queries are made to the database that could be avoided by using caches.

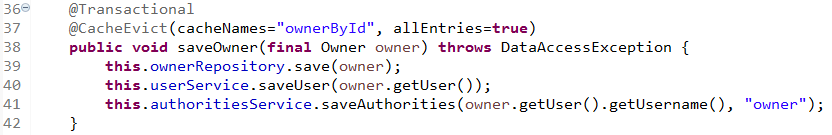
Solution:

We added a cache for findOwnerById.

We did not have to add the cache configuration as we already added it during the previous profiling (profiling 2).

We added the necessary annotations:





Effects:

With the cache, no more queries are made to the database.

