DP2 – PROFILING REPORT

GI-01

Diāna Bukša

Manuel Cañizares Juan

Yoana Dimitrova Penkova

Iván Menacho Gallardo

Álvaro Rubia Tapia

Index

Introduction 3
US-001 Vet adds a new medicine 4
US-009 Vet changes a pet’s medical record 7
US-011 Vet manages homeless pets information 11
US-017 Owner sees vet’s personal information 16
US-024 Owner sees trainer’s personal information 19

Introduction

In order to produce this report, we decide to assign a profiling to each member in the group so there would not be members with this task and others that does not even if the mandatory number user stories for this level is 3.

Each member took the user story that worst result came from the load testing. So, the distribution is as it follows:

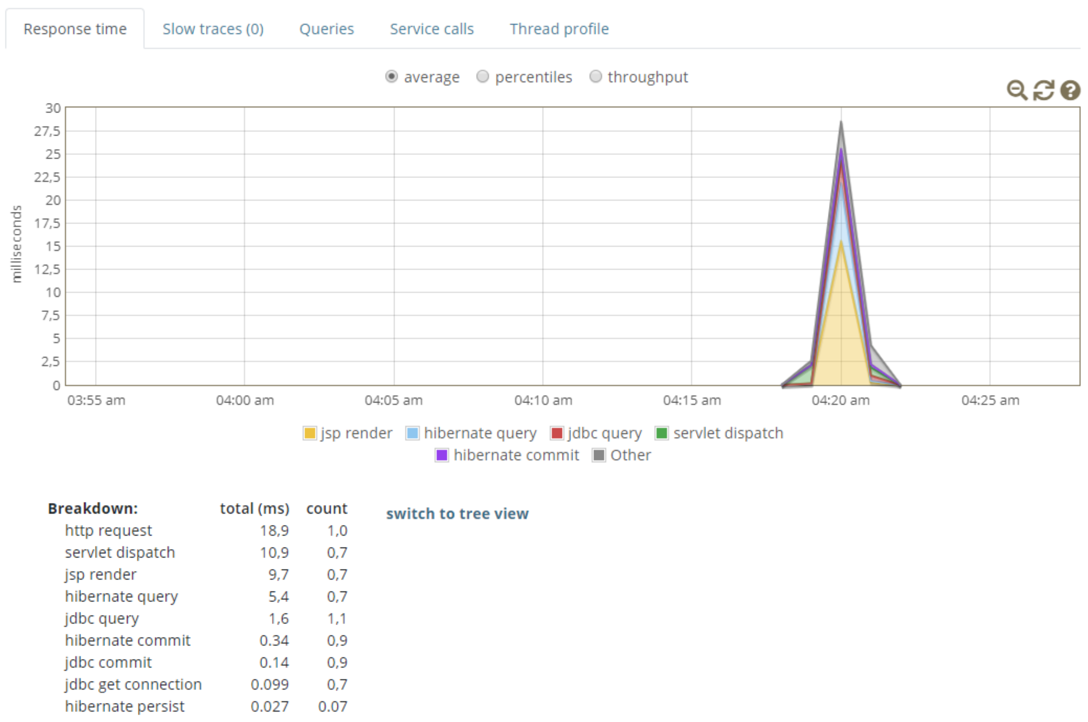
- **Diāna Bukša:** US-024 Owner sees trainer’s personal information
- **Manuel Cañizares:** US-001 Vet adds a new medicine
- **Yoana Dimitrova:** US-001 Vet manages homeless pets’ information
- **Iván Menacho:** US-009 Vet changes a pet’s medical record
- **Álvaro Rubia:** US-017 Owner sees vet’s personal information

In each user story, we can find its own conclusion.

US-001 Vet adds a new medicine

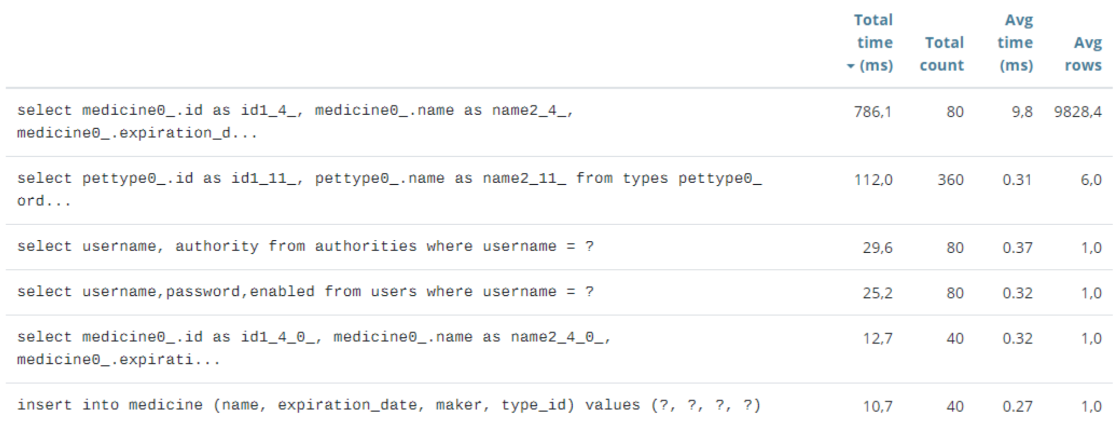
The profiling was done with the US 001 that is related to the creation and show of a medicine. This US is related to the US 002, which feature is about listing all the available medicines in the system. Due to the way is the navigation through the views done, to create a new medicine is mandatory to list all the medicine first.

Because of this casuistry, this is the US which has the worst performance in the load tests Manuel has run, so he studied about its performance in Glowroot simulating about 40 users for 50 seconds, and this is what I got from.



Breakdown:	total (ms)	count
http request	18,9	1,0
servlet dispatch	10,9	0,7
jsp render	9,7	0,7
hibernate query	5,4	0,7
jdbc query	1,6	1,1
hibernate commit	0.34	0,9
jdbc commit	0.14	0,9
jdbc get connection	0.099	0,7
hibernate persist	0.027	0.07

As we can see in the figure, one of the things that is drawing out the performance, are the jdbc and hibernate queries during the process. If we go in depth in those aspects, the following results are shown up:

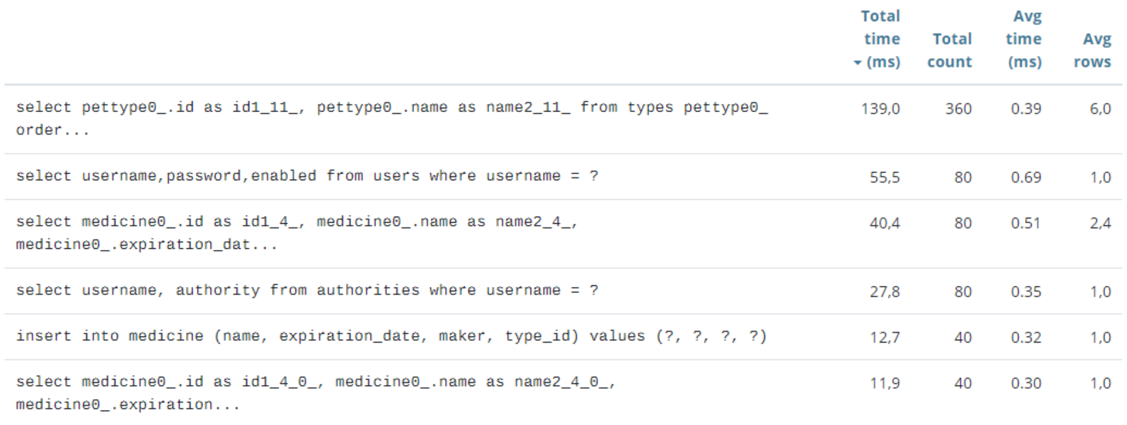


	Total time ▾ (ms)	Total count	Avg time (ms)	Avg rows
`select medicine0_.id as id1_4_, medicine0_.name as name2_4_, medicine0_.expiration_d...`	786,1	80	9,8	9828,4
`select pettype0_.id as id1_11_, pettype0_.name as name2_11_ from types pettype0_ ord...`	112,0	360	0.31	6,0
`select username, authority from authorities where username = ?`	29,6	80	0.37	1,0
`select username,password,enabled from users where username = ?`	25,2	80	0.32	1,0
`select medicine0_.id as id1_4_0_, medicine0_.name as name2_4_0_, medicine0_.expirati...`	12,7	40	0.32	1,0
`insert into medicine (name, expiration_date, maker, type_id) values (?, ?, ?, ?)`	10,7	40	0.27	1,0

The most prominent aspect is the query to request the pet types to the database done when displaying a list of them in the creation form and the one which is done during the medicine listing process, due to the default fetch configuration. The second reason is an example of a common problem, the N + 1 queries one, and this can be fixed changing the fetch configuration manually in the entity definition:

```
@ManyToOne(optional = false, fetch = FetchType.LAZY)
@JoinColumn(name = "type_id")
@NotNull
private PetType petType;
```

Once this change is done, we simulate again with the same parameters and we got this:



	Total time (ms)	Total count	Avg time (ms)	Avg rows
select pettype0_.id as id1_11_, pettype0_.name as name2_11_ from types pettype0_ order...	139,0	360	0.39	6,0
select username,password,enabled from users where username = ?	55,5	80	0.69	1,0
select medicine0_.id as id1_4_, medicine0_.name as name2_4_, medicine0_.expiration_dat...	40,4	80	0.51	2,4
select username, authority from authorities where username = ?	27,8	80	0.35	1,0
insert into medicine (name, expiration_date, maker, type_id) values (?, ?, ?, ?)	12,7	40	0.32	1,0
select medicine0_.id as id1_4_0_, medicine0_.name as name2_4_0_, medicine0_.expiration...	11,9	40	0.30	1,0

Although the count of queries to get the pet type has not changed at all, we can see an enormous difference in the total elapsed time to request all the medicines for the listing, which variance is of 746 ms.

Even that, the excessive number of queries to request the pet type is present yet, so in order to fix this problem, some cache configuration will be added to the project in the methods that request this info.

First, Manuel added those entries in the *ehchache3.xml* file:

```
<cache alias="petTypes" uses-template="default">
      <key-type></key-type>
      <value-type>java.util.Collection</value-type>
</cache>
<cache alias="medicines" uses-template="default">
      <key-type></key-type>
      <value-type>java.util.Collection</value-type>
</cache>
```

Then, Manuel set the methods which will load the cache on their call:

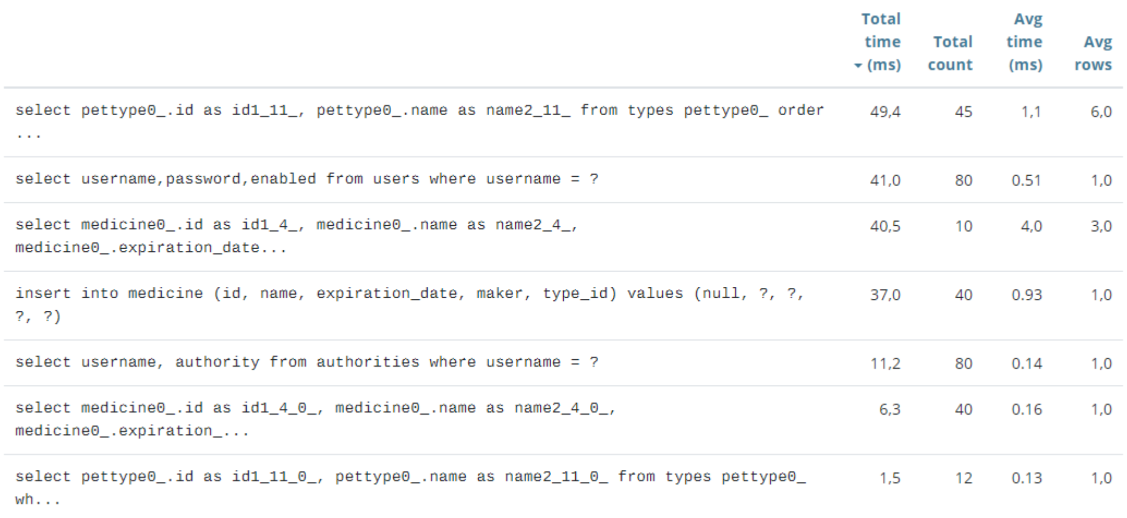
```
@Transactional(readOnly = true)
@Cacheable("petTypes")
public Collection<PetType> findPetTypes() throws DataAccessException {
    return this.petRepository.findPetTypes();
}
```

```
@Transactional(readOnly = true)
@Cacheable("medicines")
public Collection<Medicine> findManyMedicineByName(String name) {
    return repository.findByNameContaining(name);
}


@Transactional(readOnly = true)
@Cacheable("medicines")
public Collection<Medicine> findByPetType(PetType petType) {
    return repository.findByPetType(petType);
}
```

And finally, Manuel added the necessary annotations to use the cached data when a request method is called:

```
@Transactional
@CacheEvict(cacheNames = {"petTypes", "medicines"}, allEntries = true)
public void saveMedicine(Medicine medicine) {
    repository.save(medicine);
}

@Transactional
@CacheEvict(cacheNames = {"petTypes", "medicines"}, allEntries = true)
public void deleteMedicine(Medicine medicine) {
    repository.delete(medicine);
}
```

After all those changes, Manuel ran the simulation one more time:

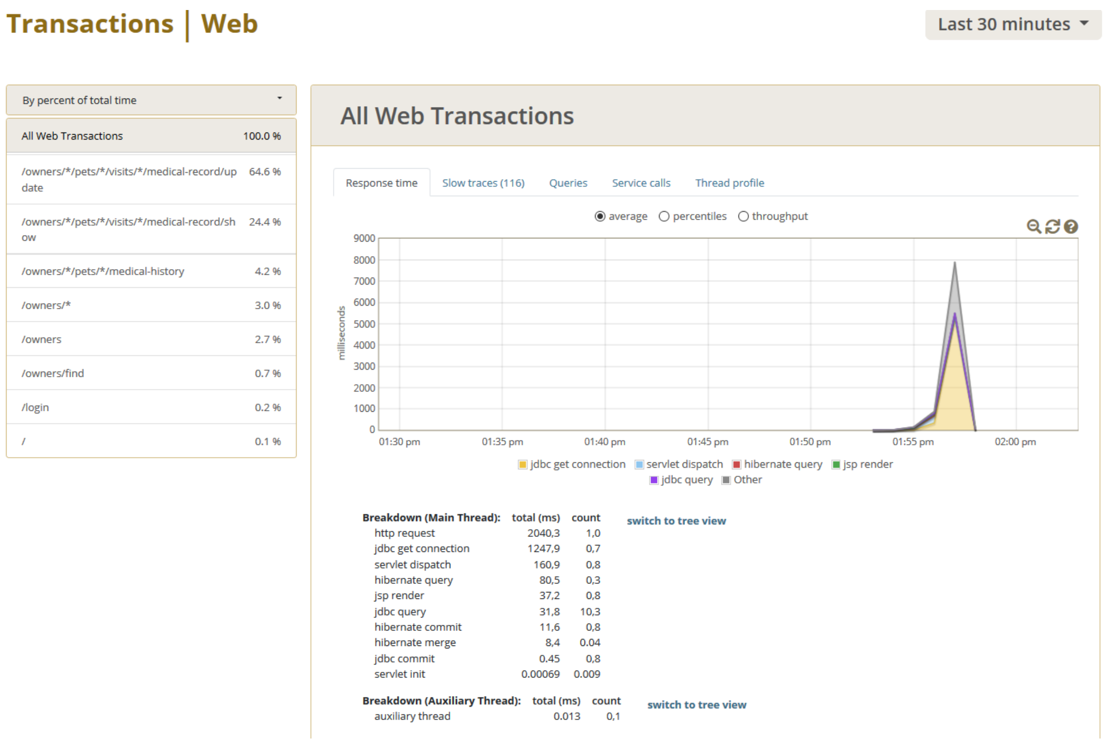


	Total time ▾ (ms)	Total count	Avg time (ms)	Avg rows
`select pettype0_.id as id1_11_, pettype0_.name as name2_11_ from types pettype0_ order ...`	49,4	45	1,1	6,0
`select username,password,enabled from users where username = ?`	41,0	80	0.51	1,0
`select medicine0_.id as id1_4_, medicine0_.name as name2_4_, medicine0_.expiration_date...`	40,5	10	4,0	3,0
`insert into medicine (id, name, expiration_date, maker, type_id) values (null, ?, ?, ?, ?)`	37,0	40	0.93	1,0
`select username, authority from authorities where username = ?`	11,2	80	0.14	1,0
`select medicine0_.id as id1_4_0_, medicine0_.name as name2_4_0_, medicine0_.expiration_...`	6,3	40	0.16	1,0
`select pettype0_.id as id1_11_0_, pettype0_.name as name2_11_0_ from types pettype0_ wh...`	1,5	12	0.13	1,0

This time, we can notice a great difference between some total times and query executions. Specifically, there are 303 less queries to request all the pet types and 70 less queries to request all the medicines, which significantly reduces the amount and time spent executing those queries.

US-009 Vet changes a pet’s medical record

The US that was chosen for the profiling was the US 009. Iván chose it because it gave us the worst performance and was the one which more potential to be improve. In order to analyze the code, he used GlowRoot.



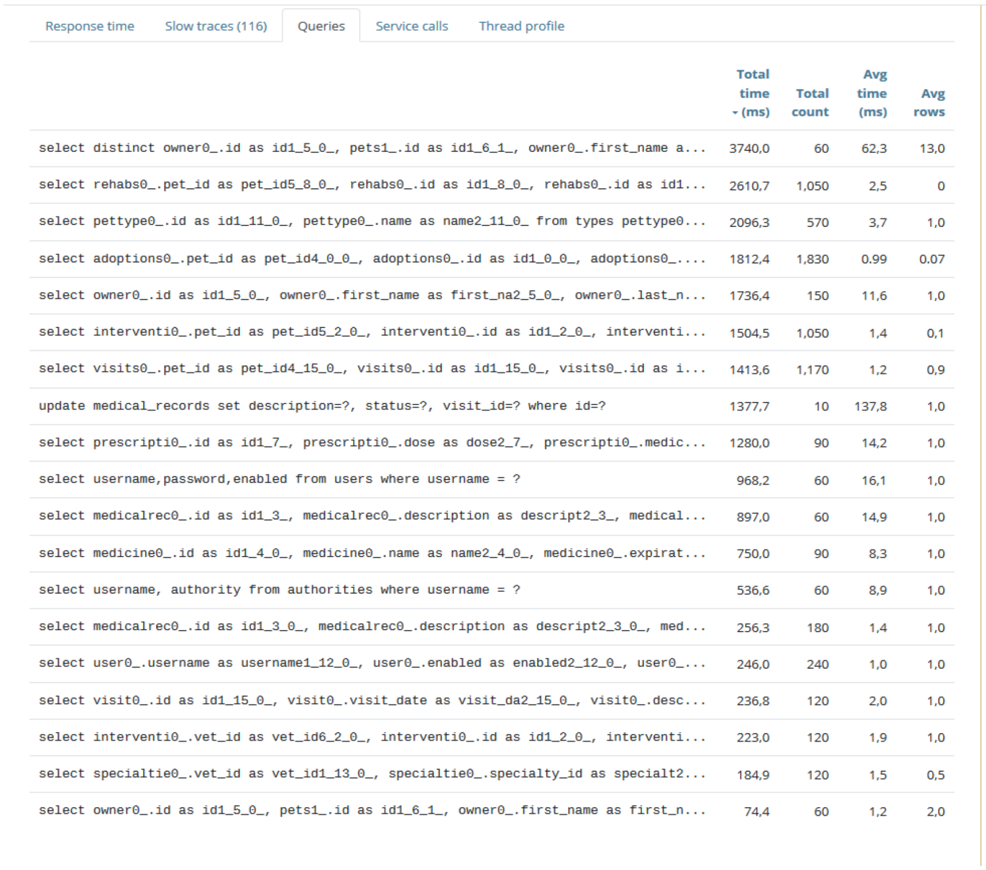
All Web Transactions	100.0 %
/owners/*/pets/*/visits/*/medical-record/update	64.6 %
/owners/*/pets/*/visits/*/medical-record/show	24.4 %
/owners/*/pets/*/medical-history	4.2 %
/owners/*	3.0 %
/owners	2.7 %
/owners/find	0.7 %
/login	0.2 %
/	0.1 %

Breakdown (Main Thread):	total (ms)	count
http request	2040,3	1,0
jdbc get connection	1247,9	0,7
servlet dispatch	160,9	0,8
hibernate query	80,5	0,3
jsp render	37,2	0,8
jdbc query	31,8	10,3
hibernate commit	11,6	0,8
hibernate merge	8,4	0.04
jdbc commit	0.45	0,8
servlet init	0.00069	0.009

switch to tree view

Breakdown (Auxiliary Thread):	total (ms)	count
auxiliary thread	0.013	0,1

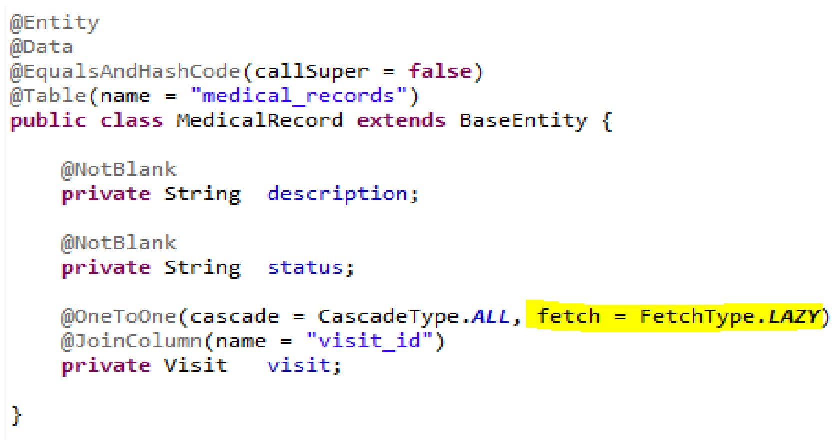
switch to tree view



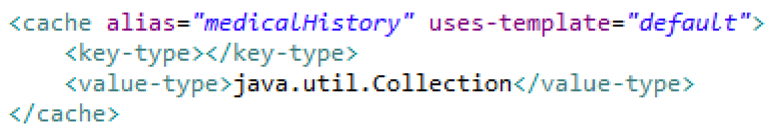
	Total time ▾ (ms)	Total count	Avg time (ms)	Avg rows
select distinct owner0_.id as id1_5_0_, pets1_.id as id1_6_1_, owner0_.first_name a...	3740,0	60	62,3	13,0
select rehabs0_.pet_id as pet_id5_8_0_, rehabs0_.id as id1_8_0_, rehabs0_.id as id1...	2610,7	1,050	2,5	0
select pettype0_.id as id1_11_0_, pettype0_.name as name2_11_0_ from types pettype0...	2096,3	570	3,7	1,0
select adoptions0_.pet_id as pet_id4_0_0_, adoptions0_.id as id1_0_0_, adoptions0_....	1812,4	1,830	0.99	0.07
select owner0_.id as id1_5_0_, owner0_.first_name as first_na2_5_0_, owner0_.last_n...	1736,4	150	11,6	1,0
select interventi0_.pet_id as pet_id5_2_0_, interventi0_.id as id1_2_0_, interventi...	1504,5	1,050	1,4	0,1
select visits0_.pet_id as pet_id4_15_0_, visits0_.id as id1_15_0_, visits0_.id as i...	1413,6	1,170	1,2	0,9
update medical_records set description=?, status=?, visit_id=? where id=?	1377,7	10	137,8	1,0
select prescripti0_.id as id1_7_, prescripti0_.dose as dose2_7_, prescripti0_.medic...	1280,0	90	14,2	1,0
select username,password,enabled from users where username = ?	968,2	60	16,1	1,0
select medicalrec0_.id as id1_3_, medicalrec0_.description as descript2_3_, medical...	897,0	60	14,9	1,0
select medicine0_.id as id1_4_0_, medicine0_.name as name2_4_0_, medicine0_.expirat...	750,0	90	8,3	1,0
select username, authority from authorities where username = ?	536,6	60	8,9	1,0
select medicalrec0_.id as id1_3_0_, medicalrec0_.description as descript2_3_0_, med...	256,3	180	1,4	1,0
select user0_.username as username1_12_0_, user0_.enabled as enabled2_12_0_, user0_...	246,0	240	1,0	1,0
select visit0_.id as id1_15_0_, visit0_.visit_date as visit_da2_15_0_, visit0_.desc...	236,8	120	2,0	1,0
select interventi0_.vet_id as vet_id6_2_0_, interventi0_.id as id1_2_0_, interventi...	223,0	120	1,9	1,0
select specialtie0_.vet_id as vet_id1_13_0_, specialtie0_.specialty_id as specialt2...	184,9	120	1,5	0,5
select owner0_.id as id1_5_0_, pets1_.id as id1_6_1_, owner0_.first_name as first_n...	74,4	60	1,2	2,0

As we can see, the main problem with the implementation is the amount of unnecessary calls that are made to the database. Using only 60 users, we got in some places more than 1000 queries, that means, for every single user that query is called more than 16 times.

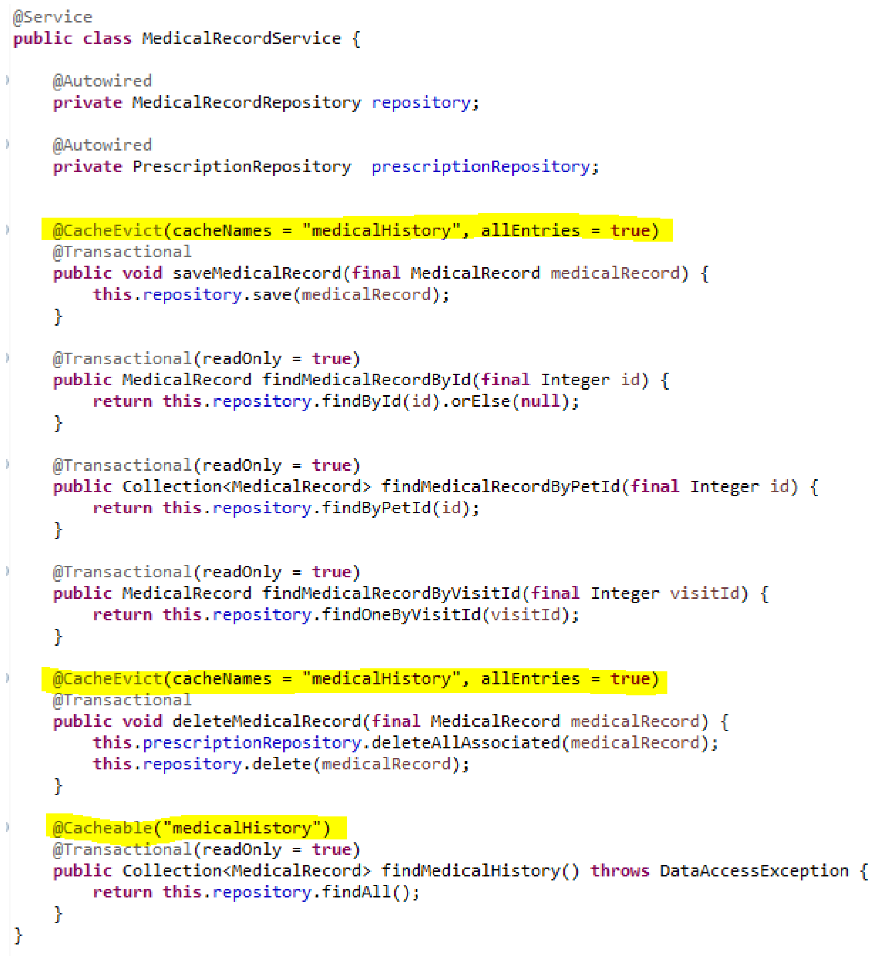
To resolve this problem, it is needed to reduce the number of callings that are being made to the Database. So, the solution that Iván end up with was by using lazy loading and implementing cache.



```java
@Entity
@Data
@EqualsAndHashCode(callSuper = false)
@Table(name = "medical_records")
public class MedicalRecord extends BaseEntity {

    @NotBlank
    private String  description;

    @NotBlank
    private String  status;

    @OneToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    @JoinColumn(name = "visit_id")
    private Visit   visit;

}
```

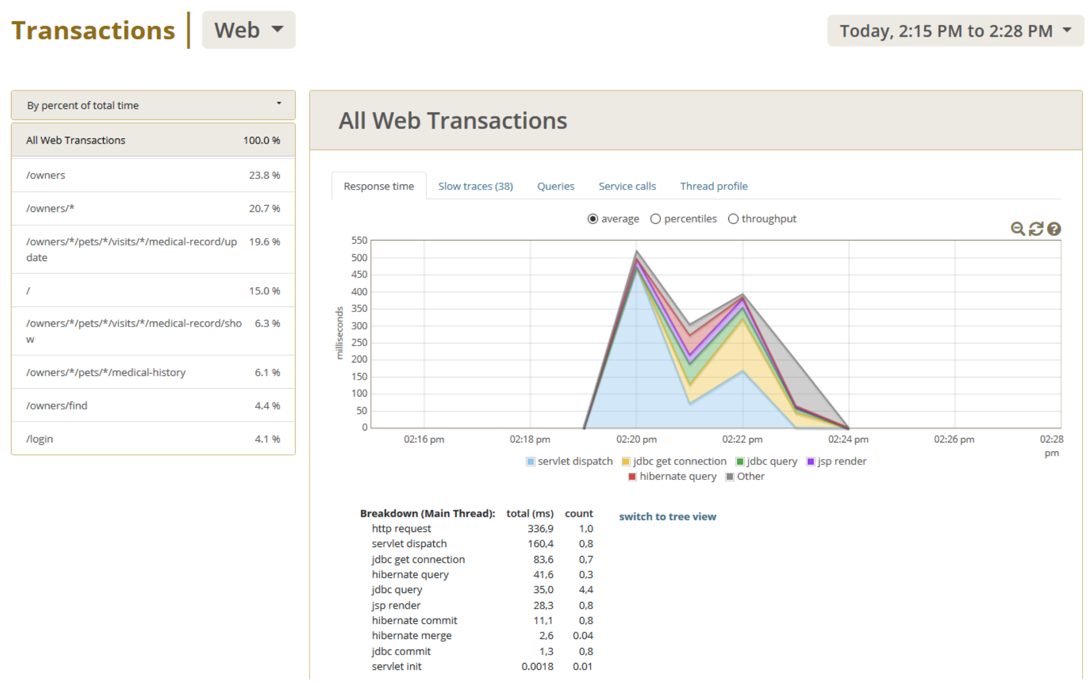


```xml
<cache alias="medicalHistory" uses-template="default">
    <key-type></key-type>
    <value-type>java.util.Collection</value-type>
</cache>
```



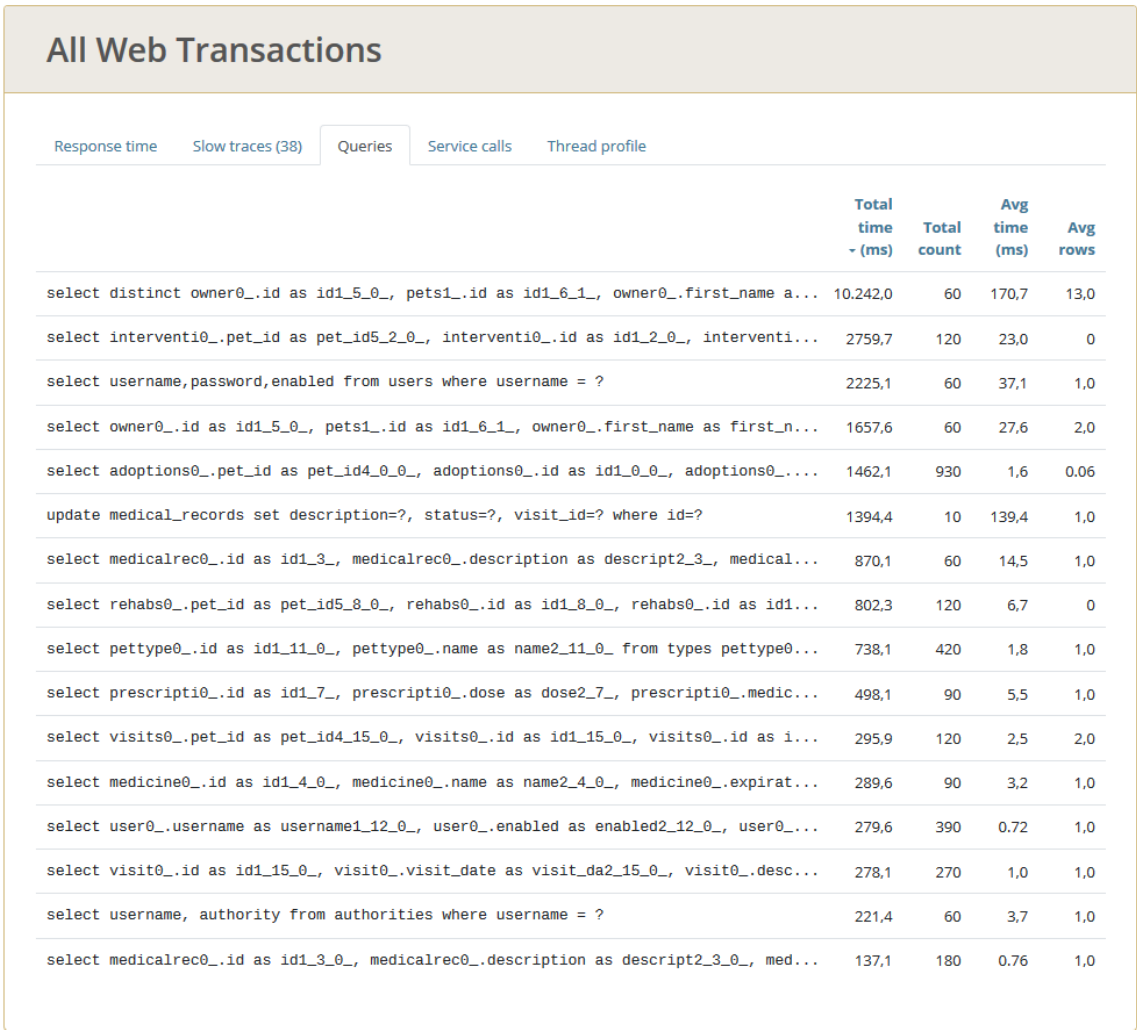
```java
@Service
public class MedicalRecordService {

    @Autowired
    private MedicalRecordRepository repository;

    @Autowired
    private PrescriptionRepository  prescriptionRepository;


    @CacheEvict(cacheNames = "medicalHistory", allEntries = true)
    @Transactional
    public void saveMedicalRecord(final MedicalRecord medicalRecord) {
        this.repository.save(medicalRecord);
    }

    @Transactional(readOnly = true)
    public MedicalRecord findMedicalRecordById(final Integer id) {
        return this.repository.findById(id).orElse(null);
    }

    @Transactional(readOnly = true)
    public Collection<MedicalRecord> findMedicalRecordByPetId(final Integer id) {
        return this.repository.findByPetId(id);
    }

    @Transactional(readOnly = true)
    public MedicalRecord findMedicalRecordByVisitId(final Integer visitId) {
        return this.repository.findOneByVisitId(visitId);
    }

    @CacheEvict(cacheNames = "medicalHistory", allEntries = true)
    @Transactional
    public void deleteMedicalRecord(final MedicalRecord medicalRecord) {
        this.prescriptionRepository.deleteAllAssociated(medicalRecord);
        this.repository.delete(medicalRecord);
    }

    @Cacheable("medicalHistory")
    @Transactional(readOnly = true)
    public Collection<MedicalRecord> findMedicalHistory() throws DataAccessException {
        return this.repository.findAll();
    }
}
```

After implementing the code previously provided, we launch again the application and check for the different results



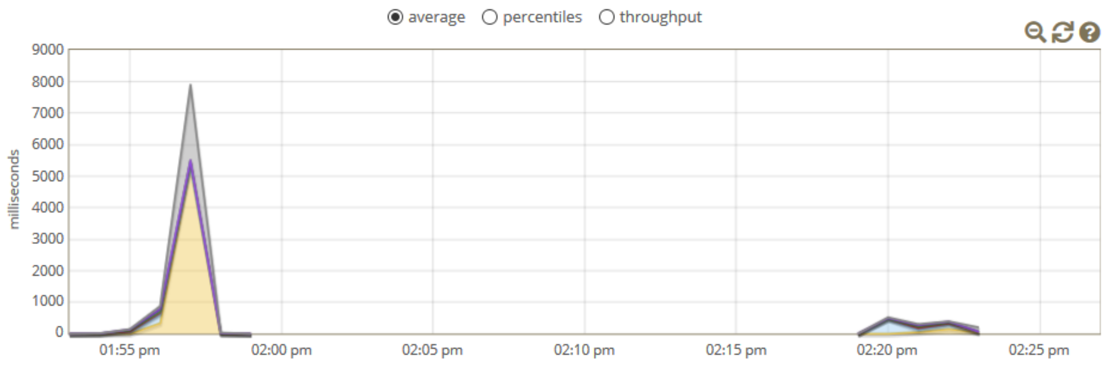
All Web Transactions	100.0 %
/owners	23.8 %
/owners/*	20.7 %
/owners/*/pets/*/visits/*/medical-record/update	19.6 %
/	15.0 %
/owners/*/pets/*/visits/*/medical-record/show	6.3 %
/owners/*/pets/*/medical-history	6.1 %
/owners/find	4.4 %
/login	4.1 %

Breakdown (Main Thread):	total (ms)	count
http request	336,9	1,0
servlet dispatch	160,4	0,8
jdbc get connection	83,6	0,7
hibernate query	41,6	0,3
jdbc query	35,0	4,4
jsp render	28,3	0,8
hibernate commit	11,1	0,8
hibernate merge	2,6	0.04
jdbc commit	1,3	0,8
servlet init	0.0018	0.01

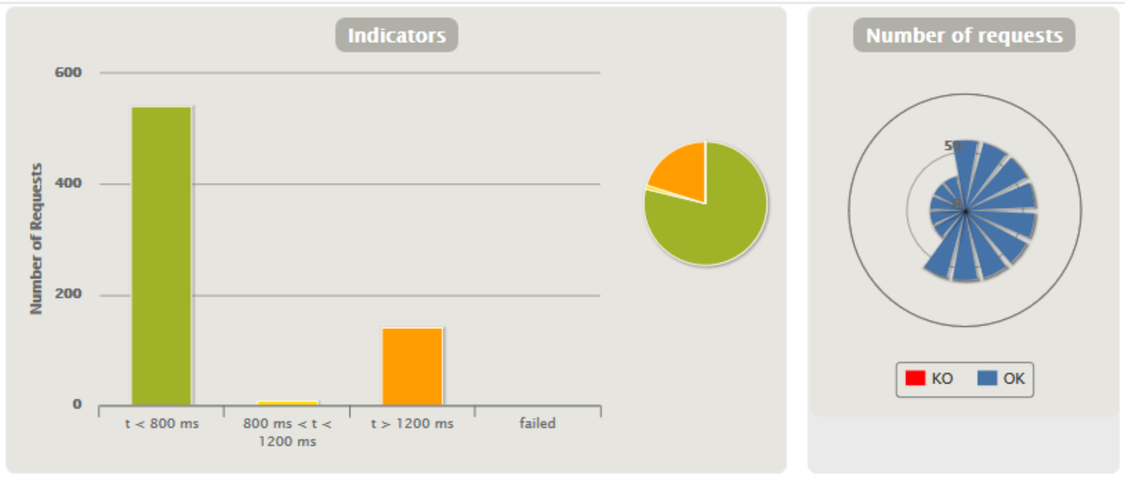


	Total time (ms)	Total count	Avg time (ms)	Avg rows
`select distinct owner0_.id as id1_5_0_, pets1_.id as id1_6_1_, owner0_.first_name a...`	10.242,0	60	170,7	13,0
`select interventi0_.pet_id as pet_id5_2_0_, interventi0_.id as id1_2_0_, interventi...`	2759,7	120	23,0	0
`select username,password,enabled from users where username = ?`	2225,1	60	37,1	1,0
`select owner0_.id as id1_5_0_, pets1_.id as id1_6_1_, owner0_.first_name as first_n...`	1657,6	60	27,6	2,0
`select adoptions0_.pet_id as pet_id4_0_0_, adoptions0_.id as id1_0_0_, adoptions0_....`	1462,1	930	1,6	0.06
`update medical_records set description=?, status=?, visit_id=? where id=?`	1394,4	10	139,4	1,0
`select medicalrec0_.id as id1_3_, medicalrec0_.description as descript2_3_, medical...`	870,1	60	14,5	1,0
`select rehabs0_.pet_id as pet_id5_8_0_, rehabs0_.id as id1_8_0_, rehabs0_.id as id1...`	802,3	120	6,7	0
`select pettype0_.id as id1_11_0_, pettype0_.name as name2_11_0_ from types pettype0...`	738,1	420	1,8	1,0
`select prescripti0_.id as id1_7_, prescripti0_.dose as dose2_7_, prescripti0_.medic...`	498,1	90	5,5	1,0
`select visits0_.pet_id as pet_id4_15_0_, visits0_.id as id1_15_0_, visits0_.id as i...`	295,9	120	2,5	2,0
`select medicine0_.id as id1_4_0_, medicine0_.name as name2_4_0_, medicine0_.expirat...`	289,6	90	3,2	1,0
`select user0_.username as username1_12_0_, user0_.enabled as enabled2_12_0_, user0_...`	279,6	390	0.72	1,0
`select visit0_.id as id1_15_0_, visit0_.visit_date as visit_da2_15_0_, visit0_.desc...`	278,1	270	1,0	1,0
`select username, authority from authorities where username = ?`	221,4	60	3,7	1,0
`select medicalrec0_.id as id1_3_0_, medicalrec0_.description as descript2_3_0_, med...`	137,1	180	0.76	1,0

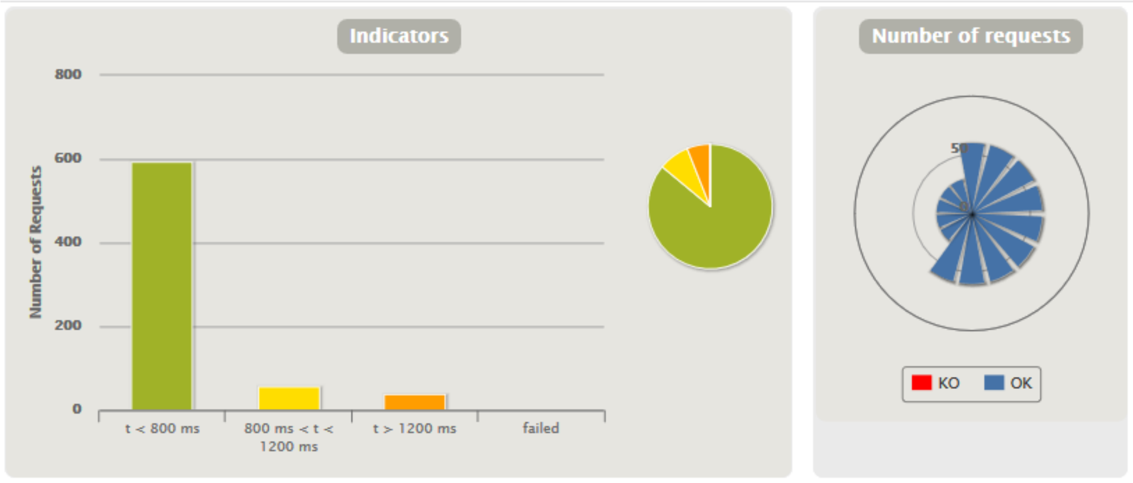
As we can see, while the total count of query's invocation is considerably low, the average time per query is practically the same. This is due to the Lazy Loading, which causes some slowdown when loading data. Even so, there are yet some unnecessary queries being called, this is due to poor code implementation and will be resolved in refactoring. Nevertheless, the performance was greatly improved by using lazy loading and cache.



Before:



After:



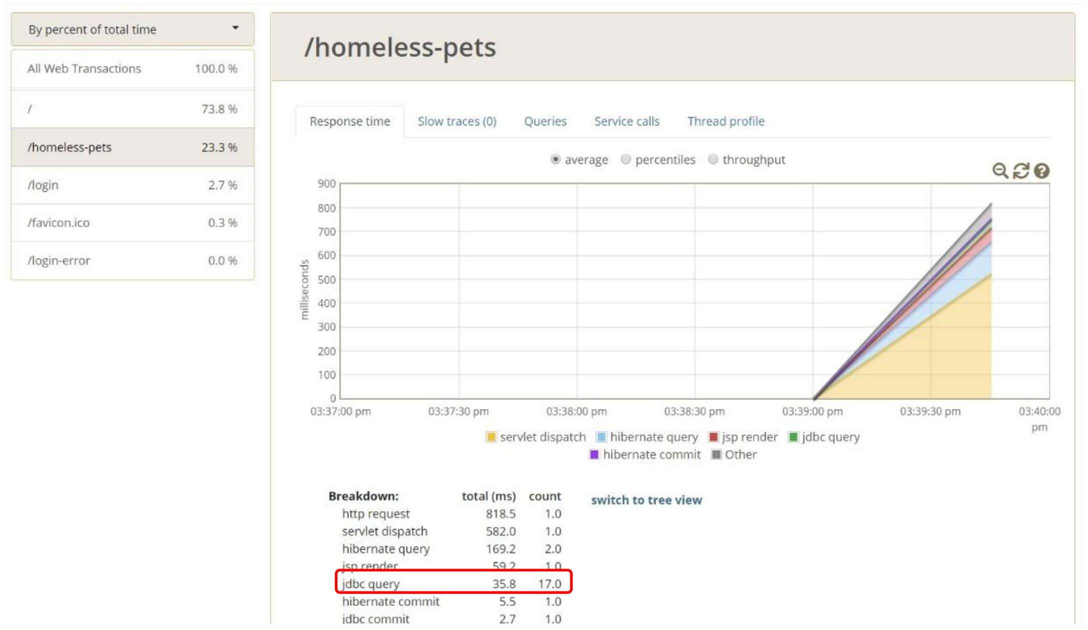
As we can see, the mean time it takes to make a response was decreased from 2046ms to 340ms, so we can conclude that the profiling was successful.

US-011 Vet manages homeless pets information

The US that was chosen for the profiling task was US 011 Homeless Pet Management because it was the one with the worst performance during the tests performed with Gatling. Getting deeper into the source of these problems Yoana chose Glowroot to see what exactly was causing them.

Yoana launched the application and performed every step of a single homeless pet creation as the specified scenario in Gatling scripts. Actually, the problem wasn't the inserting at all, it was the listing of the homeless pets (view from where she can access the new pet form). The main problem was that there were several queries that were getting executed needlessly because in the listing the info obtained from them was not used at all. It was a typical N+1 query problem.

Before performing any optimization refinements whenever the query for listing the pets was executed, several queries for their visits, interventions, rehabs and adoptions were getting executed as well and we don't need that. We can see that in the following screenshots:

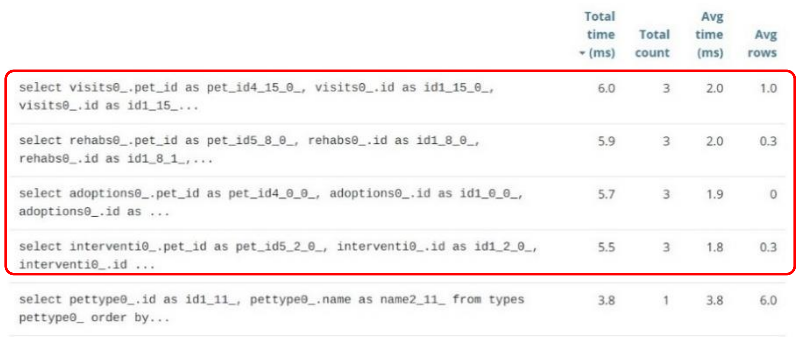


All Web Transactions	100.0 %
/	73.8 %
/homeless-pets	23.3 %
/login	2.7 %
/favicon.ico	0.3 %
/login-error	0.0 %

Breakdown:	total (ms)	count
http request	818.5	1.0
servlet dispatch	582.0	1.0
hibernate query	169.2	2.0
jsp render	59.2	1.0
jdbc query	35.8	17.0
hibernate commit	5.5	1.0
jdbc commit	2.7	1.0

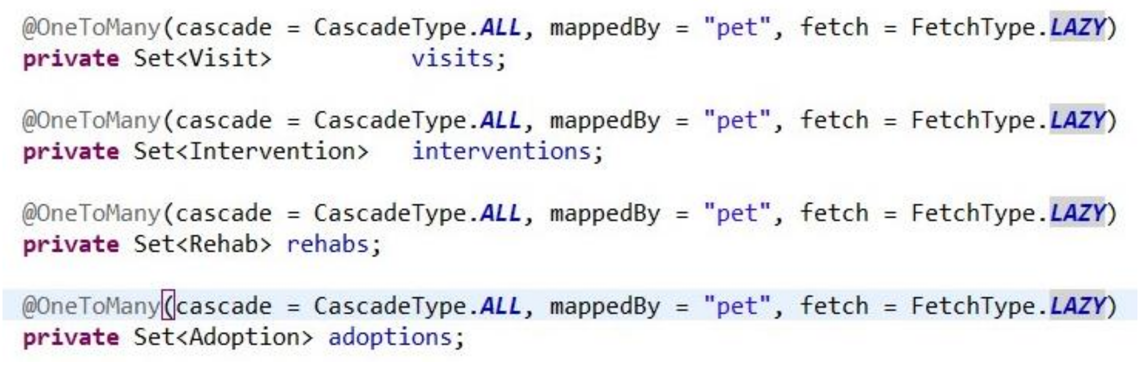
Figure 1 Response time before optimization

In Figure 1 we can see how long the queries of the listing view take, exactly 35.8ms. When we look into the different queries that get executed, we see what was mentioned before:



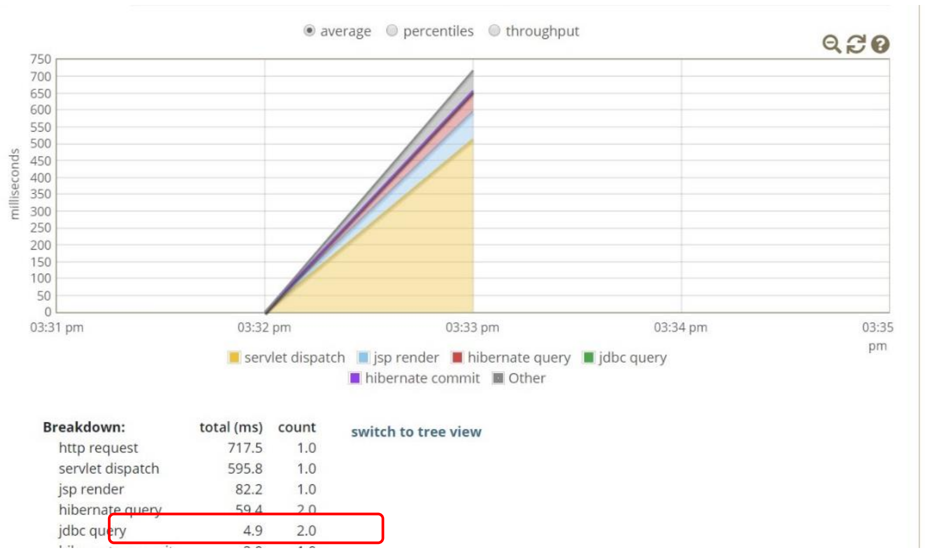
	Total time (ms)	Total count	Avg time (ms)	Avg rows
`select visits0_.pet_id as pet_id4_15_0_, visits0_.id as id1_15_0_, visits0_.id as id1_15_...`	6.0	3	2.0	1.0
`select rehabs0_.pet_id as pet_id5_8_0_, rehabs0_.id as id1_8_0_, rehabs0_.id as id1_8_1_,...`	5.9	3	2.0	0.3
`select adoptions0_.pet_id as pet_id4_0_0_, adoptions0_.id as id1_0_0_, adoptions0_.id as ...`	5.7	3	1.9	0
`select interventi0_.pet_id as pet_id5_2_0_, interventi0_.id as id1_2_0_, interventi0_.id ...`	5.5	3	1.8	0.3
`select pettype0_.id as id1_11_, pettype0_.name as name2_11_ from types pettype0_ order by...`	3.8	1	3.8	6.0

We can fix this by either lazy loading or using a join fetch. In this case it's better to use the lazy loading because we won't even use that information in the listing, we don't need it. Therefore, we need to modify the Pet entity as follows:



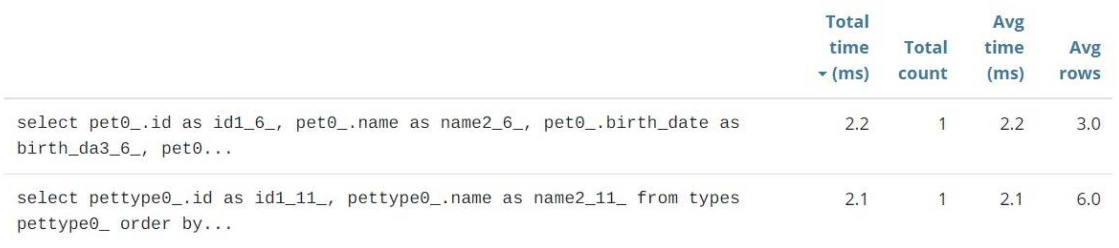
```
@OneToMany(cascade = CascadeType.ALL, mappedBy = "pet", fetch = FetchType.LAZY)
private Set<Visit>          visits;

@OneToMany(cascade = CascadeType.ALL, mappedBy = "pet", fetch = FetchType.LAZY)
private Set<Intervention>   interventions;

@OneToMany(cascade = CascadeType.ALL, mappedBy = "pet", fetch = FetchType.LAZY)
private Set<Rehab> rehabs;

@OneToMany(cascade = CascadeType.ALL, mappedBy = "pet", fetch = FetchType.LAZY)
private Set<Adoption> adoptions;
```

After this little change we can instantly see the difference in the time taken by the queries execution:



Breakdown:	total (ms)	count
http request	717.5	1.0
servlet dispatch	595.8	1.0
jsp render	82.2	1.0
hibernate query	59.4	2.0
jdbc query	4.9	2.0

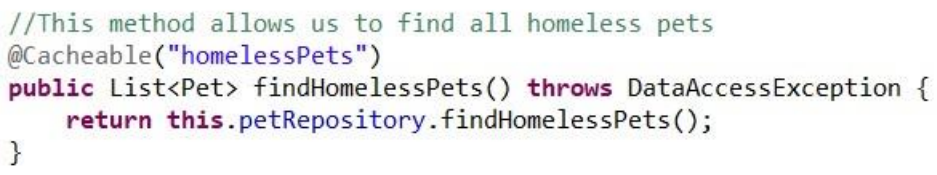
And also in the queries section (no unnecessary queries executed):



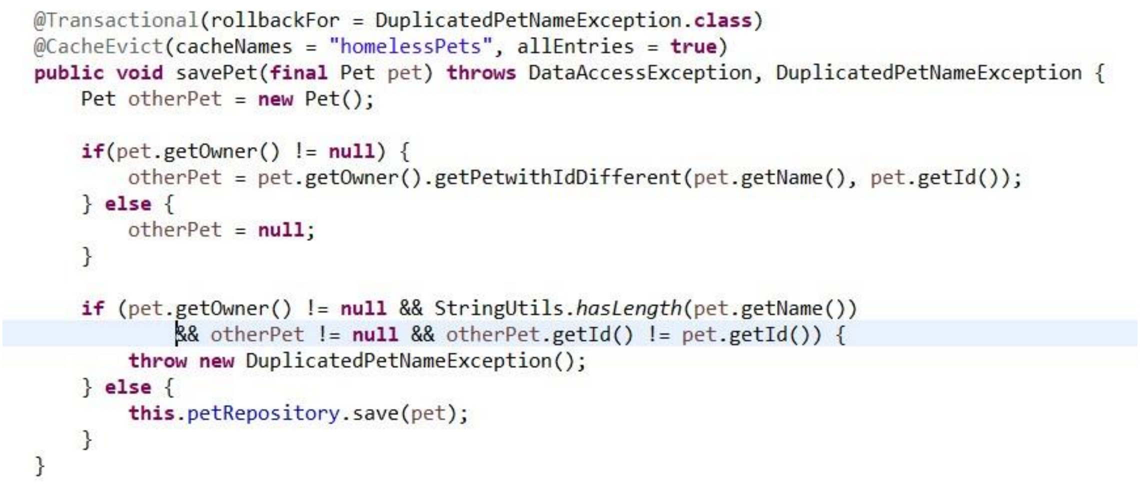
	Total time ▾ (ms)	Total count	Avg time (ms)	Avg rows
`select pet0_.id as id1_6_, pet0_.name as name2_6_, pet0_.birth_date as birth_da3_6_, pet0...`	2.2	1	2.2	3.0
`select pettype0_.id as id1_11_, pettype0_.name as name2_11_ from types pettype0_ order by...`	2.1	1	2.1	6.0

After this optimization Yoana thought "why don't we cache the listing?". To perform that task we need to make little adjustments in the code first.

As step 1 we need to add the following annotations to certain methods in PetService:

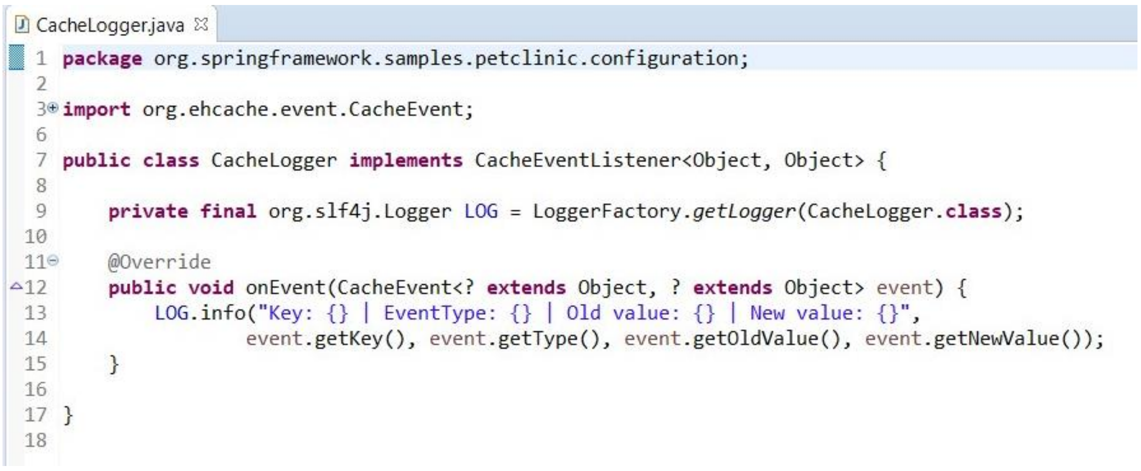


```java
//This method allows us to find all homeless pets
@Cacheable("homelessPets")
public List<Pet> findHomelessPets() throws DataAccessException {
    return this.petRepository.findHomelessPets();
}
```



```java
@Transactional(rollbackFor = DuplicatedPetNameException.class)
@CacheEvict(cacheNames = "homelessPets", allEntries = true)
public void savePet(final Pet pet) throws DataAccessException, DuplicatedPetNameException {
    Pet otherPet = new Pet();

    if(pet.getOwner() != null) {
        otherPet = pet.getOwner().getPetwithIdDifferent(pet.getName(), pet.getId());
    } else {
        otherPet = null;
    }

    if (pet.getOwner() != null && StringUtils.hasLength(pet.getName())
            && otherPet != null && otherPet.getId() != pet.getId()) {
        throw new DuplicatedPetNameException();
    } else {
        this.petRepository.save(pet);
    }
}
```

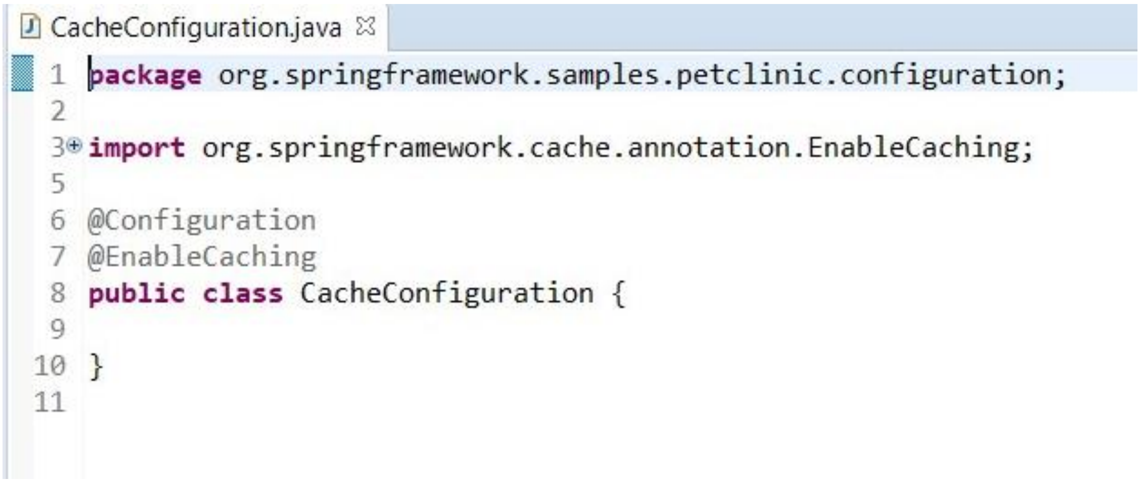
After that, as Step 2, we need to add two classes to the configuration package:



CacheLogger.java

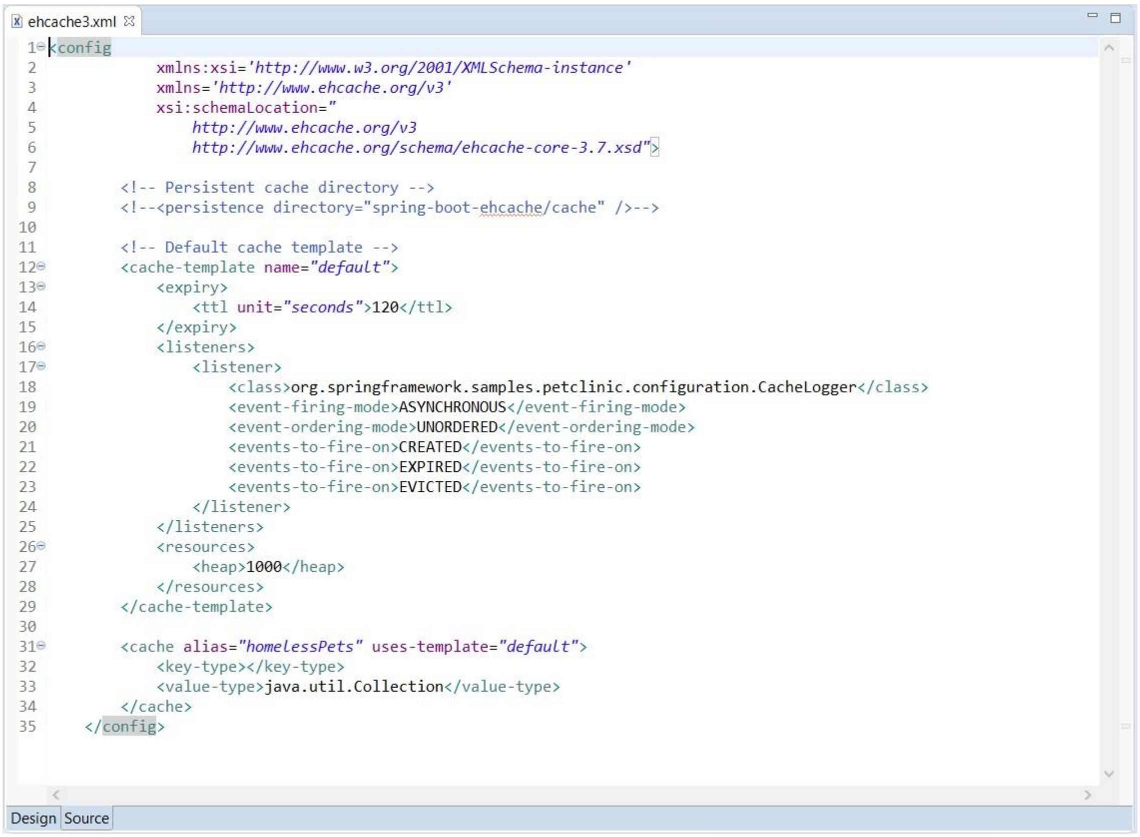
```java
package org.springframework.samples.petclinic.configuration;

import org.ehcache.event.CacheEvent;

public class CacheLogger implements CacheEventListener<Object, Object> {

    private final org.slf4j.Logger LOG = LoggerFactory.getLogger(CacheLogger.class);

    @Override
    public void onEvent(CacheEvent<? extends Object, ? extends Object> event) {
        LOG.info("Key: {} | EventType: {} | Old value: {} | New value: {}",
                event.getKey(), event.getType(), event.getOldValue(), event.getNewValue());
    }

}

```

A class that indicates us in the logs what is currently in the cache.



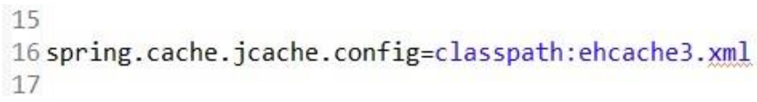
```java
package org.springframework.samples.petclinic.configuration;

import org.springframework.cache.annotation.EnableCaching;

@Configuration
@EnableCaching
public class CacheConfiguration {

}
```

As step 3 we have to add an xml file like this one in src/main/resources:



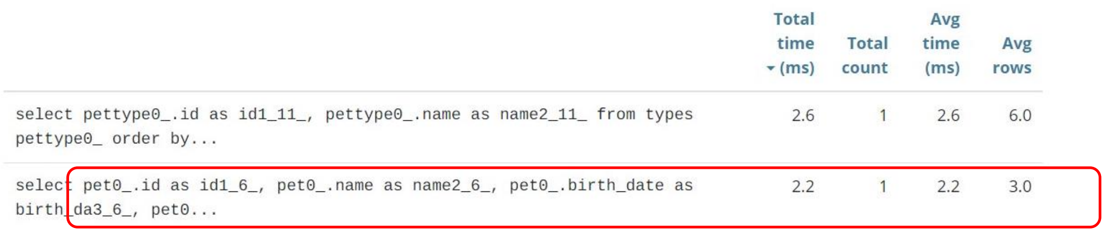
```xml
<config
            xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
            xmlns='http://www.ehcache.org/v3'
            xsi:schemaLocation="
                http://www.ehcache.org/v3
                http://www.ehcache.org/schema/ehcache-core-3.7.xsd">

        <!-- Persistent cache directory -->
        <!--<persistence directory="spring-boot-ehcache/cache" />-->

        <!-- Default cache template -->
        <cache-template name="default">
            <expiry>
                <ttl unit="seconds">120</ttl>
            </expiry>
            <listeners>
                <listener>
                    <class>org.springframework.samples.petclinic.configuration.CacheLogger</class>
                    <event-firing-mode>ASYNCHRONOUS</event-firing-mode>
                    <event-ordering-mode>UNORDERED</event-ordering-mode>
                    <events-to-fire-on>CREATED</events-to-fire-on>
                    <events-to-fire-on>EXPIRED</events-to-fire-on>
                    <events-to-fire-on>EVICTED</events-to-fire-on>
                </listener>
            </listeners>
            <resources>
                <heap>1000</heap>
            </resources>
        </cache-template>

        <cache alias="homelessPets" uses-template="default">
            <key-type></key-type>
            <value-type>java.util.Collection</value-type>
        </cache>
    </config>
```

And as step 4, indicate in the properties file that we will be using a cache file as follows:



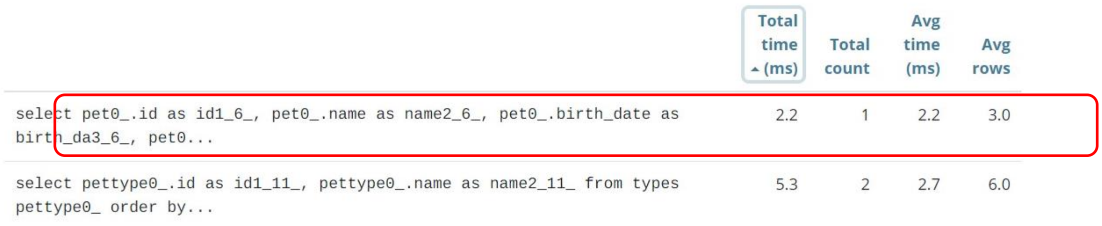
```
spring.cache.jcache.config=classpath:ehcache3.xml
```

After this, if we launch the application again to check if it works there’s a notable difference. The following screenshot is from the first time we access the listing:



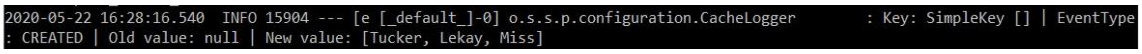
	Total time ▾ (ms)	Total count	Avg time (ms)	Avg rows
select pettype0_.id as id1_11_, pettype0_.name as name2_11_ from types pettype0_ order by...	2.6	1	2.6	6.0
select pet0_.id as id1_6_, pet0_.name as name2_6_, pet0_.birth_date as birth_da3_6_, pet0...	2.2	1	2.2	3.0

And the next one is from the second time we access the listing:



	Total time ▴ (ms)	Total count	Avg time (ms)	Avg rows
select pet0_.id as id1_6_, pet0_.name as name2_6_, pet0_.birth_date as birth_da3_6_, pet0...	2.2	1	2.2	3.0
select pettype0_.id as id1_11_, pettype0_.name as name2_11_ from types pettype0_ order by...	5.3	2	2.7	6.0

We can see that it gets executed only the first because the second time it gets the information from the cache, as it's shown in the logs:



```
2020-05-22 16:28:16.540  INFO 15904 --- [e [_default_]-0] o.s.s.p.configuration.CacheLogger        : Key: SimpleKey [] | EventType
: CREATED | Old value: null | New value: [Tucker, Lekay, Miss]
```

And more or less this makes this user story much more optimized.

US-017 Owner sees vet's personal information

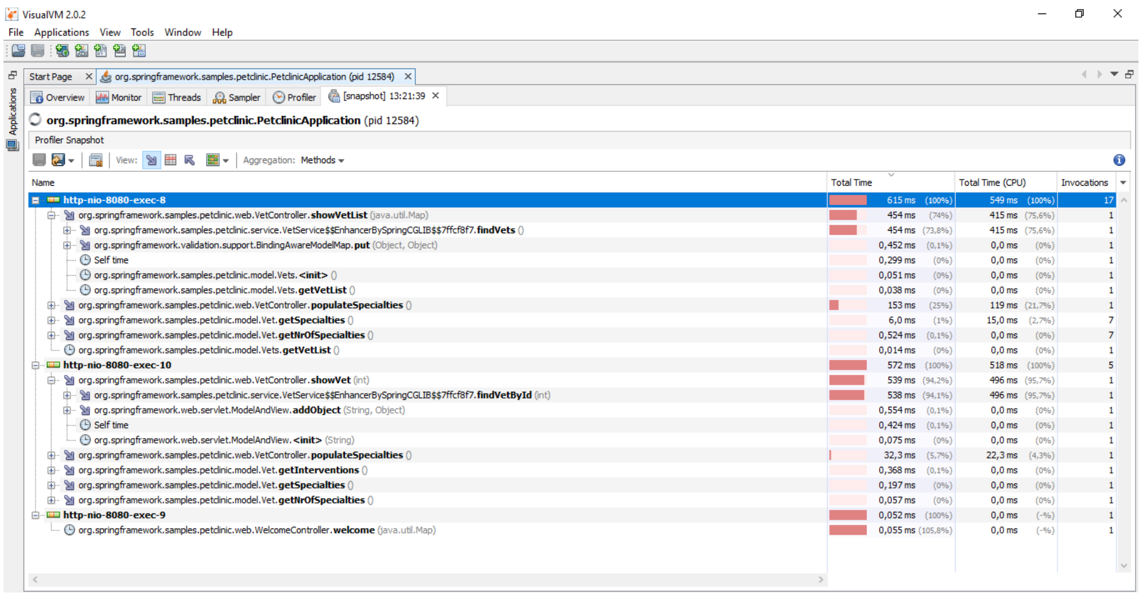
Recalling the performance testing, Álvaro selected for the profiling the US-17 (Owners sees vets' personal information).

The reason to that is, when comparing with others us, its performance was very low compared with the rest of them, taking into account that not only the performance was done with less users than the others and at the same time taking longer to perform the request. Later in the profiling Álvaro discovered the reason of that but at the beginning this was the US he had taken to perform a profiling.

The feature is simple, is just a show view of the data of the chosen vet.

The reason to use Visualvm was its simplicity and also it was straightforward when installing it, were as glowroot gave me errors and I had to lose some time to make it work, visualvm allowed me to start working from the beginning.

The result was the next.

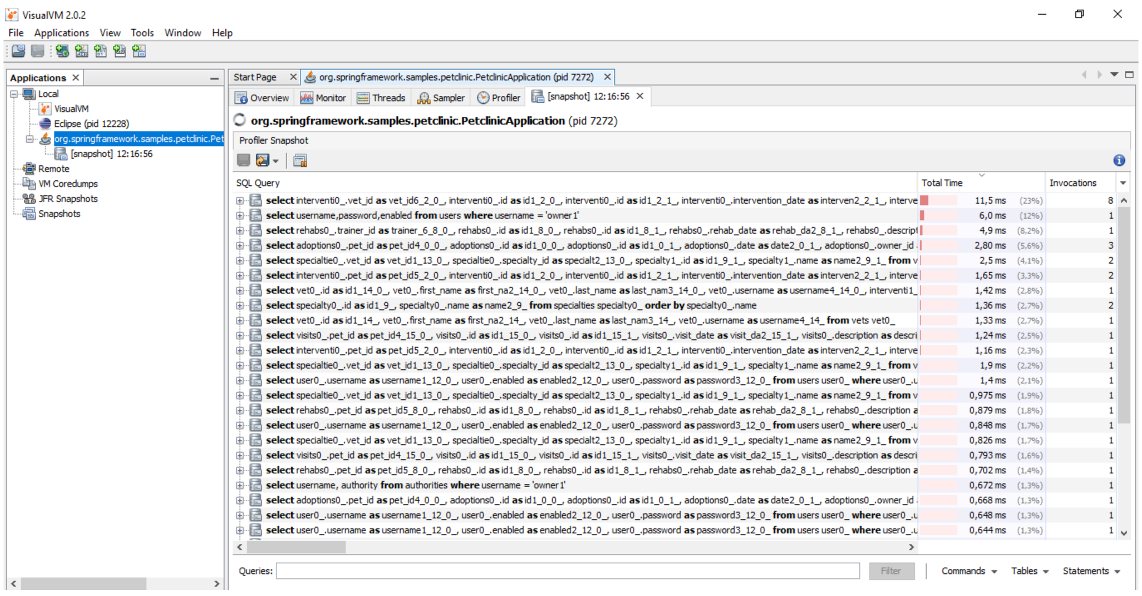


VisualVM 2.0.2
File Applications View Tools Window Help
Applications
Start Page
Overview Monitor Threads Sampler Profiler
Profiler Snapshot
View:
Aggregation: Methods

org.springframework.samples.petclinic.PetclinicApplication (pid 12584)

[snapshot] 13:21:39

Name	Total Time		Total Time (CPU)		Invocations
http-nio-8080-exec-8	615 ms	(100%)	549 ms	(100%)	17
org.springframework.samples.petclinic.web.VetController.**showVetList** (java.util.Map)	454 ms	(74%)	415 ms	(75,6%)	1
org.springframework.samples.petclinic.service.VetService$$EnhancerBySpringCGLIB$$7ffcf8f7.**findVets** ()	454 ms	(73,8%)	415 ms	(75,6%)	1
org.springframework.validation.support.BindingAwareModelMap.**put** (Object, Object)	0,452 ms	(0,1%)	0,0 ms	(0%)	1
Self time	0,299 ms	(0%)	0,0 ms	(0%)	1
org.springframework.samples.petclinic.model.Vets.**<init>** ()	0,051 ms	(0%)	0,0 ms	(0%)	1
org.springframework.samples.petclinic.model.Vets.**getVetList** ()	0,038 ms	(0%)	0,0 ms	(0%)	1
org.springframework.samples.petclinic.web.VetController.**populateSpecialties** ()	153 ms	(25%)	119 ms	(21,7%)	1
org.springframework.samples.petclinic.model.Vet.**getSpecialties** ()	6,0 ms	(1%)	15,0 ms	(2,7%)	7
org.springframework.samples.petclinic.model.Vet.**getNrOfSpecialties** ()	0,524 ms	(0,1%)	0,0 ms	(0%)	7
org.springframework.samples.petclinic.model.Vets.**getVetList** ()	0,014 ms	(0%)	0,0 ms	(0%)	1
http-nio-8080-exec-10	572 ms	(100%)	518 ms	(100%)	5
org.springframework.samples.petclinic.web.VetController.**showVet** (int)	539 ms	(94,2%)	496 ms	(95,7%)	1
org.springframework.samples.petclinic.service.VetService$$EnhancerBySpringCGLIB$$7ffcf8f7.**findVetById** (int)	538 ms	(94,1%)	496 ms	(95,7%)	1
org.springframework.web.servlet.ModelAndView.**addObject** (String, Object)	0,554 ms	(0,1%)	0,0 ms	(0%)	1
Self time	0,424 ms	(0,1%)	0,0 ms	(0%)	1
org.springframework.web.servlet.ModelAndView.**<init>** (String)	0,075 ms	(0%)	0,0 ms	(0%)	1
org.springframework.samples.petclinic.web.VetController.**populateSpecialties** ()	32,3 ms	(5,7%)	22,3 ms	(4,3%)	1
org.springframework.samples.petclinic.model.Vet.**getInterventions** ()	0,368 ms	(0,1%)	0,0 ms	(0%)	1
org.springframework.samples.petclinic.model.Vet.**getSpecialties** ()	0,197 ms	(0%)	0,0 ms	(0%)	1
org.springframework.samples.petclinic.model.Vet.**getNrOfSpecialties** ()	0,057 ms	(0%)	0,0 ms	(0%)	1
http-nio-8080-exec-9	0,052 ms	(100%)	0,0 ms	(-%)	1
org.springframework.samples.petclinic.web.WelcomeController.**welcome** (java.util.Map)	0,055 ms	(105,8%)	0,0 ms	(-%)	1



org.springframework.samples.petclinic.PetclinicApplication (pid 7272)

[snapshot] 12:16:56

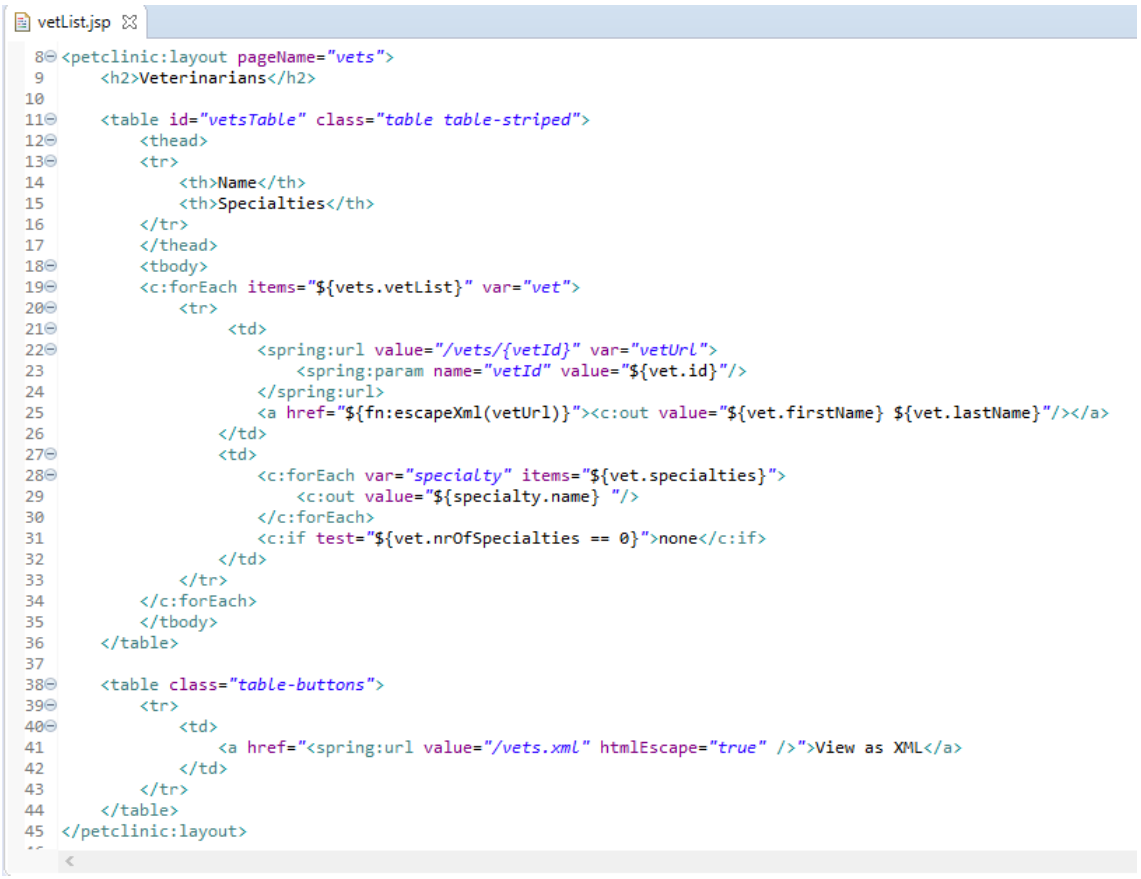
SQL Query	Total Time		Invocations
select interventi0_.vet_id **as** vet_id6_2_0_, interventi0_.id **as** id1_2_0_, interventi0_.id **as** id1_2_1_, interventi0_.intervention_date **as** interven2_2_1_, interve	11,5 ms	(23%)	8
select username,password,enabled **from** users **where** username = 'owner1'	6,0 ms	(12%)	1
select rehabs0_.trainer_id **as** trainer_6_8_0_, rehabs0_.id **as** id1_8_0_, rehabs0_.id **as** id1_8_1_, rehabs0_.rehab_date **as** rehab_da2_8_1_, rehabs0_.descript	4,9 ms	(8,2%)	1
select adoptions0_.pet_id **as** pet_id4_0_0_, adoptions0_.id **as** id1_0_0_, adoptions0_.id **as** id1_0_1_, adoptions0_.date **as** date2_0_1_, adoptions0_.owner_id	2,80 ms	(5,6%)	3
select specialtie0_.vet_id **as** vet_id1_13_0_, specialtie0_.specialty_id **as** specialt2_13_0_, specialty1_.id **as** id1_9_1_, specialty1_.name **as** name2_9_1_ **from** v	2,5 ms	(4,1%)	2
select interventi0_.pet_id **as** pet_id5_2_0_, interventi0_.id **as** id1_2_0_, interventi0_.id **as** id1_2_1_, interventi0_.intervention_date **as** interven2_2_1_, interve	1,65 ms	(3,3%)	2
select vet0_.id **as** id1_14_0_, vet0_.first_name **as** first_na2_14_0_, vet0_.last_name **as** last_nam3_14_0_, vet0_.username **as** username4_14_0_, interventi1_	1,42 ms	(2,8%)	1
select specialty0_.id **as** id1_9_, specialty0_.name **as** name2_9_ **from** specialties specialty0_ **order by** specialty0_.name	1,36 ms	(2,7%)	2
select vet0_.id **as** id1_14_, vet0_.first_name **as** first_na2_14_, vet0_.last_name **as** last_nam3_14_, vet0_.username **as** username4_14_ **from** vets vet0_	1,33 ms	(2,7%)	1
select visits0_.pet_id **as** pet_id4_15_0_, visits0_.id **as** id1_15_0_, visits0_.id **as** id1_15_1_, visits0_.visit_date **as** visit_da2_15_1_, visits0_.description **as** descri	1,24 ms	(2,5%)	1
select interventi0_.pet_id **as** pet_id5_2_0_, interventi0_.id **as** id1_2_0_, interventi0_.id **as** id1_2_1_, interventi0_.intervention_date **as** interven2_2_1_, interve	1,16 ms	(2,3%)	1
select specialtie0_.vet_id **as** vet_id1_13_0_, specialtie0_.specialty_id **as** specialt2_13_0_, specialty1_.id **as** id1_9_1_, specialty1_.name **as** name2_9_1_ **from** v	1,9 ms	(2,2%)	1
select user0_.username **as** username1_12_0_, user0_.enabled **as** enabled2_12_0_, user0_.password **as** password3_12_0_ **from** users user0_ **where** user0_.u	1,4 ms	(2,1%)	1
select specialtie0_.vet_id **as** vet_id1_13_0_, specialtie0_.specialty_id **as** specialt2_13_0_, specialty1_.id **as** id1_9_1_, specialty1_.name **as** name2_9_1_ **from** v	0,975 ms	(1,9%)	1
select rehabs0_.pet_id **as** pet_id5_8_0_, rehabs0_.id **as** id1_8_0_, rehabs0_.id **as** id1_8_1_, rehabs0_.rehab_date **as** rehab_da2_8_1_, rehabs0_.description a	0,879 ms	(1,8%)	1
select user0_.username **as** username1_12_0_, user0_.enabled **as** enabled2_12_0_, user0_.password **as** password3_12_0_ **from** users user0_ **where** user0_.u	0,848 ms	(1,7%)	1
select specialtie0_.vet_id **as** vet_id1_13_0_, specialtie0_.specialty_id **as** specialt2_13_0_, specialty1_.id **as** id1_9_1_, specialty1_.name **as** name2_9_1_ **from** v	0,826 ms	(1,7%)	1
select visits0_.pet_id **as** pet_id4_15_0_, visits0_.id **as** id1_15_0_, visits0_.id **as** id1_15_1_, visits0_.visit_date **as** visit_da2_15_1_, visits0_.description **as** descri	0,793 ms	(1,6%)	1
select rehabs0_.pet_id **as** pet_id5_8_0_, rehabs0_.id **as** id1_8_0_, rehabs0_.id **as** id1_8_1_, rehabs0_.rehab_date **as** rehab_da2_8_1_, rehabs0_.description a	0,702 ms	(1,4%)	1
select username, authority **from** authorities **where** username = 'owner1'	0,672 ms	(1,3%)	1
select adoptions0_.pet_id **as** pet_id4_0_0_, adoptions0_.id **as** id1_0_0_, adoptions0_.id **as** id1_0_1_, adoptions0_.date **as** date2_0_1_, adoptions0_.owner_id	0,668 ms	(1,3%)	1
select user0_.username **as** username1_12_0_, user0_.enabled **as** enabled2_12_0_, user0_.password **as** password3_12_0_ **from** users user0_ **where** user0_.u	0,648 ms	(1,3%)	1
select user0_.username **as** username1_12_0_, user0_.enabled **as** enabled2_12_0_, user0_.password **as** password3_12_0_ **from** users user0_ **where** user0_.u	0,644 ms	(1,3%)	1

Just by looking at the lines we figure out that something is happening with a query that is called 8 times for the feature. This happens because to get to the vet show view first we have to go throw a listing of vet. The problem resides here, in the listing, there is shown part of the information of the vets, and then, when a user access to the vet show view the query is call one more time. This is a N+1 problem.

Another problem was a query regarding adoptions that was called. Although it is not wrong, it does not contribute any functionality rather than showing more things and it can be avoided.

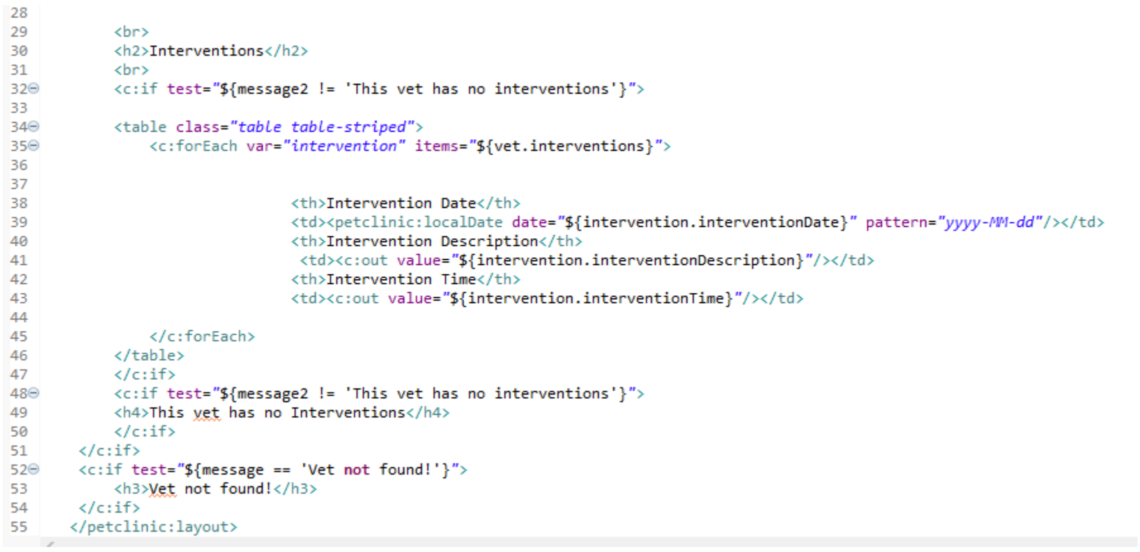
The first step is to amend the N+1 problem. In the database there are 7 vets, so the query is executed 7 times (one for each vet) + 1 (for the show view).

The correct answer would be deleting the listing thus improving a lot the performance, but it would affect other users stories, so the solution was to minimize the data shown in the listing, giving only the strictly necessary when finding a vet (the chosen attributes were the name and the specialty).



```
<petclinic:layout pageName="vets">
    <h2>Veterinarians</h2>

    <table id="vetsTable" class="table table-striped">
        <thead>
        <tr>
            <th>Name</th>
            <th>Specialties</th>
        </tr>
        </thead>
        <tbody>
        <c:forEach items="${vets.vetList}" var="vet">
            <tr>
                 <td>
                    <spring:url value="/vets/{vetId}" var="vetUrl">
                        <spring:param name="vetId" value="${vet.id}"/>
                    </spring:url>
                    <a href="${fn:escapeXml(vetUrl)}"><c:out value="${vet.firstName} ${vet.lastName}"/></a>
                </td>
                <td>
                    <c:forEach var="specialty" items="${vet.specialties}">
                        <c:out value="${specialty.name} "/>
                    </c:forEach>
                    <c:if test="${vet.nrOfSpecialties == 0}">none</c:if>
                </td>
            </tr>
        </c:forEach>
        </tbody>
    </table>

    <table class="table-buttons">
        <tr>
            <td>
                <a href="<spring:url value="/vets.xml" htmlEscape="true" />">View as XML</a>
            </td>
        </tr>
    </table>
</petclinic:layout>
```

The other problem, Álvaro just performed a simple change in the view file deleting the adoptions.



```

        <br>
        <h2>Interventions</h2>
        <br>
        <c:if test="${message2 != 'This vet has no interventions'}">

        <table class="table table-striped">
            <c:forEach var="intervention" items="${vet.interventions}">


                            <th>Intervention Date</th>
                            <td><petclinic:localDate date="${intervention.interventionDate}" pattern="yyyy-MM-dd"/></td>
                            <th>Intervention Description</th>
                             <td><c:out value="${intervention.interventionDescription}"/></td>
                            <th>Intervention Time</th>
                            <td><c:out value="${intervention.interventionTime}"/></td>

            </c:forEach>
        </table>
        </c:if>
        <c:if test="${message2 != 'This vet has no interventions'}">
        <h4>This vet has no Interventions</h4>
        </c:if>
    </c:if>
    <c:if test="${message == 'Vet not found!'}">
        <h3>Vet not found!</h3>
    </c:if>
   </petclinic:layout>
```

As a result, one query that took 5.6% (4th) of the time was erased and the N+1 problem was affected from 11.5ms (21%) to 8.9ms (21%).

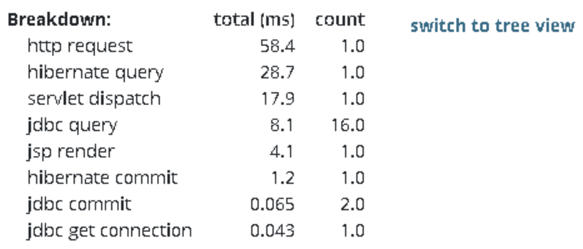
It is not a significant result considering the large result in the performance testing, but it couldn't been change any more code because it would affect the functionality of the feature.

US-024 Owner sees trainer’s personal information

For profiling Diāna chose US-024, where owner is accessing trainer’s personal information. As a profiler she used Glowroot.

After executing the steps for US-024 that have been defined before (as in, for example, positive UI test), Diāna checked the queries that have been executed together with loading an exact page.

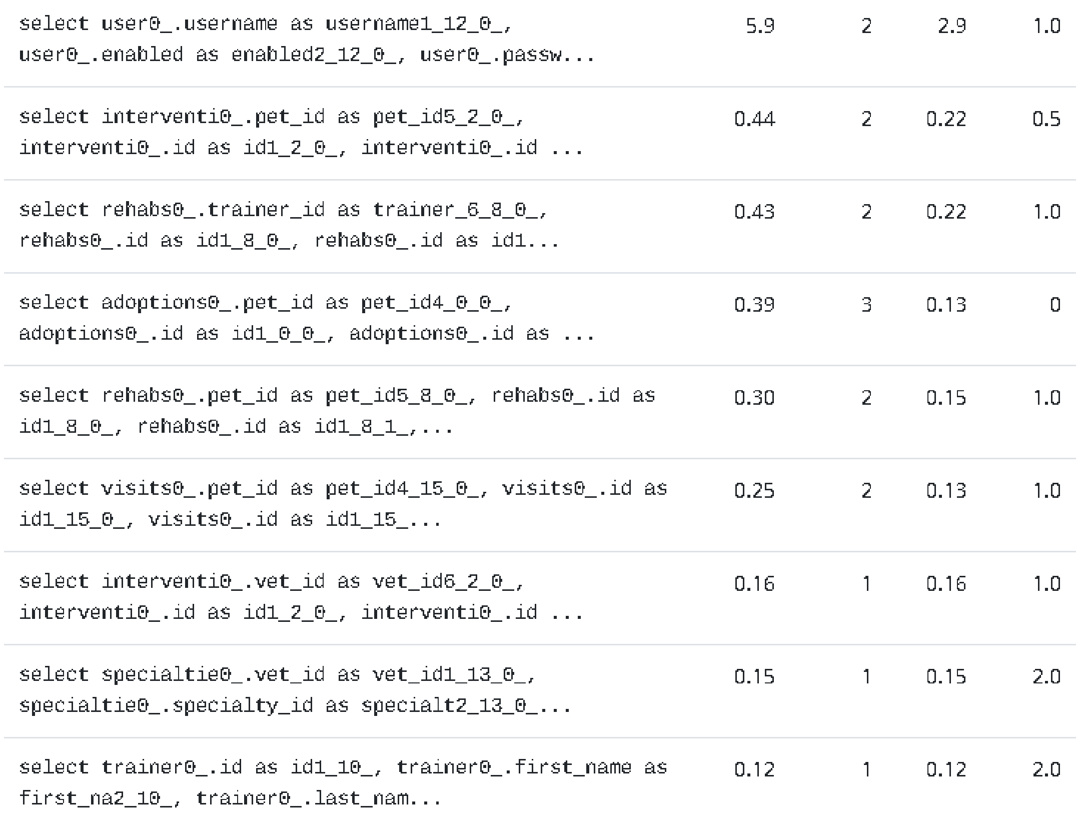
She noticed that when the trainer’s list page is being opened (it shows the list of all trainers), 16 queries have been executed. This page only shows the first, last name of the trainer, his email and phone number. It definitely should not be 16 queries for the information that has been shown.



Breakdown:	total (ms)	count
http request	58.4	1.0
hibernate query	28.7	1.0
servlet dispatch	17.9	1.0
jdbc query	8.1	16.0
jsp render	4.1	1.0
hibernate commit	1.2	1.0
jdbc commit	0.065	2.0
jdbc get connection	0.043	1.0

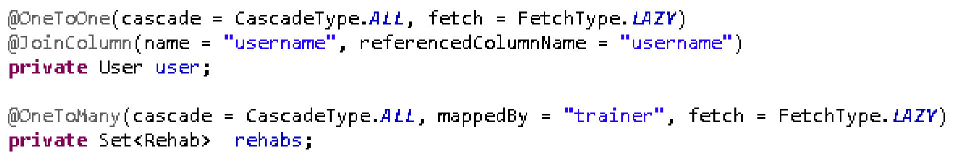
switch to tree view

Then Diāna took a more detailed look inside what those 16 queries were:



select user0_.username as username1_12_0_, user0_.enabled as enabled2_12_0_, user0_.passw...	5.9	2	2.9	1.0
select interventi0_.pet_id as pet_id5_2_0_, interventi0_.id as id1_2_0_, interventi0_.id ...	0.44	2	0.22	0.5
select rehabs0_.trainer_id as trainer_6_8_0_, rehabs0_.id as id1_8_0_, rehabs0_.id as id1...	0.43	2	0.22	1.0
select adoptions0_.pet_id as pet_id4_0_0_, adoptions0_.id as id1_0_0_, adoptions0_.id as ...	0.39	3	0.13	0
select rehabs0_.pet_id as pet_id5_8_0_, rehabs0_.id as id1_8_0_, rehabs0_.id as id1_8_1_,...	0.30	2	0.15	1.0
select visits0_.pet_id as pet_id4_15_0_, visits0_.id as id1_15_0_, visits0_.id as id1_15_...	0.25	2	0.13	1.0
select interventi0_.vet_id as vet_id6_2_0_, interventi0_.id as id1_2_0_, interventi0_.id ...	0.16	1	0.16	1.0
select specialtie0_.vet_id as vet_id1_13_0_, specialtie0_.specialty_id as specialt2_13_0_...	0.15	1	0.15	2.0
select trainer0_.id as id1_10_, trainer0_.first_name as first_na2_10_, trainer0_.last_nam...	0.12	1	0.12	2.0

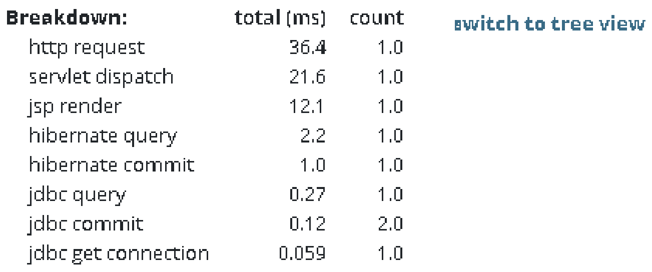
As we can see, many unnecessary queries have been loaded. For example, we can see that queries such those regarding to visits and vets have also been executed, although trainer does not have any relationship with these entities mentioned before. That means that those queries are taking more time to load the page and it means that there is a place for improvement. In Trainer’s model Diāna changes the fetch type strategy from “Eager” to “Lazy”, so that only necessary queries would be executed.



```
@OneToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
@JoinColumn(name = "username", referencedColumnName = "username")
private User user;

@OneToMany(cascade = CascadeType.ALL, mappedBy = "trainer", fetch = FetchType.LAZY)
private Set<Rehab>  rehabs;
```

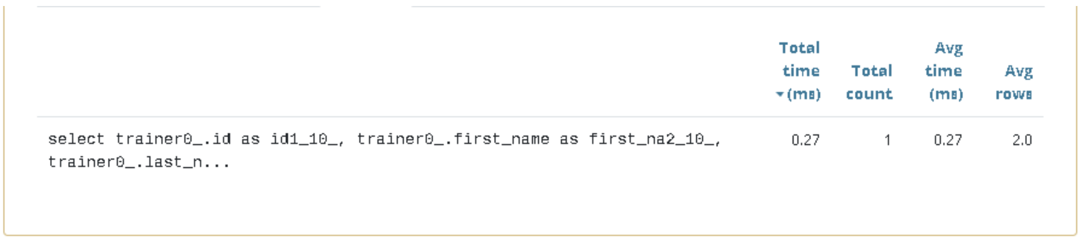
Since the trainer has a relationship with 2 entities – user and rehab, then she changed fetch type for both of the entities.

After doing so, Diāna executed US-24 again, she checked if there are any changes regarding to queries execution.



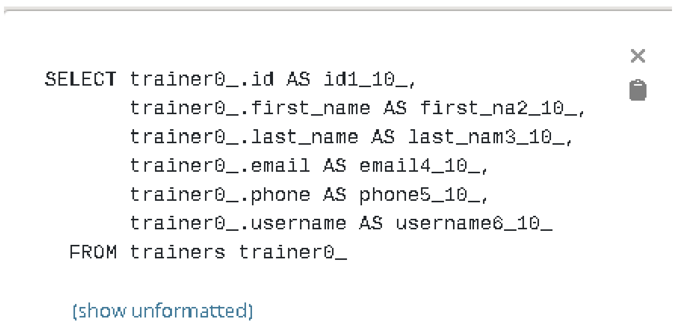
Breakdown:	total (ms)	count
http request	36.4	1.0
servlet dispatch	21.6	1.0
jsp render	12.1	1.0
hibernate query	2.2	1.0
hibernate commit	1.0	1.0
jdbc query	0.27	1.0
jdbc commit	0.12	2.0
jdbc get connection	0.059	1.0

We can see that the number of queries has been significantly reduced, from 16 to 1. That one query is:



	Total time ▾(ms)	Total count	Avg time (ms)	Avg rows
select trainer0_.id as id1_10_, trainer0_.first_name as first_na2_10_, trainer0_.last_n...	0.27	1	0.27	2.0

As Diāna opened the detailed query description, she could see that this query is responsible for fetching trainer’s name, last name, email and phone number, what is exactly what we need for this view and nothing more.



```
SELECT trainer0_.id AS id1_10_,
       trainer0_.first_name AS first_na2_10_,
       trainer0_.last_name AS last_nam3_10_,
       trainer0_.email AS email4_10_,
       trainer0_.phone AS phone5_10_,
       trainer0_.username AS username6_10_
  FROM trainers trainer0_
```

As the performance has been improved, the page is loading faster. Before (as shown in Figure 1) we can see that the time for this request took 58.4ms while now it is 36.4ms (as shown in the Figure 5).

In general, especially when working with multi-functional websites, that have many pages to load, such improvements can significantly improve user experience while using the website. For example, if there are big amount of information included in the page, that consists of many predefined entities, then such optimization steps will have bigger effect, since the http request time will be reduced.