Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

DISEÑO Y PRUEBAS II

PROFILING

Autores:
Francisco Manuel Cordero Vela
Alejandro Fuentes Gómez
José Manuel Gónzalez Mancilla
Miguel Ponce Melero
María Teresa Monge Zambrano
Pedro Padilla Molina

Grupo:
G3-11

Lunes 1 de Junio de 2020

1. Introducción

Para el desarrollo de la profiling, el equipo ha analizado exhastuviamente el informe de las pruebas de rendimiento previamente, con el objetivo de solucionar los posibles problemas de rendimiento. En este informe se muestran 3 profilings, las historias de usuario que el equipo ha considerado más graves. Para ello el equipo ha utilizado la herramienta de glowroot.

Como viene siendo usual en el equipo de trabajo, se ha dividido las tareas en las siguientes parejas:

- Pareja 1: Formada por María Teresa Monge Zambrano y Francisco Manuel Cordero Vela: Profiling para la historia de usuario 17.
- Pareja 2: Formada por Jose Manuel González Mancilla y Alejandro Fuentes Gómez: Profiling para la historia de usuario 10.
- Pareja 3: Formada por Miguel Ponce Melero y Pedro Padilla Molina: Profiling para la historia de usuario 19.

2. Profiling para la historia de usuario 10.

Para recordar la historia de usuario 10, consistía en que se podía organizar las visitas de un veterinario por fecha, siempre y cuando se le haya añadido una visita a dicho veterinario.

Como se puede observar en el documento de las pruebas de rendimiento, dicha historia de usuario no tenía un rendimiento adecuado. Por lo tanto el equipo ha decido intervenir para mejorar el rendimiento de esta historia de usuario. A modo de recordatorio se adjunta el rendimiento de la misma:

Para la mejora del rendimiento de esta historia de usuario, el equipo ha realizado dos cambios importantes:

- **Mapeo de tablas**: Se ha añadido la notación @Index para la optimización de las querys en la entidad Visits.
- **Implementación de una caché**: Se ha añadido con el objetivo de optimizar la lista de veterinarios, ya que es una lista que usualmente no se modifica.

Hemos tomado una serie de capturas de la aplicación Glowroot en la que nos muestra los tiempos de carga de las vistas correspondientes a la lista de vetrinarios y la vista de las visitas ordenadas antes de hacer profiling.

	Total time (ms)	Total count	Avg time (ms)	Avg rows
select specialtie0_.vet_id as vet_id1_11_0_, specialtie0_.specialty_id as specialt2_11_...	32,3	18	1,8	0,8
select vet0_.id as id1_12_, vet0_.first_name as first_na2_12_, vet0_.last_name as last_...	7,1	3	2,4	6,0

Lista de veterinarios

	Total time (ms)	Total count	Avg time (ms)	Avg rows
select pet0_.id as id1_4_0_, pet0_.name as name2_4_0_, pet0_.birth_date as birth_da3_4_...	39,3	3	13,1	6,0
select visit0_.id as id1_13_, visit0_.visit_date as visit_da2_13_, visit0_.description ...	28,5	3	9,5	4,0
select specialtie0_.vet_id as vet_id1_11_0_, specialtie0_.specialty_id as specialt2_11_...	9,2	6	1,5	1,5
select vet0_.id as id1_12_0_, vet0_.first_name as first_na2_12_0_, vet0_.last_name as l...	4,0	3	1,3	1,0
select medicine0_.id as id1_2_0_, medicine0_.name as name2_2_0_, medicine0_.description...	2,6	3	0.87	1,0

Lista de visitas ordenadas por veterinario

Viendo estos resultados, hemos decidido de implementar lo que hemos comentado antes y mapeo de tabla en la entidad Visit y una caché para la lista e los veterinarios. El código lo hemos cambiado de la siguiente manera, empezamos con la entidad Visit:

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

import java.time.LocalDate;


/**
 * Simple JavaBean domain object representing a visit.
 *
 * @author Ken Krebs
 */
@Entity
@Table(name = "visits")
public class Visit extends BaseEntity {
```

Antes

```java
/**
 * Simple JavaBean domain object representing a visit.
 *
 * @author Ken Krebs
 */
@Entity
@Table(name = "visits", indexes = {
        @Index(columnList = "vet_id, visit_date"),
        @Index(columnList = "visit_date, visit_time, vet_id")})
public class Visit extends BaseEntity {

    /**
     * Holds value of property date.
     */
    @NotNull
    @Column(name = "visit_date")
    @DateTimeFormat(pattern = "yyyy/MM/dd")
    private LocalDate date;

    @NotNull
    @Column(name = "visit_time")
    @DateTimeFormat(pattern = "HH:mm")
    private LocalTime time;

    /**
     * Holds value of property description.
     */
    @NotEmpty
    @Column(name = "description")
    private String description;
```

Después

Continuamos con VetService:

```java
@Service
public class VetService {

    @Autowired
    private VetRepository vetRepository;


    @Autowired
    public VetService(VetRepository vetRepository) {
        this.vetRepository = vetRepository;
    }

    @Transactional(readOnly = true)
    public Collection<Vet> findVets() throws DataAccessException {
        return vetRepository.findAll();
    }
```

Antes

```java
@Service
public class VetService {

    @Autowired
    private VetRepository vetRepository;

    @Autowired
    public VetService(VetRepository vetRepository) {
        this.vetRepository = vetRepository;
    }

    @Transactional(readOnly = true)
    @Cacheable("allVets")
    public Collection<Vet> findVets() throws DataAccessException {
        return vetRepository.findAll();
    }
```

Después

```
    </cache-template>

    <cache alias="allVets" uses-template="default">
        <key-type>org.springframework.cache.interceptor.SimpleKey</key-type>
        <value-type>java.util.Collection</value-type>
    </cache>
</config>
```

Creación de la caché

Analizando las capturas adjuntadas, podemos observar que en la entidad Visit se ha añadido en la notación "@Table" los índices con el objetivo de optimizar el tiempo de búsqueda en la base de datos. Seguidamente, podemos ver la modificación en el servicio de los veterinarios (VetService), en el método "findVets()", la anotación "@Cacheable".La notación consiste en conseguir que el resultado de la invocación a dicho método se quede almacenada en caché mediante el nombre "AllVets". La memoria caché se destruirá cuando pase 120 segundos, ya que así se ha configurado. Tomamos la configuración general debido a que no se tiene implementado la posibilidad de añadir nuevos veterinarios. En el caso contrario, se debería destruir la caché en el momento que un veterinario es añadido al sistema.

Gracias a los cambios comentados, se ha obtenido los siguientes resultados:

Response time | Slow traces (2) | Queries | Service calls | Thread profile

	Total time (ms)	Total count	Avg time (ms)	Avg rows
select specialtie0_.vet_id as vet_id1_11_0_, specialtie0_.specialty_id as specialt2_11_...	24,4	24	1,0	0,8
select vet0_.id as id1_12_, vet0_.first_name as first_na2_12_, vet0_.last_name as last_...	7,3	4	1,8	6,0

Lista de veterinarios

Response time | Slow traces (0) | Queries | Service calls | Thread profile

	Total time (ms)	Total count	Avg time (ms)	Avg rows
select pet0_.id as id1_4_0_, pet0_.name as name2_4_0_, pet0_.birth_date as birth_da3_4_0_...	4,4	2	2,2	6,0
select specialtie0_.vet_id as vet_id1_11_0_, specialtie0_.specialty_id as specialt2_11_0_...	2,6	4	0.66	1,5
select medicine0_.id as id1_2_0_, medicine0_.name as name2_2_0_, medicine0_.description a...	1,7	2	0.87	1,0
select visit0_.id as id1_13_, visit0_.visit_date as visit_da2_13_, visit0_.description as...	1,4	2	0.69	4,0
select vet0_.id as id1_12_0_, vet0_.first_name as first_na2_12_0_, vet0_.last_name as las...	1,2	2	0.62	1,0

Lista de visitas ordenadas por veterinario

Gracias a los resultados obtenidos antes y después de las modificaciones, se puede concluir:

- En el acceso a la vista de todos los veterinarios, el tiempo se ha reducido en 1,4 ms. En el caso de hacer 1000 peticiones se reducirá el tiempo total de respuesta en 1 segundo.
- En el acceso a la lista de visitas ordenadas por veterinarios, el tiempo se ha reducido en 21 ms. En el caso de hacer 1000 peticiones se reducirá el tiempo total de respuesta en 21 segundos aproximadamente.

Consecuentemente, gracias a la utilización de las técnicas de optimización, se puede observar una gran mejoría en el rendimiento de la historia de usuario que estamos contemplando.

3. Profiling para la historia de usuario 17.

Para recordar la historia de usuario 17, consistía que logueado como administrador se puede conocer la lista de diagnosticos ya realizados.

Como se puede observar en el documento de las pruebas de rendimiento, dicha historia de usuario no tenía un rendimiento adecuado. Por lo tanto el equipo ha decido intervenir para mejorar el rendimiento de esta historia de usuario. A modo de recordatorio se adjunta el rendimiento de la misma:

ASSERTIONS

Assertion	Status
Global: max of response time is less than 5000.0	OK
Global: mean of response time is less than 1000.0	OK
Global: percentage of successful events is greater than 95.0	OK

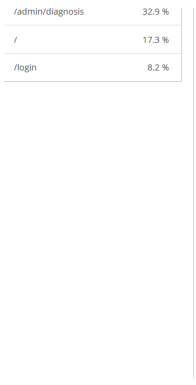
STATISTICS — Expand all groups | Collapse all groups

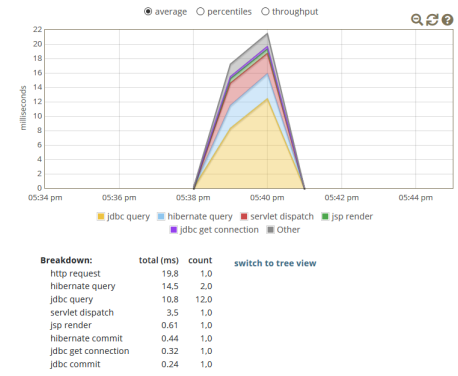
Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	36000	36000	0	0%	122.449	0	5	14	22	74	808	11	29
Home	6000	6000	0	0%	20.408	2	4	6	9	15	145	5	6
Login	6000	6000	0	0%	20.408	0	2	3	5	8	132	3	4
Logged	6000	6000	0	0%	20.408	1	4	5	7	10	103	4	3
Logged Redirect 1	6000	6000	0	0%	20.408	1	4	5	8	13	167	4	4
Dashboard	6000	6000	0	0%	20.408	12	18	20	35	304	808	25	49
ListDiagnosis	6000	6000	0	0%	20.408	10	15	17	31	305	734	23	46

Para la mejora del rendimiento de esta historia de usuario, el equipo ha realizado dos cambios importantes:

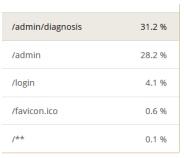
- **Queries N+1**: Se ha cambiado la anotacion fetch = FetchType.EAGER por LAZY en la entidad Vet.
- **Implementación de una caché**: Se ha añadido con el objetivo de optimizar la lista de diagnósticos.

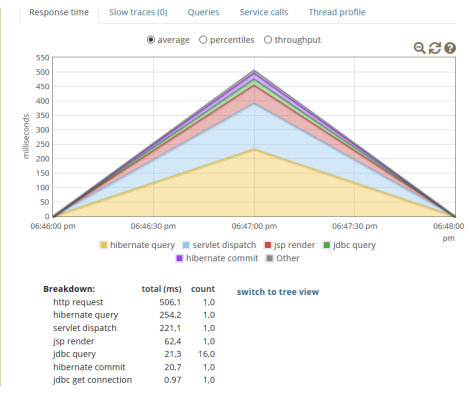
Hemos tomado una serie de capturas de la aplicación Glowroot en la que nos muestra los tiempos de carga de las vistas correspondientes a la lista de diagnosticos y dashboard (/admin) antes y después de hacer profiling.



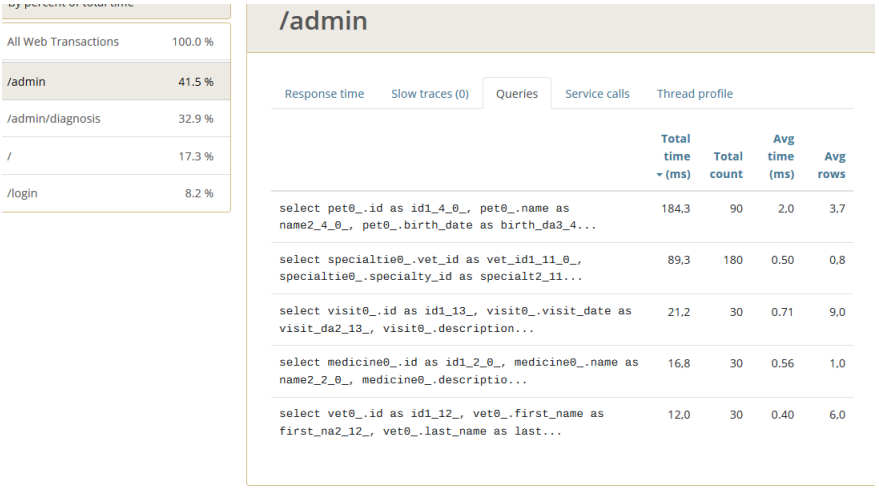


Dashboard





Lista de diagnósticos



Dashboard queries.

Viendo estos resultados, hemos decidido de implementar lo que hemos comentado antes y y una caché para la lista de diagnósticos. El código lo hemos cambiado de la siguiente manera, empezamos con la entidad Vet:

```java
public class Vet extends Person {

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name = "vet_specialties", joinColumns = @JoinColumn(name = "vet_id"),
            inverseJoinColumns = @JoinColumn(name = "specialty_id"))
    private Set<Specialty> specialties;
```

Antes

```java
public class Vet extends Person {

    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(name = "vet_specialties", joinColumns = @JoinColumn(name = "vet_id"),
            inverseJoinColumns = @JoinColumn(name = "specialty_id"))
    private Set<Specialty> specialties;
```

Después

Continuamos con DiagnosisService:

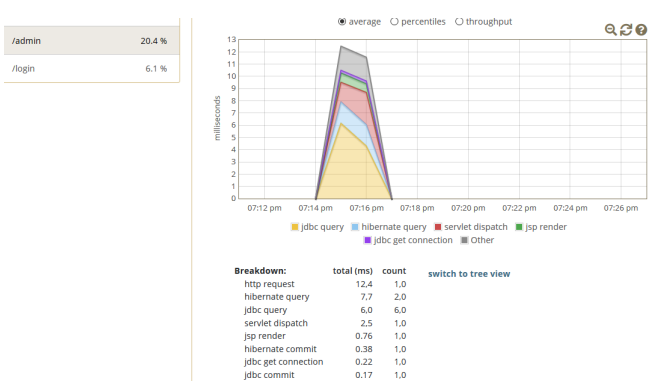
```java
@Transactional(readOnly=true)
@Cacheable("allDiagnosis")
public Collection<Diagnosis> findAll() {
    // TODO Auto-generated method stub
    Collection<Diagnosis> diagnosis = new ArrayList<Diagnosis>();
    for(Diagnosis d: this.diagnosisRepository.findAll()) {
        diagnosis.add(d);
    }
    return diagnosis;
}
```

```xml
<cache alias="allDiagnosis" uses-template="default">
    <key-type>org.springframework.cache.interceptor.SimpleKey</key-type>
    <value-type>java.util.Collection</value-type>
</cache>
```

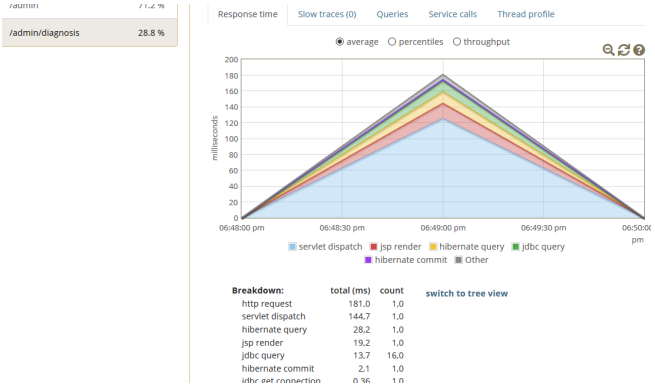
Creación de la caché

Analizando las capturas adjuntadas, podemos observar que en la entidad Visit se ha añadido en la notación "LAZY" para así optimizar las queries, ya que al realizar (/admin, que es el Dashboard) se realizaban muchas queries de datos que no eran necesarios en ese momento y por tanto aumentaba el tiempo. (Como consecuencia de haber realizado este cambio una de las queries no se realiza, por tanto el tiempo disminuye). Seguidamente, podemos ver la modificación en el servicio de Diagnóstico (DiagnosisService), en el método "findAll()", la anotación "@Cacheable".Esto consiste en conseguir que el resultado de la invocación a dicho método se quede almacenada en caché mediante el nombre "allDiagnosis". La memoria caché se destruirá cuando pasen 2 minutos. En el caso contrario, se debería destruir la caché en el momento que un diagnóstico es añadido al sistema.

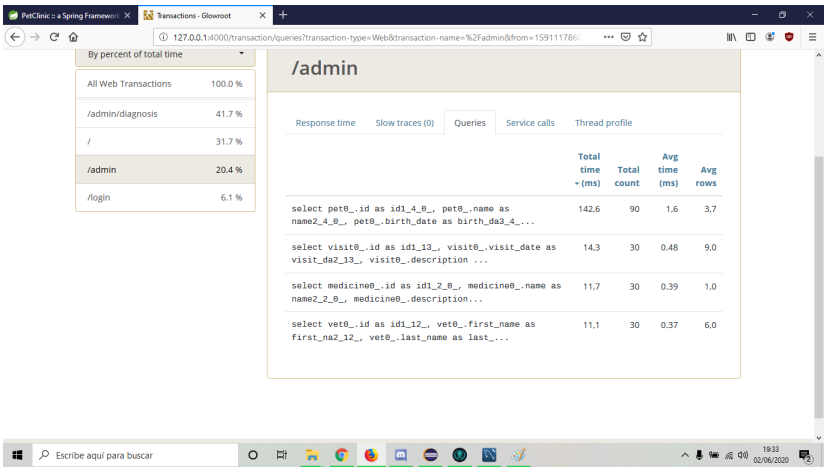
Debido a los cambios comentados, se ha obtenido los siguientes resultados:



Dashboard



Lista de diagnósticos



Dashboard queries

Gracias a los resultados obtenidos antes y después de las modificaciones, se puede concluir:

- En el acceso a dashboard, el tiempo se ha reducido en 7,4ms al haber realizado 5 peticiones.
- En el acceso a la lista de diagnósticos, el tiempo se ha reducido en 325,1 ms habiendo hecho 5 peticiones.

Como conclusión, gracias a la utilización de optimización, se puede observar una gran mejoría en el rendimiento de la historia de usuario 17.

4. Profiling para la historia de usuario 19.

Para recordar la historia de usuario 19, consistía que como administrador se puede conocer la lista de residentes activos en la clínica.

Esta HU no tenía un rendimiento adecuado como se puede observar en el documento de rendimiento. Por lo tanto el equipo ha decido intervenir para mejorar el rendimiento de esta historia de usuario. A modo de recordatorio se adjunta el rendimiento de la misma:

ASSERTIONS

Assertion	Status
Global: max of response time is less than 5000.0	OK
Global: mean of response time is less than 1000.0	OK
Global: percentage of successful events is greater than 95.0	OK

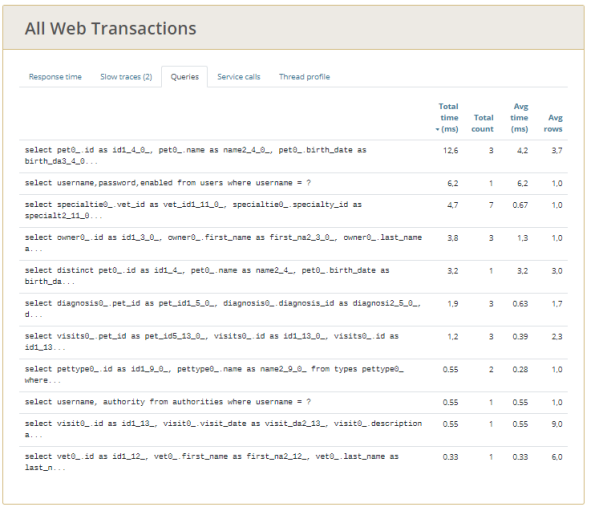
STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	39000	39000	0	0%	131.757	0	5	20	178	465	1791	36	115
Home	6500	6500	0	0%	21.959	2	5	7	10	25	1791	14	119
Login	6500	6500	0	0%	21.959	0	2	3	6	12	1788	11	115
Logged	6500	6500	0	0%	21.959	1	4	5	8	16	1787	12	111
Logged Redirect 1	6500	6500	0	0%	21.959	1	4	6	9	16	1382	6	27
Dashboard	6500	6500	0	0%	21.959	12	31	87	384	535	1380	85	122
ListResidences	6500	6500	0	0%	21.959	12	32	106	406	536	1404	92	128

Para la mejora del rendimiento de esta historia de usuario, el equipo ha realizado un cambio importante:

- **Implementación de una caché**: Se ha añadido con el objetivo de optimizar la lista de residencias activas.

Hemos tomado una serie de capturas de la aplicación Glowroot en la que nos muestra los tiempos de carga de las vistas correspondientes a la lista de residencias activas y dashboard (/admin) antes y después de hacer profiling.



Dashboard queries antes de los cambios.

Con estos resultados, decidimos implementar una caché para la lista de residencias. El código lo hemos cambiado en el xml y en el servicio de ResidenceService:

```java
@Transactional(rollbackFor = NoElementOnListException.class)
public Collection<Pet> findAllResicenceNotEnded() throws DataAccessException, NoElementOnListException{
    Collection<Pet> res = new ArrayList<Pet>();
    res = this.residenceRepository.findAllNotEnded();
    if(res.isEmpty()) {
        throw new NoElementOnListException();
    }
    return res;
}
```

Antes

```java
@Transactional(rollbackFor = NoElementOnListException.class)
@Cacheable("activeResidences")
public Collection<Pet> findAllResicenceNotEnded() throws DataAccessException, NoElementOnLi
    Collection<Pet> res = new ArrayList<Pet>();
    res = this.residenceRepository.findAllNotEnded();
    if(res.isEmpty()) {
        throw new NoElementOnListException();
    }
    return res;
}
```

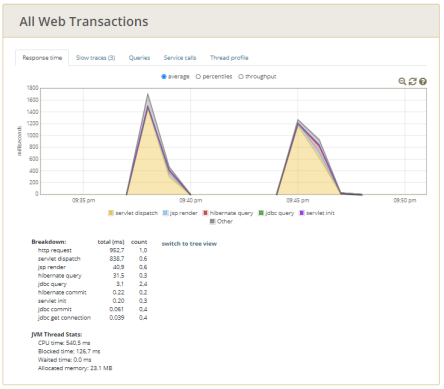
Después

```xml
<cache alias="allDiagnosis" uses-template="default">
    <key-type>org.springframework.cache.interceptor.SimpleKey</key-type>
    <value-type>java.util.Collection</value-type>
</cache>
```

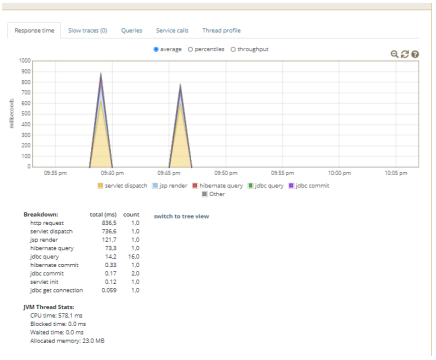
Creación de la caché

Analizando las capturas adjuntadas, podemos observar que hemos añadido la anotación "@Cacheable". Así logramos que el resultado de esta consulta se quede almacenado en caché con el nombre de "activeResidences". La memoria caché se destruirá cuando pasen 2 minutos. En el caso contrario, se debería destruir la caché en el momento que una residencia es añadida al sistema.

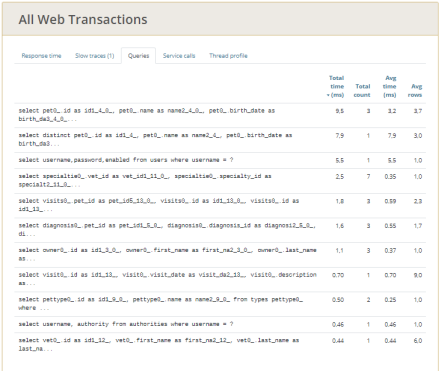
Debido a los cambios comentados, se ha obtenido los siguientes resultados:



Diferencias en Dashboard



Lista de residencias activas



Dashboard queries

Gracias a los resultados obtenidos antes y después de las modificaciones, se puede concluir que en el acceso a dashboard, el tiempo se ha reducido en 3.1 ms al haber realizado 3 peticiones. Por tanto se observa una mejoría en la HU 19 gracias a este proceso de optimización.