Escuela Técnica Superior de
Ingeniería Informática

DP2

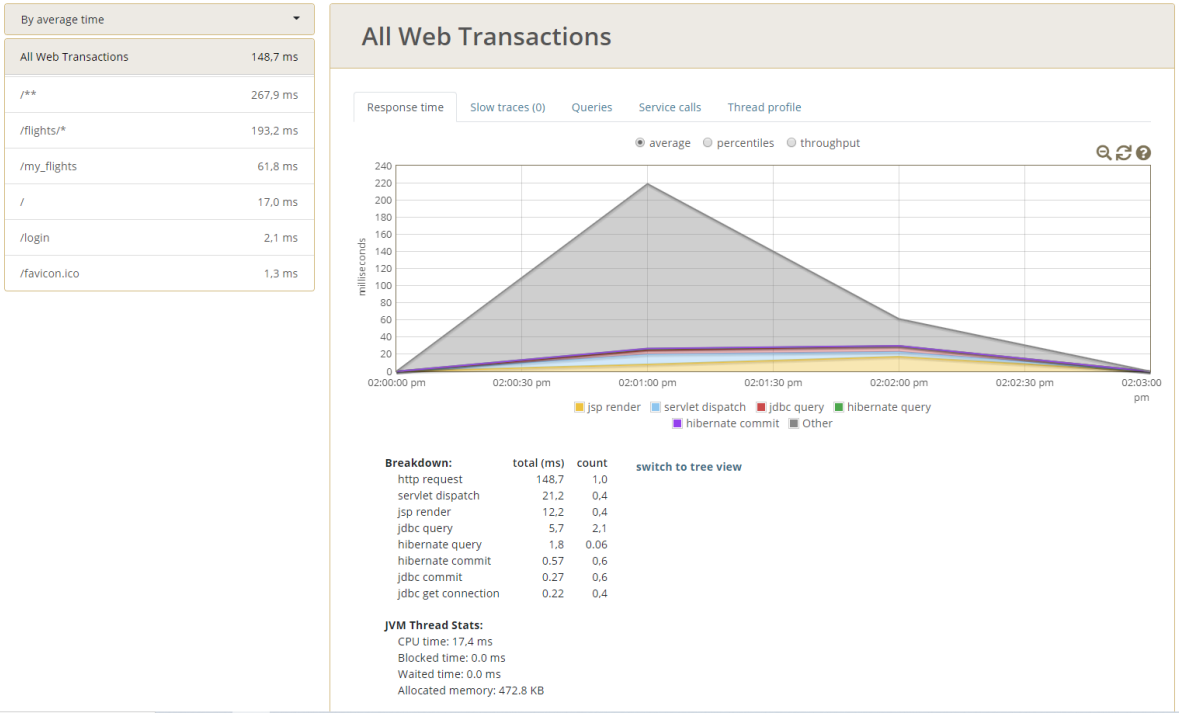
PROFILING

Grupo - G2-1
Miembros del grupo
DANIEL ARELLANO MARTÍNEZ
EDUARDO MIGUEL BOTÍA DOMINGO
JOSE MARTÍN SÁNCHEZ
JUAN NOGUEROL TIRADO
JOSÉ MANUEL SÁNCHEZ RUIZ
JAVIER VÁZQUEZ ZAMBRANO

ÍNDICE

Flight

Por nuestra parte, agrupamos historias de usuario de diferentes entidades para realizar profiling de todas ellas, y en este caso, la historia de usuario correspondiente al listado de los vuelos creados por un trabajador, nos ofrecía el siguiente rendimiento con respecto a las transacciones web:



Para mejorar esto, usamos 3 herramientas de profiling:

- Índices

Para la entidad de Flight, usaremos los siguientes índices:

```
@Table(name = "flights", indexes = { @Index(columnList = "depart_date"), @Index(columnList="reference") })
```

- Cachés

En el xml visto en teoría para tener en cuenta los atributos cacheables (ehcache.xml), se añadió el siguiente código:

```
<cache alias="airlineFlights" uses-template="default">
                <key-type>java.lang.String</key-type>
                <value-type>java.util.Collection</value-type>
</cache>
```

- Projections

Crearemos una clase de projections con los atributos necesarios para listar los vuelos.

```
public interface FlightListAttributes {
	Integer		getLandAirportId();
	String		getLandAirportCity();
	String		getDepartAirportCity();
	Integer		getDepartAirportId();
	String		getPlaneModel();
	Integer		getPlaneId();
	String		getAirlineName();
	Integer		getAirlineId();
	Integer		getId();
	String		getReference();
	Double		getPrice();
	Date		getLandDate();
	Date		getDepartDate();
}
```

Usaremos la clase de projections en servicio y repositorio para crear métodos que se usarán en el controlador.

```
@Override
@Query("SELECT f.id AS id, f.reference AS reference, f.price AS price, "
			+ "f.landDate AS landDate, f.departDate AS departDate, f.airline.name
AS airlineName, f.airline.id AS airlineId, "
			+ "f.plane.id AS planeId, f.plane.model AS planeModel,
f.departes.airport.id AS departAirportId, "
			+ "f.departes.airport.city AS departAirportCity, f.lands.airport.id AS
landAirportId, f.lands.airport.city AS landAirportCity"
			+ " FROM Flight f WHERE f.airline.user.username =:username AND
f.departDate >= current_date()")
List<FlightListAttributes> findAllAirlineFlightListAttributes(@Param("username") String
username) throws DataAccessException;
-----------------------------------------------------------------------------------------------------------------------
List<FlightListAttributes> findAllAirlineFlightListAttributes(String username) throws
DataAccessException;
-----------------------------------------------------------------------------------------------------------------------
@Transactional(readOnly = true)
@Cacheable("airlineFlights")
public List<FlightListAttributes> findAllAirlineFlightListAttributes(String username) throws
DataAccessException{
		return flightRepository.findAllAirlineFlightListAttributes(username);
	}
```

Hacemos la siguiente modificación en el código del método del controlador para el listado de vuelos:

```
Collection<Flight> flights = this.flightService.findAirlineFlight(username);
Collection<FlightListAttributes> flights =
this.flightService.findAllAirlineFlightListAttributes(username);
model.put("flights", flights);
```

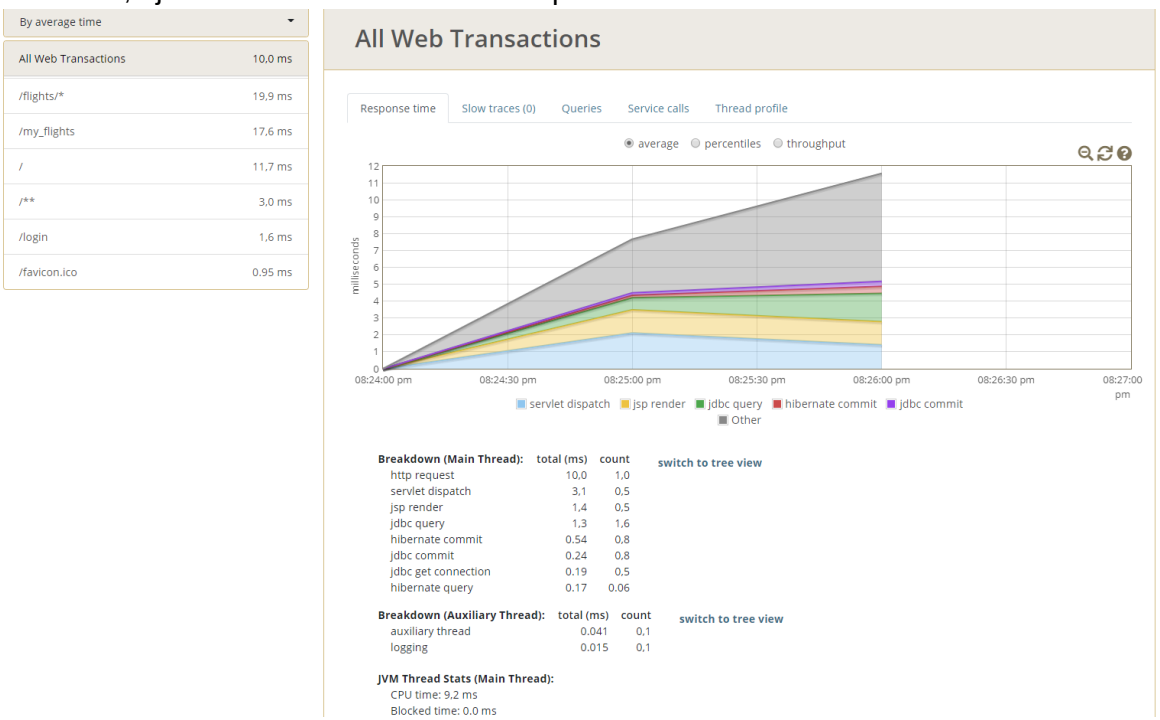
Para aplicar estos cambios en la vista de la aplicación, sustituimos el código de acuerdo a la nueva clase de projections

```
<td>
<spring:url value="/airports/{airportId}" var="landsUrl">
  <spring:param name="airportId" value="${flight.lands.airport.id}" />
</spring:url>
        <a href="${fn:escapeXml(landsUrl)}">
        <c:out value="${flight.lands.airport.city}" /></a>
</td>
```

Lo sustituimos de la siguiente forma:

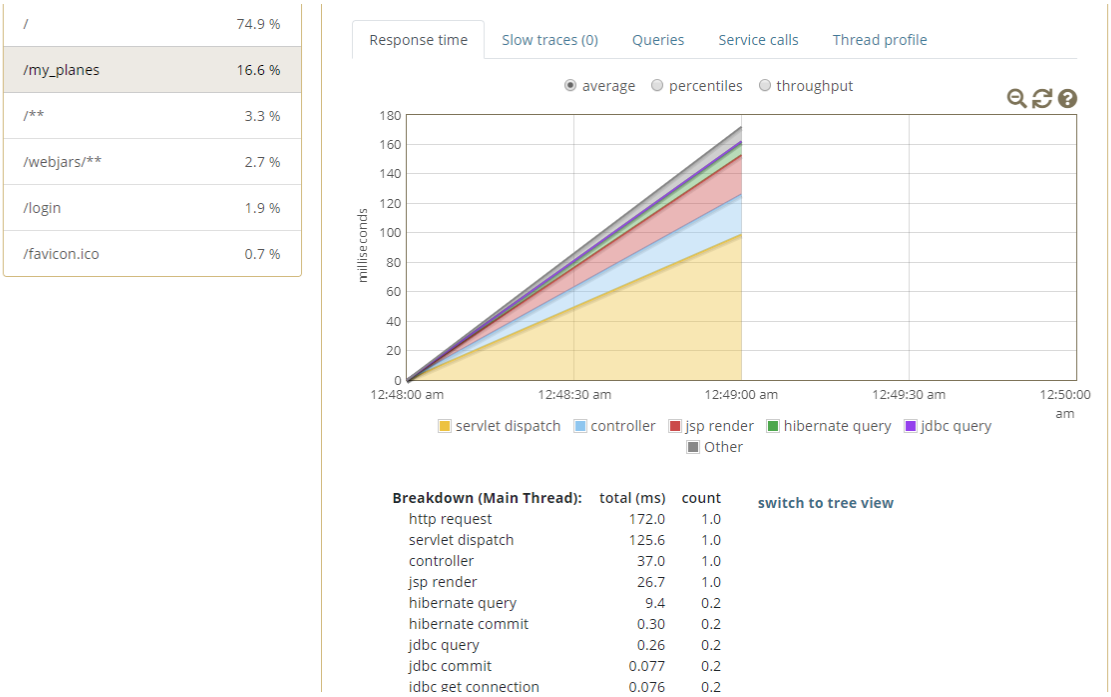
```
<td>
<spring:url value="/airports/{airportId}" var="landsUrl">
  <spring:param name="airportId" value="${flight.landAirportId}" />
</spring:url>
        <a href="${fn:escapeXml(landsUrl)}">
                <c:out value="${flight.landAirportCity}" /></a>
</td>
```

Finalmente, ejecutamos de nuevo la misma prueba en Glowroot:



Airline

La siguiente historia de usuario que hemos tratado de mejorar el rendimiento se trata de la HU-T02, Trabajador lista sus aviones. El rendimiento que hemos obtenido antes de realizar el profiling del código, es el siguiente que se muestra a continuación:



(seleccionamos “/my_planes”, ya que es la vista en la queremos mejorar el rendimiento)

Para mejorarlo, hemos utilizado dos herramientas de profiling:

- Cachés

En el xml visto en teoría para tener en cuenta los atributos cacheables (ehcache.xml), se añadió el siguiente código:

```
<cache alias="planesByAirline" uses-template="default">
        <key-type>java.lang.String</key-type>
        <value-type>java.util.Collection</value-type>
</cache>
```

- Projections

Crearemos una clase de projections con los atributos necesarios para listar los aviones.

```
public interface PlaneListAttributes {
    String      getId();
    String      getReference();
    String      getMaxSeats();
    String      getDescription();
    String      getManufacter();
    String      getModel();
    String      getNumberOfKm();
    String      getMaxDistance();
    Date        getLastMaintenance();
```

```
}
```

Usaremos la clase de projections en servicio y repositorio para crear métodos que se usarán en el controlador.

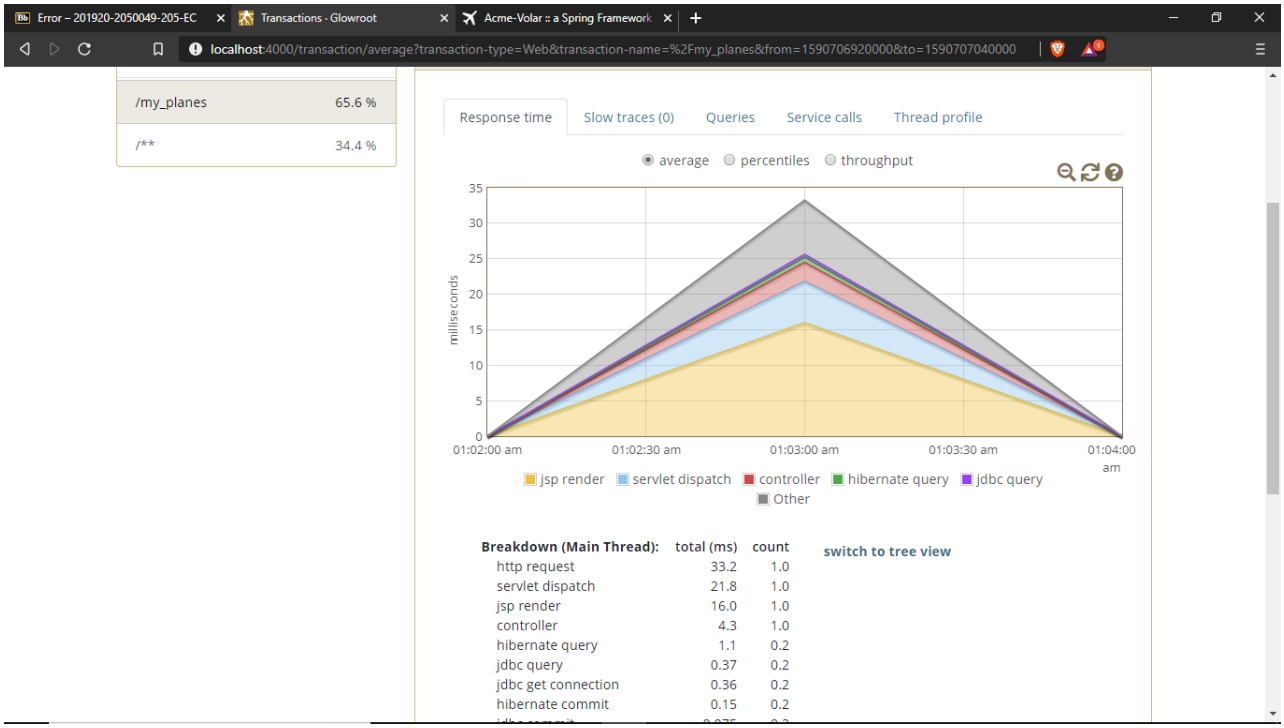
```
@Override
@Query("SELECT p.id AS id, p.reference AS reference, "
+ "p.maxSeats AS maxSeats, p.description AS description, p.manufacter AS manufacter, "
+ "p.model AS model, p.numberOfKm AS numberOfKm, p.maxDistance AS maxDistance,
p.lastMaintenance AS lastMaintenance "
                    + " FROM Plane p WHERE p.airline.user.username =:airline"
)
List<PlaneListAttributes> findAllAirlinePlaneListAttributes(@Param("airline") String airline)
throws DataAccessException;
-----------------------------------------------------------------------------------------------------------------
List<airline> airline(String airline) throws DataAccessException;
-----------------------------------------------------------------------------------------------------------------
@Transactional(readOnly = true)
@Cacheable("planesByAirline")
public List<PlaneListAttributes> findAllAirlinePlaneListAttributes(String airline) throws
DataAccessException{
                    return planeRepository.findAllAirlinePlaneListAttributes(airline);
          }
```

Hacemos la siguiente modificación en el código del método del controlador para el listado de vuelos:

```
Collection<Plane> planes= this.planeService.getAllPlanesFromAirline(username);
Collection<PlaneListAttributes> planes =
this.planeService.findAllAirlinePlaneListAttributes(username);
model.put("planes", planes);
```

Para aplicar estos cambios en la vista de la aplicación, no tenemos nada más que hacer, ya que hemos tratado de usar los mismos nombres que se utilizaban en la vista antigua.

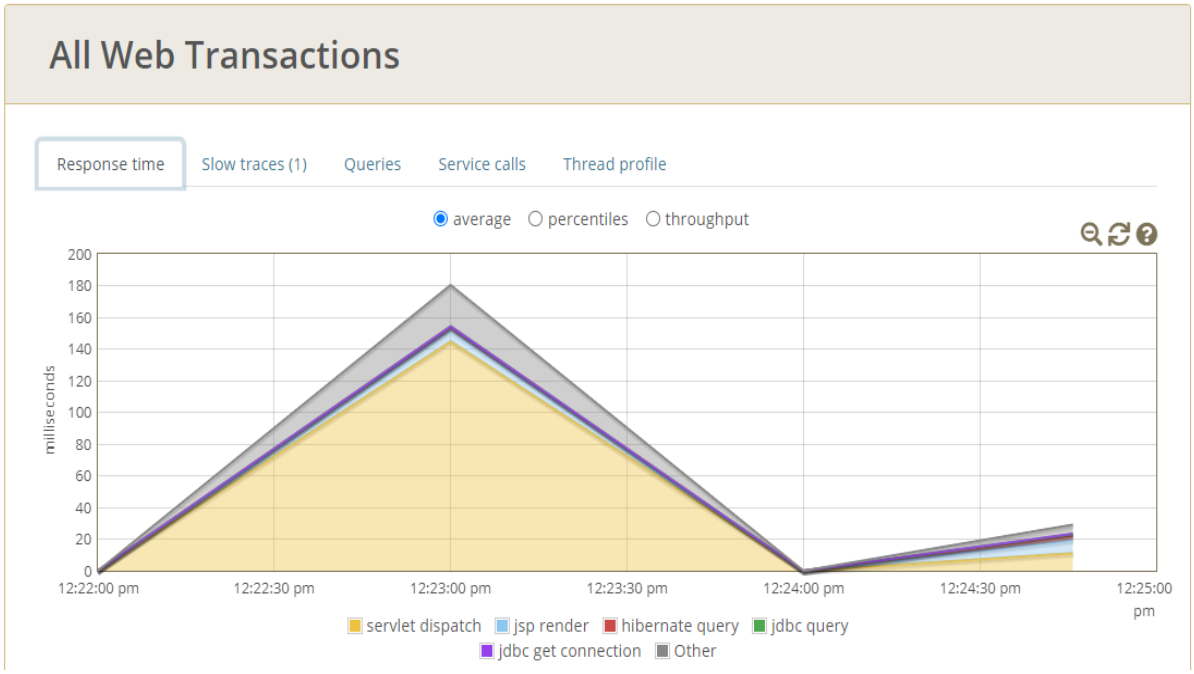
Ejecutamos de nuevo la misma prueba en Glowroot para observar la mejora:



(seleccionamos de nuevo “/my_planes”, ya que es la vista cuyo rendimiento hemos tratado de mejorar)

Airport

La historia de usuario de la que hemos hecho profiling ha sido ‘Trabajador lista aeropuerto’. Glowroot nos proporcionó este análisis antes de realizar los cambios:



Para optimizar esta historia, hemos usado dos herramientas:

- Cachés

En primer lugar, hemos añadido al fichero ehcache3.xml el código presentado a continuación:

```
<cache alias=”listAirports” uses-template=”default”>
        <value-type>java.util.List</value-type>
</cache>
```

A su vez, añadimos el siguiente código señalado a la clase AirportService:

```
@Transactional(readOnly=true)
@Cacheable(“listAirports”)
public List<AirportListAttributes> findAirportListAttributes() throws DataAccessException {
        return this.airportRepository.findAllAirportAttributes();
}
```

También se añadió esta línea de código a los métodos saveAirport y deleteAirport:

```
@CacheEvict(cacheName = “listAirpots”, allEntries = true)
```

Como se puede observar, en este método se hace uso de la clase AirportListAttributes, que explicaremos más adelante.

Podemos observar que este cambio tuvo efecto y mejoró el rendimiento simplemente observando la consola de Eclipse. En esta primera imagen, si accedemos al listado de aeropuertos dos veces, vemos que se realizan 2 consultas:
Mientras que, tras los cambios realizados, la misma operación solo realiza una consulta, lo que se ve también reflejado en el análisis de Glowroot:

```
Hibernate:
    select
        airport0_.id as id1_2_,
        airport0_.name as name2_2_,
        airport0_.city as city3_2_,
        airport0_.code as code4_2_,
        airport0_.latitude as latitude5_2_,
        airport0_.longitude as longitud6_2_,
        airport0_.max_number_of_clients as max_numb7_2_,
        airport0_.max_number_of_planes as max_numb8_2_
    from
        airports airport0_
Hibernate:
    select
        airport0_.id as id1_2_,
        airport0_.name as name2_2_,
        airport0_.city as city3_2_,
        airport0_.code as code4_2_,
        airport0_.latitude as latitude5_2_,
        airport0_.longitude as longitud6_2_,
        airport0_.max_number_of_clients as max_numb7_2_,
        airport0_.max_number_of_planes as max_numb8_2_
    from
        airports airport0_
```

```
Hibernate:
    select
        airport0_.id as id1_2_,
        airport0_.name as name2_2_,
        airport0_.city as city3_2_,
        airport0_.code as code4_2_,
        airport0_.latitude as latitude5_2_,
        airport0_.longitude as longitud6_2_,
        airport0_.max_number_of_clients as max_numb7_2_,
        airport0_.max_number_of_planes as max_numb8_2_
    from
        airports airport0_
2020-05-27 12:30:05.786  INFO 20992 --- [e [_default_]-0] acmevolar.configuration.CacheLogger      : Key: SimpleKey [] |
 EventType: CREATED | Old value: null | New value: [Sevilla Airport, Adolfo Suárez Madrid-Barajas Airport, El Prat Airpo
rt, Charles de Gaulle Airport, Aeropuerto Federico García Lorca Granada-Jaén, Aeropuerto de Almería, Aeropuerto de Alica
nte-Elche, Aeropuerto de Málaga-Costa del Sol, Aeropuerto de Huesca-Pirineos]
```

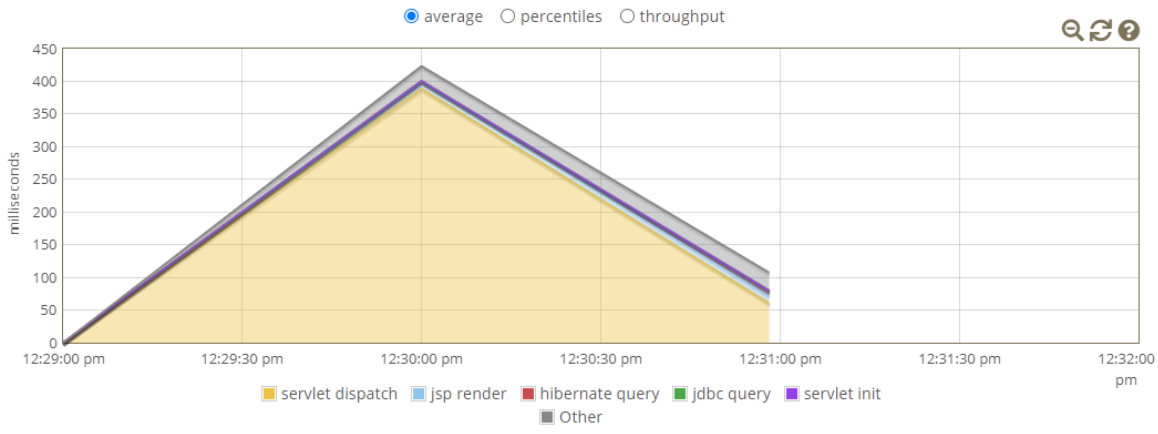
All Web Transactions

Response time | Slow traces (1) | Queries | Service calls | Thread profile

	Total time (ms)	Total count	Avg time (ms)	Avg rows
`select username,password,enabled from users where username = ?`	2,0	1	2,0	1,0
`select airport0_.id as id1_2_, airport0_.name as name2_2_, airport0_.city as city3_2_, ai...`	0.73	1	0.73	9,0
`select username, authority from authorities where username = ?`	0.57	1	0.57	1,0

All Web Transactions

Response time | Slow traces (1) | Queries | Service calls | Thread profile



- Projections

En primer lugar creamos la clase AirportListAttributes, que incluye el siguiente código, con los atributos de la clase Airport que queremos mostrar:

```
package acmevolar.projections;

public interface AirportListAttributes {

	String getId();
	String getName();
	Integer getMaxNumberOfPlanes();
	Integer getMaxNumberOfClients();
	Double getLatitude();
	Double getLongitude();
	String getCode();
	String getCity();

}
```

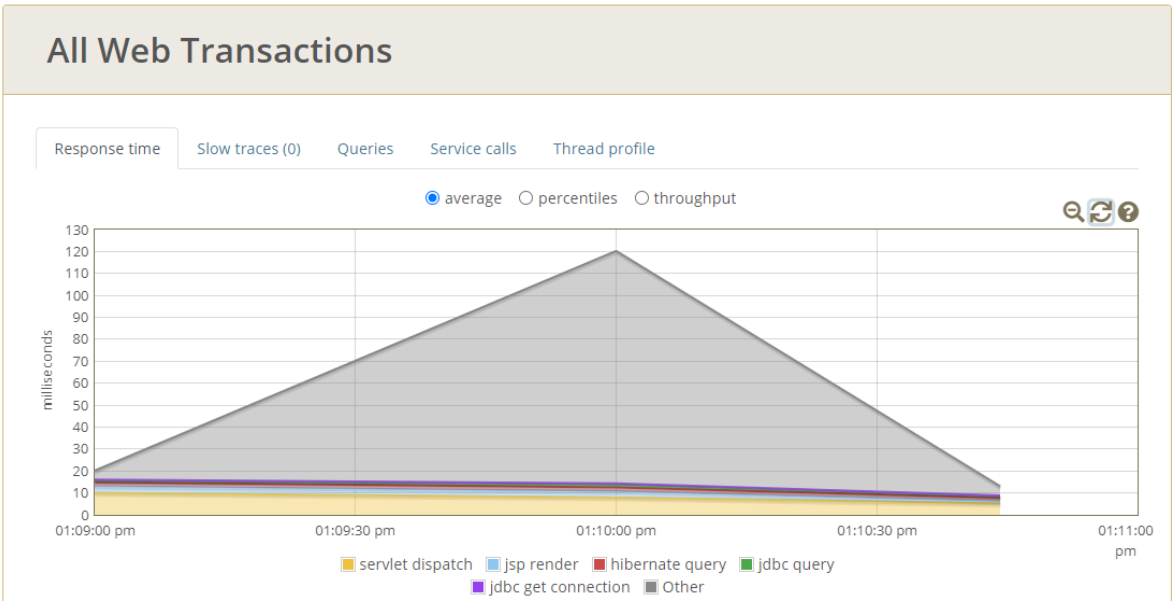
A su vez, introducimos la siguiente consulta en AirportRepository:

```
@Override
@Query("SELECT a.id AS id, a.name AS name, a.maxNumberOfPlanes AS maxNumberOfPlanes, a.maxNumberOfClients AS maxNumberOfClients, a.latitude AS latitude, a.longitude AS longitude, a.code AS code, a.city AS city FROM Airport a")
List<AirportListAttributes> findAllAirportAttributes();
```

Así mismo, habría que alterar servicio, controlador y vistas para llevar a cabo estas modificaciones.

Runway

La historia de usuario de la que hemos hecho profiling ha sido ‘Trabajador lista runway’. Glowroot nos proporcionó este análisis:



	Total time (ms)	Total count	Avg time (ms)	Avg rows
select airport0_.id as id1_2_, airport0_.name as name2_2_, airport0_.city as city3_2_, a...	1,5	3	0.49	1,0
select runway0_.id as id1_11_, runway0_.airport_id as airport_3_11_, runway0_.name as na...	1,2	2	0.62	5,0
select runwaytype0_.id as id1_12_0_, runwaytype0_.name as name2_12_0_ from runway_type r...	1,1	4	0.29	1,0
select airport0_.id as id1_2_0_, airport0_.name as name2_2_0_, airport0_.city as city3_2...	0.81	2	0.41	1,0

De nuevo optamos por aplicar la herramienta de caché.

- Cachés

En primer lugar, añadimos al fichero ehcache3.xml el siguiente código:

```
<cache alias=”listRuwaysByAirportId” uses-template=”default”>
        <key-type>java.lang.Integer</key-type>
        <value-type>java.util.List>/value-type>
</cache>
```

Añadimos el siguiente código a la clase RunwaytService:

```
@Transactional(readOnly=true)
@Cacheable(“listRuwaysByAirportId”)
public List<Runway> findRunwaysByAirportId(final Integer airportId) throws
DataAccessException {
        return this.runwayRepository.findRunwaysByAirportId(airportId);
}
```

También se añadió esta línea de código a los métodos saveRunway y deleteRunway:

```
@CacheEvict(cacheName = “listRunwaysByAirpotId”, allEntries = true)
```

De nuevo, podemos ver que la consulta se realiza 2 veces al acceder en 2 ocasiones al listado de runways de un aeropuerto y, al realizar los cambios realizados, solo una:

Después de los cambios:

```
Hibernate:
    select
        runwaytype0_.id as id1_12_0_,
        runwaytype0_.name as name2_12_0_
    from
        runway_type runwaytype0_
    where
        runwaytype0_.id=?
Hibernate:
    select
        runwaytype0_.id as id1_12_0_,
        runwaytype0_.name as name2_12_0_
    from
        runway_type runwaytype0_
    where
        runwaytype0_.id=?
2020-05-31 13:33:55.713  INFO 10312 --- [e [_default_]-0] acmevolar.configuration.CacheLogger      : Key: 1 | EventType:
 CREATED | Old value: null | New value: [Runway [name=A-01, type=take_off, airport=Sevilla Airport], Runway [name=A-06,
type=landing, airport=Sevilla Airport], Runway [name=A-08, type=landing, airport=Sevilla Airport], Runway [name=A-09, ty
pe=take_off, airport=Sevilla Airport], Runway [name=A-38, type=take_off, airport=Sevilla Airport]]
```

Antes de los cambios:

```
        runwaytype0_.id=?
Hibernate:
    select
        runwaytype0_.id as id1_12_0_,
        runwaytype0_.name as name2_12_0_
    from
        runway_type runwaytype0_
    where
        runwaytype0_.id=?
Hibernate:
    select
        airport0_.id as id1_2_,
        airport0_.name as name2_2_,
        airport0_.city as city3_2_,
        airport0_.code as code4_2_,
        airport0_.latitude as latitude5_2_,
        airport0_.longitude as longitud6_2_,
        airport0_.max_number_of_clients as max_numb7_2_,
        airport0_.max_number_of_planes as max_numb8_2_
    from
        airports airport0_
    where
        airport0_.id=?









Hibernate:
    select
        runway0_.id as id1_11_,
        runway0_.airport_id as airport_3_11_,
        runway0_.name as name2_11_,
        runway0_.runway_type_id as runway_t4_11_
    from
        runway runway0_
    left outer join
        airports airport1_
            on runway0_.airport_id=airport1_.id
    where
        airport1_.id=?
```