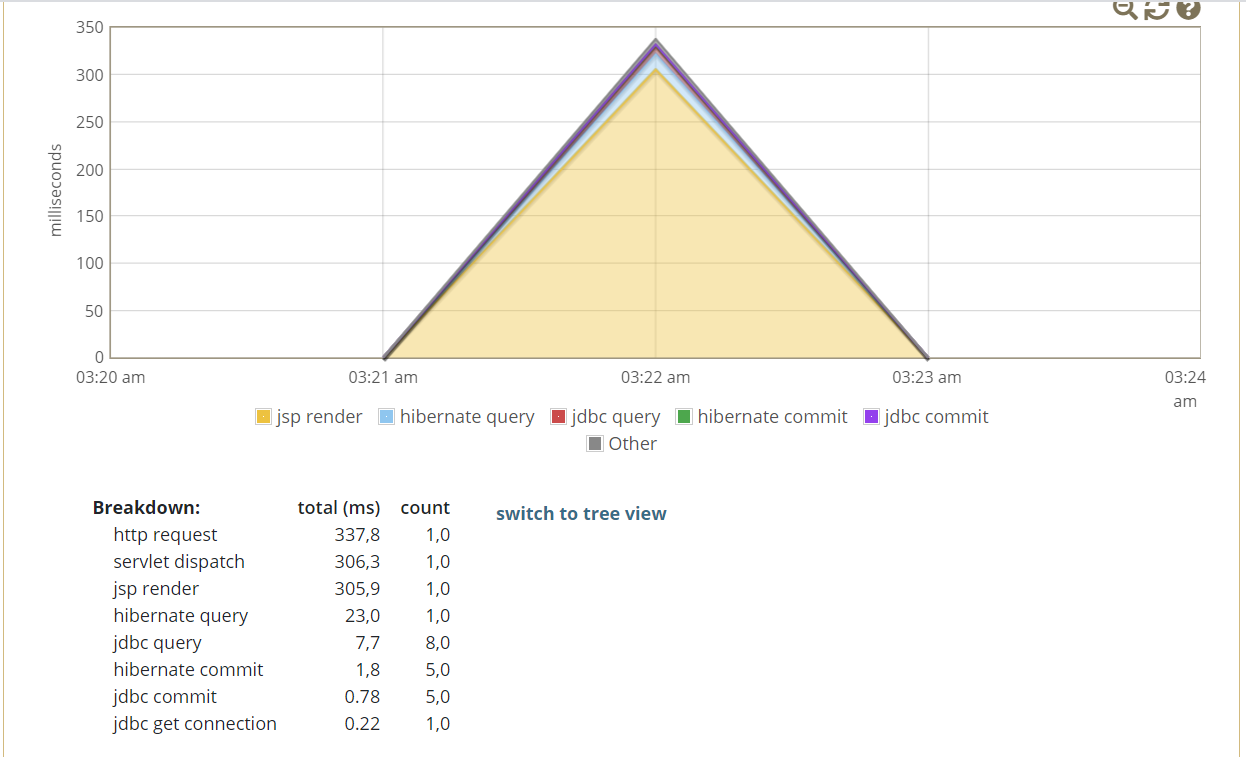
**-Profiling mostrar tratamientos de informe-**

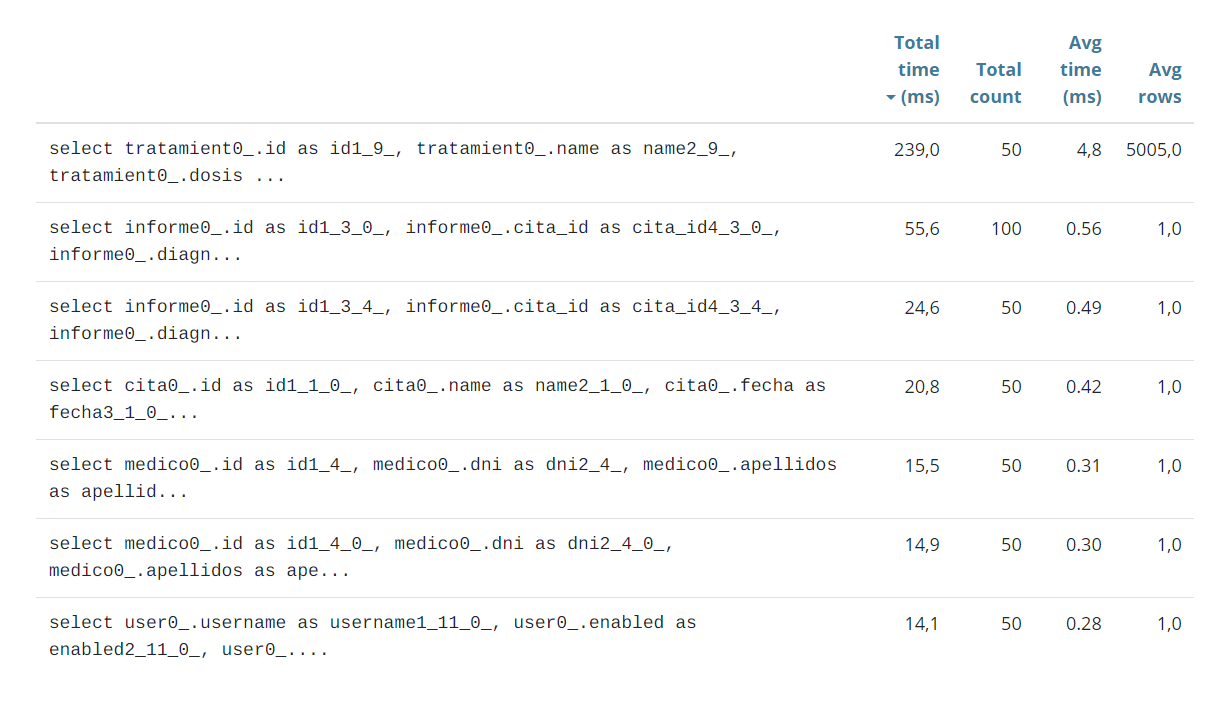
Al realizar las pruebas de performance nos dimos cuenta de que uno de los casos más llamativos se producía al mostrar un informe que contenía muchos tratamientos pues la query devolvía todos los que un médico ha creado en el informe, así como el render de la vista debía mostrar un gran número de tratamientos en la vista de Informe. Esto se traduce en un coste computacional demasiado alto y provoca que la experiencia de usuario se viera afectada.

En la siguiente imagen podemos ver un trozo del report Mostrar Tratamiento en el que podemos ver como esto ralentiza nuestro sistema.

Para confirmar esta idea hemos utilizado un script de Gatling para poblar la base de datos con 5000 tratamientos en un Informe, otro script para realizar accesos por parte de 50 usuarios a este informe y finalmente hemos consultado los tiempos de respuesta en una herramienta de profiling (Glowroot).



Se ha de notar, que el tiempo que consume el render del jsp es muy alto. Esto se debe a que la consulta recoge todos los tratamientos del informe. Podemos observar cómo, de todas las consultas realizadas para este listado, la query que no devuelve los tratamientos asociados a un respectivo informe es la que consume un mayor tiempo en ejecutarse.



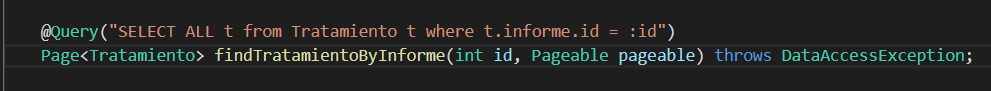
Esta consulta nos devuelve todo lo necesario para el listado, pero podemos observar cómo nos devuelve una gran cantidad de datos y requiere de varias relaciones entre tablas.

Imagen que contiene pájaro, ave

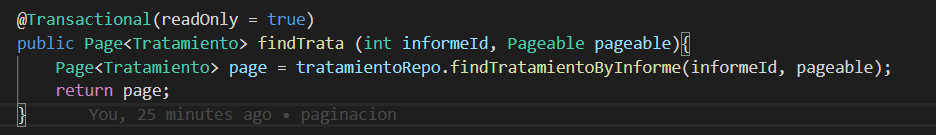
Descripción generada automáticamente

Para solucionar este problema creemos que la mejor solución es añadir paginación.

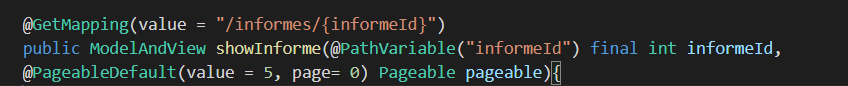
Para añadir paginación hemos añadido un nuevo query con una respuesta Page (Tipo presente en las librerías de Sprint-Boot):

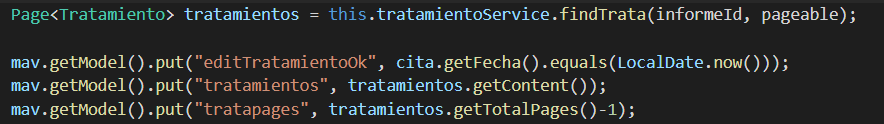


Un método de servicio que hace una llamada a este query:

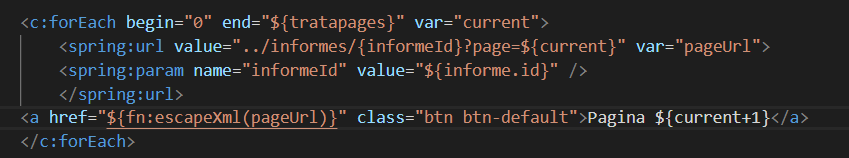


Hemos modificado la sección del método del controlador que se encarga de mostrar los informes para que añada al modelo los datos que requerimos (El contenido del Page y el número total de páginas) así como la cabecera para establecer los valores por defecto de la paginación.

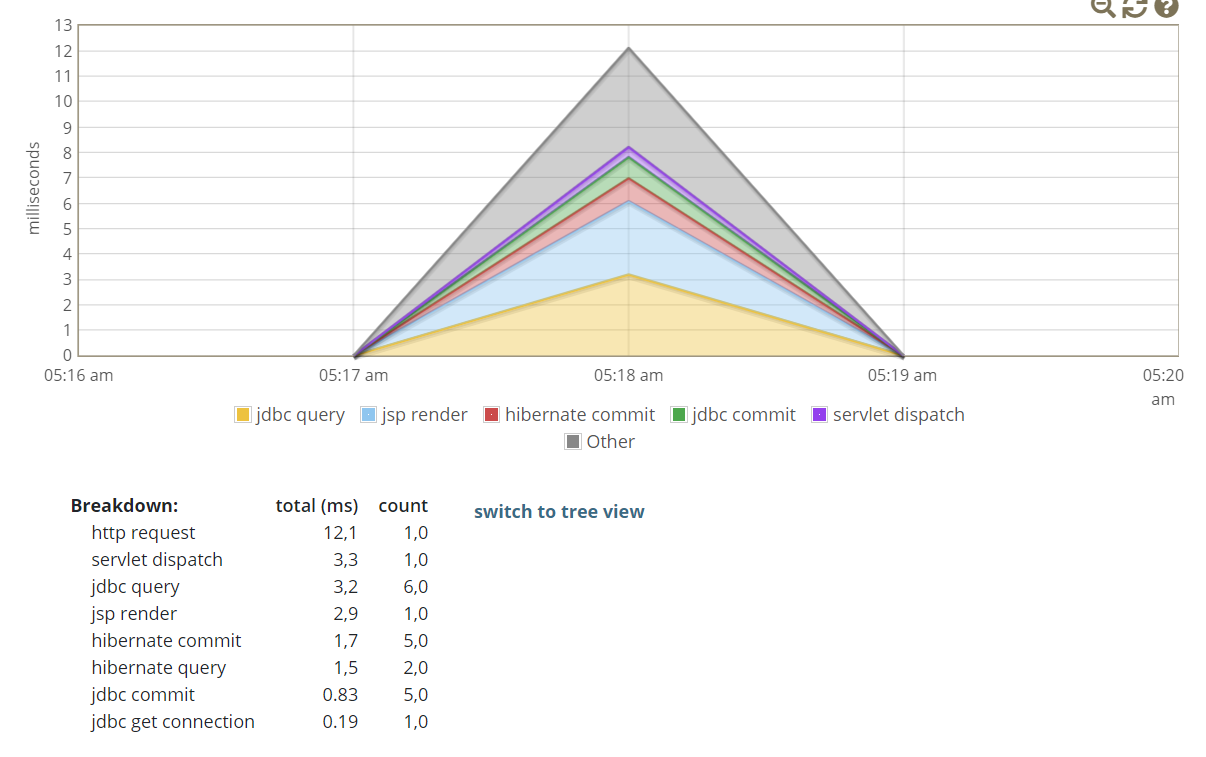




Y finalmente hemos añadido el bucle que se encarga de añadir los botones para las páginas en la vista jsp:



Tras realizar estos cambios, hemos poblado nuevamente la base de datos con 5000 tratamientos mediante un script de Gatling y posteriormente hemos usado otro script para realizar accesos por parte de 50 usuarios. Así, recibimos estos nuevos datos de la herramienta Glowroot:



Como podemos observar, tras realizar los cambios y añadir paginación, los tiempos de respuesta se reducen drásticamente. De esta forma conseguimos que mejore la experiencia del usuario al usar el sistema. En la imagen podemos ver que el tiempo total de ‘’jdbc query’’ ha bajado hasta los 3,2ms y su count; mientras que el tiempo total que consume el render jsp ha sido notablemente reducido hasta 2,9 ms.

Captura de pantalla de un celular

Descripción generada automáticamente

En esta imagen podemos observar que el tiempo de ejecución de dicha query también se ha reducido hasta un tiempo medio de 0.41ms.

Imagen que contiene ave

Descripción generada automáticamente