Instituto Politécnico Nacional

Escuela Superior de Cómputo

Desarrollo de Sistemas Distribuidos

Prof. **Benjamín Cruz Torres**

Práctica No. 10
Coordinación en Sistemas Distribuidos

Grupo: 4CV3

Equipo: 6

Integrantes:

1. Acosta Rosales Jair Sebastián
2. De Jesús López David
3. Galicia Vargas Gerardo
4. Martínez Marcos Luis Eduardo
5. Octaviano Lima Elvia Jaqueline

Fecha:{06/12/2018 }

Práctica 10: Coordinación en sistemas distribuidos

Objetivo de la Práctica

Dentro de un sistema distribuido replicado, cuya función es repartir cartas, aplicar un algoritmo de coordinación para mantener la transparencia.

Tecnologías a aplicar

Sockets, RMI, SOAP, Hilos (threads), POO, Protocolos de comunicación, Bases de Datos, algoritmos de sincronización, algoritmos de exclusión mutua, algoritmos de coordinación, replicación, consistencia y coordinación.

Actividades

Desarrollar una aplicación que reparta cartas a los usuarios a través de una Base de Datos, la aplicación estará compuesta por un sistema distribuido con tres equipos servidores (equipos separados o máquinas virtuales). Los tres equipos funcionarán como uno solo y proporcionarán el servicio a los usuarios. Utiliza un modelo de replicación para trabajar las réplicas. El sistema deberá seguir funcionando aunque uno o dos de los coordinadores (coordinador principal) dejen de funcionar.

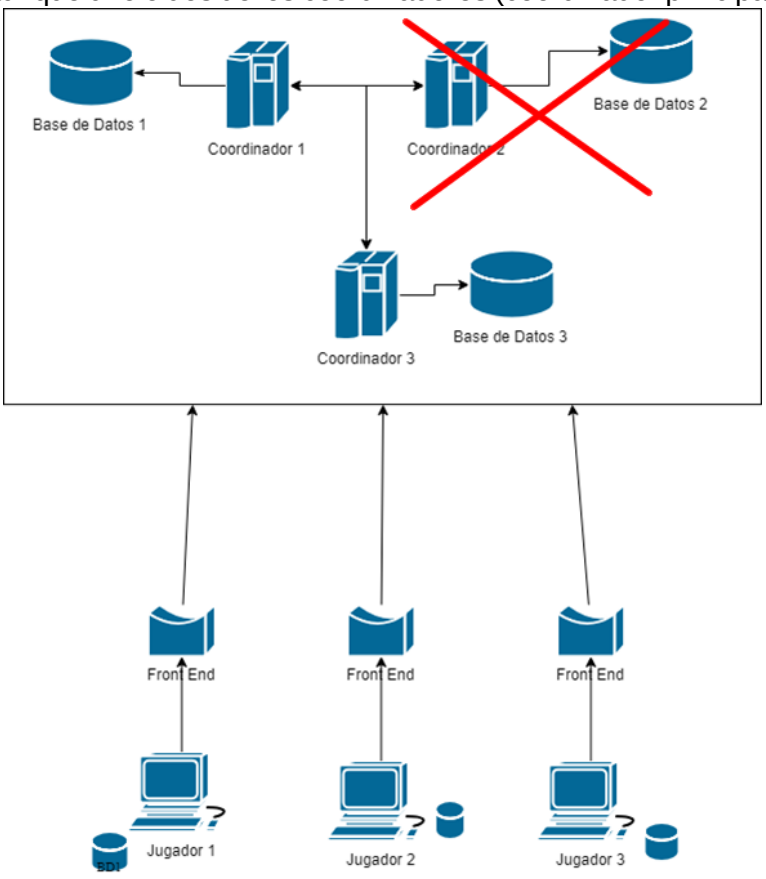


Figura 1. Replicación y Coordinación de Sistemas Distribuidos

Requerimientos funcionales

- Los jugadores se conectarán al sistema principal a través de los Front Ends.
- Los Front Ends elegirán a qué coordinador conectarse (use el modelo de replicación que guste).
- Los coordinadores mantendrán un registro de sus actividades en sus BD correspondiente.
- Las Bases de Datos de los coordinadores siempre tendrán la misma información

- Cada vez hay alguna modificación (escritura) en la BD, se copiarán los cambios a las demás.
- Aunque uno o dos de los coordinadores dejen de funcionar el sistema deberá seguir proporcionando el servicio.
- *Cuando el coordinador caído vuelva a levantarse se reincorporará a la lista de coordinadores activos.

Requerimientos no funcionales

- Los jugadores verán al sistema como una sola computadora.
- Utilice un modelo de replicación para trabajar las réplicas.
- Mantenga la consistencia en las bases de datos.
- Utilice un algoritmo de sincronización para mantener una mejor consistencia.

Introducción

Parte de la investigación para la realización de esta práctica se centro en el front end, debido a que en un inicio el concepto de este parecía muy confuso, y chocaba con otro concepto de front end que teníamos, por lo que hubo que aclarar dudas con el profesor respecto al funcionamiento correcto, y al final se concluyo que el front end no es más que una clase encargada de hacer la conexión al servidor primario en ese momento, que constantemente recibe actualizaciones para saber a qué servidor hacer las peticiones del cliente, lo que le permite hacer una conexión inmediata en lugar de ir probando cada ip de cada servidor y perder tiempo esperando la respuesta.

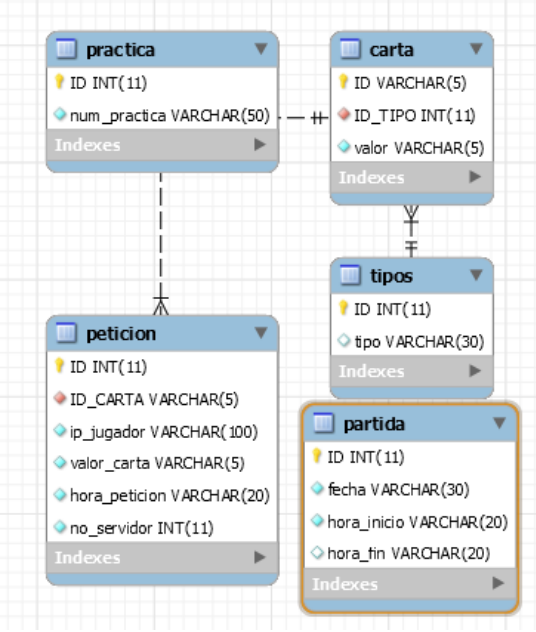
También hubo que hacer un reajuste de todo lo que era el levantamiento de servidores, debido a que originalmente los servidores se iniciaban con cierto orden en nuestros programas, primeramente S1, S2 y S3 esto debido a que cada servidor nuevo que iniciaba requería de los primero que se iniciaban para hacer la conexión entre estos y que pudieran estar en constante comunicación bidireccional(gracias a la técnica de callbacks de RMI).

Desarrollo de la práctica

Considerando el desarrollo de las anteriores prácticas, se partió del punto en que ya se contaba con la funcionalidad de los relojes y la comunicación entre los clientes y los servidores.

Para la comunicación entre servidores se hizo uso de algo conocido como callbacks, de modo pudiéramos conectar dos servidores y ambos hicieran inserts remotos a la base de datos del otro servidor, además se hizo uso de un contador lógico que, con la ayuda de la implementación del algoritmo de Lamport fue de utilidad para llevar una correcta sincronización entre los eventos que pasaban uno después del otro entre ambos servidores, aunque como bien ya se mencionó en la introducción se eliminó la parte de inicio de orden de los servidores, ahora es posible iniciar cualquiera de los tres servidores y estos deben ser capaces de conectarse entre si.

Para la persistencia se consideró un modelo de base datos con las tablas necesarias para almacenar la información justa sobre las partidas, las cartas y la asignación de estas a los jugadores. El modelo que se realizó es el mostrado a continuación:



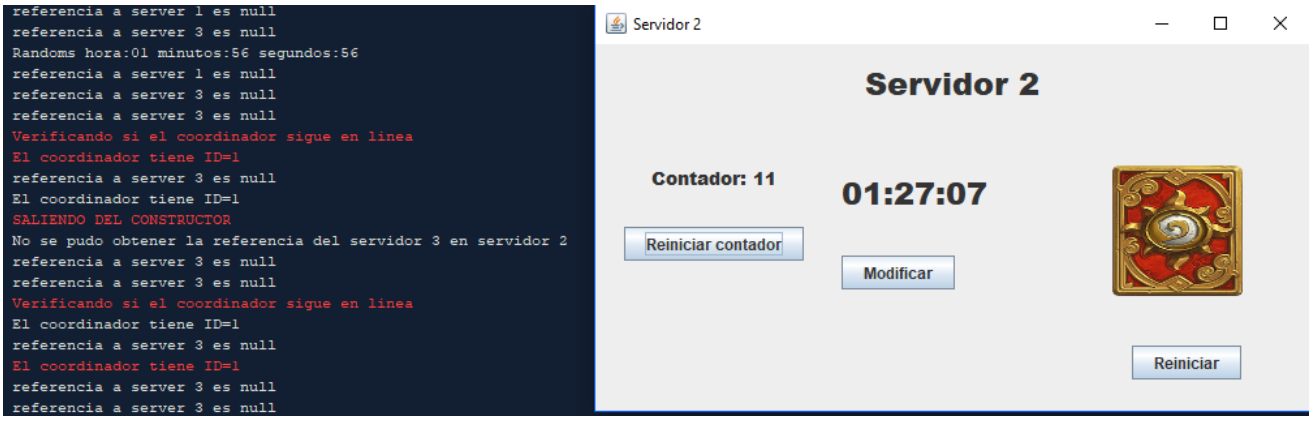
Pruebas de servidores y clientes

Iniciando server 1

Al momento de iniciar comienza a verificar si hay otros servidores conectados y además si los front end de los jugadores también ya se conectaron.

El id de este server es 1 por lo que tiene la prioridad más alta para ser el server primario.

Iniciando server 2

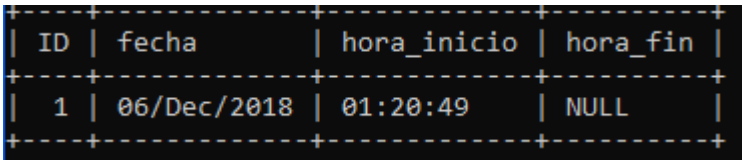


Vemos que el servidor al iniciar no tiene referencia ni a S1 y S3, pero pasado unos instantes detecta a S1 por lo que solo desconoce a S3, además es conciente de que el servidor primario tiene id = 1 es decir el server 1 y constantemente le manda mensaje para saber si sigue en línea.

Iniciando server 3

Al iniciarse el server 3 este detecta a S1 y S2 por lo que vemos que se efectúan las consultas remotas para agregar el registro de una partida nueva.

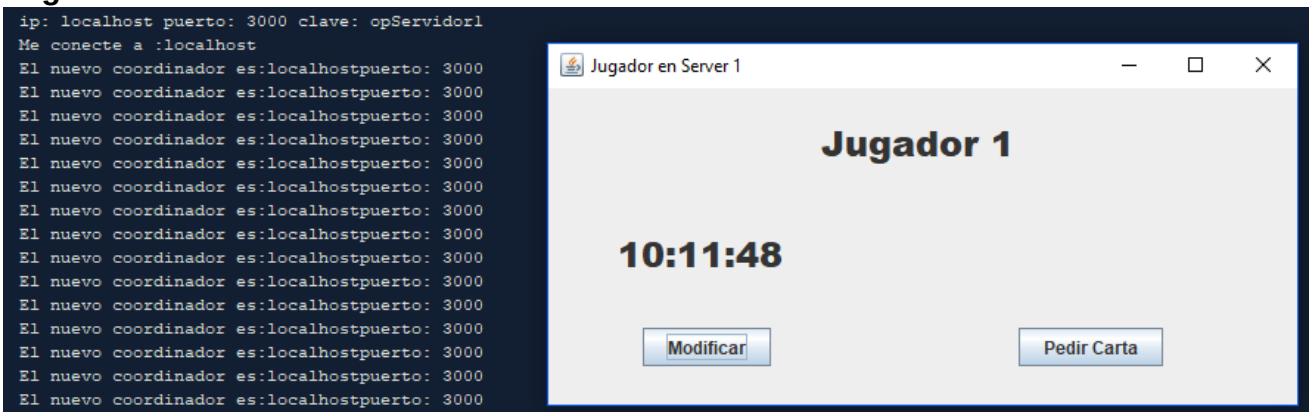
Además después de esto tanto S1 y S2 tienen referencia a S3 por lo tanto deja de ser null.

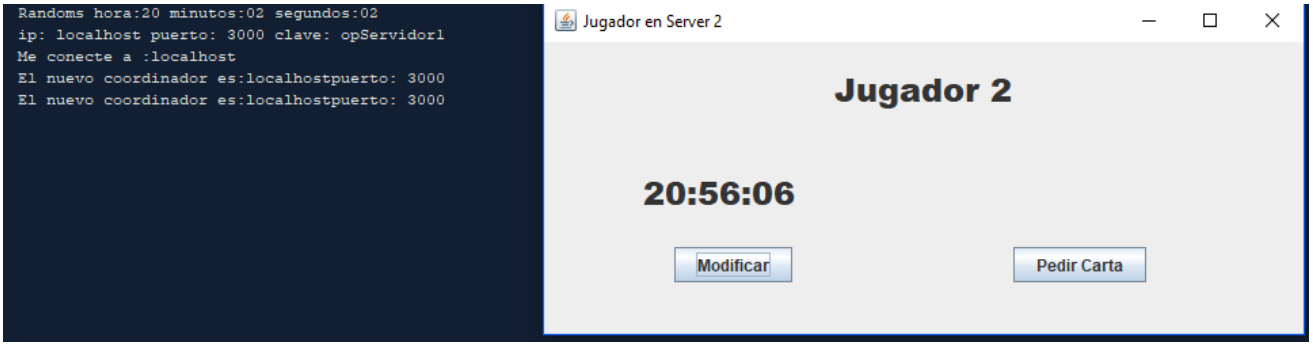


ID	fecha	hora_inicio	hora_fin
1	06/Dec/2018	01:20:49	NULL

Se hace el registro de la nueva partida en la BD

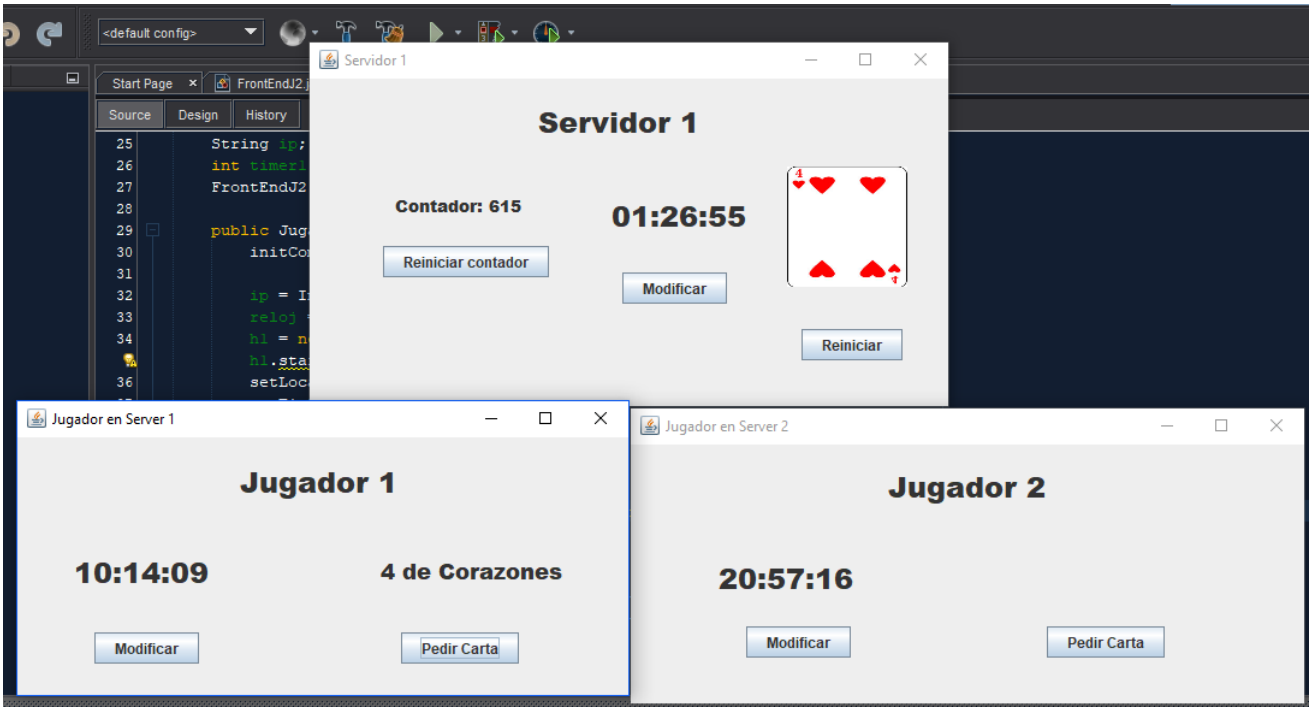
Jugadores conectándose

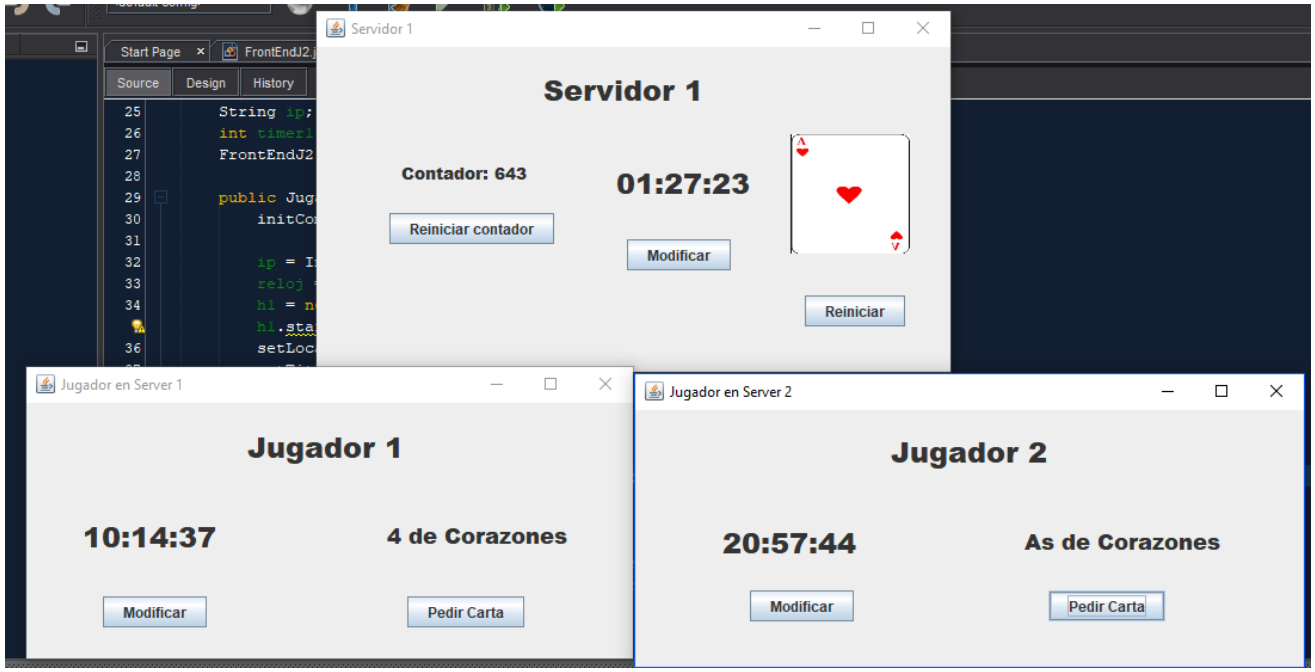




Cada jugador recibe constantemente la actualización de quien es el nuevo coordinador

Pidiendo cartas





Tirando Server 1

El server 2 es quien atiende ahora las peticiones

Tirando Server 2

S3 ahora responde a las peticiones

Termina la partida

Revisando las bases de datos

BD en server 1

```
mysql> SELECT * FROM peticion;
+----+----------+-----------------------------+-------------+---------------+-------------+
| ID | ID_CARTA | ip_jugador                  | valor_carta | hora_peticion | no_servidor |
+----+----------+-----------------------------+-------------+---------------+-------------+
|  1 | 14       | Sebastián-AR/192.168.1.74   | 4           | 01:26:40      |           1 |
|  2 | 11       | Sebastián-AR/192.168.1.74   | As          | 01:27:22      |           1 |
+----+----------+-----------------------------+-------------+---------------+-------------+
2 rows in set (0.00 sec)
```

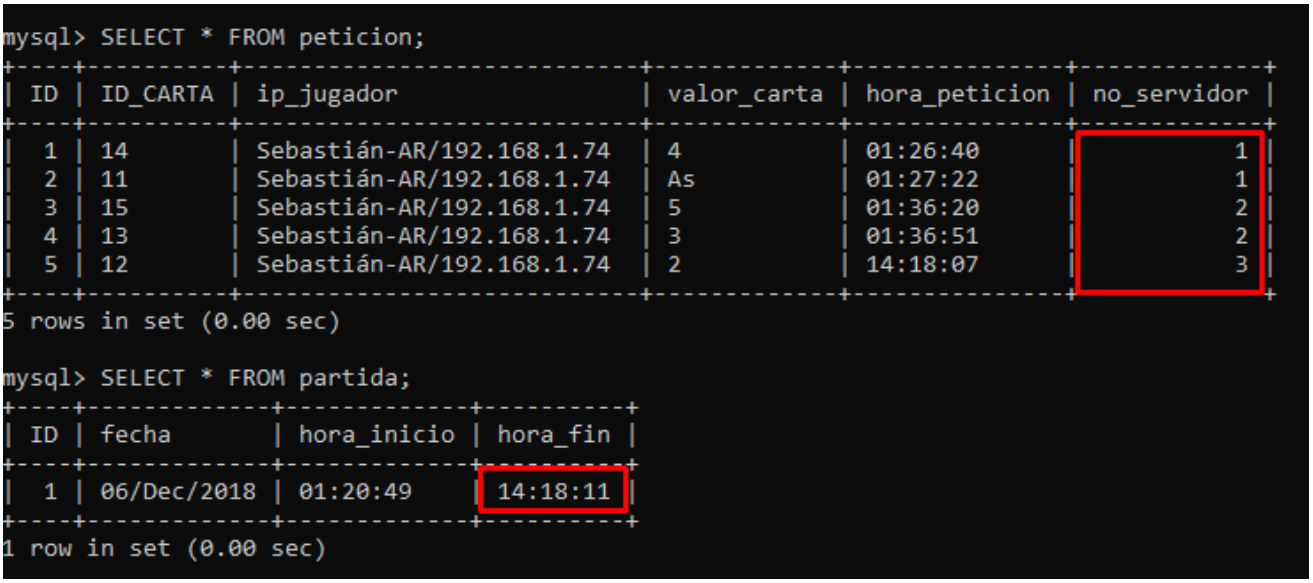
Solo tiene las peticiones que atendió

BD en server 2

```
mysql> SELECT * FROM peticion;
+----+----------+-----------------------------+-------------+---------------+-------------+
| ID | ID_CARTA | ip_jugador                  | valor_carta | hora_peticion | no_servidor |
+----+----------+-----------------------------+-------------+---------------+-------------+
|  1 | 14       | Sebastián-AR/192.168.1.74   | 4           | 01:26:40      |           1 |
|  2 | 11       | Sebastián-AR/192.168.1.74   | As          | 01:27:22      |           1 |
|  3 | 15       | Sebastián-AR/192.168.1.74   | 5           | 01:36:20      |           2 |
|  4 | 13       | Sebastián-AR/192.168.1.74   | 3           | 01:36:51      |           2 |
+----+----------+-----------------------------+-------------+---------------+-------------+
4 rows in set (0.00 sec)
```

Tiene las que atendió y las del server 1 antes de caer

BD en server 3



```
mysql> SELECT * FROM peticion;
+----+----------+---------------------------+-------------+---------------+-------------+
| ID | ID_CARTA | ip_jugador                | valor_carta | hora_peticion | no_servidor |
+----+----------+---------------------------+-------------+---------------+-------------+
|  1 | 14       | Sebastián-AR/192.168.1.74 | 4           | 01:26:40      |           1 |
|  2 | 11       | Sebastián-AR/192.168.1.74 | As          | 01:27:22      |           1 |
|  3 | 15       | Sebastián-AR/192.168.1.74 | 5           | 01:36:20      |           2 |
|  4 | 13       | Sebastián-AR/192.168.1.74 | 3           | 01:36:51      |           2 |
|  5 | 12       | Sebastián-AR/192.168.1.74 | 2           | 14:18:07      |           3 |
+----+----------+---------------------------+-------------+---------------+-------------+
5 rows in set (0.00 sec)

mysql> SELECT * FROM partida;
+----+-------------+-------------+----------+
| ID | fecha       | hora_inicio | hora_fin |
+----+-------------+-------------+----------+
|  1 | 06/Dec/2018 | 01:20:49    | 14:18:11 |
+----+-------------+-------------+----------+
1 row in set (0.00 sec)
```

Tiene todos los registros de las cartas entregadas de la partida y tiene la hora en que acabo.

Conclusiones

Esta práctica fue sumamente interesante e importante para comprender mejor como es la comunicación entre servidores y los front end del lado del cliente, pues juegan un papel sumamente importante a la hora de tener que hacer los cambios para saber a qué servidor se hace la petición.

Bibliografía

Comunicación entre un servidor y múltiples clientes. (s.f.). Webtutoriales.com. Recuperado de: http://www.webtutoriales.com/articulos/comunicacion-entre-un-servidor-y-multiples-clientes

González-J., Agustín. (2009). *Remote Method Invocation (Invocación Remota de Métodos)*. Recuperado de: http://profesores.elo.utfsm.cl/~agv/elo330/2s09/lectures/RMI/RMI.html

Alejandro Calderón Mateos, Javier García Blas, David Expósito Singh, Laura Prada Camacho,(2012). JAVA RMI: CALLBACK DE CLIENTE
Recuperado de: http://ocw.uc3m.es/ingenieria-informatica/desarrollo-de-aplicaciones-distribuidas/materiales-de-clase/rmi-callback