|  |  |  |  |
| --- | --- | --- | --- |
| logo_udlap    **EDEI** | **Reporte Final Sistemas Distribuidos**  **Practicas - Laboratorio** | | |
| **DEPARTAMENTO** | Computación, Electrónica y Mecatrónica | **MATERIA** | LIS3061 |
| **PROFESOR** | Dra. Ofelia Cervantes | **PERIODO** | Primavera 2015 |
| **Alumno** | Diego Rodriguez | **Id** | 146247 |

**Introducción**

En este documento se realizara una recopilación de todos los reportes correspondientes a cada una de las prácticas desarrolladas durante el semestre, este documento contendrá todas las tareas completadas en cada práctica, los objetivos generales, conclusiones y preguntas que fueron desarrolladas, al mismo tiempo se incluirá los resultados de cada una de las implementaciones en capturas de pantalla que mostraran y comprobara que la práctica fue realizada.

El código y archivos que se generaron como resultado de cada una de las practicas se encontrara disponible en la siguiente liga.

**Objetivo**

Conocer las principales tecnologías para desarrollar aplicaciones en sistemas distribuidos.

Feminizarse con los términos y conceptos relacionados con el ambiente distribuido.

Comprender la diferencia entre las principales herramientas de sistemas distribuidos.

Poner en práctica los conceptos aprendidos en clase y entender su importancia durante la práctica correspondiente.

**Índice de prácticas.**

RPCGEN.

RMI.

DINING PHILOSOPHERS.

PLAY WEB SERVICE.

WEBSOCKET PLAY.

WEBSOCKET PLAY ACTORS.

METEOR TUTORIAL.

Conclusiones

Durante el desarrollo del semestre fueron completadas cada una de los objetivos que se planteaban en las prácticas, se dieron respuesta a cada una de las preguntas que se incluían en cada tema.

En el semestre durante las sesiones de teoría se fueron conociendo conceptos con el ambiente distribuido, que fueron definidos y detallados durante estas sesiones, en el laboratorio estos conceptos fueron puestos en práctica has familiarizarse con cada uno de ellos, cumpliendo con el objetivo del curso.

En el laboratorio se atendió y completo exitosamente cada una de las prácticas planteadas en el sitios del curso, se resolvieron todos los inconvenientes que se presentaron con respecto al sistema operativo o versiones de ciertas aplicación que fueron instaladas con el fin de completar cada una de las practicas, a su vez las practicas fueron completadas al 100%, se atendieron todas implementaciones como lo podemos ver en cada uno de los reportes anexos al documento.

Código

El código que fue generado como resultado de las prácticas se encuentra disponible en la siguiente dirección:

<https://github.com/diego2102/SistemasDistribuidosUdlap>

|  |  |  |  |
| --- | --- | --- | --- |
| logo_udlap    **EDEI** | **RPCGEN** | | |
| **DEPARTAMENTO** | Computación, Electrónica y Mecatrónica | **MATERIA** | LIS- 3061 |
| **PROFESOR** | Dra. Ofelia Cervantes | **PERIODO** | Primavera 2015 |
| **ID** | 146247 | **Nombre** | Diego Rodriguez |

INTRODUCCION.

Es este documento se muestra los resultados obtenidos en la primer practica de laboratorio, la practica fue dividida en dos partes, en el documento se muestran todas las dificultades que se presentaron para completar la actividad.

OBJETIVO.

El objetivo principal de esta práctica es poner en funcionamiento dos servidores y conectarse a ellos para utilizar sus procesos.

Uno de los objetivos intermedios es completar la parte 1 de la práctica, que comprende utilizar los comandos de Rpcgen para generar los archivos del servidor y del cliente, mediante ellos hacer funcionar el servidor, toda la practica

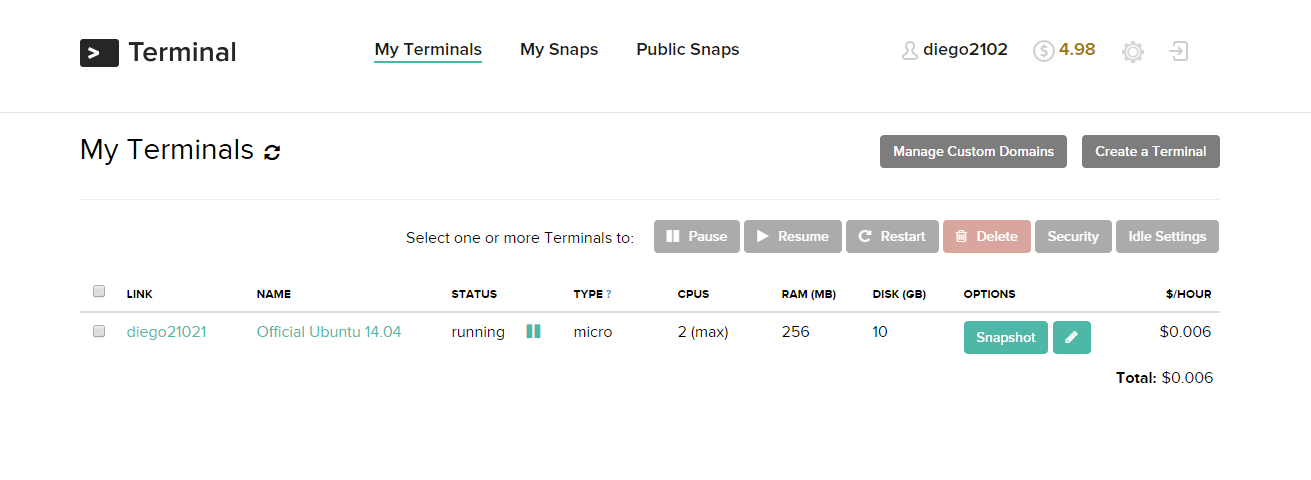
La segunda parte de la práctica es poner a prueba los conocimientos obtenidos en la parte uno, la aplicación debe de recibir un nombre, calcular la fecha y escribirla en un archivo de texto, otra función debe de recibir un nombre, buscar en el archivo de texto y mostrar la fecha.

El último objetivo es entender e incrementar el conocimiento del lenguaje C.

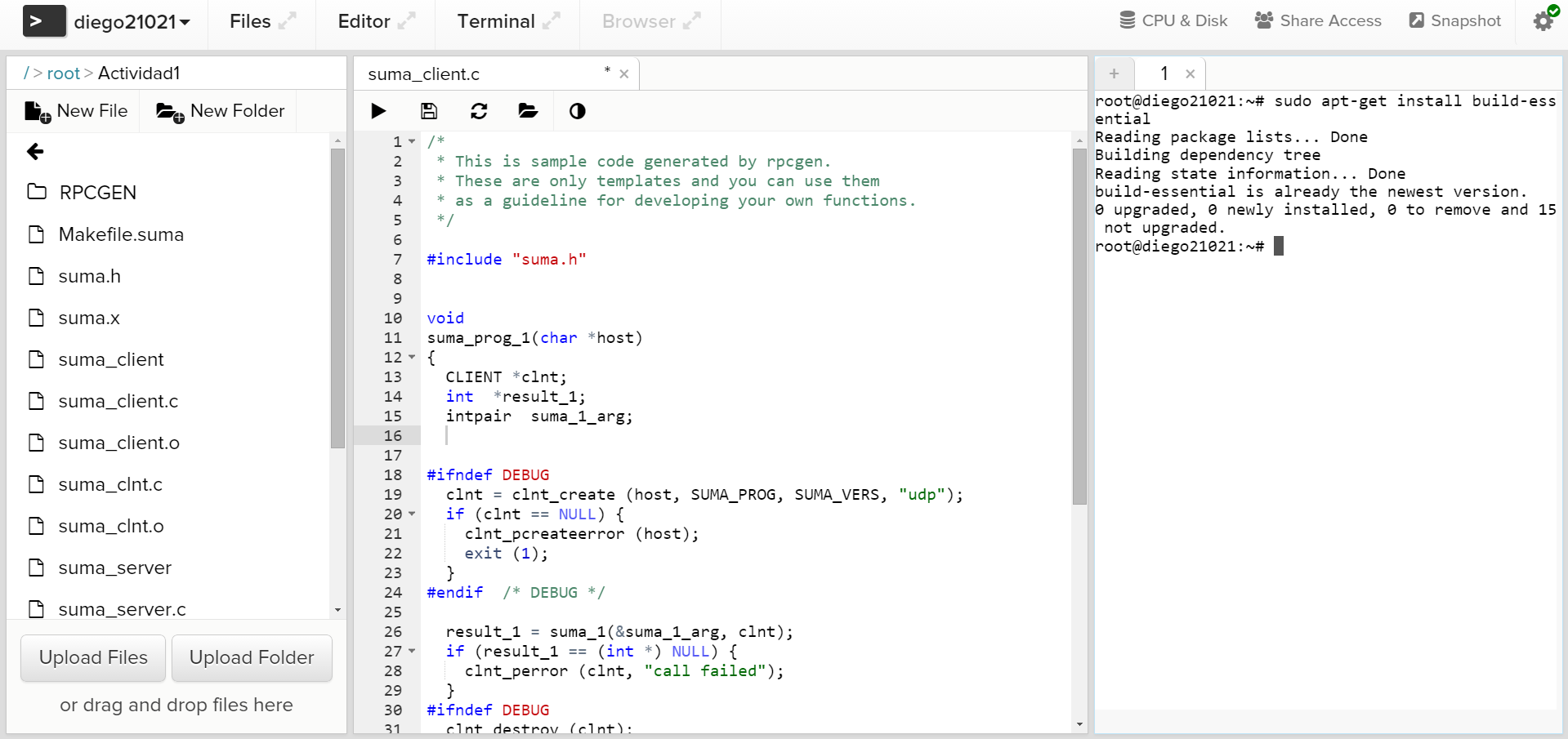
CONTENIDO.

Toda la práctica se debe de realizar en un sistema Linux, por lo que fue necesario virtualizar uno en nuestras maquinas, en mi caso particular no fue posible utilizar el sistema, por lo que fue necesario otra alternativa.

La página de internet terminal.com, ofrece un servicio de servidores en diferentes sistemas, por lo que me permitió realizar la práctica en un servidor de tipo Linux, aunque es un servicio de paga la página ofrece 5 dólares de regalo para probar la aplicación. A continuación se muestra la página donde se da inicio a los servidores ligados a una cuenta personal relacionada con google+.



Una vez iniciada la terminal de Linux, la página ofrece una interfaz donde se puede aperturar cuantas terminales se requiera, un editor de archivos y un navegador, en este navegador se puede organizar los archivos, como lo veremos en la próxima imagen, del lado derecho se muestran los archivos que se generaron durante la practica 1 parte 1, en el centro se muestra uno de los archivos que se editaron, y final mente del lado izquierdo aparece las terminales que se encuentran funcionando.



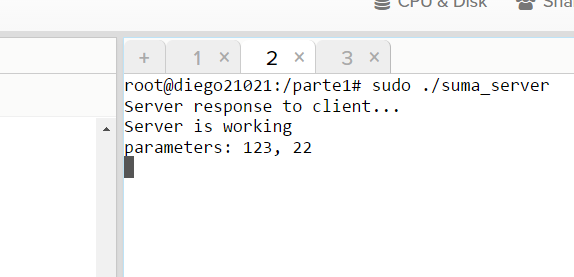
La primer parte la practica fue muy fácil de completar, solo consistió en copiar los comandos que se encontraban en la dirección de gitHub, <https://github.com/Innova4DLab/RPCGen>.

Mediante el archivo suma que contenía el siguiente código.

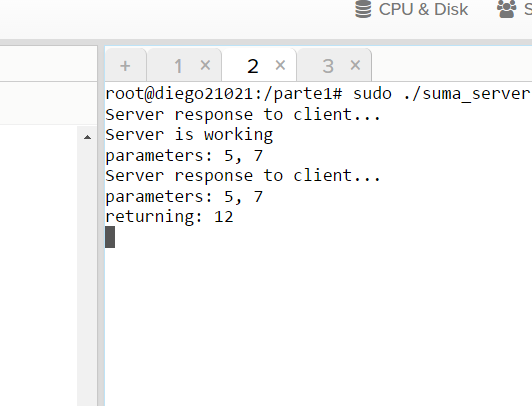
|  |
| --- |
| struct intpair { |
|  | int a; |
|  | int b; |
|  | }; |
|  |  |
|  | program SUMA\_PROG { |
|  | version SUMA\_VERS { |
|  | int SUMA(intpair) = 1; |
|  | } = 1; |
|  | } = 0x23451111; |

Este archivo genera un conjunto de archivos en leguaje C, que contiene las partes principales de la aplicación, el programador no tiene que hacer construir este código que permite la conexión entre cliente y el servidor, y única mente se debe concentrar en la construcción de funciones para la aplicación.

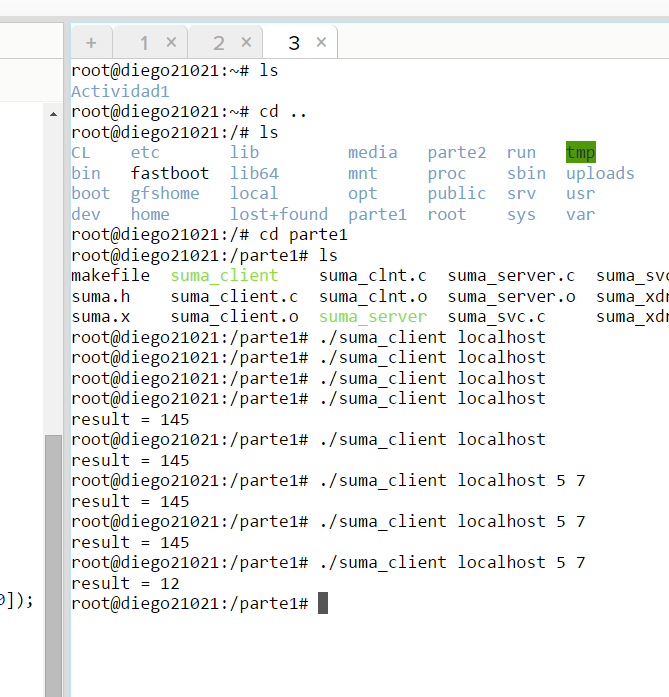
En la suigente imagen podemos ver los resultdos obtenidos después de enviar los parámetros al servidor, la imagen que se muestra abajo corresponde a la información que arroja el servidor



La siguiente imagen muestra el primer procesamiento que se desarrolla desde el servidor, este proceso no regresa nada al cliente, pero ya realiza una operación con datos que fueron enviados desde el cliente



Final mente en la siguiete imagen podemos ver como el cliente puede ver el resultado obtenido y que fue enviado desde el servidor, los datos que el cliente envía son introducido desde la terminal, a diferencia de las imágenes anteriores donde los datos están almacenadas en el código del programa. En esta imagen podemos ver las ejecuciones por parte del cliente a las que el servidor atendió, los resultados fueron mostrados en las imágenes anteriores.



Una vez cubierto estos puntos, la parte 1 de la práctica ha sido completada satisfactoriamente.

Parte dos de la práctica.

La parte dos de la practica consistió en hacer uso de los comandos y habilidades obtenidas en la práctica uno y aplicarlas a un nuevo problema, a continuación se muestran los objetivos de la práctica, que nuevamente se encontraron en gitHub.

* Generar un nueva especificación RPCGen (Como add.x) en donde se definan dos funciones:
  + La función "agregar" deberá:
    - Recibir un nombre y obtener la fecha con la librería [Date](http://goo.gl/tSrZ46) de C.
    - [Guardar](http://goo.gl/HQeG4H) en un archivo .txt la fecha y el nombre.
    - Retornar un mensaje si los datos se han escrito satisfactoriamente.
  + La función "buscar" deberá:
    - Recibir un nombre, mediante una [búsqueda](http://stackoverflow.com/a/13450846/1206078) obtener la fecha(s) relacionada(s) al nombre.
    - Retornar el la fecha relacionada para el nombre indicado y/o el número de registros con el mismo nombre.
    - Realizar el [paso 6 de la actividad 1](http://goo.gl/Qm2m59), en cuanto a limpieza del código y comentarios.

El archivo add.x debería contener el siguiente código.

program ADD\_PROG {

version ADD\_VERS {

string ADD(string) = 1;

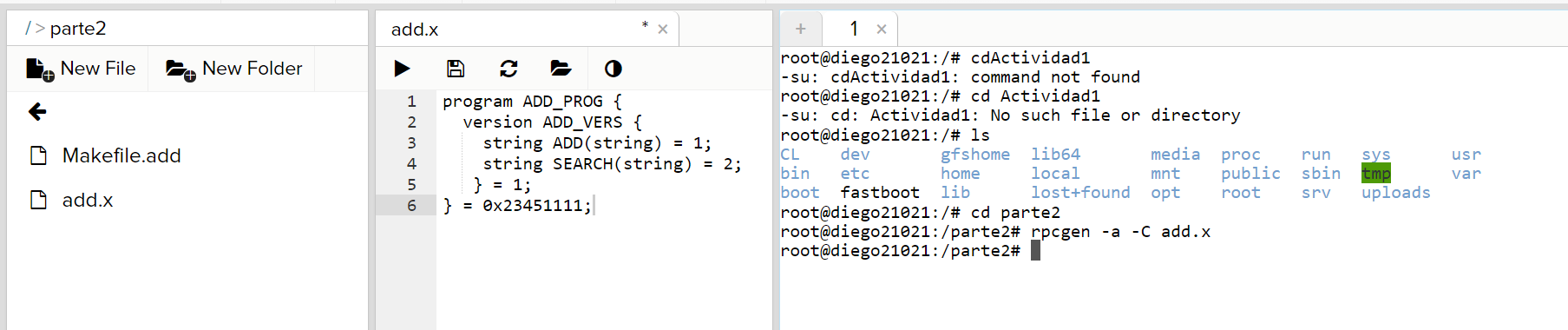
string SEARCH(string) = 2;

} = 1;

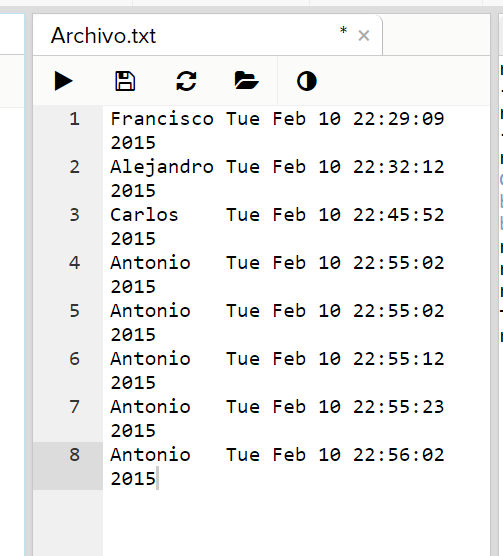
} = 0x23451111;

Con el commando rpcgen -a -C add.x

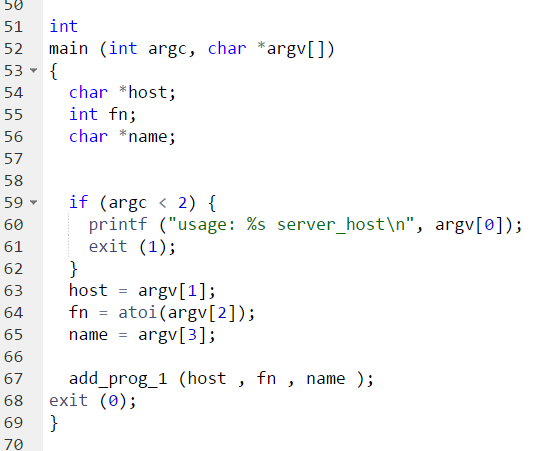
Con este comando se generaron todos los archivos necesarios para realizar la práctica.



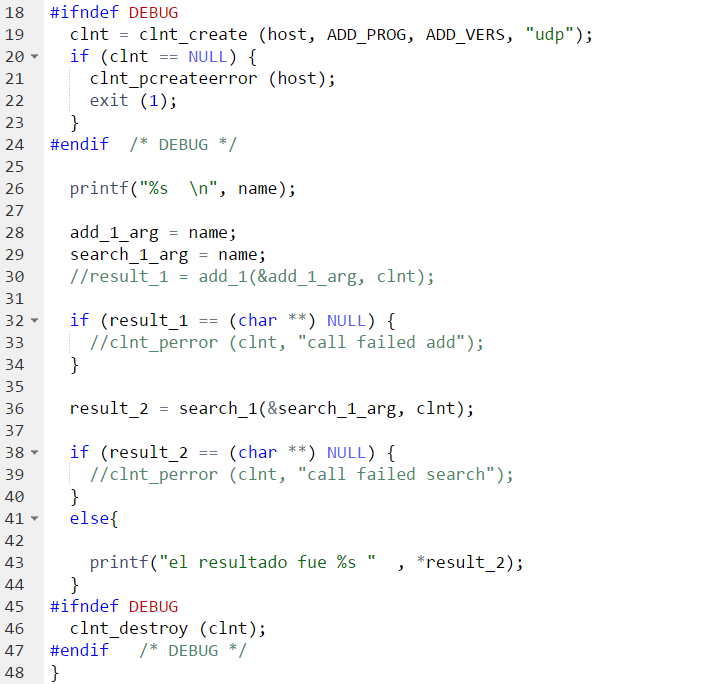
Archivo que genera la aplicación fue el siguiente.



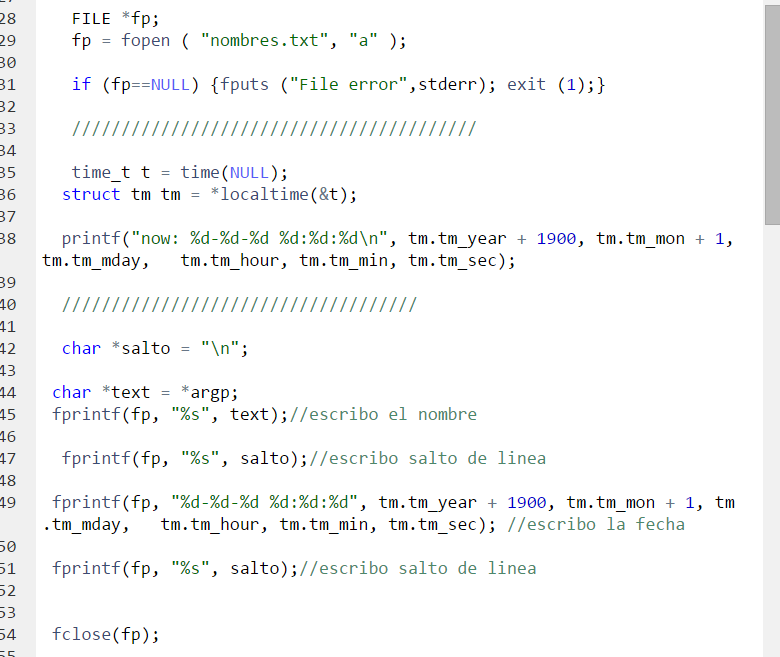
La primer parte a modificar del programa fue el main principal, localizado en el cliente, es el método que se ejecuta al momento de correr el programa, podemos observar que los datos obtenidos desde la terminal son enviados al meto del programa para que inicie el proceso.



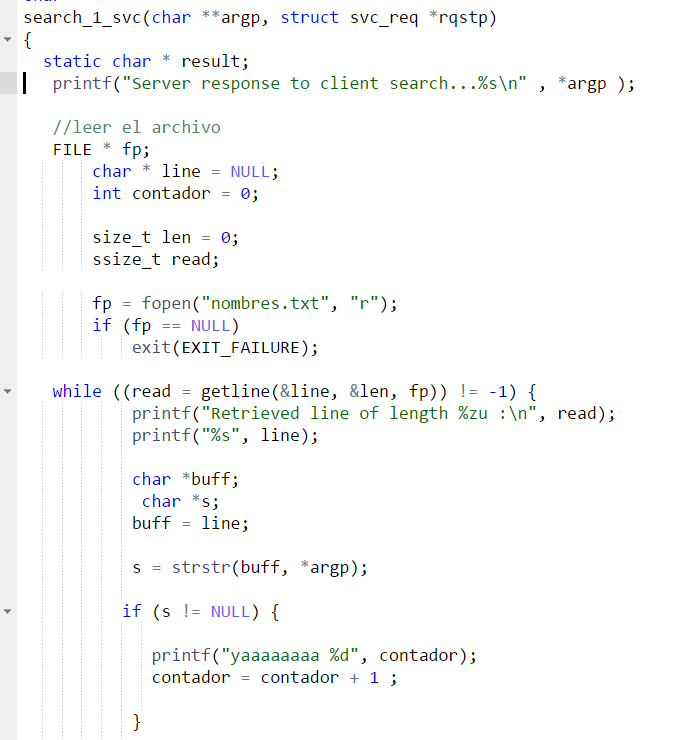
Una vez que los datos llegan al método, automáticamente se genera el cliente y envía los datos que utilizara el servidor además de los datos del cliente para realizar la conexión, en nuestro caso será el local host.



El servidor de la aplicación realiza varias funciones, escribe datos y la fecha en un archivo de texto.

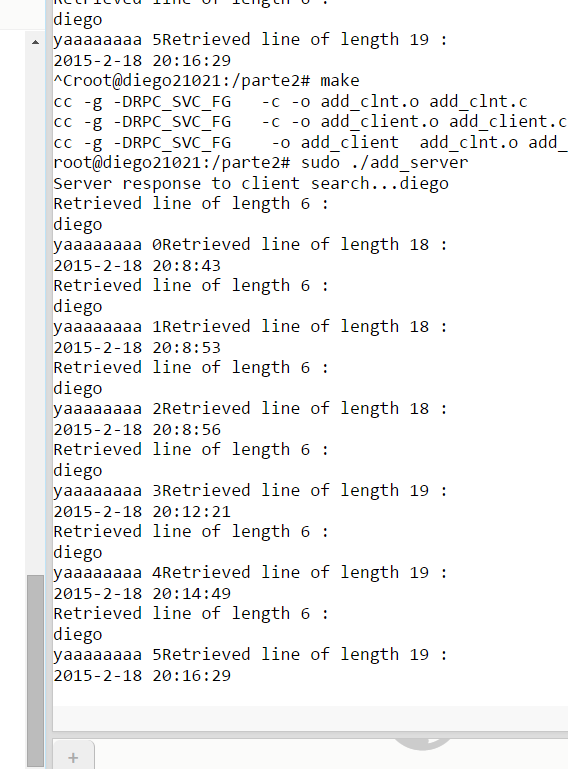


En su defecto busca un nombre en el archivo de texto.

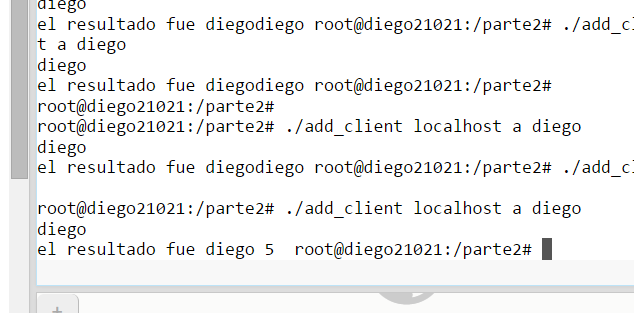


El cliente selecciona que función debe ejecutar el cliente.

Finalmente los resultados son los siguientes, el servidor ejecuta la búsqueda dentro del archivo, y muestra la fecha cuando se realizó la inserción, he imprime los valores.



Y el cliente recibe el número total de veces que se encontró el nombre que envio al iniciar el programa, en este caso la respuesta es de 5.



El archivo que se genero fue el siguiente



CLONCLUCIONES.

Como conclusión podemos decir que se completaron al 100% por ciento los objetivos, se completó en sus totalidad las dos partes de la practica 1, y se obtuvieron los resultados a adecuados, pero una falta mayor habilidad y trabajo en el lenguaje C, lo que hubiera permitido realizar la practica con mayor velocidad, no obstante con esta práctica se logró comprender el funcionamiento de RPCGEN, y aprender nuevas instrucciones en el lenguaje c.

Es muy importante mencionar que cuando se trabaja con un lenguaje que no está dominado, se debe ser muy cuidadoso con el código que se está desarrollando, debido a que se puede introducir errores de sintaxis o de omisión que puede retrasar el desarrollo.

Final mente la práctica fue completada con éxito, y los objetivos establecidos al inicio del documento fueron completados.

|  |  |  |  |
| --- | --- | --- | --- |
| logo_udlap    **EDEI** | **RMI** | | |
| **DEPARTAMENTO** | Computación, Electrónica y Mecatrónica | **MATERIA** | LACLVE DE LA MATERIA |
| **PROFESOR** | Dra. Ofelia Cervantes | **PERIODO** | Primavera 2015 |

INTRODUCCION

El documento es el resultado de la práctica correspondiente al tema RMI, RMI ***Remote Method Invocation,*** invocación de objetos remotos es el paradigma utilizado para desarrollar aplicaciones orientadas a objetos en el ambiente distribuido.

Durante el trascurso de este periodo se abordó en clase todos los aspectos técnicos que engloban este paradigma, y que serán puestos en práctica en el desarrollo de esta práctica

OBJETIVO

* Completar la primera parte de la practica
  + importar el proyecto
  + crear un método que multiplique dos números enteros
  + crear un método para analizar si una palabra es palíndromo o no
* completar la segunda parte de la practica
  + importar el proyecto
  + analizar el funcionamiento de la aplicación
  + agregar un elemento a la aplicación (moto, avión, etc.).
* Dar respuestas a las preguntas planteadas en la practica.

DESARROLLO

* PARTE 1

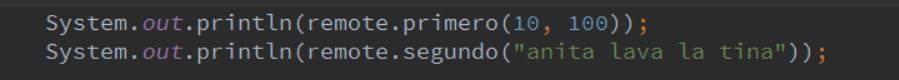
La práctica comenzó descargando el proyecto desde el sitio: <https://github.com/Innova4DLab/RMI>. Las instrucciones de toda la práctica también están descritas en la misma liga.

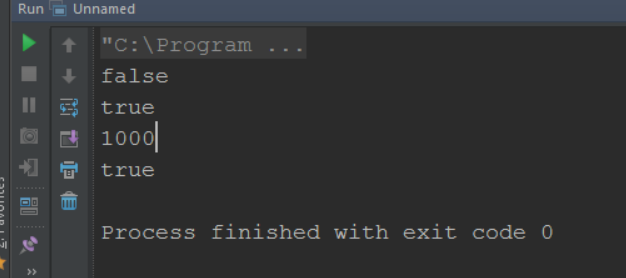
Para la realización de la practica el proyecto fue importado a InteliJ, para su ejecución, en este ide fue desarrollada toda la práctica, se configuro la localización de los paquetes y fue ejecutado para asegurar su óptimo funcionamiento.

Una vez terminando la ejecución del programa, se prosiguió a modificar el servidor, en esta ocasión debería ejecutar una multiplicación y averiguar si una palabra es un palíndromo o no.

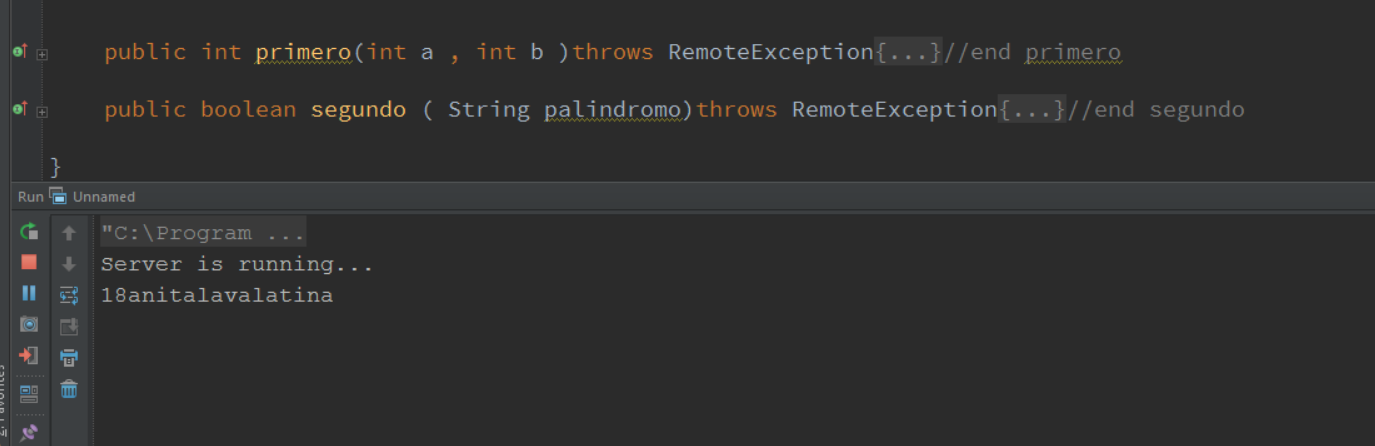
Desde el cliente se ejecutaban los métodos en el servidor, se le envían dos números enteros y se le pide que imprima el resultado, el segundo método envía una cadena de texto, que el servidor procesa para analizar si es un palíndromo, regresa true o false dependiendo si es o no es palíndromo.

A continuación se muestra la invocación del metod.



El resultado de esta invocación fue el siguiente, las dos últimas líneas muestran lo obtenido después de la ejecución.

En el servidor podemos observar como recibió los numero y una cadena de texto y la proceso para regresar la respuesta.



Finalmente podemos decir que la primera parte de la practica correspondiente a RMI fue completada con éxito, los problemas a los que se les dio solución fueron principalmente en el procesamiento de la cadena de texto, a la cual se le quitaron los espacios y se invirtió para averiguar si es un palíndromo o no.

* PARTE 2

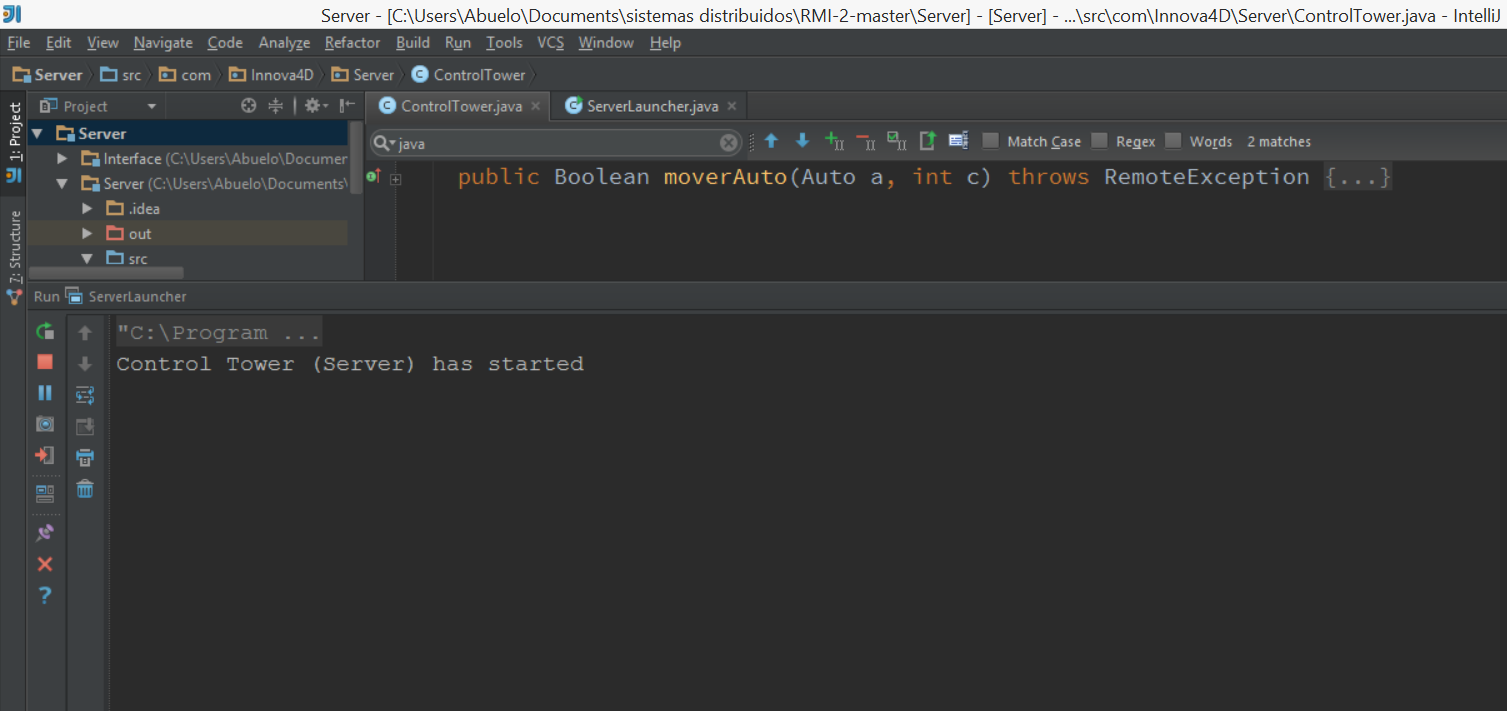
La segunda parte consistido en elaborar una torre de control que permitiera la administración de una pista de aterrizaje, la torre de control administra la velocidad y el movimiento de los carro o aviones que desean utilizar o aterrizar sobre la pista.

La primera parte de la práctica fue importar el proyecto que se descargó de la misma dirección de internet que la parte 1, el proyecto fue modificado debido a que la versión de java en la que fue desarrollado sufrió modificación en la forma de recibir parámetros particularmente en el switch case que contenía la aplicación, por lo que fue sustituida por dos instrucciones de tipo IF.

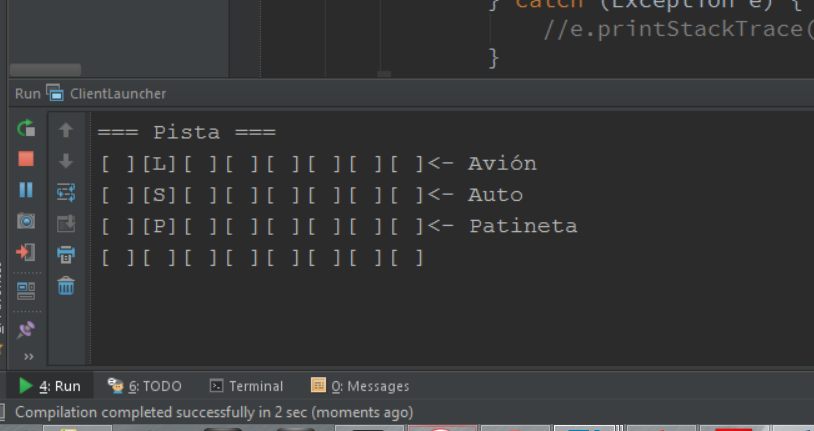
El proyecto importado funciono a la perfección, la torre de control asignaba el tiempo a cada objeto que quería hacer uso de la pista, un avión y un auto se movían a diferentes velocidades dentro de la aplicación.

En concreto el objetivo de la parte dos fue agregar nuevos tipos de objetos que hicieran uso de la pista que es administrada por la torre, para esto fue necesario modificar la inteface, donde se declararon tres nuevos métodos idénticos a los utilizados por el avión o por el auto, estos métodos fueron programados en el servidor, la clase que permitía generar nuevas instancias de un obejto nuevo fue declarada en la interface.

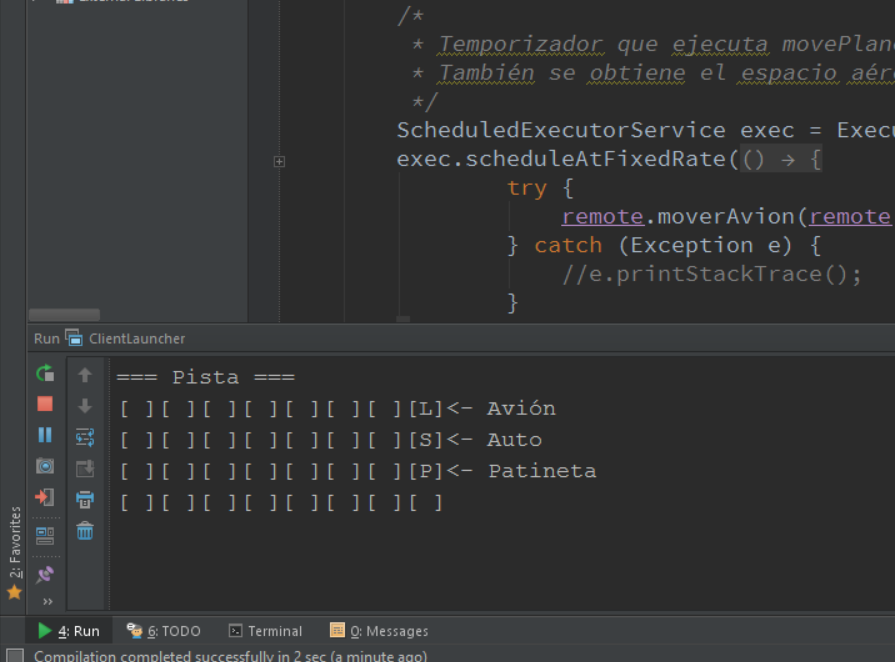
A continuación se muestra una captura de pantalla del servidor corriendo y esperando a un cliente que se conecte a él.



La segunda imagen permite observar el estado intermedio de los objetos, durante la ejecución, las letras dentro de los corchetes indican el espacio que ocupa el objeto dentro de la pista.



Por ultimo en la siguiente imagen se muestra el resultado final de la ejecución del programa, la torre principal detecta que los objetos llegaron a su punto final, pero la ejecución sigue aún que los objetos ya no se mueven.



Para terminar podemos decir que la segunda parte también se completó con éxito, los tres objetos pueden utilizar la pista y son controlados por la torre de control hasta llegar a la última parte de ella.

PREGUNTAS

* PARTE 1
  + ¿Cómo se definen nuevos métodos en **Server**?
    - Cada método nuevo deberá ser declarado en el server y en intefece, este método va a ser llamado desde cualquier cliente.
  + ¿Cuáles son los pasos necesarios para registrar un nuevo método?
    - Ir a la interface, escribir el nombre del método incluyendo los parámetros de salida y de entrada, en la clase servidor se desarrollara el cuerpo del método.
  + ¿Cuál es la función de **Registry**?
    - Su función es almacenar la localización del servidor, en esta ocasión será la dirección local, pero se puede almacenar cualquier IP de donde se localice el servidor.
  + ¿Qué se hace en **Interface**?
    - La interface asegura que se programen todos los métodos necesarios para la ejecución del programa, evitando posibles errores al memento de la ejecución.
  + ¿Cómo es la comunicación entre **Client** y **Server**?
    - El cliente es un programa que está corriendo esperando a que un cliente necesite recursos y otorgarle respuesta, el servidor tiene asignada una dirección y el cliente conoce esta dirección a la cual se conecta para poder acceder a los recursos del servidor.
  + ¿Cómo podrías ejecutar más **Clientes**?
    - Se pueden ejecutar cuantos clientes sean necesarios desde una misma computadora, pero ciertamente muchas computadoras se pueden conectar al mismo servidor al mismo tiempo.
* PARTE 2
  + ¿Cómo se definen nuevas clases para usarlas con **Server**?
    - Toda nueva clase debe ser definida junto con la interface, para poder utilizar un objeto de esa clase debe ser declarada desde la interface, al igual que los métodos que necesiten atributos u objetos de esa clase.
  + ¿Cuáles son los pasos necesarios para crear una nueva clase y utilizar el objeto en el Servidor?
    - Primero se debe definir la clase junto a la interface, luego cualquier objeto que necesite utilizar el servidor debe ser declarado en la interface, posteriormente deberá ser utilizado desde el servidor.
  + ¿Cuál es la utilidad de **Constant.java**?
    - Es una clase que se puede implementar junto con un buen diseño, permite mantener los valores constantes.
  + ¿Cómo se manejan los errores en el Servidor?
    - Todo error provocado por el servidor provoca una excepción y el servidor deja de funcionar, y el cliente termina la ejecución del programa.
  + ¿Cuál es la utilidad de la clase **RemoteException**?
    - Este tipo de excepción que provoca el programa permite que la aplicación no termine su ejecución, y solo el componente que no puedo completar su ejecución sea el que se detenga y la aplicación continúe funcionando.
  + ¿Por qué es necesario utilizar **Serializable** en las nuevas clases?
    - La implementación de esta clase estamos permitiendo convertir cualquier instancia de la clase que se implemente en bites, para poder ser enviados atreves de la red.
  + Dada la comunicación entre **Client** y **Server**
    - ¿Por qué es necesario que el método avionClient() y los objetos **RemoteInterface** y **Avión** sean de tipo **final**?
      * Para que no puedan ser modificados durante la ejecución de la aplicación.
  + ¿Por qué es necesario un método **getAvion(String id, int c)**?
    - Este método permite crear el objeto avión después de ser enviado atreves de la red.
  + ¿Cómo se manejan los errores en el Cliente?
    - Se provoca una excepción pero el programa sigue funcionando.
  + ¿Qué significa y cuál es la utilidad de **NotBoundException**?
    - Permite que no se sobre pase la dimensión del arreglo.

CONCLUSIONES

Finalmente podemos decir que las dos partes iniciales de la práctica fueron desarrolladas exitosamente, también podemos asegurar que los conceptos teóricos de esta práctica fueron asimilados y usados durante esta práctica.

Adicionalmente podemos concluir que en esta práctica se repasaron y aprendieron aspectos que tiene que ver con la parte técnica y de implementación en lenguaje java, como lo fue la función toString en la primera parte de la práctica, o el funcionamiento de los threads en la parte dos.

En cuanto a los aspectos a mejorar, tenemos que la aplicación aún es muy sencilla y debemos de ir subiendo la complejidad de los proyectos para poder adentrarse más en la materia así como comprender y aprende más al respecto.

|  |  |  |  |
| --- | --- | --- | --- |
| logo_udlap    **EDEI** | **Practica: Problemas de los Filósofos** | | |
| **DEPARTAMENTO** | Computación, Electrónica y Mecatrónica | **MATERIA** | Sistemas Distribuidos |
| **PROFESOR** | Dra Ofelia Cervantes | **PERIODO** | Primavera 2015 |

**Introducción.**

El problema de los filósofos es uno de los problemas mas conocidos, consiste en grupo de dos o mas filósofos, para poner el ejemplo tomaremos en cuenta un grupo de 5 filósofos, que están sentados a la mesa, mientras algunos de ellos comen los otros filósofos piensa, cada filosofo necesita forzosamente dos cubiertos para poder comer, el problema radica que en la mesa solo existen 5 cubiertos uno por cada filosofo sentado, si todos los filósofos deciden comer al mismo tiempo y todos toman un cubierto ninguno de los filósofos puede comer, a este evento se le conoce como un deadlock.

Un deadlock es cuando todos los recursos se encuentras acaparados por un elemento, y este elemento no puede continuar por que otro elemento tiene y no libera los recursos que necesita para completar sus actividades, por lo que todos los elementos no se encuentran en un mismo estado y no pueden avanzar.

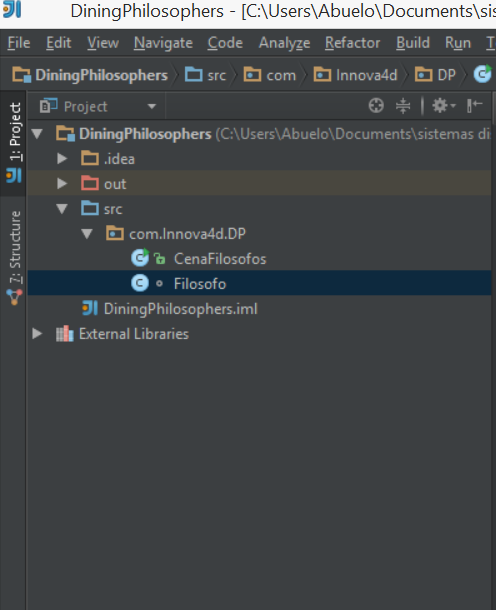
El problema se puede resolver asignando tiempo para comer con los dos cubiertos, mientras los filósofos que no pueden comer están pensando, tiempo después vuelve a verificar que pueda tomar los dos cubiertos y comer, aun así existe la posibilidad que uno o más filósofos nunca pueda tener los dos cubiertos, por consiguiente no pueda comer y muera de hambre.

**Desarrollo.**

La actividad consiste en completar un programa que permita resolver este problema, organizando a los filósofos para que alternen entre pensar y comer sin provocar un deadlock.

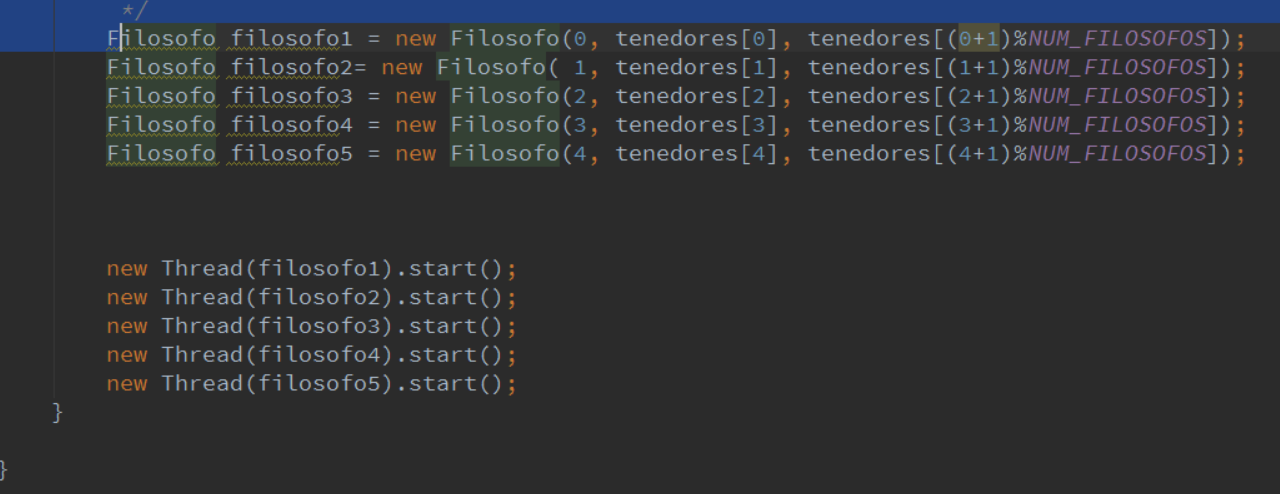
Primera Parte 1

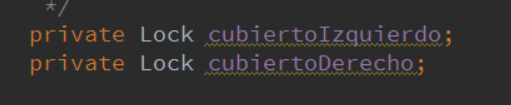
La primera parte consiste en a completar el código faltante el programa, por lo que primero debemos de importar el proyecto en java a nuestro ide que ocuparemos para el desarrollo de la actividad.

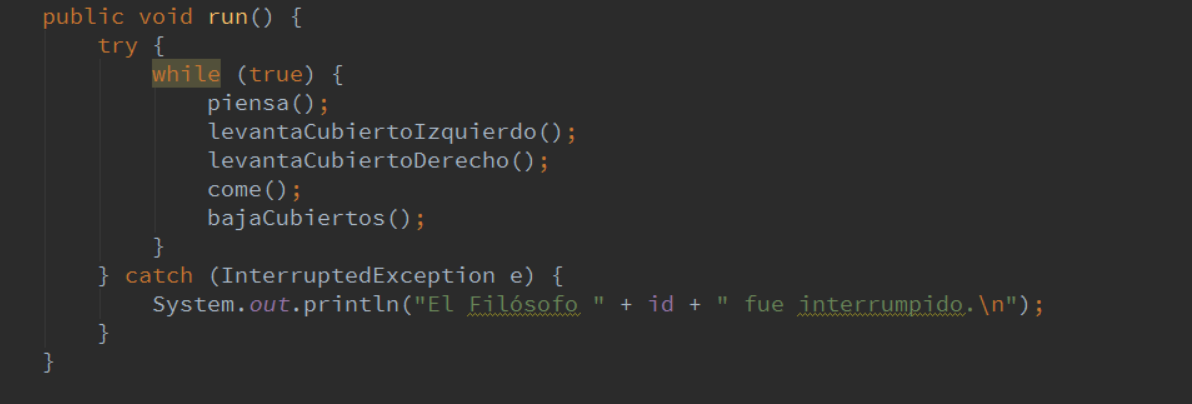


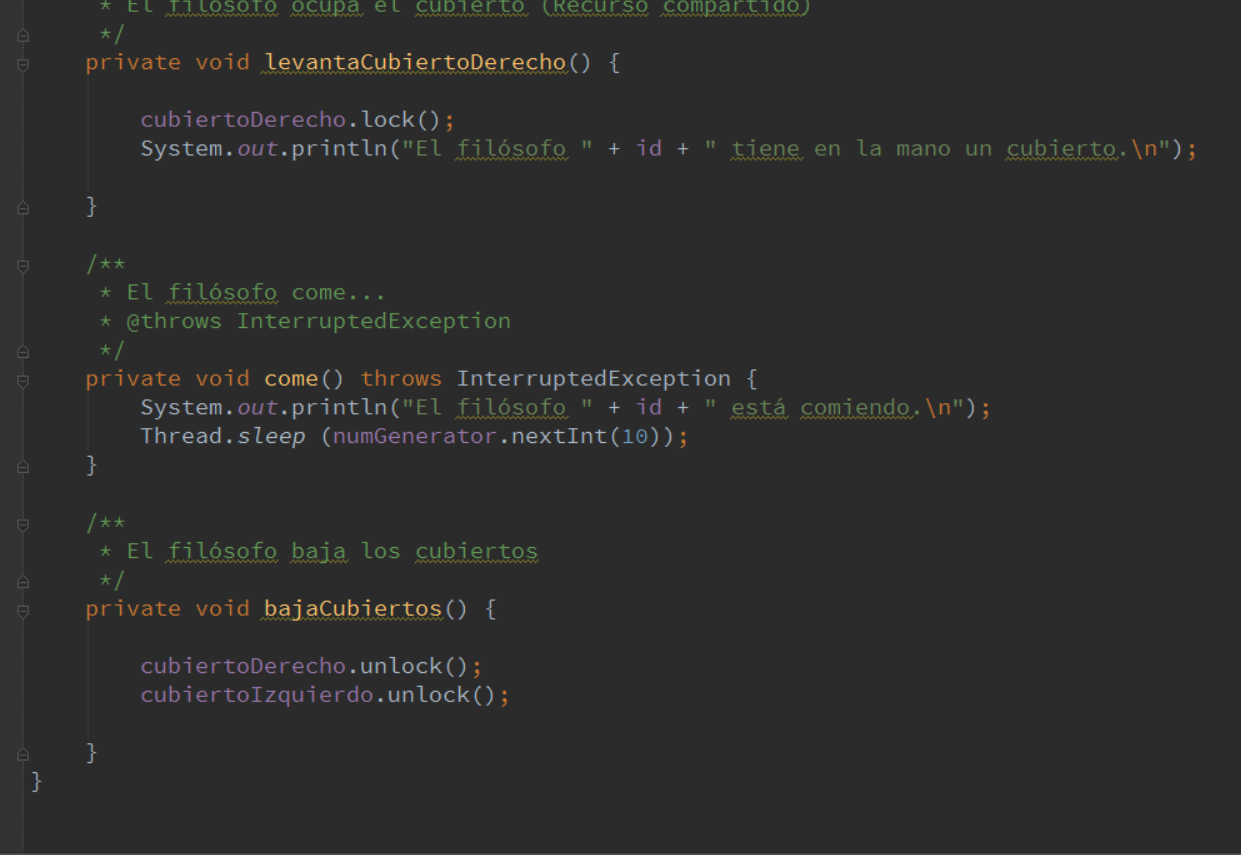
Una vez importado el proyecto, nos percatemos de que el código de dicho proyecto se encuentra incompleto, la tarea es completar el código y observa que hace el programa cuando por fin funcione completamente, además debemos de incrementar el número de filósofos a 5 y observar que ocurre.

Para aumentar el número de filósofos debemos de crear 5 objetos instancia de la clase filósofos, para que cada uno de los filósofos existe un hilo de proceso asignado a ellos, es decir vamos a crear y asignar un thread para cada filósofo.



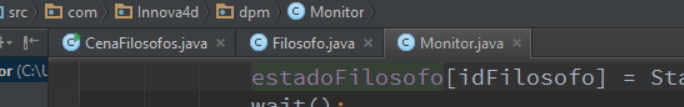




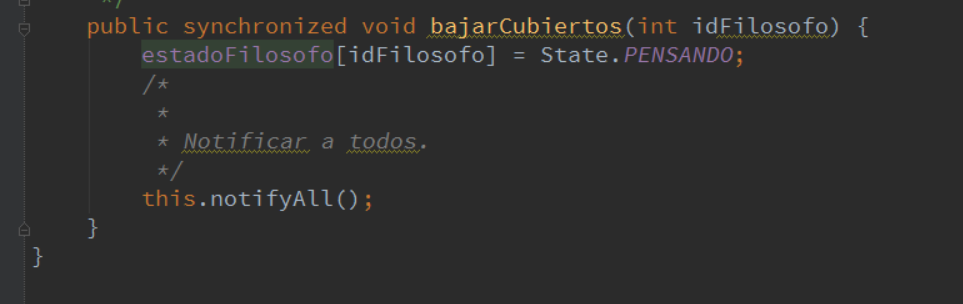


Parte 2

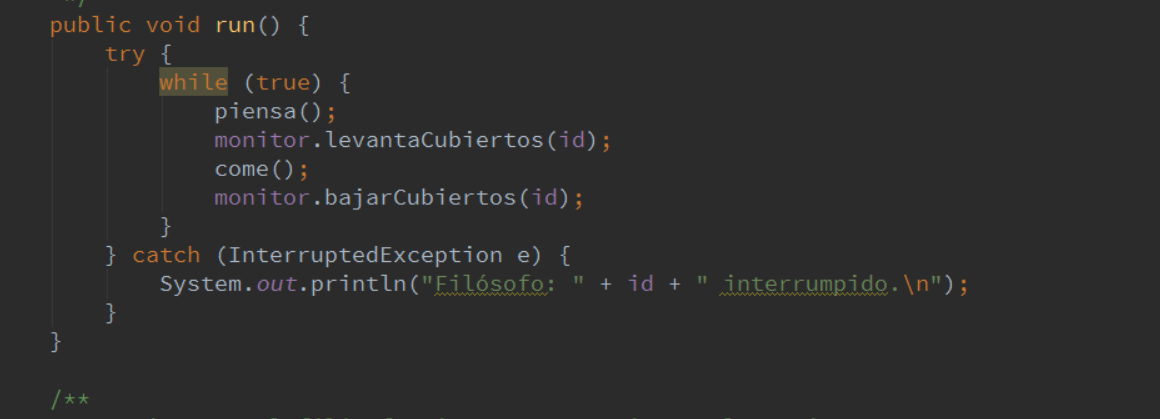
Para dar inicio debemos de importar el proyecto a nuestro IDE, podemos observar que el proyecto tiene una nueva clase llamada monitor, esta nueva clase nos permitirá administrar el uso de recurso evitando que se generen abrazos mortales durante la ejecución de nuestra aplicación.



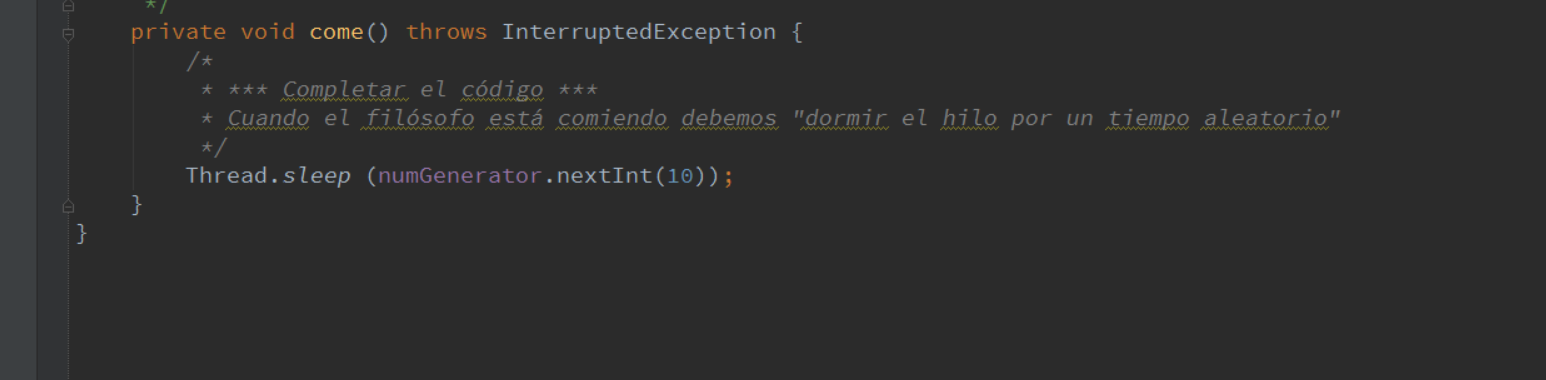
Para nuestra practica el objetivo es completar las partes de código faltantes en nuestro proyecto, un ejemplo se muestra a continuación, el meto bajarCubiertos() indica que el hilo dejara los cubiertos y cambiara su estado a pensar, el método notificara a todos los hilos que hubo recursos liberados.



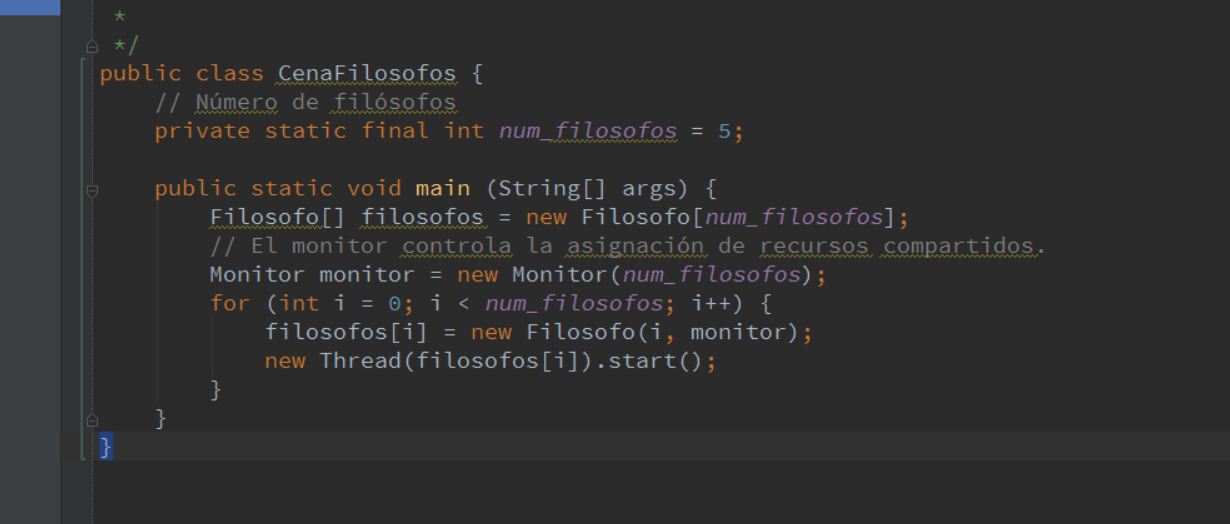
En la siguiente imagen podemos ver el código que contiene el método run(), el cual le pedirá al thread que valla al monitor y verifique que hay recursos liberados, levantara los cubiertos, comerá y dejara nuevamente los recursos disponibles.

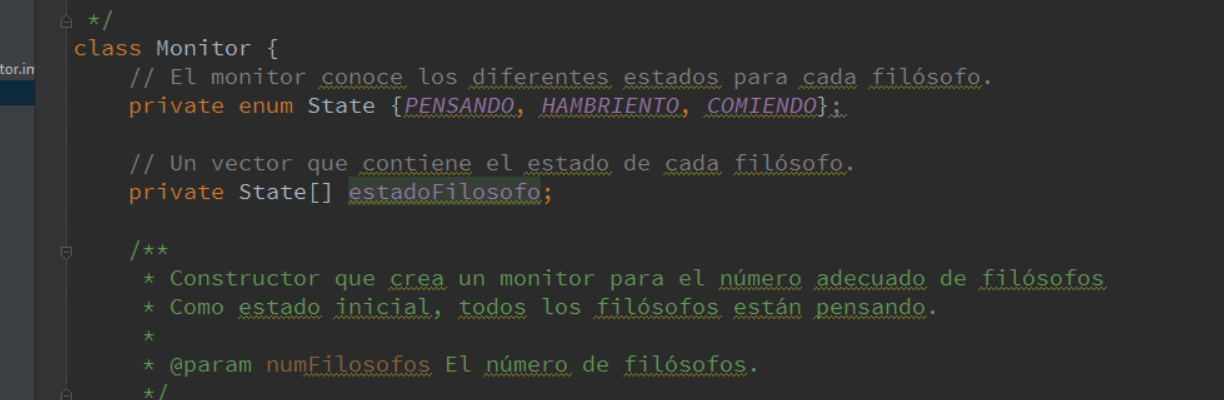


En la siguiente imagen podemos ver que el método come(), manda a dormir el thread durante 10 segundos, periodo en el cual el thread hace uso de los recursos y no los libera hasta que termina de comer.



En la clase main se encuentra todos los hilos de ejecución, instancias de thread, cada uno de estos thread fungirá como filósofo y tratara de llevar a cabo sus dos acciones principales, comer y pensar, alternando entre ellas dos, siempre y cuando tenga los recursos disponibles para hacerlo.



Finalmente tenemos el monitos, el monitor guarda los estados de cada uno de los filósofos, cada filosofo es representado como un thread, este monitor va a ser monitoreado por los hilos de ejecución para ver si puede acceder a los recursos que necesita cada subproceso.

**Preguntas.**

* ¿Cuál es la diferencia entre **Wait()** y **Sleep()**?
  + El método Wait() al ser invocado por un thread provoca que el proceso se mantenga en espera sin utilizar recursos, en cambio Sleep() se apodera de los recursos pero no hace ninguna función hasta que el tiempo termina y el thread continua con si ejecución para más tarde liberar los recursos.
* ¿Para qué se utiliza **Lock**?
  + Lock es una herramienta que se utiliza cuando hay recursos compartidos, permitiendo acceder a estos recursos cuando los lock se encuentran desbloqueados, de lo contrario evita que alguien haga uso de estos y provoque inconsistencias.
* ¿Cuál es la diferencia entre **Runnable()** y **Thread()**?
  + Runnable() es una interface que obliga a implementar el método run(), todo thread que se instancie ejecutara los metod indicados en el metod run(), en cambio para thread la clase que quiera implementar un hilo deberá extender de la clase thead, problema es que java no permite la múltiple herencia por lo que si la clase hereda de otra clase esta opción no es posible de utilizar.
* ¿Por qué es importante el uso de **Synchronized**?
  + Asigna recursos y tiempo de ejecución a un solo thread, evitando que otros hilos hagan uso de esos recursos o métodos.
* ¿Cuál es la diferencia entre un **Semáforo** y un **Monitor**?
  + El semáforo permite que los hilos ejecuten los métodos y recursos, en cambio el monitor controla quien puede hacer uso de los recursos y de los métodos, asignando tiempo y oportunidad a cada hilo para ejecutar su procesamiento.
* ¿Por qué los filósofos aún podrían morir de hambre **(Starvation)** en el ejercicio con el monitor?
  + Por qué podría darse la posibilidad que un filósofo nunca tenga tiempo de comer y cada vez que lo intenta los recursos se encuentran ocupados.

**Conclusión.**

Para concluir podemos decir que la práctica se completó al 100%, en ella se repasaron conceptos vistos en sistemas operativos, además que se dominó el código java para el usos de hilos de ejecución, además se aprendió la nueva herramienta de monitor que permite administrar el accesos a los recursos y evitar que se generen incoherencias durante la ejecución del programa.

Aunque el código java ya se comprendió, se entiende completamente y se conocieron detalles de su implementación, falta practicar y poner en uso el código en un proyecto.

Finalmente hemos terminado la práctica correspondiente a los filósofos, cumpliendo y completando todos los objetivos planteados en ella.

|  |  |  |  |
| --- | --- | --- | --- |
| logo_udlap    **EDEI** | **Reporte Servicios Web** | | |
| **DEPARTAMENTO** | Computación, Electrónica y Mecatrónica | **MATERIA** |  |
| **PROFESOR** | Dra. Ofelia Cervantes | **PERIODO** | Primavera 2015 |

Objetivo

El objetivo de esta práctica es conocer y poner en práctica lo que compete a servicios web, en esta práctica se llevara a cabo en dos partes, la primera crear un servicio web y la segunda crear funciones que pueda ser implementadas.

Contenido

Antes de Empezar

Antes de poder iniciar la actividad debemos de instalar todos los elementos que debemos de utilizar.

* IDE como Eclipse o InteliJ
* Java JDK 6 o superior
* Instalar SBT

Para instalar SBT debemos de completar los siguientes pasos.

Windows:

* Descargar el instalador de Play [aquí](https://www.playframework.com/).
* Ejectuar el archivo **.bat**
* Agregar activator a la variable **Path**.

Cuando todo el proceso de complete con éxito, debemos de continuar con la actividad.

Parte 1

Debemos de abrir una ventana de comandos y ejecutar el código

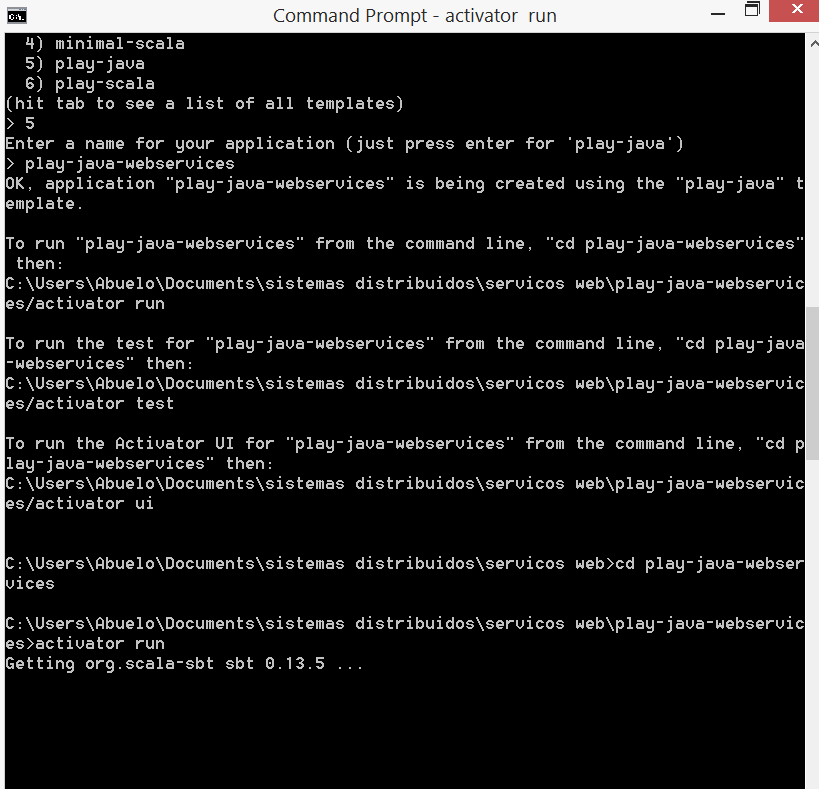
$ activator new

Este código ejecutar un menú donde crearemos nuestro servicio web, escogeremos la opción 5 java play, posteriormente colocaremos el nombre de nuestro servicio, y damos enter nuevamente, en la ventana de comandos, posteriormente debemos de entrar al directorio que se creó y tiene el nombre del servicio y ejecutar el comando.

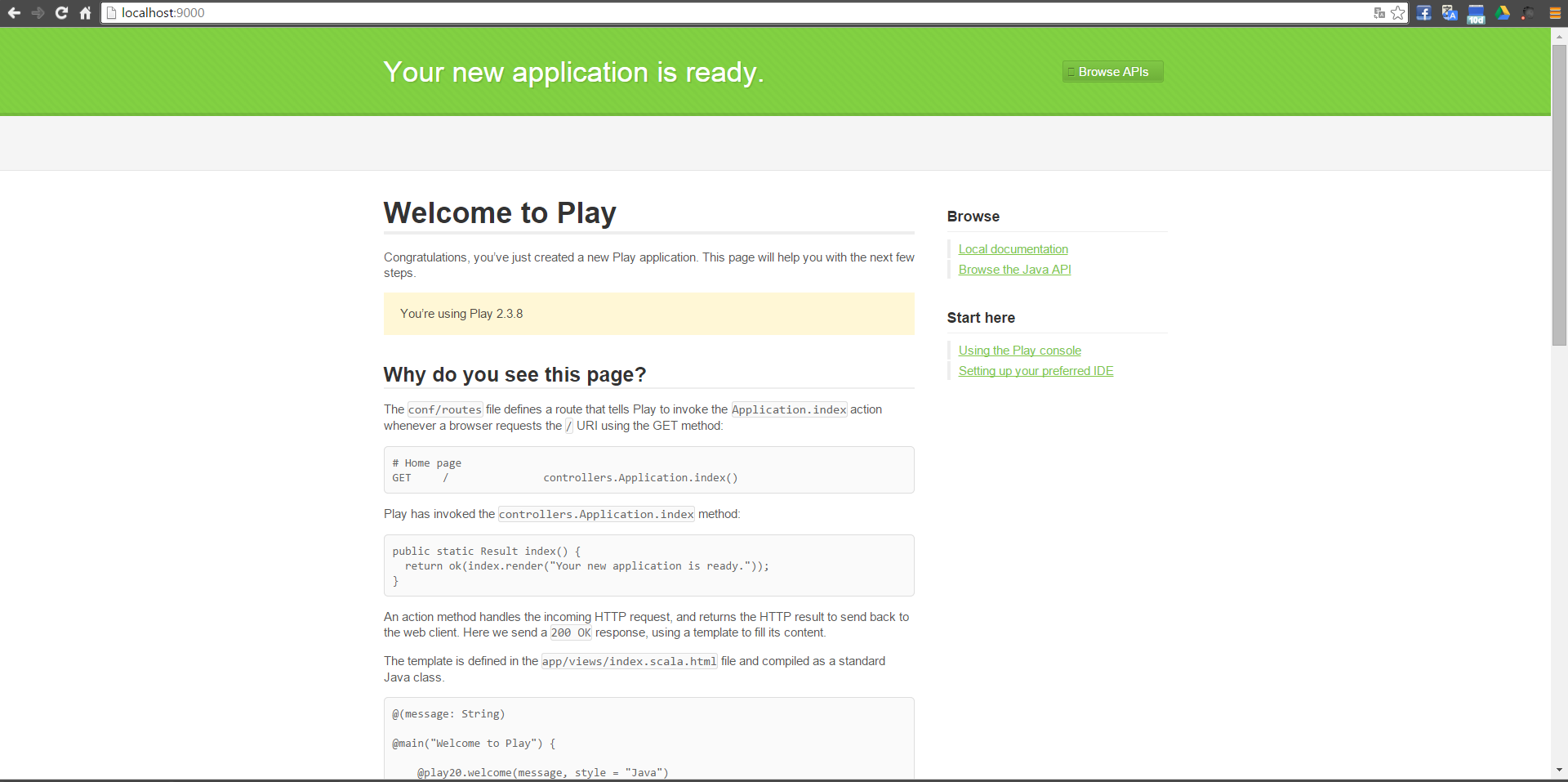
$ activator run

A continuación se ejecutaran todas las instrucciones necesarias para completar la creación del servicio, este proceso puede durar varios minutos.

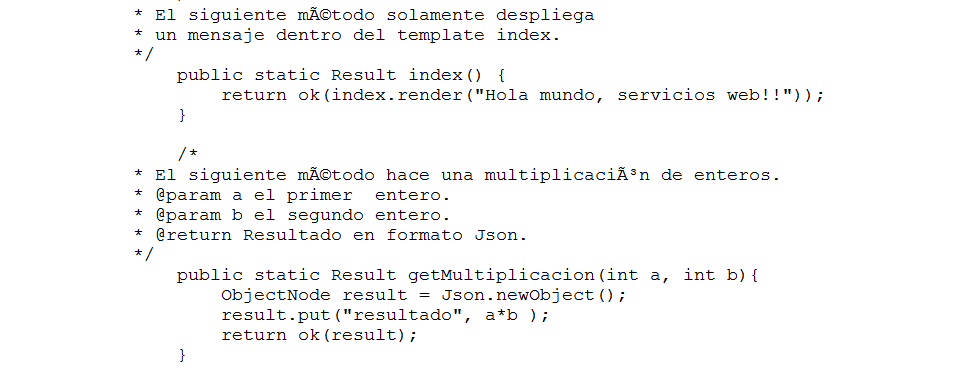
La siguiente imagen muestra en la parte de arriba el menú que se despliega, el nombre del servicio web, el comando run y el inicio del proceso de creación del servicio.



Cuando el proceso se termina, debemos de ir un navegador, he ir a la dirección **"localhost:9000", esta dirección nos llevar a la página que se muestra a continuación.**



El siguiente paso de la actividad es crear un método dentro de las clases creadas para el servicio, el siguiente código fue insertado en la aplicación. El codigo fue insertado en la clase aplicación dentro de la carpeta app//controller.

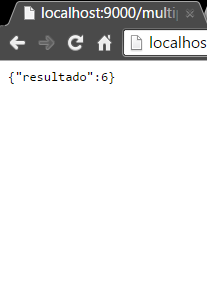


Para ejecutar este método debemos de declararlo también routes, en el debemos de declarar el tipo de método, los parámetros de salida, y el nombre del método para que se pueda ejecutar.

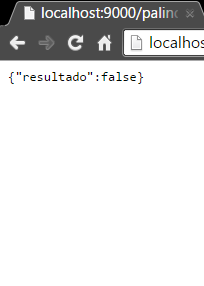
La nueva página para ejecutar este método es la siguiente:

**localhost:9000/multiplicacion/2/3**:

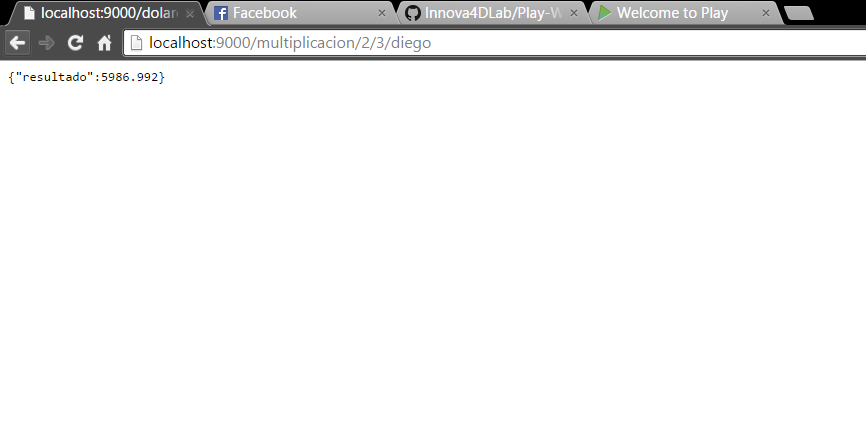
La dirección pide al servicio que ejecute la operación multiplicación y recibe como parámetros un 2 y 3, el resultado se muestra en la siguiente imagen.



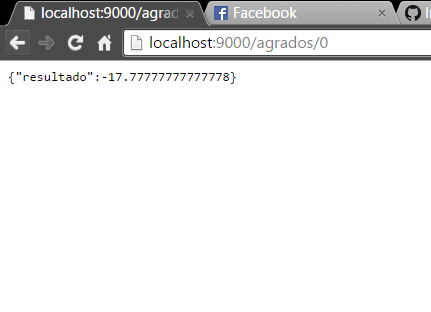
Función palíndromo



Función convertir de dólares a pesos.



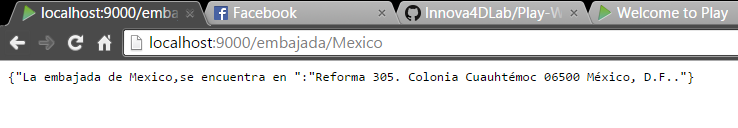
Convertir a grados Fahrenheit.



Obtener la capital de un país,



Obtener la dirección de la embajada.



Conclusiones

Esta práctica se completó correctamente, se aprendió y entendió el concepto referente a servicios web, principalmente a crear un nuevo servicio, declararlo, y usarlo desde un navegador, en modo local, aunque la práctica fue de muy fácil ejecución tuvo complicaciones, el tiempo de desarrollo de las funciones tomo mucho tiempo y fue completado fuera del periodo de laboratorio, además de que el servicio se probó en modo local, faltaría hacerlo desde un servidor para que fuera ocupado desde cualquier parte.

|  |  |  |  |
| --- | --- | --- | --- |
| logo_udlap    **EDEI** | **Web Service Mashup Master** | | |
| **DEPARTAMENTO** | Computación, Electrónica y Mecatrónica | **MATERIA** | Sistemas Distribuidos |
| **PROFESOR** | Dra Ofelia Cervantes | **PERIODO** | Primavera 2015 |

Introducción.

Durante las prácticas pasadas trabajamos en servicios web, junto con el framework de Play, facilitando el desarrollo de aplicaciones distribuidas, en las prácticas pasadas el objetivo fue crear y utilizar los servicios que nosotros mismos creamos, en esta nueva practica utilizaremos servicios ya existentes.

Objetivo.

El objetivo principal de esta práctica es utilizar los servicios ya existentes de google maps y de flickr, haciendo uso de nuestros servicios web de las prácticas anteriores.

Desarrollo.

Para estas actividades debemos de descargar los archivos que se encuentran en la dirección de gitHub correspondiente a la práctica y sustituirlos en el proyecto de la práctica pasada, estos archivos, contiene una página en HTML que incorpora el uso de algunos servicio, que junto con los servicios que desarrollamos formaremos una pequeña aplicación.

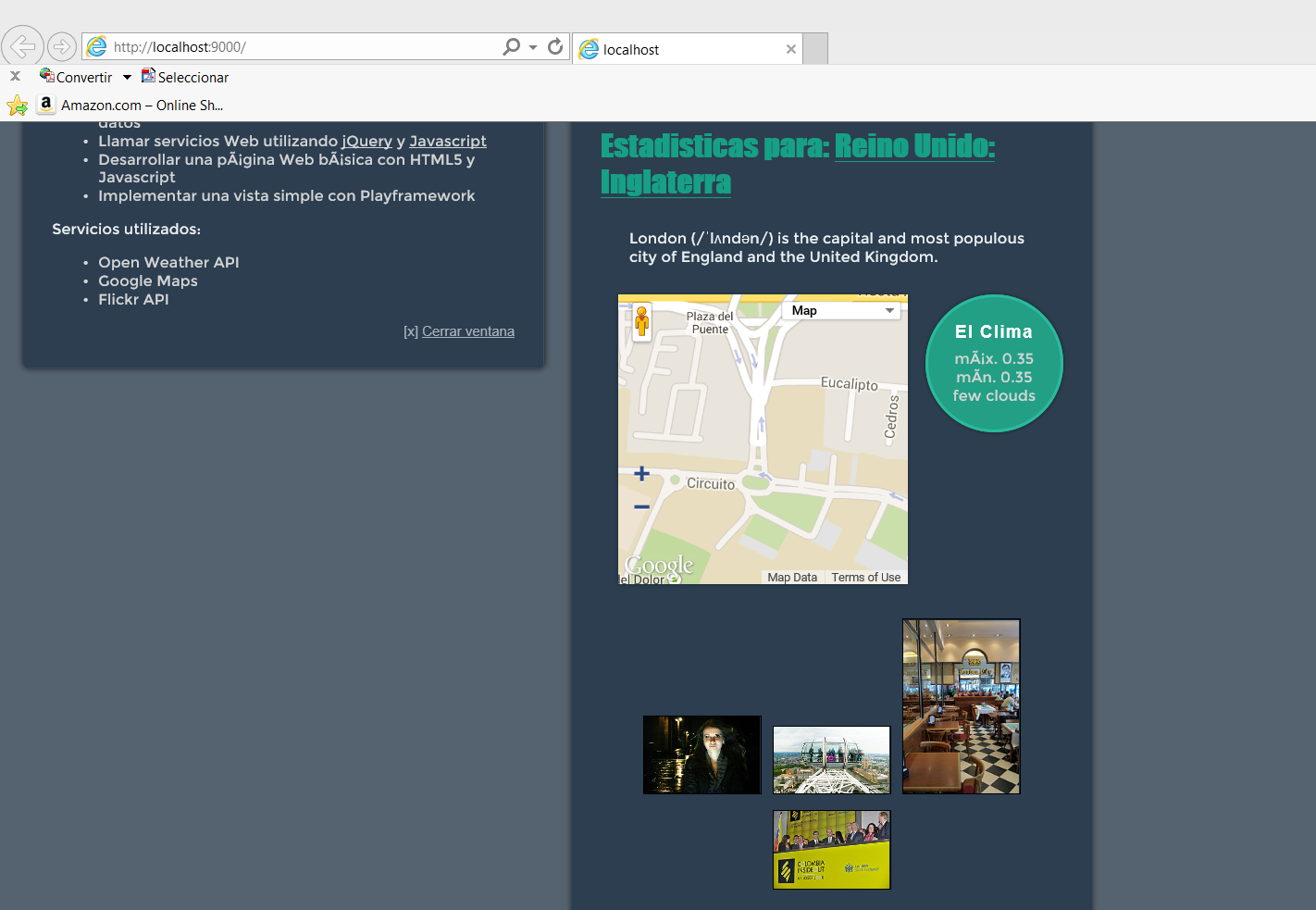
Los archivos que debemos de sustituir en el proyecto son: main.scala.html y main.css, estos archivos están contenidos en el GitHub, como main(2).scala.html y main(2).css, como resultados tendremos una página en de internet que ya incorpora varios servicios existentes como google maps.

El segundo paso de la práctica es incorporar nuestros servicios web, al que le pediremos la localización de la embajada, obtendremos un String del cual extraeremos la primera coordenada y la segunda para enviarlas al servicio de google maps y permitir que en el mapa muestre la localización de la embajada.

Otro servicio nos permitirá recuperar la temperatura promedio de dicho país, utilizar nuestro servicio web para convertir a grados Fahrenheit y proyectarlo en la página web.

Final mente con flickr debemos obtener las fotos que estén cercanas a la embajada del país que seleccionamos, en la parte de abajo del mapa se mostraran las imágenes que se encuentren en determinado radio teniendo como punto la embajada, y una foto que corresponda a la embajada para que los usuarios puedan identificarla fácilmente.

Para mostrar que los tres servicios integrados funcionan perfectamente, a continuación se muestra una imagen con los resultados.



Conclusiones.

Esta práctica fue relativamente sencilla, solo fue incorporar funciones que ya estaba desarrolladas, aun así la practica tomo tiempo, ya que no se comprendía la estructura de los nuevos servicios, la práctica tuvo sus dificultades para poder ser terminada, pero fue resulta fácilmente gracias a una librería de JAVA que nos permitió obtener las cadenas correspondientes a las coordenadas para poder hacer uso del servicio web.

|  |  |  |  |
| --- | --- | --- | --- |
| logo_udlap    **EDEI** | **Reporte Practica WebSocket Parte 1** | | |
| **DEPARTAMENTO** | Computación, Electrónica y Mecatrónica | **MATERIA** | Sistemas Distribuidos |
| **PROFESOR** | Dr Ofelia Cervantes | **PERIODO** | Primavera 2015 |

Introducción

En el desarrollo de estos documentos, se presentaran los detalles de implementación de la práctica correspondiente a websockets, haciendo uso del sistema de actores.

Los actores son entidades que se ejecutan concurrentemente y se envían mensajes asíncronos en ellos, el modelo de actores está planeado en base a la idea de que los actores no comparten el estado y que de esta manera no hay problema de deadlock para acceder al estado.

Objetivo

El objetivo de esta práctica es implementar un chat anónimo en modo local, utilizando sistema de actores de Akka atravez de Websockets con playFramework.

Desarrollo

Para iniciar con la práctica debemos de descargar el proyecto de la página de Git del curso, colocarlo en una carpeta estratégica en nuestro equipo y atreves de ventana de comandos debemos de ejecutar el comando “activator run”, dentro de la carpeta del proyecto, esto nos permitirá iniciar el servidor donde se va ejecutar el chat, la aplicación por el momento cuenta con errores que deberán ser resueltos durante el trascurso de nuestra práctica.

La segunda parte de la práctica es corregir los errores que se presentaron al iniciar el proyecto, el primer paso para ello es definir un servicio Rest **assets/javascripts/index.js** agregando el siguiente código:

GET /ws controllers.Application.ws

Posteriormente debemos de copiar los siguientes fragmentos de código, el primero será el encargado de recibir los datos, el segunda parte del código deberá enviar el mensaje.

Código de recibir:

var ws;

ws = new WebSocket($("body").data("ws-url")); //Se toma La URL del Tag en Body

ws.onmessage = function(event) {

var message;

message = JSON.parse(event.data);

switch (message.type) {

case "message":

return $("#board tbody").append("<tr><td>"+ message.msg + "</td></tr>");

default:

return console.log(message);

}

};

Código de enviar:

$("#msgform").submit(function(event) {

event.preventDefault();

console.log($("#msgtext").val());

ws.send(JSON.stringify({ //Enviar el mensaje en el formulario como JSON.

msg: $("#msgtext").val()

}));

return $("#msgtext").val("");

});

Con esto debemos de poder ejecutar el servicio web en su totalidad el chat debe funcionar por completo, por lo que ahora dado un vector de string, debemos de enviar cada elemento contenido en el vector utilizando una función que los envié cada 5 segundos.

Vector:

var speech = ["Hello there!","Welcome to the world of Pokémon!",

"My name is Oak!","People call me the Pokémon Prof!",

"This world is inhabited by creatures called Pokémon!",

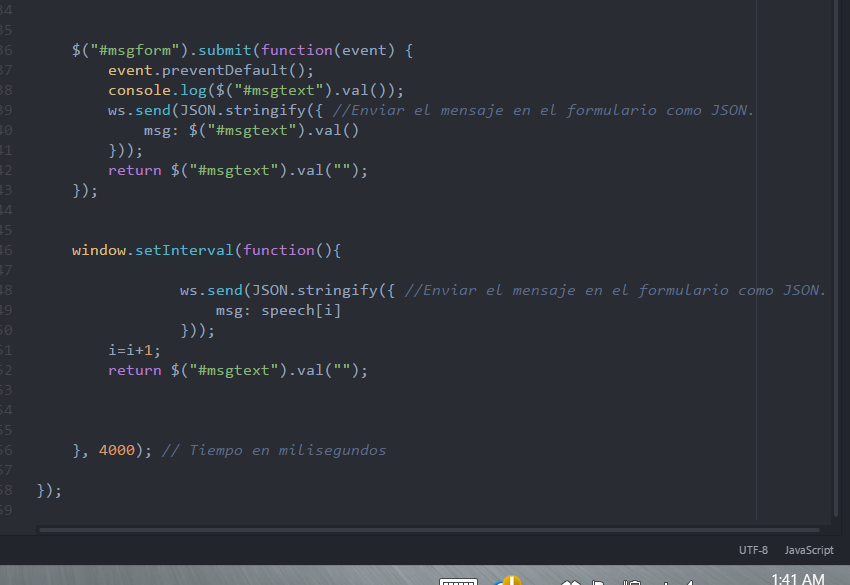
"For some people, Pokémon are pets",

"Other use them for fights",

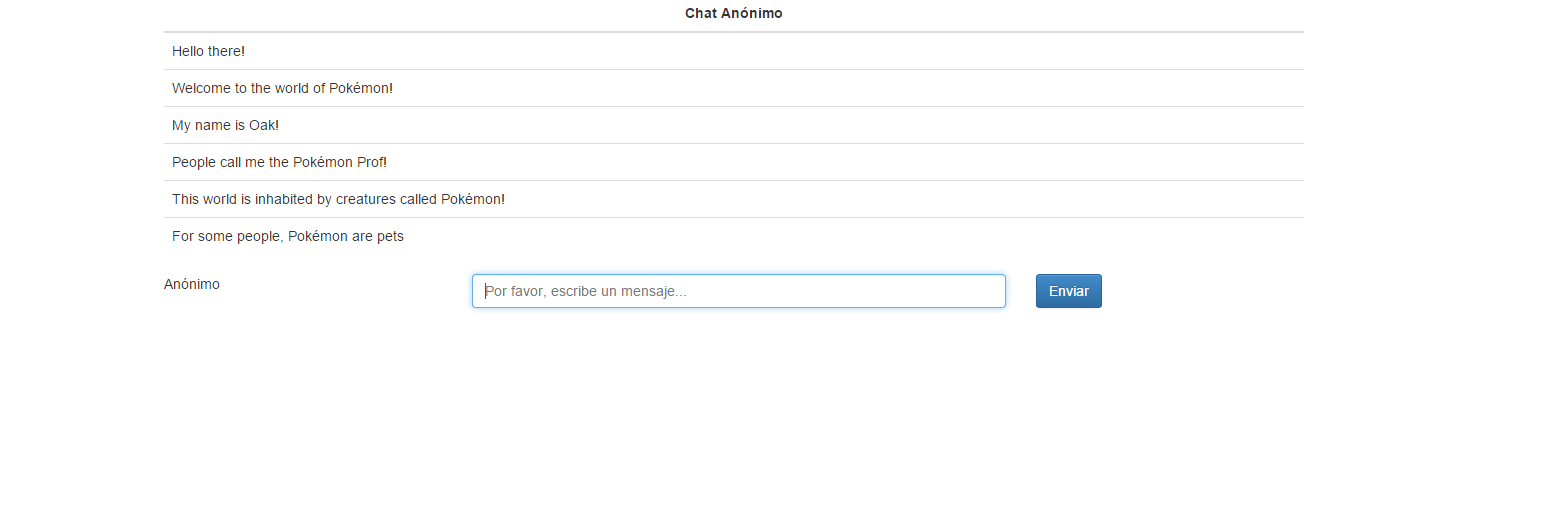
"Myself… I study Pokémon as a profession",

"First, what is your name?"];

Código de la implementación:



Aplicación



Conclusiones

En esta práctica se comprendieron los temas de websockets dentro del ámbito distribuido, los conceptos teóricos aprendidos durante la clase se pusieron en práctica durante el desarrollo de esta práctica, básicamente estos conceptos ayudaron a comprender lo que se estaba llevando a cabo en el código del proyecto que implementaría nuestra aplicación de chat anónimo.

La práctica se completó al 100%, no obstante falta mejorar el desarrollo de aplicaciones utilizando websockets y comprender des los detalles de implementación que se están llevando a cabo en el código.

Preguntas

* ¿**Qué es WebSocket?**
  + Es una tecnología que permite una conexión bidireccional sobre un mismo socket, está diseñada para navegadores y servicios web.
* **¿Cuál es la diferencia entre WebSocket y Servicios REST?**
  + La diferencia es tipo de protocolo en el que están basados, REST está basado en comunicación cliente servidor, mientras que websocket el cliente está conectado al servidor atreves de un conector.
* **¿Cómo se establece una conexión con WebSocket?**
  + el cliente manda una negociación de tipo websocket y el servidor manda una respuesta a la negociaion.
* **Explica la integración de datos en la interfaz utilizando JavaScript.**
* **¿Cómo se implementa el patrón MVC en esta aplicación?**
  + **¿Qué se hace en el controlador?**
    - El controlador en esta aplicación es el encargado de enviar el mensaje, mantener la conexión con el cliente y estar al pendiente de cualquier mensaje que se envié para actualizar la vista o el view,
  + **¿Cómo se implementa la vista?**
    - El view es la página que se muestra en nuestro navegador y esta implementada con html y JavaScrip.

|  |  |  |  |
| --- | --- | --- | --- |
| logo_udlap    **EDEI** | Reporte WebSocket-Play-Actors | | |
| **DEPARTAMENTO** | Computación, Electrónica y Mecatrónica | **MATERIA** | Sistemas Distribuidos |
| **PROFESOR** | Dra. Ofelia Cervantes | **PERIODO** | Primavera 2015 |

**Introducción**

El modelo de actores es un modelo matemático de computación concurrente que trata como "actores" a los principios de la computación concurrente: En respuesta a un mensaje que un actor recibe el actor puede tomar decisiones, como por ejemplo, crear actores, enviar mensajes y/o determinar como responder al siguiente mensaje. El modelo de actores adopta como filosofía que todo es un actor. Éste concepto es similar al de todo es un objeto utilizado en el paradigma orientado a objetos pero difiere en cuanto a que en un modelo orientado a objetos, el cómputo se ejecuta típicamente de manera secuencial, mientras que el modelo de actores es en esencia concurrente.

**Objetivo**

El objetivo de esta práctica es completar es completar métodos necesario que permita ejecutar un chat haciendo uso del framework de play he implementar el uso de actores de Akka atreves de websockets.

Desarrollo

Para dar inicio a la practica debemos de descargar el proyecto base desde la página del curso en la dirección: <https://github.com/Innova4DLab> , en esta dirección se encuentra todos los archivos necesarios para llevar acabo la actividad, debemos de crear un clon en nuestro escritorio, acceder a este directorio atreves de una ventana de comandos, por esta venta podemos ejecutar el comando: activaitor run, para iniciar con nuestro proyecto, después de unos minutos el proyecto queda listo para trabajar aunque con algunos errores de compilación.

Para editar nuestro proyecto utilizaremos un programa llamado Atom, el editor nos permite abrir diferentes tipos de archivos, para nuestra practica es muy útil ya que nos permite visualizar un archivo de cualquier tipo, por el momento debemos de localizar el archivo Future.succesful, para modificar el método UserActor.props, si el método no existe debemos de crearlo y luego insertar el siguiente código.

def ws = WebSocket.tryAcceptWithActor[JsValue, JsValue] { implicit request =>

Future.successful(request.session.get(UID) match {

case None => Left(Forbidden)

case Some(uid) => Right(UserActor.props(uid))

})

}

Este código implemente Websocket recibe dos valores, siempre y cuando la petición HTTP sea exitosa, se analiza si existe o no el Id, si a si es permite el acceso llama al método UserActor.props

Posteriormente debemos de ir al directorio BoardActor y en el archivo UserScala.sacala debemos de incertar el siguiente código:

override def preStart() = {

BoardActor() ! Subscribe

}

Aparte debemos de insertar este código, el código que se muestra a continuación crea un objeto JSON enviado mediante our a BoardActor.

def receive = LoggingReceive {

case Message(muid, s) if sender == board => {

val js = Json.obj("type" -> "message", "uid" -> muid, "msg" -> s)

out ! js

}

case js: JsValue => (js \ "msg").validate[String] map { Utility.escape(\_) } map { board ! Message(uid, \_ ) }

case other => log.error("Error, no se creó el objeto JSON: " + other)

}

Y posteriormente el siguiente fragmento de código:

object UserActor {

def props(uid: String)(out: ActorRef) = Props(new UserActor(uid, BoardActor(), out))

}

Finalmente debemos de insertar este ultimo fragmento:

def receive = LoggingReceive {

case m:Message => users map { \_ ! m} //Recibe el mensaje y lo envía a los actores suscritos

case Subscribe => {

users += sender // Agregar a usuarios al sender.

context watch sender // Observar al sender

}

case Terminated(user) => users -= user

}

object BoardActor {

lazy val board = Akka.system().actorOf(Props[BoardActor])

def apply() = board // Cuando un actor se suscribe retorna Board.

}

El código agrega a un usuario en la lista, el actor empieza a ser observado, en cuanto el actor envía un mensaje este lo revisa y lo envía a todos los demás.

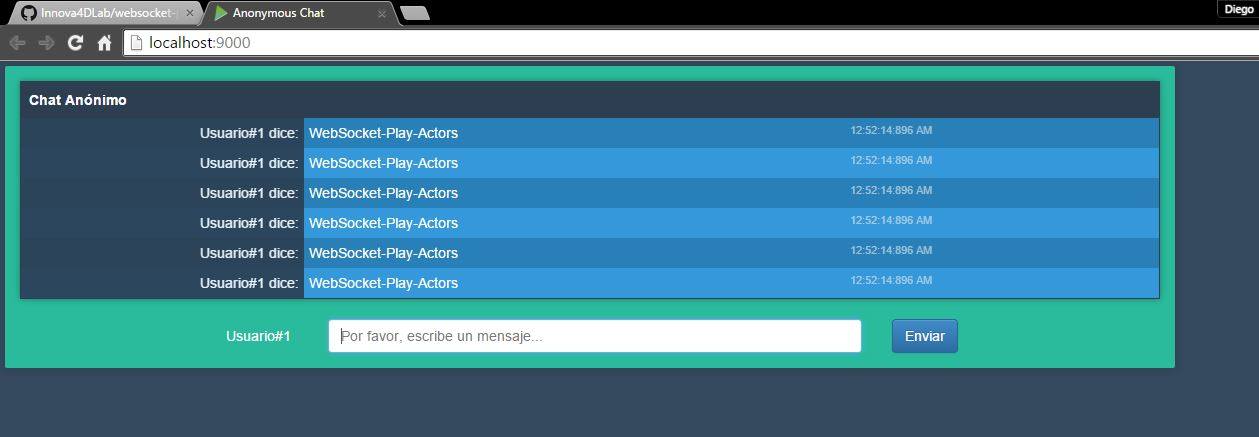
Antes de volver a correr el servidor para verificar que funciona debemos de ir al directorio conf/routes y comidficar el archivo index, este archivo se ejecuta automáticamente al cargar la dirección local en nuestro navegador.

GET / controllers.Application.index

GET /ws controllers.Application.ws

Detenemos el servidor que se encuentra corriendo en nuestra ventana de comando, y lo volvemos a ejecutar, en esta ocasión no debe de marcar errores, si es así, debemos de ir a nuestro navegador favorito y asegurar que el servidor está haciendo su función adecuada.

La siguiente imagen muestra el resultado obtenido durante la práctica



Para completar la practica debemos de implementar la siguiente función, cada uno de los mensajes que los actores envían deben de incluir la fecha en la que se realiza cada uno de ellos, debemos de agregar el código necesario antes de crear el objeto de tipo JSON, el código necesario para realizar esta función es el siguiente.

val hour = new SimpleDateFormat("hh").format(today)

val ampm = new SimpleDateFormat("a").format(today)

val minute = new SimpleDateFormat("mm").format(today)

val seconds = new SimpleDateFormat("s").format(today)

val miliseconds = new SimpleDateFormat("S").format(today)

val time = hour + ":" + minute + ":" + seconds + ":" + miliseconds + " " + ampm

Claramente el código concatena la fecha que se va a imprimir junto con cada mensaje en nuestro chat, para eso debemos de modificar el objeto JSON que enviamos, para que este incluya la fecha y la forma de regresar este tipo de objeto.

val js = Json.obj("type" -> "message", "uid" -> muid, "msg" -> s, "timestamp" -> time)

return $("#board tbody").append("<tr><td class='user'>Usuario#"+ message.uid +" dice:</td><td>"+ message.msg + "</td><td class='timestamp'>"+message.timestamp+"</td></tr>");

Conclusión.

Esta práctica se completó exitosamente en la mitad del tiempo especificado para ella, durante el desarrollo de esta se pusieron en práctica los conocimientos obtenidos durante la clase teórica, se implementó el uso de actores junto con web servicie, en esta práctica se comprendió el funcionamiento de WebSoscket y actores, de esta forma el servidor está esperando que el actor envie un mensaje y lo replica a todos los demás, en lugar de asignarle tiempo a cada uno, mejor cada actor le dice atreves de su socket al servidor que tiene algo que procesar y necesita de sus recursos.

Preguntas

* ¿Un chat es una aplicación concurrente?
  + El chat es una aplicación concurrente, ya que cada chat por separado es un proceso concurrente.
* Explica como harías un Chat sin el modelo de actores
  + El chat sin modelo de actores puede ser implementado mediante un sistema donde el servidor le asigna un determinado tiempo a cada usuario para que actualice sus mensajes y el servidor actualice los mensajes enviados.
* Explica como harías un Chat sin Websocket
* ¿Que representa **UserActor**?
  + Representa a todos los actors que estan conectados al servidor.
* ¿Cuál es el objetivo de **BoardActor**?
  + Es el encargado de actualizar a los demás actores atreves de WebSocket
* ¿Cuál es el significado de Future en el método **Application.ws**?
  + Cuando la petición es exitosa indica los siguientes pasos a realizar, o prohibir el acceso.
* ¿En qué momento se ejecuta el método **Application.index**?
  + Al momento de cargar la dirección del servidor en este caso el local host
* Explica el uso de la variable **users** dentro de **BoardActor.scala**
  + esta variable permite almacenar todos los usuarios que están conectados al servidor.

|  |  |  |  |
| --- | --- | --- | --- |
| logo_udlap    **EDEI** | **Reporte Meteor** | | |
| **DEPARTAMENTO** | Computación, Electrónica y Mecatrónica | **MATERIA** | Sistemas Distribuidos |
| **PROFESOR** | Dra. Ofelia Cervantes | **PERIODO** | Primavera 2015 |

Introducción

**Meteor** es una plataforma muy completa para desarrollar aplicaciones móviles y Web únicamente utilizando Javascript. Algunas características de **Meteor**:

* **Interfaz de usuario moderna**
  + Construye aplicaciones de alta calidad, como Facebook ó Twitter.
* **Mucho menos código**
  + Gracias a su modelo reactivo, desarrolla en pocas líneas lo que tomaría mucho más tiempo.
* **Un lenguaje en todas partes**
  + El mismo lenguaje se utiliza para el **cliente** como para el **servidor**.
* **Actualizaciones en tiempo real**
  + Los usuarios pueden observar cómo cambian los datos instantáneamente y colaborar sin interrupciones.
* **Altamente responsivo**
  + El código en JavaScript se adapta a cualquier plataforma.

Objetivo

Es crear una lista reactiva en esta lista tendremos tareas pendientes que no hemos completado.

Desarrollo

Primero debemos de instalar Meteor en nuestro equipo, el instalador lo obtendremos de la siguiente dirección:

<https://install.meteor.com/windows>

Una vez descargado debemos de ejecutar el instalador, cuando el procesos termine las computadoras que tengan Windows 8.1 deberán ser reiniciadas para que los comandos funcionen correctamente.

Cuando la computadora este lista debemos de colocarnos en un nuestra carpeta de preferencia, atreves de una ventana de comandos debemos de llegar a ese directorio y ejecutar el siguiente comando:

meteor create simple-todos

En esta ocasión el nombre de nuestra aplicación será simple-todos, cuando la ejecución termine, se generara los siguientes archivos.

//////foto

Para ejecutar la aplicación debemos de entrar al directorio que creo el primer comando y ejecutar el siguiente tal como lo muestra el siguiente bloque de instrucciones.

cd simple-todos

meteor

En nuestro navegador web de nuestra preferencia debemos de ir a la siguiente dirección para ver el resultado de nuestra aplicación.

**http://localhost:3000**

Para continuar debemos de insertar el siguiente código en el archivo todo-list.html y todo-list.js y simple-todos.js.

<!-- todo-list.html -->

<head>

<title>Lista de tareas</title>

</head>

<body>

<div class="container">

<header>

<h1>Lista de tareas</h1>

</header>

<ul>

{{#each tasks}}

{{> task}}

{{/each}}

</ul>

</div>

</body>

<template name="task">

<li>{{text}}</li>

</template>

// todo-list.js

if (Meteor.isClient) {

// Este código se ejecuta únicamente en el cliente

Template.body.helpers({

tasks: [

{ text: "Tarea 1" },

{ text: "Tarea 2" },

{ text: "Tarea 3" }

]

});

}

// simple-todos.js

Tasks = new Mongo.Collection("tasks");

if (Meteor.isClient) {

// Este código se ejecuta únicamente en el cliente

Template.body.helpers({

tasks: function () {

return Tasks.find({});

}

});

}

Continuamos abriendo una nueva ventana de comandos en la que ejecutaremos los siguientes comandos para crear nuevos archivos en la base de datos.

meteor mongo

db.tasks.insert({ text: "Hello world!", createdAt: new Date() });

Debemos de completar el siguiente código en los archivos correspondientes.

<header>

<h1>Lista de tareas</h1>

<!-- Agregar un formulario después de H1 -->

<form class="new-task">

<input type="text" name="text" placeholder="Escribir para agregar nuevas tareas" />

</form>

</header>

// Dentro de el bloque if (Meteor.isClient) después de Template.body.helpers:

Template.body.events({

"submit .new-task": function (event) {

// Esta función será llamada despues de agregar un nuevo elemento

var text = event.target.text.value;

Tasks.insert({

text: text,

createdAt: new Date() // current time

});

// Clear form

event.target.text.value = "";

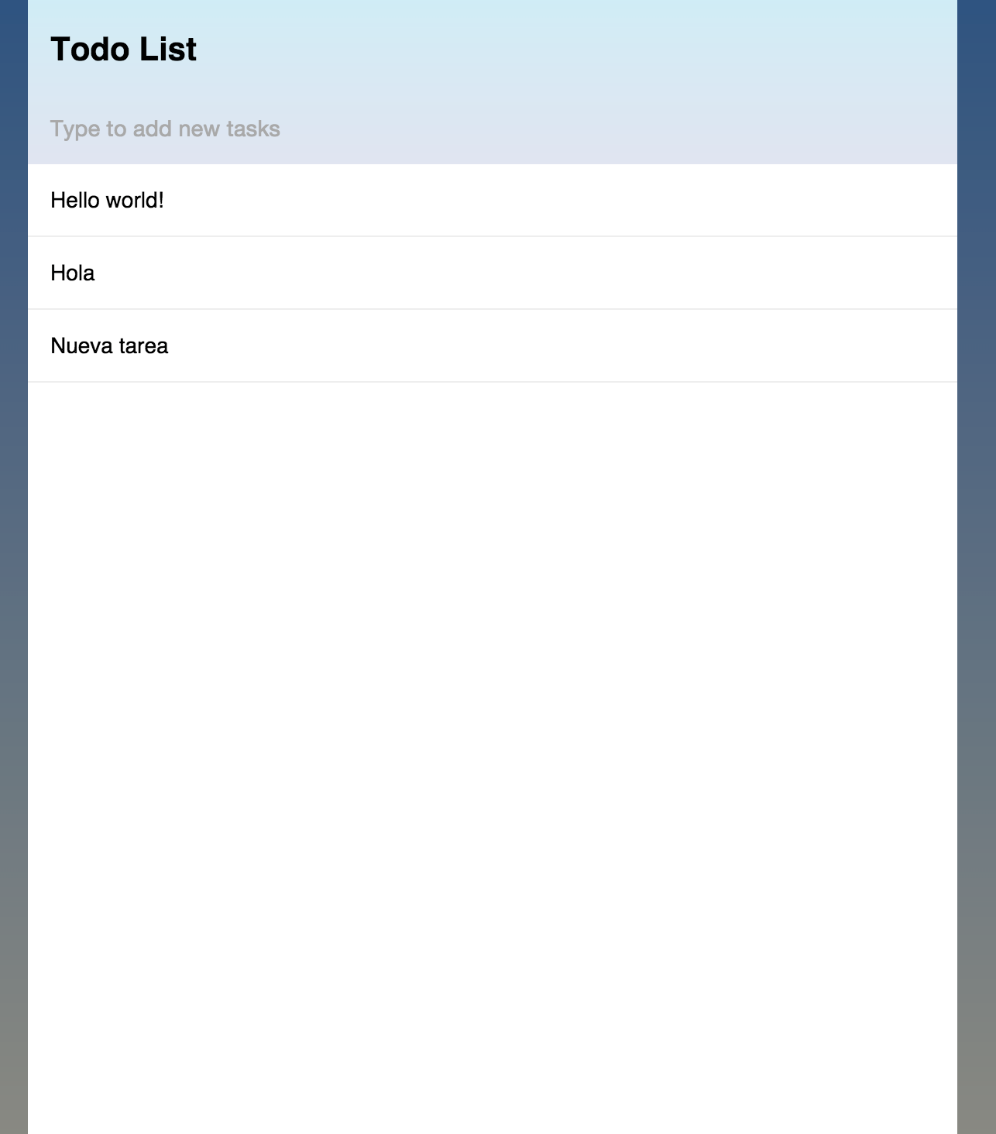
// Prevenir que se envíe por default

return false;

}

});

El resultado debe verse como la siguiente imagen.



Para poder eliminar tareas en la lista, debemos de insertar el siguiente código.

<!-- reemplazar el template actual por el siguiente -->

<template name="task">

<li class="{{#if checked}}checked{{/if}}">

<button class="delete">&times;</button>

<input type="checkbox" checked="{{checked}}" class="toggle-checked" />

<span class="text">{{text}}</span>

</li>

</template>

// En el código del cliente, debajo de todo lo demás.

Template.task.events({

"click .toggle-checked": function () {

Tasks.update(this.\_id, {$set: {checked: ! this.checked}});

},

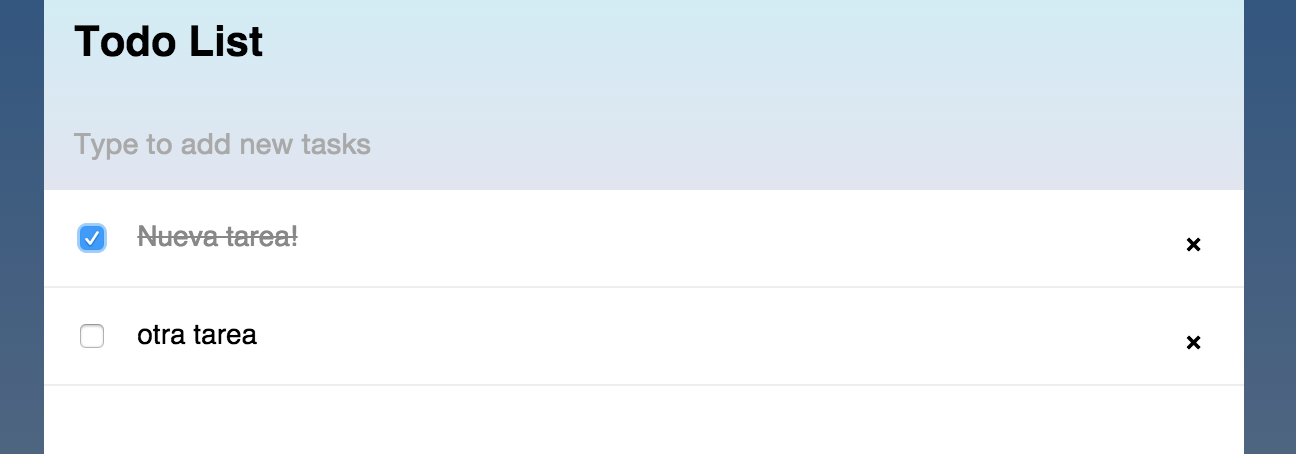
"click .delete": function () {

Tasks.remove(this.\_id);

}

});

La aplicación lista debería de verse de la siguiente manera



Conclusión.

En esta práctica se aprendió a utilizar la tecnología Meteor que nos permite crear aplicación móvil de una manera más fácil, los resultados son de una mejor calidad.

Meteor permite el uso de Java Scrip lo que permite una flexibilidad y que nuestra aplicación se adapte a diferentes plataformas.

Preguntas

* ¿Qué es una aplicación reactiva?
  + Las aplicaciones reactivas son aquellas que tienen componentes activos y listos para recibir eventos, una aplicación reactiva debe ser escalable, sensible, resistente.
* ¿Qué es **MongoDB**?
  + n sistema de base de datos NoSQL orientado a documentos. **MongoDB** guarda estructuras de datos en documentos tipo **JSON** con un esquema dinámico, por lo que es muy amigable con aplicaciones **javascript**.
* Explica el uso de los templates.
  + Un templete permite al cliente pedir información a la base de datos de MongoDB
* Describe ampliamente la conexión "automática" entre **Meteor** y **MongoDB**.
  + Meteor y MongoDB son tecnologías que utilizan como lenguaje javaScript por lo que las tecnologías se acoplan de muy buena forma permitiendo un funcionamiento adecuado y eficiente para el programador
* ¿Qué tendrías que hacer para desarrollar la aplicación en **Play Framework**?
  + Debería de crear un servicio web que recupere los mensajes de una base de datos distribuida y los coloque como tareas, para esto se necesita que otro servicio que utiliza base de datos interactuare con la aplicación, además de que la pagina web debería permitir agregar y eliminar tareas, lo que podría resultar complicado de realizar.