

# **Sistemas Distribuidos**

## **PRA2.**

**Phase 2-3-4. Informe**

**Kepa Sarasola Bengoetxea. Grupo 202\_75\_644\_02**

## Índice de contenidos

Phase 2.....	3
1. Introduction.....	3
1.1 Estructura de la solución.....	3
1.2 Decisiones tomadas.....	3
2. Implementation.....	5
2.1 Partes añadidas al código.....	5
2.2 Limitaciones de la solución aportada.....	10
3. Test.....	11
3.1 Test en entorno local.....	11
3.2 Test en DSLab.....	14
4. Conclusiones Phase2.....	17
4.1 Dificultades.....	17
Phase 3.....	18
1. Introduction.....	18
1.1 Estructura de la solución.....	18
1.2 Decisiones tomadas.....	18
2. Implementation.....	19
2.1 Partes añadidas al código.....	19
3. Test.....	23
3.1 Test en entorno local.....	23
3.2 Test en DSLab.....	24
4. Conclusiones Phase3.....	25
4.1 Dificultades.....	25
Phase 4.....	26
1. Introduction.....	26
1.1 Estructura de la solución.....	26
1.2 Decisiones tomadas.....	26
2. Implementation.....	27
2.1 Partes añadidas al código.....	27
3. Test.....	30
3.1 Test en entorno local.....	30
3.2 Test en DSLab.....	31
4. Conclusiones Phase4.1.....	32
4.1 Dificultades.....	32

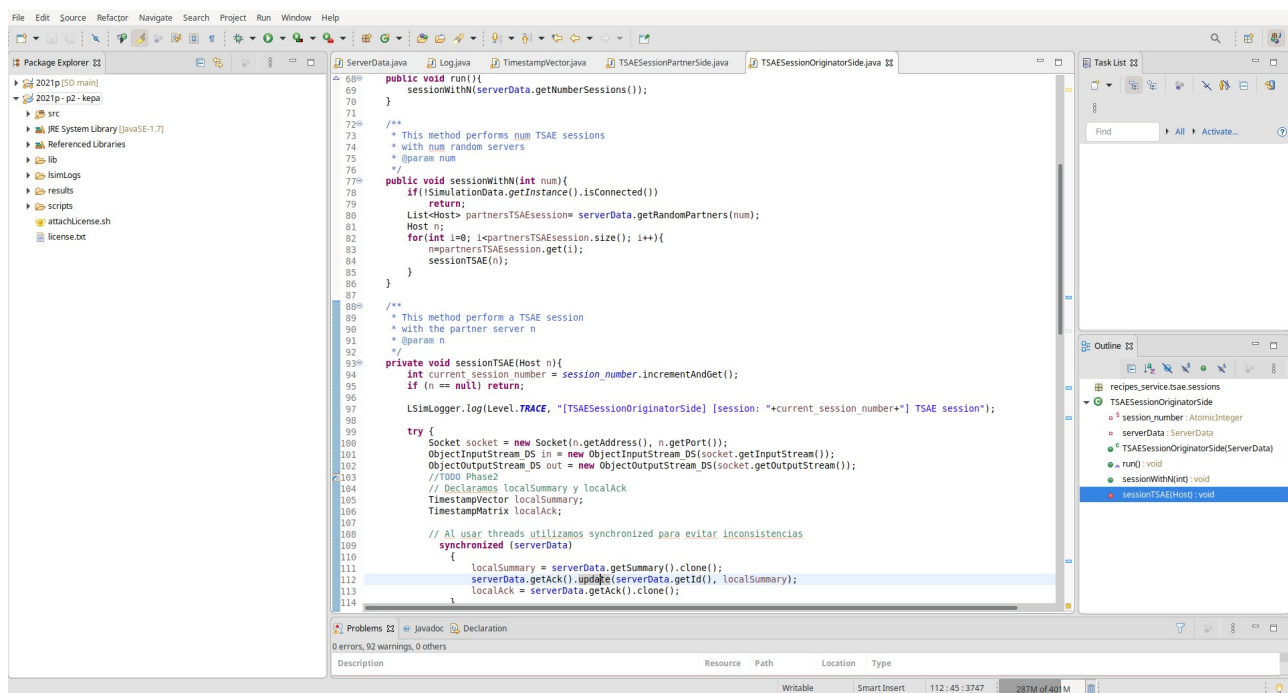
# Phase 2

## 1. Introduction

### 1.1 Estructura de la solución

En esta segunda fase, se nos ha pedido la implementación de una versión reducida del protocolo TSAE, con el objetivo de añadir operaciones sin purgar el log. Al igual que en la phase1, lo primero ha sido cargar el proyecto en el IDE Eclipse, tal y como lo deje en la Phase1, centrandome en entender el funcionamiento del protocolo y de las clases implicadas en esta Phase2.

Atendiendo al enunciado, las clases implicadas en esta segunda fase son, *ServerData*, *Log*, *TimestampVector*, *TSAESessionPartnerSide* y *TSAESessionOriginatorSide*.



### 1.2 Decisiones tomadas.

#### 1.2.1 Métodos desarrollados para realizar esta práctica

Aunque en los ficheros .java descargados junto con el enunciado hay varios métodos no implementados, solamente he desarrollado aquellos que he creído necesarios para cumplir con los requisitos de la Phase2, que recordemos era la implementación de las opciones de añadir operaciones al log.

- **Clase Log**

La clase Log registra las operaciones recibidas por el cliente e implementa un log, la información sobre las operaciones recibidas se guardan en un *ConcurrentHashMap*. Esta clase implementa algunos métodos que no están desarrollados todavía. Para desarrollar esta segunda fase, "Phase2", he desarrollado el método *listNewer(TimestampVector sum)*, que se encarga de añadir un elemento a una lista de Operaciones, si esta es posterior a la última operación realizada

- **Clase TimestampVector**

La clase TimestampVector se encarga de gestionar un resumen de marcas de tiempo para cada host y se encarga de guardar en el *summary* la marca de tiempo (*timestamp*) de la última operación recibida desde este host.

Para realizar esta segunda fase he implementado los siguientes métodos:

- *updateMax(TimestampVector tsVector)*, este método se encarga de actualizar el timestamp de un vector de tiempos en caso de que el vector de tiempo pasado por parametro sea posterior al del vector de tiempos de la clase.
- *Timestamp getLast(String node)*, devuelve el valor del *node* pasado por parametro dentro del vector de tiempos de la clase.
- *mergeMin(TimestampVector tsVector)*, este método se encarga de actualizar el timestamp de un vector de tiempos, con el valor minimo entre el valor pasado por parametro y el que contiene el parametro de la clase.
- *clone()*, este metodo se encarga de devolver un objeto de tipo TimestampVector que es un clon del que contiene la clase.

- **Clase ServerData**

- *addRemoveOperation(MessageOperation message)* este método es nuevo y su objetivo es simplificar la adición o la eliminación de operaciones desde los métodos que controlan la comunicación entre los nodos *TSAESessionPartnerSide* y *TSAESessionOriginatorSide* que en algunos casos tendrán como objetivo añadir operaciones y en otras eliminarlas.

- **Clases TSAESessionPartnerSide y TSAESessionOriginatorSide:** en este caso he seguido el pseudocódigo o plantilla que se indica en el enunciado y que esta parcialmente implementada en el código. Básicamente he tratado de implementar las partes necesarias para que la clase realice las operaciones necesarias según el tipo de mensaje, es decir, la labor que se quiera realizar en cada momento. En este punto desatacaría la necesidad de bloquear ciertas variables durante las operaciones con mediante *synchronized* concretamente en el momento que necesitamos clonar los parametros de la clase ServerData localSummary y localAck que contienen el summary y el ack que han de ser clonados para realizar el protocolo TSAE de forma exitosa.

Se puede observar que las operaciones que se realizan desde estas clases y que utilizan los métodos de la clase ServerData han de estar protegidos (o bloqueados) para que aseguran la integridad de la información contenida en estos mediante la palabra clave *synchronized*.

### 1.2.2 Acceso simultáneo a los datos

Como bien se ha hecho notar en el enunciado, puede existir un acceso simultáneo a los datos, es decir dos instancias de objetos recibidos desde diferentes *threads* pueden mezclarse, produciendo confusión y resultados erróneos.

Ante esta situación en la que puede darse *Multi-thread*, es necesario asegurarse de que mientras un hilo esta accediendo a un recurso dado, solamente el *thread* implicado pueda realizar cambios en los recursos comprometidos.

Para evitar estas interferencias y posibles errores, he usado la palabra clave *synchronized* en la declaración de varios metodos. Java ofrece esta palabra clave para poder sincronizar tareas mediante el uso de bloques sincronizados, asegurando así que todos los bloques sincronizados sobre el mismo objeto sólo pueden tener un hilo ejecutándose dentro de ellos a la vez.

Todos los demás hilos que intentan entrar en el bloque sincronizado se bloquean, hasta que el hilo que está usando los recursos del bloque sale de este liberando los recursos.

## 2. Implementation

### 2.1 Partes añadidas al código

Como ya he comentado en el punto anterior, los métodos que añadido en el código son las siguientes:

- **Clase Log**
  - **listNewer(TimestampVector sum)**

Este método es usado para añadir una nueva operación a una lista de Operaciones, si esta es posterior a la última operación realizada.

El método cruza la información contenida en el parametro log de la clase, obtiene el node de este log y compara su timestamp con el timestamp del objeto *TimestampVector* que recibe como parametro. Una vez realizada una comparación entre ambos añade la operación a la lista de operaciones posteriores a la que contiene la clase.

El método es precedido por la palabra clave *synchronized* tal y como hemos indicado anteriormente con el objeto de proteger la integridad de los recursos que utiliza el método.

Imagen con la implementación del método:

```
public synchronized List<Operation> listNewer(TimestampVector sum){
    //TODO Phase2
    List<Operation> newerListOp = new Vector<Operation>();
    List<String> participants = new Vector<String>(this.log.keySet());

    for (Iterator<String> it = participants.iterator(); it.hasNext(); ){
        // necesitamos el siguiente nodo para obtener las operaciones
        String node = it.next();
        List<Operation> operations = new Vector<Operation>(this.log.get(node));
        Timestamp timestampToCompare = sum.getLast(node);
        // Recorremos las operaciones que obtenemos del nodo
        for (Iterator<Operation> opIteration = operations.iterator(); opIteration.hasNext(); ) {
            Operation op = opIteration.next();
            if (op.getTimestamp().compare(timestampToCompare) > 0) {
                // Comparamos los Timestamp y lo añadimos a la lista en caso de que sea mayor que el que tenemos por parametro
                newerListOp.add(op);
            }
        }
    }

    return newerListOp;
}
```

- **Clase TimestampVector**

- **updateMax(TimestampVector tsVector)**

Este método se encarga de actualizar el timestamp de un vector de tiempos en caso de que el vector de tiempo pasado por parametro sea posterior al del vector de tiempos de la clase.

El método itera sobre el objeto *TimestampVector* que recibe como parámetro, va obteniendo el timestamp contenido en cada iteración y lo va comparando con el que contiene el parámetro *tsVector* de la clase. Una vez comparados los dos timestamps actualiza el timestamp del parametro de la clase con el mayor de los dos.

El método es precedido por la palabra clave *synchronized* tal y como hemos indicado anteriormente con el objeto de proteger la integridad de los recursos que utiliza el método.

Imagen con la implementación del método:

```
public synchronized void updateMax(TimestampVector tsVector){
    //TODO Phase2
    // Recorremos el hashmap con un iterator
    for (Iterator<String> itString = tsVector.timestampVector.keySet().iterator(); itString.hasNext(); ){
        String str = itString.next();
        // Guardamos los dos timestamp en dos variables para compararlos
        Timestamp ts1 = tsVector.timestampVector.get(str);
        Timestamp ts2 = timestampVector.get(str);
        // Actualizamos el timestamp a través de una comparación para actualizarlo con el mayor
        updateTimestamp(ts1.compare(ts2) > 0 ? ts1 : ts2);
    }
}
```

- **getLast(String node)**

Este método tiene una funcionalidad muy sencilla, simplemente devuelve el valor del node recibido por parametro.

El método es precedido por la palabra clave *synchronized* tal y como hemos indicado anteriormente con el objeto de proteger la integridad de los recursos que utiliza el método.

Imagen con la implementación del método:

```
public synchronized Timestamp getLast(String node){
    //TODO Phase2
    //Devuelve el valor del node
    return this.timestampVector.get(node);
}
```

- **mergeMin(TimestampVector tsVector)**

Este método es parecido y funciona de manera parecida al método explicado anteriormente *updateMax(TimestampVector tsVector)* pero tiene justo el objetivo contrario. Si en el método *updateMax* el objetivo era realizar una comparación y actualizar el parámetro de la clase con el mayor valor obtenido en la comparación, en este caso se actualiza el valor timestamp del parámetro de la clase con el menor valor en la comparación.

El método itera sobre el objeto *TimestampVector* que recibe como parámetro, va obteniendo el timestamp contenido en cada iteración y lo va comparando con el que contiene el parámetro *tsVector* de la clase. Una vez comparados los dos timestamps actualiza el timestamp del parametro de la clase con la de menor valor de las dos comparadas.

El método es precedido por la palabra clave *synchronized* tal y como hemos indicado anteriormente con el objeto de proteger la integridad de los recursos que utiliza el método.

Imagen con la implementación del método:

```
public synchronized void mergeMin(TimestampVector tsVector){
    //TODO Phase2
    // Recorremos el hashmap con un iterator
    for (Iterator<String> iteratorKeys = tsVector.timestampVector.keySet().iterator(); iteratorKeys.hasNext(); ){
        // Guardamos la siguiente key del iterator
        String nextKey = iteratorKeys.next();
        // Guardamos los dos timestamps en una variable para compararlos posteriormente
        Timestamp ts1 = tsVector.timestampVector.get(nextKey);
        Timestamp ts2 = timestampVector.get(nextKey);
        // Actualizamos el timestamp a través de una comparación para actualizarlo con el menor
        updateTimestamp(ts1.compare(ts2) < 0 ? ts1 : ts2);
    }
}
```

- **clone()**

Este método se encarga de devolver un objeto de tipo `TimestampVector` que es un clon del que contiene la clase.

El método itera sobre el parámetro *timestampVector* de la clase con el objetivo de construir un nuevo objeto `TimestampVector` idéntico al alojado como parámetro de la clase.

El método es precedido por la palabra clave *synchronized* tal y como hemos indicado anteriormente con el objeto de proteger la integridad de los recursos que utiliza el método.

Imagen con la implementación del método:

```
public synchronized TimestampVector clone(){
    //TODO Phase2
    TimestampVector timestamp = new TimestampVector(new ArrayList<String>());
    // Recorremos el hashmap con un iterator
    for (Iterator<String> iteratorTimestamp = timestampVector.keySet().iterator(); iteratorTimestamp.hasNext(); ){
        String keyId = iteratorTimestamp.next();
        // Añadimos la keyId al hashmap
        timestamp.timestampVector.put(keyId, timestampVector.get(keyId));
    }
    //Devuelve el timestamp clonado
    return timestamp;
}
```

- **Clase ServerData**

- **addRemoveOperation(MessageOperation message)**

Este método es nuevo y su objetivo es simplificar la adición o la eliminación de operaciones desde los métodos que controlan la comunicación entre los nodos `TSAESessionPartnerSide` y `TSAESessionOriginatorSide` que en algunos casos tendrán como objetivo añadir operaciones y en otras eliminarlas. se encarga de actualizar el timestamp de un vector de tiempos en caso de que el vector de tiempo pasado por parámetro sea posterior al del vector de tiempos de la clase.

El método recibe como parámetro un mensaje de tipo `MessageOperation` que contiene una operación que puede ser de tipo *ADD* o de Tipo *REMOVE*, según cual sea el tipo de operación que recibe realizará la operación que corresponda.

El método es precedido por la palabra clave *synchronized* tal y como hemos indicado anteriormente con el objeto de proteger la integridad de los recursos que utiliza el método.

Imagen con la implementación del método:



```

public synchronized void addRemoveOperation(MessageOperation message){
    //TODO Phase 2
    Operation operation = message.getOperation();
    if (operation != null) {
        // Tenemos dos tipos operaciones ADD y REMOVE
        if(operation.getType() == OperationType.ADD) {
            // Creamos una nueva variable casteada que sera la operacion que se piensa añadir
            AddOperation addOp = (AddOperation) operation;
            // Añadir la operacion al log, en caso de que log.add sea true añadimos recipe
            if(addOp.getRecipe() != null && log.add(addOp)) recipes.add(addOp.getRecipe());
        } else if(operation.getType() == OperationType.REMOVE){
            RemoveOperation removeOp = (RemoveOperation) operation;
            // Añadir la operacion al log, en caso de que log.add sea true borramos recipe
            if(removeOp.getRecipeTitle() != null && log.add(removeOp)) recipes.remove(removeOp.getRecipeTitle());
        }
    }
}

```

- **Clase TSAESessionPartnerSide**

El trabajo realizado sobre la clase ha sido el de completar los puntos que faltaban por implementar dentro de las plantillas. Básicamente he tratado de implementar las partes necesarias para que la clase realice las operaciones necesarias según el tipo de mensaje, es decir, la labor que se quiera realzar en cada momento.

Voy a a realizar una exposición solamente de las partes añadidas al código:

#### Trabajo sobre el tipo MessageAERequest:

Antes de nada necesitamos rellenar con datos las variables que representan al Summary y al Ack para lo cual clonamos los que se contienen en la clase ServerData. Subrayar que usamos synchronized para evitar que otro Thread intervenga y modifique los datos mientras se realiza esta operación.

Posteriormente se obtienen los logs y se crear los objetos necesarios con las operaciones que contienen:

```

// Al usar threads utilizamos synchronized para evitar inconsistencias
synchronized (serverData) {
    localSummary = serverData.getSummary().clone();
    serverData.getAck().update(serverData.getId(), localSummary);
    localAck = serverData.getAck().clone();
}

// Obtenemos los logs
List<Operation> newLogs = serverData.getLog().listNewer(messageAER.getSummary());
// Recorremos todos los logs
for (Operation op : newLogs) {
    out.writeObject(new MessageOperation(op));
}

```

#### Trabajo sobre el tipo MessageOperation:

Añadimos la operación a la lista de *MessageOperations* listOperations.

```

// receive operations
msg = (Message) in.readObject();
LSimLogger.log(Level.TRACE, "[TSAESessionPartnerSide] [session: "+current_session_number+"] received message: "+
while (msg.type() == MessageType.OPERATION){
    //TODO Phase2
    listOperations.add((MessageOperation) msg);

    msg = (Message) in.readObject();
    LSimLogger.log(Level.TRACE, "[TSAESessionPartnerSide] [session: "+current_session_number+"] received message:
}

```



**Trabajo sobre el tipo MessageEndTSAE:**

Se recorren la lista de operaciones, realizando (add o remove) cada una de ellas según el tipo de cada una de ellas llamando al método *addRemoveOperation(Operation)*. Posteriormente se actualizan los timestamps del summary y del ack mediante el método *updateMax*.

```
if (msg.type() == MsgType.END_TSAE){
    // send and "end of TSAE session" message
    msg = new MessageEndTSAE();
    msg.setSessionNumber(current_session_number);
    out.writeObject(msg);
    LSimLogger.log(Level.TRACE, "[TSAESessionPartnerSide] [session: "+current_session_number+"] sent message: "+

    //TODO Phase2
    // Update server's Summary and Acknowledgement vectors
    // Al usar threads utilizamos synchronized para evitar inconsistencias
    synchronized (serverData) {
        // Recorremos la lista de operaciones
        for (MessageOperation operation : listOperations) {
            serverData.addRemoveOperation(operation);
        }

        serverData.getSummary().updateMax(messageAER.getSummary());
        serverData.getAck().updateMax(messageAER.getAck());
        serverData.getLog().purgeLog(serverData.getAck());
    }
}
```

- **Clase TSAESessionOriginatorSide**

Al igual que ocurre con la clase anterior, el trabajo realizado sobre la clase ha sido el de completar los puntos que faltaban por implementar dentro de las plantillas. Básicamente he tratado de implementar las partes necesarias para que la clase realice las operaciones necesarias según el tipo de mensaje, es decir, la labor que se quiera realizar en cada momento.

Voy a a realizar una exposición solamente de las partes añadidas al código:

**Obtener datos del summary y el ack:**

Antes de nada necesitamos rellenar con datos las variables que representan al Summary y al Ack para lo cual clonamos los que se contienen en la clase ServerData. Subrayar que usamos *synchronized* para evitar que otro Thread intervenga y modifique los datos mientras se realiza esta operación.

```
//TODO Phase2
// Declaramos localSummary y localAck
TimestampVector localSummary;
TimestampMatrix localAck;

// Al usar threads utilizamos synchronized para evitar inconsistencias
synchronized (serverData)
{
    localSummary = serverData.getSummary().clone();
    serverData.getAck().update(serverData.getId(), localSummary);
    localAck = serverData.getAck().clone();
}
```

**Trabajo sobre el tipo MessageAerequest:**

Se obtienen los logs y se crean los objetos necesarios con las operaciones que contienen:

```
//TODO Phase2
MessageAerequest messageAER = (MessageAerequest) msg;
List<Operation> newLogs = serverData.getLog().listNewer(messageAER.getSummary());

for (Operation op : newLogs)
{
    out.writeObject(new MessageOperation(op));
    LSimLogger.log(Level.TRACE, "[TSAESessionOriginatorSide] [session: "+current_session_number+"] sent message: "+
}
```

**Trabajo sobre el tipo MessageOperation:**

Añadimos la operación a la lista de *MessageOperations* listOperations.

```
while (msg.type() == MsgType.OPERATION){
    //TODO Phase2
    listOperations.add((MessageOperation) msg);
    msg = (Message) in.readObject();
    LSimLogger.log(Level.TRACE, "[TSAESessionOriginatorSide] [session: "+current_session_number+"] received message:");
}
```

**Trabajo sobre el tipo MessageEndTSAE:**

Se recorren la lista de operaciones, realizando (add o remove) cada una de ellas según el tipo de cada una de ellas llamando al método *addRemoveOperation(Operation)*. Posteriormente se actualizan los timestamps del summary y del ack mediante el método *updateMax*.

```
// receive partner's summary and ack
if (msg.type() == MsgType.AE_REQUEST){
    //TODO Phase2
    MessageAerequest messageAER = (MessageAerequest) msg;
    List<Operation> newLogs = serverData.getLog().listNewer(messageAER.getSummary());

    for (Operation op : newLogs)
    {
        out.writeObject(new MessageOperation(op));
        LSimLogger.log(Level.TRACE, "[TSAESessionOriginatorSide] [session: "+current_session_number+"] sent message:");
    }

    // send and "end of TSAE session" message
    msg = new MessageEndTSAE();
    msg.setSessionNumber(current_session_number);
    out.writeObject(msg);
    LSimLogger.log(Level.TRACE, "[TSAESessionOriginatorSide] [session: "+current_session_number+"] sent message: "+m

    // receive message to inform about the ending of the TSAE session
    msg = (Message) in.readObject();
    LSimLogger.log(Level.TRACE, "[TSAESessionOriginatorSide] [session: "+current_session_number+"] received message:");
    if (msg.type() == MsgType.END_TSAE){

        // Update server's Summary and Acknowledgement vectors
        // Al usar threads utilizamos synchronized para evitar inconsistencias
        synchronized (serverData) {
            for (MessageOperation operation : listOperations) {
                serverData.addRemoveOperation(operation);
            }

            serverData.getSummary().updateMax(messageAER.getSummary());
            serverData.getAck().updateMax(messageAER.getAck());
            serverData.getLog().purgeLog(serverData.getAck());
        }
    }
}
```

**2.2 Limitaciones de la solución aportada**

Como ya he comentado en el primer punto, la solución aportada solamente trata el problema planteado en el enunciado de la práctica “Phase 2”, es decir, solamente implementa la solución necesaria para poder realizar operaciones de agregación de operaciones.

Con respecto a la realización de un proyecto para la implementación del protocolo TSAE, la solución aportada tiene bastantes deficiencias, tanto en cuanto deja sin implementar los métodos correspondientes al borrado de operaciones y a purgar el log.

## 3. Test

### 3.1 Test en entorno local

#### 3.1.1 Test de funcionamiento del proyecto con la implementación del proyecto en Phase1

El punto de inicio en este momento es el proyecto con las implementaciones realizadas en la Phase1.

Se han vuelto a realizar los mismos test antes de iniciar a codificar para la Phase 2:

##### Test1 – recipes\_service.Server

- **Instrucción para el test:** `java -cp ../bin:../lib/* recipes_service.Server --phase1`
- **Descripción:** el test ejecuta el programa java y pone en marcha un servidor el cual permite realizar ciertas operaciones como añadir recetas, obtener el listado de recetas, ver el contenido de los logs y el contenido de los summary.
- **Resultado:** se puede observar el resultado de varias interacciones con la aplicación en la imagen del terminal.

```

kepa@Belkoain:~/Dokumentuak/Keparenak/UOC/SD/PRA1/2021p_SD_practical_assignment/2021p_SD--students/scripts$ (main) java -cp ../bin:../lib/* recipes_service.Server --phase1

Server localhost:9000

Select a command:
1: Add a recipe
3: Show the list of recipes
4: Show the Log
5: Show the summary
0: Exit
1
Enter the title of the recipe to add
recipel title
Enter the recipe to add
recipel

Server localhost:9000

Select a command:
1: Add a recipe
3: Show the list of recipes
4: Show the Log
5: Show the summary
0: Exit
3
Recipes:
{recipel title=[recipel title, recipel, localhost:9000]}

Server localhost:9000

Select a command:
1: Add a recipe
3: Show the list of recipes
4: Show the Log
5: Show the summary
0: Exit
4
Log:
AddOperation [recipe=[recipel title, recipel, localhost:9000], timestamp=localhost:9000: 0]

Server localhost:9000

Select a command:
1: Add a recipe
3: Show the list of recipes
4: Show the Log
5: Show the summary
0: Exit
5
Summary:
localhost:9000: 0

Server localhost:9000

Select a command:
1: Add a recipe
3: Show the list of recipes
4: Show the Log
5: Show the summary
0: Exit

```

- **Conclusión:** la aplicación realiza las operaciones que ofrece el menú de manera correcta, además se observa que implementa el método add de manera correcta, ya que añade acciones al log.

## Test2 – start.sh

- **Instrucción para el test:** ./start.sh 20004 --phase1
- **Descripción:** el test ejecuta el programa java y trata de realizar las operaciones TSAE, al final compara si los ficheros log y summary de los diferentes servidores son iguales
- **Resultado:** se puede observar el resultado de varias interacciones con la aplicación en la imagen del terminal.

```

Fitxategia  Editatu  Ikusi  Terminala  Fitxak  Laguntza
user2: 10
user0: 6
user3: 11
user4: 11

15-03-2021 18:57:18:651 Phase_1 [INFO] : Log:
AddOperation [recipe=[mrtsccko, Content=-mrtsccko, user1], timestamp=user1: 0]
AddOperation [recipe=[fbybqmkf, Content=-fbybqmkf, user1], timestamp=user1: 1]
AddOperation [recipe=[qycxccpi, Content=-qycxccpi, user1], timestamp=user1: 2]
AddOperation [recipe=[gsktxxsh, Content=-gsktxxsh, user1], timestamp=user1: 3]
AddOperation [recipe=[hlvdoryn, Content=-hlvdoryn, user1], timestamp=user1: 4]
AddOperation [recipe=[hauelsfc, Content=-hauelsfc, user1], timestamp=user1: 5]
AddOperation [recipe=[kpetxjq, Content=-kpetxjq, user1], timestamp=user1: 6]
AddOperation [recipe=[yjcauehb, Content=-yjcauehb, user1], timestamp=user1: 7]
AddOperation [recipe=[vgrxismv, Content=-vgrxismv, user2], timestamp=user2: 0]
AddOperation [recipe=[igutjmas, Content=-igutjmas, user2], timestamp=user2: 1]
AddOperation [recipe=[enwxuoaq, Content=-enwxuoaq, user2], timestamp=user2: 2]
AddOperation [recipe=[bycjuiap, Content=-bycjuiap, user2], timestamp=user2: 3]
AddOperation [recipe=[clibbrbl, Content=-clibbrbl, user2], timestamp=user2: 4]
AddOperation [recipe=[rlretsf, Content=-rlretsf, user2], timestamp=user2: 5]
AddOperation [recipe=[vqkrfrdd, Content=-vqkrfrdd, user2], timestamp=user2: 6]
AddOperation [recipe=[srhvdadm, Content=-srhvdadm, user2], timestamp=user2: 7]
AddOperation [recipe=[skotncrs, Content=-skotncrs, user2], timestamp=user2: 8]
AddOperation [recipe=[tlugowex, Content=-tlugowex, user2], timestamp=user2: 9]
AddOperation [recipe=[fdfyucma, Content=-fdfyucma, user2], timestamp=user2: 10]
AddOperation [recipe=[ftyqqryp, Content=-ftyqqryp, user0], timestamp=user0: 0]
AddOperation [recipe=[wfvfcioe, Content=-wfvfcioe, user0], timestamp=user0: 1]
AddOperation [recipe=[ayxlnxfj, Content=-ayxlnxfj, user0], timestamp=user0: 2]
AddOperation [recipe=[jeqkicvn, Content=-jeqkicvn, user0], timestamp=user0: 3]
AddOperation [recipe=[gjtvehkq, Content=-gjtvehkq, user0], timestamp=user0: 4]
AddOperation [recipe=[hnsfjsrm, Content=-hnsfjsrm, user0], timestamp=user0: 5]
AddOperation [recipe=[tustxmqr, Content=-tustxmqr, user0], timestamp=user0: 6]
AddOperation [recipe=[naenayif, Content=-naenayif, user3], timestamp=user3: 0]
AddOperation [recipe=[paqlrlar, Content=-paqlrlar, user3], timestamp=user3: 1]
AddOperation [recipe=[brqcmawf, Content=-brqcmawf, user3], timestamp=user3: 2]
AddOperation [recipe=[vribeukc, Content=-vribeukc, user3], timestamp=user3: 3]
AddOperation [recipe=[lupjjovh, Content=-lupjjovh, user3], timestamp=user3: 4]
AddOperation [recipe=[qbmyxmeo, Content=-qbmyxmeo, user3], timestamp=user3: 5]
AddOperation [recipe=[jvuacyfw, Content=-jvuacyfw, user3], timestamp=user3: 6]
AddOperation [recipe=[booxoevo, Content=-booxoevo, user3], timestamp=user3: 7]
AddOperation [recipe=[cefxgrfy, Content=-cefxgrfy, user3], timestamp=user3: 8]
AddOperation [recipe=[nikvppdg, Content=-nikvppdg, user3], timestamp=user3: 9]
AddOperation [recipe=[phmfqbsu, Content=-phmfqbsu, user3], timestamp=user3: 10]
AddOperation [recipe=[fewjqkrh, Content=-fewjqkrh, user3], timestamp=user3: 11]
AddOperation [recipe=[mtxhcsxk, Content=-mtxhcsxk, user4], timestamp=user4: 0]
AddOperation [recipe=[guimxcvn, Content=-guimxcvn, user4], timestamp=user4: 1]
AddOperation [recipe=[jdpmsrot, Content=-jdpmsrot, user4], timestamp=user4: 2]
AddOperation [recipe=[prrrbyyq, Content=-prrrbyyq, user4], timestamp=user4: 3]
AddOperation [recipe=[nhmswiev, Content=-nhmswiev, user4], timestamp=user4: 4]
AddOperation [recipe=[dtuaotah, Content=-dtuaotah, user4], timestamp=user4: 5]
AddOperation [recipe=[jfyqtts, Content=-jfyqtts, user4], timestamp=user4: 6]
AddOperation [recipe=[xakmtotn, Content=-xakmtotn, user4], timestamp=user4: 7]
AddOperation [recipe=[oogduvgv, Content=-oogduvgv, user4], timestamp=user4: 8]
AddOperation [recipe=[lvckthrq, Content=-lvckthrq, user4], timestamp=user4: 9]
AddOperation [recipe=[pytljyec, Content=-pytljyec, user4], timestamp=user4: 10]
AddOperation [recipe=[roppuwew, Content=-roppuwew, user4], timestamp=user4: 11]

15-03-2021 18:57:18:651 Phase_1 [INFO] : Summary:
user1: 7
user2: 10
user0: 6
user3: 11
user4: 11

Result is correct: Log and Summary are correct

```

- **Conclusión:** la aplicación realiza las operaciones y logra realizar el protocolo TSAE de forma correcta, el mensaje al final de la ejecución “Result is correct: Log and Summary are correct” indica que el protocolo TSAE esta correctamente implementado.

### 3.1.2 Test de funcionamiento del proyecto con la implementación del proyecto en Phase2

El punto de inicio en este momento es el proyecto con las implementaciones realizadas en la Phase1.

Se han vuelto a realizar los mismos test antes de iniciar a codificar para la Phase 2:

#### Test1 – start.sh

- **Instrucción para el test:** `./start.sh 20004 3 --nopurge`
- **Descripción:** el test ejecuta el programa java y trata de realizar las operaciones TSAE, al final compara si los ficheros log y summary de los diferentes servidores son iguales
- **Resultado:** se puede observar el resultado de varias interacciones con la aplicación en la imagen del terminal.

```

erabltzallea@port: ~/Proiektuak/2021p_5D--students--kepa/scripts
Recipes: {cdecqftt=[cdecqftt, Content--cdecqftt, Server@127.0.1.1:35001], chwvypck=[chwvypck, Content--chwvypck, Server@127.0.1.1:35000], cykcvwkp=[cykcvwkp, Content--cykcvwkp, Server@127.0.1.1:35001], l
tayowjb=[ltayowjb, Content--ltayowjb, Server@127.0.1.1:35000], jldsgchq=[jldsgchq, Content--jldsgchq, Server@127.0.1.1:35000], kvctewss=[kvctewss, Content--kvctewss, Server@127.0.1.1:35002], lsptlwuu=[ls
ptlwuu, Content--lsptlwuu, Server@127.0.1.1:35000], mwlfrnk=[mwlfrnk, Content--mwlfrnk, Server@127.0.1.1:35002], twayqyh=[twayqyh, Content--twayqyh, Server@127.0.1.1:35001], vhlcvfnq=[vhlcvfnq, Con
tent--vhlcvfnq, Server@127.0.1.1:35001], yccnoyqe=[yccnoyqe, Content--yccnoyqe, Server@127.0.1.1:35001], ywlvfnx=[ywlvfnx, Content--ywlvfnx, Server@127.0.1.1:35000]}
Log: AddOperation [recipe=[lsptlwuu, Content--lsptlwuu, Server@127.0.1.1:35000], timestamp=Server@127.0.1.1:35000: 0]
AddOperation [recipe=[ltayowjb, Content--ltayowjb, Server@127.0.1.1:35000], timestamp=Server@127.0.1.1:35000: 1]
AddOperation [recipe=[ywlvfnx, Content--ywlvfnx, Server@127.0.1.1:35000], timestamp=Server@127.0.1.1:35000: 2]
AddOperation [recipe=[jldsgchq, Content--jldsgchq, Server@127.0.1.1:35000], timestamp=Server@127.0.1.1:35000: 3]
AddOperation [recipe=[chwvypck, Content--chwvypck, Server@127.0.1.1:35000], timestamp=Server@127.0.1.1:35000: 4]
AddOperation [recipe=[kvctewss, Content--kvctewss, Server@127.0.1.1:35002], timestamp=Server@127.0.1.1:35002: 0]
AddOperation [recipe=[mwlfrnk, Content--mwlfrnk, Server@127.0.1.1:35002], timestamp=Server@127.0.1.1:35002: 1]
AddOperation [recipe=[twayqyh, Content--twayqyh, Server@127.0.1.1:35001], timestamp=Server@127.0.1.1:35001: 0]
AddOperation [recipe=[cdecqftt, Content--cdecqftt, Server@127.0.1.1:35001], timestamp=Server@127.0.1.1:35001: 1]
AddOperation [recipe=[cykcvwkp, Content--cykcvwkp, Server@127.0.1.1:35001], timestamp=Server@127.0.1.1:35001: 2]
AddOperation [recipe=[yccnoyqe, Content--yccnoyqe, Server@127.0.1.1:35001], timestamp=Server@127.0.1.1:35001: 3]
AddOperation [recipe=[vhlcvfnq, Content--vhlcvfnq, Server@127.0.1.1:35001], timestamp=Server@127.0.1.1:35001: 4]

Summary: Server@127.0.1.1:35000: 4
Server@127.0.1.1:35002: 1
Server@127.0.1.1:35001: 4

Ack: Server@127.0.1.1:35000: Server@127.0.1.1:35000: -1000
Server@127.0.1.1:35002: -1000
Server@127.0.1.1:35001: -1000

Server@127.0.1.1:35002: Server@127.0.1.1:35000: -1000
Server@127.0.1.1:35002: -1000
Server@127.0.1.1:35001: -1000

Server@127.0.1.1:35001: Server@127.0.1.1:35000: -1000
Server@127.0.1.1:35002: -1000
Server@127.0.1.1:35001: -1000

Results are equal      Nodes converged at the iteration 0

=====

***** num received results: 2
***** % received results: 66
***** minimal required number of results: 2

erabltzallea@port: ~/Proiektuak/2021p_5D--students--kepa/scripts$

```

- **Conclusión:** la aplicación realiza las operaciones y logra realizar el protocolo TSAE de forma correcta, el mensaje al final de la ejecución “Result are equal” indica que el protocolo TSAE esta correctamente implementado.



### 3.2 Test en DSLab

Una vez realizadas las pruebas en el entorno local, se han realizado los test en la herramienta DSLab (<https://sd.uoc.edu/dslab>). Para realizar el test se ha seguido la serie de instrucciones que indica el enunciado:

1. Crear un grupo para realizar el proyecto, en este caso como no he trabajado en grupo sino de manera individual he creado un grupo con un único componente.

**UOC** **DSLab**

User : Kepa Sarasola Bengoetxea (STUDENT) Classroom: 202\_75\_644\_02 Help Mattermost Logout Laboratorio Sistemas distribuidos aula 2

Principal

#### Editar Group

Code: 911402191

User (1): ksarasola@uoc.edu

User (2):

Check the members of the group before confirming.  
Once created, the composition of the group can't be modified

Discard Confirm

Universitat Oberta de Catalunya - 2020

2. Crear un proyecto y cargar los ficheros que contienen las clases afectadas por la Phase2 en el proyecto:

**UOC** **DSLab**

User : Kepa Sarasola Bengoetxea (STUDENT) Classroom: 202\_75\_644\_02 Help Mattermost Logout Laboratorio Sistemas distribuidos aula 2

Principal Projects Submissions

#### Mostrar PRA2 - Fase2

Profile Files

#### User uploaded files

Path	Name	Logs added	Date	Size
src/recipes_service/	ServerData.java	No	12:53:14 16-05-2021	8.2 KB
src/recipes_service/tsae/sessions/	TSAESessionPartnerSide.java	No	12:53:41 16-05-2021	5.9 KB
src/recipes_service/tsae/sessions/	TSAESessionOriginatorSide.java	No	12:53:35 16-05-2021	6.7 KB
src/recipes_service/tsae/data_structures/	TimestampVector.java	No	12:53:22 16-05-2021	5.8 KB
src/recipes_service/tsae/data_structures/	Log.java	No	12:53:27 16-05-2021	6.5 KB


#### Log files generated by build

Name	Date	Size
log_compile.txt	12:54:30 16-05-2021	1.7 KB
log_build_project.txt	12:54:30 16-05-2021	2.2 KB
log_common_part.txt	12:54:30 16-05-2021	787 bytes

Submit

Universitat Oberta de Catalunya - 2020

3. Crear un experimento que ejecute el proyecto




**DSLab**  
 Laboratorio Sistemas distribuidos aula 2

User : Kepa Sarasola Bengoetxea (STUDENT) Classroom: 202\_75\_644\_02
 [Help](#)
[Mattermost](#)
[Logout](#)

[Principal](#)
[Submissions](#)

### Mostrar Submission

Submission with id "1.220" is running. The duration of the execution will depend on the parameters of the execution. In case of an error, this execution will last a maximum of 22 minutes.
 

**Id** 1.220

**Group** 911402191

**Users** Kepa Sarasola Bengoetxea (ksarasola@uoc.edu)

**Assignment** Lab session 2

**Task** Fase 2

**Project** PRA2 - Fase2


**Submission Date** 13:55:52 16-05-2021

**State** EXECUTING

**Success** Executing

**Logs** [View logs](#)

Universitat Oberta de Catalunya - 2020



**DSLab**  
 Laboratorio Sistemas distribuidos aula 2

User : Kepa Sarasola Bengoetxea (STUDENT) Classroom: 202\_75\_644\_02
 [Help](#)
[Mattermost](#)
[Logout](#)

[Principal](#)
[Submissions](#)

### Mostrar Submission

**Id** 1.220

**Group** 911402191

**Users** Kepa Sarasola Bengoetxea (ksarasola@uoc.edu)

**Assignment** Lab session 2

**Task** Fase 2

**Project** PRA2 - Fase2

**Submission Date** 13:55:52 16-05-2021

**State** FINISHED

**Result Date** 09:06:05 16-05-2021

**Success** ✓

**Result Summary** Correct. Results are equal Nodes converged at the iteration 1

**Result** [View result details](#)

**Logs** [View logs](#)

Universitat Oberta de Catalunya - 2020



## 4. Verificar los resultados

```

Server--Instructor_0@10.20.30.18:35001: -1000
Server--Student_1@10.20.30.13:35000: -1000
Server--Student_5@10.20.30.18:35000: -1000
Server--Student_6@10.20.30.20:35000: -1000
Server--Student_2@10.20.30.17:35000: -1000
Server--Student_3@10.20.30.11:35000: -1000
Server--Instructor_3@10.20.30.10:35000: -1000
Server--Instructor_1@10.20.30.20:35001: -1000
Server--Student_0@10.20.30.22:35000: -1000

Server--Instructor_0@10.20.30.18:35001: Server--Instructor_4@10.20.30.19:35000: -1000
Server--Student_4@10.20.30.21:35000: -1000
Server--Instructor_6@10.20.30.14:35000: -1000
Server--Instructor_2@10.20.30.16:35001: -1000
Server--Student_7@10.20.30.16:35000: -1000
Server--Instructor_5@10.20.30.15:35000: -1000
Server--Instructor_0@10.20.30.18:35001: -1000
Server--Student_1@10.20.30.13:35000: -1000
Server--Student_5@10.20.30.18:35000: -1000
Server--Student_6@10.20.30.20:35000: -1000
Server--Student_2@10.20.30.17:35000: -1000
Server--Student_3@10.20.30.11:35000: -1000
Server--Instructor_3@10.20.30.10:35000: -1000
Server--Instructor_1@10.20.30.20:35001: -1000
Server--Student_0@10.20.30.22:35000: -1000

Server--Student_1@10.20.30.13:35000: Server--Instructor_4@10.20.30.19:35000: -1000
Server--Student_4@10.20.30.21:35000: -1000
Server--Instructor_6@10.20.30.14:35000: -1000
Server--Instructor_2@10.20.30.16:35001: -1000
Server--Student_7@10.20.30.16:35000: -1000
Server--Instructor_5@10.20.30.15:35000: -1000
Server--Instructor_0@10.20.30.18:35001: -1000
Server--Student_1@10.20.30.13:35000: -1000
Server--Student_5@10.20.30.18:35000: -1000
Server--Student_6@10.20.30.20:35000: -1000
Server--Student_2@10.20.30.17:35000: -1000
Server--Student_3@10.20.30.11:35000: -1000
Server--Instructor_3@10.20.30.10:35000: -1000
Server--Instructor_1@10.20.30.20:35001: -1000
Server--Student_0@10.20.30.22:35000: -1000

Server--Student_5@10.20.30.18:35000: Server--Instructor_4@10.20.30.19:35000: -1000
Server--Student_4@10.20.30.21:35000: -1000
Server--Instructor_6@10.20.30.14:35000: -1000
Server--Instructor_2@10.20.30.16:35001: -1000
Server--Student_7@10.20.30.16:35000: -1000
Server--Instructor_5@10.20.30.15:35000: -1000
Server--Instructor_0@10.20.30.18:35001: -1000

```

En esta segunda captura se ve el resultado final del test en DSLab:

```

Server--Student_0@10.20.30.22:35000: -1000

Server--Instructor_1@10.20.30.20:35001:  Server--Instructor_4@10.20.30.19:35000:  -1000
Server--Student_4@10.20.30.21:35000:  -1000
Server--Instructor_6@10.20.30.14:35000:  -1000
Server--Instructor_2@10.20.30.16:35001:  -1000
Server--Student_7@10.20.30.16:35000:  -1000
Server--Instructor_5@10.20.30.15:35000:  -1000
Server--Instructor_0@10.20.30.18:35001:  -1000
Server--Student_1@10.20.30.13:35000:  -1000
Server--Student_5@10.20.30.18:35000:  -1000
Server--Student_6@10.20.30.20:35000:  -1000
Server--Student_2@10.20.30.17:35000:  -1000
Server--Student_3@10.20.30.11:35000:  -1000
Server--Instructor_3@10.20.30.10:35000:  -1000
Server--Instructor_1@10.20.30.20:35001:  -1000
Server--Student_0@10.20.30.22:35000:  -1000

Server--Student_0@10.20.30.22:35000:  Server--Instructor_4@10.20.30.19:35000:  -1000
Server--Student_4@10.20.30.21:35000:  -1000
Server--Instructor_6@10.20.30.14:35000:  -1000
Server--Instructor_2@10.20.30.16:35001:  -1000
Server--Student_7@10.20.30.16:35000:  -1000
Server--Instructor_5@10.20.30.15:35000:  -1000
Server--Instructor_0@10.20.30.18:35001:  -1000
Server--Student_1@10.20.30.13:35000:  -1000
Server--Student_5@10.20.30.18:35000:  -1000
Server--Student_6@10.20.30.20:35000:  -1000
Server--Student_2@10.20.30.17:35000:  -1000
Server--Student_3@10.20.30.11:35000:  -1000
Server--Instructor_3@10.20.30.10:35000:  -1000
Server--Instructor_1@10.20.30.20:35001:  -1000
Server--Student_0@10.20.30.22:35000:  -1000

***** num received results: 11
***** % received results: 73
***** minimal required number of results: 10

```

[< Previous](#)

## 4. Conclusiones Phase2

### 4.1 Dificultades

En mi caso la mayor dificultad ha sido la de entender el enunciado ya que al principio no entendí muy bien cual era el alcance de esta segunda fase “Phase2”. Aunque los métodos que hay que desarrollar en las clases Log y TimestampVector, son bastante claros, la mayor dificultad en mi caso ha sido la de entender que es lo que se esperaba de las clases TSAESessionPartnerSide y TSAESessionOriginatorSide.

## Phase 3

### 1. Introduction

#### 1.1 Estructura de la solución

Hasta esta tercera fase, se nos ha pedido la implementación de una versión reducida del protocolo TSAE, con el objetivo de añadir operaciones sin purgar el log. En esta tercera fase el objetivo es lograr que se realice el purgado del log.

Atendiendo al enunciado, las clases implicadas en esta segunda fase son las clases, *Log*, y *TimestampMatrix*.

#### 1.2 Decisiones tomadas.

##### 1.2.1 Métodos desarrollados para realizar esta práctica

Las clases *Log* y *TimestampVector* tienen los métodos necesarios para realizar esta Phase 3 declarados pero no están desarrollados, por lo que han sido desarrollados para esta Phase3.

- **Clase Log**

La clase *Log* registra las operaciones recibidas por el cliente e implementa un log, la información sobre las operaciones recibidas se guardan en un *ConcurrentHashMap*.

Para desarrollar esta tercera fase, "Phase3", he desarrollado el método *purgeLog(TimestampMatrix ack)*, que se encarga de eliminar las operaciones de los logs que hayan sido reconocidos por todos los nodos participantes y que se hayan registrado en el *ackSummary*.

- **Clase TimestampMatrix**

La clase *TimestampVector* se encarga de gestionar la matrix de ACKnowledge, siendo una matriz que está conformada por vectores de tiempo que pertenecen a los diferentes nodos.

Para realizar esta tercera fase he implementado los siguientes métodos:

- *getTimestampVector(String node)*, devuelve el valor del *node* pasado por parametro dentro de la matriz de vectores de tiempos de la clase.
- *updateMax(TimestampMatrix tsMatrix)*, este método se encarga de actualizar el timestamp de una matriz de vectores de tiempos en caso de que el vector de tiempo de la matriz pasada por parametro sea posterior al del vector de tiempos de la clase.
- *update(String node, TimestampVector tsVector)*, este método sustituye el *TimestampVector* de la matriz actual por el *TimestampVector* pasado por parametro en caso de que exista, en caso contrario lo crea.

- *minTimestampVector()*, este método devuelve para cada nodo el TimestampVector reconocido por todos los nodos.
- *clone()*, este metodo se encarga de devolver un objeto de tipo TimestampMatrix que es un clon del que contiene la clase.
- *equals()*, comprueba si dos objetos de tipo timestampMatrix son iguales devolviendo true o false.

## 2. Implementation

### 2.1 Partes añadidas al código

Como ya he comentado en el punto anterior, los métodos que añadido en el código son las siguientes:

- **Clase Log**
  - **purgeLog(TimestampMatrix ack)**

Este método se encarga de eliminar las operaciones de los logs que hayan sido reconocidos por todos los nodos participantes y que se hayan registrado en el ackSummary.

El método se encarga de recorrer el log mediante un iterator, compara los timestamp de las operaciones que contienen estos y los borra en el caso de que sus timestamps sean menores de los de la última operación reconocida por todos los nodos y que se guarda en el ackSummary.

El método es precedido por la palabra clave *synchronized* tal y como hemos indicado anteriormente con el objeto de proteger la integridad de los recursos que utiliza el método.

Imagen con la implementación del método:

```
public synchronized void purgeLog(TimestampMatrix ack) {
    //Phase 3
    List<String> keys = new Vector<String>(this.log.keySet());
    TimestampVector minTimestamp = ack.minTimestampVector();

    // Usamos el Iterator para recorrer las key del log
    for (Iterator<String> iter = keys.iterator(); iter.hasNext(); ){
        // necesitamos el siguiente key para obtener las operaciones
        String key = iter.next();
        // Recorremos las operaciones que obtenemos mediante el key y un iterator
        for (Iterator<Operation> operationIterator = log.get(key).iterator(); operationIterator.hasNext(); ){
            // Comprobamos que no sea null o que la comparación con el menor timestamp sea menor que 0
            // Es decir que ya haya sido recibido por los demas nodos y que podamos borrarlo
            if (minTimestamp.getLast(key) != null && operationIterator.next().getTimestamp().compare(minTimestamp.getLast(key)) < 0)
                // Borramos la operación
                operationIterator.remove();
        }
    }
}
```

- **Clase TimestampMatrix**
  - **getTimestampVector(String node)**

Este método tiene una funcionalidad muy sencilla, simplemente devuelve el valor del node recibido por parametro.

El método es precedido por la palabra clave `synchronized` tal y como hemos indicado anteriormente con el objeto de proteger la integridad de los recursos que utiliza el método..

Imagen con la implementación del método:

```
TimestampVector getTimestampVector(String node){
    //Phase 3
    //Devuelve el valor del node
    return timestampMatrix.get(node);
}
```

- **updateMax(TimestampMatrix tsMatrix)**

Este método se encarga de actualizar el timestamp de una matriz de vectores de tiempos en caso de que el vector de tiempo de la matriz pasada por parametro sea posterior al del vector de tiempos de la clase.

Para ello recorre el hashmap para ir comparando los timestampVector de cada key, se crean dos variables, una que corresponde al TimestampVector de la clave que se esta evaluando y otra con el TimestampVector del timestampMatrix de la clase. Tras evaluar ambas se queda con la mayor de las dos.

El método es precedido por la palabra clave `synchronized` tal y como hemos indicado anteriormente con el objeto de proteger la integridad de los recursos que utiliza el método..

Imagen con la implementación del método:

```
public synchronized void updateMax(TimestampMatrix tsMatrix){
    //Phase 3
    // Recorremos el hashmap ya que necesitamos ir comparando los timestampVector de cada key
    for (Map.Entry<String, TimestampVector> keys : tsMatrix.timestampMatrix.entrySet()) {
        // Obtenemos la clave y declaramos los dos timestamp para actualizar el maximo
        String key = keys.getKey();
        // Guardamos los dos timestampVector en dos variables para compararlos
        TimestampVector tsv1 = keys.getValue();
        TimestampVector tsv2 = this.timestampMatrix.get(key);
        // Revisamos que no sea null antes de actualizar, ya
        if (tsv2 != null) {
            // Finalmente actualizamos
            tsv2.updateMax(tsv1);
        }
    }
}
```

- **update(String node, TimestampVector tsVector)**

Este método sustituye el TimestampVector de la matriz actual por el TimestampVector pasado por parametro en caso de que exista, en caso contrario lo crea..

Para ello en primer lugar realiza una comprobación para ver que existe un nodo en el timestampMatrix de la clase ya que en caso de exista lo actualizará por el que recibe por parámetro, en caso contrario lo asignará.

El método es precedido por la palabra clave `synchronized` tal y como hemos indicado anteriormente con el objeto de proteger la integridad de los recursos que utiliza el método..

Imagen con la implementación del método

```

public synchronized void update(String node, TimestampVector tsVector) {
    //Phase 3
    // Tenemos que reemplazar el nodo por el nuevo vector si este ya existia
    //En caso contrario debemos de crearlo
    if (this.timestampMatrix.get(node) == null) {
        this.timestampMatrix.put(node, tsVector);
    } else {
        this.timestampMatrix.replace(node, tsVector);
    }
}

```

- **minTimestampVector()**

Este método devuelve para cada nodo el TimestampVector reconocido por todos los nodos.

Para ello recorre el hashmap para ir comparando los timestampVector de cada key, Si el timestamp es null, realiza un clon y en caso contrario llama al método mergeMin.

El método es precedido por la palabra clave synchronized tal y como hemos indicado anteriormente con el objeto de proteger la integridad de los recursos que utiliza el método.

Imagen con la implementación del método

```

public synchronized TimestampVector minTimestampVector(){
    //Phase3
    TimestampVector timestampVector = null;
    // Iteramos los valores del hashmap
    for (TimestampVector eachTimestampVector : this.timestampMatrix.values()) {
        // Si el timestamp es null lo clonamos y en caso contrario llamamos al metodo merge
        if (timestampVector == null)
            timestampVector = eachTimestampVector.clone();
        else
            timestampVector.mergeMin(eachTimestampVector);
    }
    // Finalmente retornamos el timestampVector
    return timestampVector;
}

```

- **clone()**

Este método se encarga de devolver un objeto de tipo TimestampMatrix que es un clon del que contiene la clase.

Para ello recorre el hashmap para ir comparando los timestampVector de cada key, Si el timestamp es null, realiza un clon y en caso contrario llama al método mergeMin.

Imagen con la implementación del método

```

public synchronized TimestampMatrix clone(){
    //Phase 3
    TimestampMatrix timestampCloner = new TimestampMatrix();
    // Iteramos las keys del hashmap y vamos construyendo el clon con los datos de este
    for (Map.Entry<String, TimestampVector> data : timestampMatrix.entrySet()) {
        timestampCloner.timestampMatrix.put(data.getKey(), data.getValue().clone());
    }

    return timestampCloner;
}

```

- **equals(Object obj)**

Este método comprueba si dos objetos de tipo timestampMatrix son iguales devolviendo true o false.

Para ello realiza varias comprobaciones anteriores para ver que no son null o que corresponden al mismo tipo de clase.

Imagen con la implementación del método:

```
public boolean equals(Object obj) {  
    //TODO Phase 3  
    //comprobamos que el objeto que hemos recibido no es null y que es una instancia de la clase Log  
    if ((obj == null) || !(obj instanceof TimestampMatrix)) {  
        //Si no es una instancia de la clase TimestampMatrix, devolvemos false  
        return false;  
    }  
    TimestampMatrix temp = (TimestampMatrix) obj;  
    //Tras chequear que tanto el objeto recibido como el de la clase no son null,  
    //realizamos la comprobación de que ambos objetos sean iguales o no mediante la función equals  
    if ((this.timestampMatrix == null) || (temp.timestampMatrix == null)) {  
        return false;  
    } else {  
        TimestampMatrix newLog = (TimestampMatrix) obj;  
        return newLog.timestampMatrix.equals(timestampMatrix);  
    }  
}
```



### 3. Test

#### 3.1 Test en entorno local

##### 3.1.1 Test de funcionamiento del proyecto con la implementación del proyecto en Phase3

Se ha realizado el test marcado por el enunciado y que validan que se añaden las recetas y que se purgan los logs:

##### Test1 – start.sh

- **Instrucción para el test:** `./start.sh 20004 15 --logResults -path ../results --noremove`
- **Descripción:** el test ejecuta el programa java y trata de realizar las operaciones TSAE, añadiendo las recetas en los nodos participantes, en este caso 15.
- **Resultado:** se puede observar el resultado de varias interacciones con la aplicación en la imagen del terminal.

```

erabiltzailea@port: ~/Prolektuak/SD - Onak/2021p_SD--students--kepa--Phase3/scripts
Server@127.0.1.1:35000: 6
Server@127.0.1.1:35007: -1000
Server@127.0.1.1:35009: 0

Server@127.0.1.1:35000: Server@127.0.1.1:35000: 5
Server@127.0.1.1:35011: 7
Server@127.0.1.1:35010: 3
Server@127.0.1.1:35002: 7
Server@127.0.1.1:35013: 2
Server@127.0.1.1:35001: 4
Server@127.0.1.1:35012: 6
Server@127.0.1.1:35004: 3
Server@127.0.1.1:35003: 4
Server@127.0.1.1:35014: 5
Server@127.0.1.1:35006: 3
Server@127.0.1.1:35005: 4
Server@127.0.1.1:35008: 6
Server@127.0.1.1:35007: -1000
Server@127.0.1.1:35009: 0

Server@127.0.1.1:35007: Server@127.0.1.1:35000: -1000
Server@127.0.1.1:35011: 0
Server@127.0.1.1:35010: 1
Server@127.0.1.1:35002: -1000
Server@127.0.1.1:35013: 1
Server@127.0.1.1:35001: 1000
Server@127.0.1.1:35012: 2
Server@127.0.1.1:35004: -1000
Server@127.0.1.1:35003: -1000
Server@127.0.1.1:35014: 2
Server@127.0.1.1:35006: -1000
Server@127.0.1.1:35005: 1
Server@127.0.1.1:35008: 1
Server@127.0.1.1:35007: -1000
Server@127.0.1.1:35009: 0

Server@127.0.1.1:35009: Server@127.0.1.1:35000: 4
Server@127.0.1.1:35011: 6
Server@127.0.1.1:35010: 3
Server@127.0.1.1:35002: 6
Server@127.0.1.1:35013: 1
Server@127.0.1.1:35001: 4
Server@127.0.1.1:35012: 5
Server@127.0.1.1:35004: 2
Server@127.0.1.1:35003: 3
Server@127.0.1.1:35014: 5
Server@127.0.1.1:35006: 2
Server@127.0.1.1:35005: 3
Server@127.0.1.1:35008: 4
Server@127.0.1.1:35007: -1000
Server@127.0.1.1:35009: 0

10-05-2021 19:24:54:432 TSAE_Server@127.0.1.1:35012 [INFO] : END
erabiltzailea@port: ~/Prolektuak/SD - Onak/2021p_SD--students--kepa--Phase3/scripts$

```

**Conclusión:** la aplicación realiza las operaciones y logra realizar el protocolo TSAE de forma correcta, además realiza el purgado de los logs con éxito.

### 3.2 Test en DSLab

Una vez realizadas las pruebas en el entorno local, se han realizado los test en la herramienta DSLab (<https://sd.uoc.edu/dslab>). Para realizar el test se ha seguido la serie de instrucciones que indica el enunciado:

1. Crear un proyecto y cargar los ficheros que contienen las clases afectadas por la Phase3 en el proyecto, al igual que en la fase 2:

Path	Name	Date	Size	Show	Remove
src/recipes_service/	ServerData.java	19:46:50 16-05-2021	8.2 KB		
src/recipes_service/tsae/data_structures/	TimestampMatrix.java	19:47:00 16-05-2021	5.5 KB		
src/recipes_service/tsae/data_structures/	TimestampVector.java	19:47:05 16-05-2021	5.8 KB		
src/recipes_service/tsae/data_structures/	Log.java	19:47:16 16-05-2021	7.4 KB		
src/recipes_service/tsae/sessions/	TSAESessionPartnerSide.java	19:47:30 16-05-2021	5.9 KB		
src/recipes_service/tsae/sessions/	TSAESessionOriginatorSide.java	19:47:40 16-05-2021	6.7 KB		

2. Crear un experimento que ejecute el proyecto

Path	Name	Logs added	Date	Size
src/recipes_service/tsae/data_structures/	TimestampVector.java	No	19:47:05 16-05-2021	5.8 KB
src/recipes_service/	ServerData.java	No	19:46:50 16-05-2021	8.2 KB
src/recipes_service/tsae/sessions/	TSAESessionOriginatorSide.java	No	19:47:40 16-05-2021	6.7 KB
src/recipes_service/tsae/data_structures/	TimestampMatrix.java	No	19:47:00 16-05-2021	5.5 KB
src/recipes_service/tsae/data_structures/	Log.java	No	19:47:16 16-05-2021	7.4 KB
src/recipes_service/tsae/sessions/	TSAESessionPartnerSide.java	No	19:47:30 16-05-2021	5.9 KB

Name	Date	Size
log_compile.txt	19:49:51 16-05-2021	1.7 KB
log_build_project.txt	19:49:51 16-05-2021	2.2 KB
log_common_part.txt	19:49:51 16-05-2021	868 bytes

[Submit](#)

### 3. Verificar los resultados

```

Server--Instructor_0@10.20.30.15:35002: 8
Server--Instructor_6@10.20.30.21:35001: 11
Server--Student_4@10.20.30.19:35001: 7
Server--Instructor_1@10.20.30.14:35002: 10

Server--Student_4@10.20.30.19:35001:  Server--Instructor_5@10.20.30.11:35001: 5
Server--Student_1@10.20.30.20:35002: 8
Server--Student_0@10.20.30.18:35001: 2
Server--Student_5@10.20.30.15:35001: 6
Server--Instructor_3@10.20.30.13:35001: 4
Server--Student_2@10.20.30.16:35002: 5
Server--Student_7@10.20.30.22:35001: 7
Server--Student_6@10.20.30.14:35001: 3
Server--Student_3@10.20.30.10:35002: 5
Server--Instructor_4@10.20.30.17:35001: 3
Server--Instructor_2@10.20.30.22:35002: 8
Server--Instructor_0@10.20.30.15:35002: 8
Server--Instructor_6@10.20.30.21:35001: 10
Server--Student_4@10.20.30.19:35001: 7
Server--Instructor_1@10.20.30.14:35002: 10

Server--Instructor_1@10.20.30.14:35002:  Server--Instructor_5@10.20.30.11:35001: 6
Server--Student_1@10.20.30.20:35002: 8
Server--Student_0@10.20.30.18:35001: 3
Server--Student_5@10.20.30.15:35001: 8
Server--Instructor_3@10.20.30.13:35001: 5
Server--Student_2@10.20.30.16:35002: 7
Server--Student_7@10.20.30.22:35001: 9
Server--Student_6@10.20.30.14:35001: 5
Server--Student_3@10.20.30.10:35002: 5
Server--Instructor_4@10.20.30.17:35001: 5
Server--Instructor_2@10.20.30.22:35002: 11
Server--Instructor_0@10.20.30.15:35002: 8
Server--Instructor_6@10.20.30.21:35001: 11
Server--Student_4@10.20.30.19:35001: 7
Server--Instructor_1@10.20.30.14:35002: 10

***** num received results: 13
***** % received results: 86
***** minimal required number of results: 10

```

[< Previous](#)

## 4. Conclusiones Phase3

### 4.1 Dificultades

En esta fase, el reto ha sido realizar el desarrollo de la clase `TimestampMatrix` que por otra parte me ha parecido muy similar a la clase `TimestampVector`. Las dos clases aunque no son exactamente iguales, comparten un modo de funcionamiento muy similar, pero existen diferencias: mientras que la segunda trabaja principalmente con el objeto `TimestampVector`, la primera realiza acciones muy parecidas sobre el objeto `TimestampMatrix` que incluye también el objeto `TimestampVector`.

Por otra parte quiero señalar que he tenido que modificar el método `add` de la clase `Log` y el método `equals` de la clase `TimestampVector`, posteriormente a su utilización con éxito en Phase1 y Phase2. Este detalle ha supuesto un quebradero de cabeza importante ya que en principio suponía que al estar ambas fases superadas, todos los métodos estarían implementados de manera correcta y que funcionarían sin problemas en las fases posteriores.

## Phase 4

### 1. Introduction

#### 1.1 Estructura de la solución

Esta cuarta fase tiene como objetivo añadir la operación *removeRecipe(String recipeTitle)*, que se encarga de lanzar una operación de tipo *RemoveOperation*, con el objetivo de crear una nueva sesión para borrar una receta en todos los nodos del sistema implicados.

Atendiendo al enunciado, la clase implicada en esta cuarta fase es la clase *ServerData* y más concretamente el método *removeRecipe(String recipeTitle)*. En esta fase de la práctica se incorpora el trabajo con un tipo de objeto nuevo que no se ha usado hasta ahora *tombstones*.

#### 1.2 Decisiones tomadas.

##### 1.2.1 Métodos desarrollados para realizar esta práctica

Para la realización de esta Phase 4, se ha desarrollado el método *removeRecipe(String recipeTitle)* de la clase *ServerData*. Este método trabaja con el objeto *RemoveOperation*, como consecuencia he tenido que crear un método *removeOperation(RemoveOperation removeOperation)* que se encarga de borrar el objeto que recibe y de añadir al objeto *tombstone* la operación que se ha borrado.

El objeto *tombstone* se incorpora en esta fase de la práctica como valedor o registro de la marca de tiempo de las operaciones borradas en el transcurso de la operación TSAE.

Dado que la implementación en cuanto al objeto *tombstone* es diferente si de lo que tratamos es de añadir una operación o de borrarla (en cuanto al objeto *tombstone* se realiza la operación contraria), he dividido en dos métodos el método *addRemoveOperation(MessageOperation message)* que incorpore en la Phase 2, dado que los quehaceres de los dos funciones de añadir y de borrar han variado considerablemente al tener que tener en cuenta el objeto *tombstone*.

## 2. Implementation

### 2.1 Partes añadidas al código

Como ya he comentado en el punto anterior, se ha desarrollado el método `removeRecipe` de la clase `ServerData`:

- **Clase `ServerData`**
  - **`removeRecipe(String recipeTitle)`**

El objetivo de este método es el de borrar una receta del nodo que recibe la orden y de propagar esta operación en todos los nodos del sistema, creando una nueva instancia del tipo `RemoveOperation`. Posteriormente actualiza tanto los los como los summary del sistema. Imagen con la implementación del método:

```
public synchronized void removeRecipe(String recipeTitle){
    //TODO Phase 4.1
    // Declaramos e inicializamos variables que necesitaremos
    Timestamp timestamp = nextTimestamp();
    Recipe recipe = this.recipes.get(recipeTitle);
    // Creamos una nueva operación de tipo RemoveOperation
    Operation operation = new RemoveOperation(recipeTitle, recipe.getTimestamp(), timestamp);
    // Añadimos la operation al log
    this.log.add(operation);
    // Actualizamos el timestamp
    this.summary.updateTimestamp(timestamp);
    // Borramos la receta
    this.recipes.remove(recipeTitle);
}
```

- **`addOperation(AddOperation addOperation)`**

Tal y como se ha comentado en el primer punto, he eliminado la función genérica que añade o elimina una operación dividiendo esta en dos métodos específicos que son usados por las clases `TSAESessionOriginatorSide` y `TSAESessionPartnerSide`.

En este caso el método recibe un objeto `AddOperation` y comprueba que la marca de tiempo de esta no se haya sido registrada en el tombstones, es decir, que no se haya borrado ya anteriormente, y en caso de que esta marca coincida, se encarga de borrarla.

```
public synchronized void addOperation(AddOperation addOperation){
    // Palabra reservada necesaria para el uso de threads
    synchronized(tombstones) {
        // Obtenemos la receta pasada por parametro y su correspondiente titulo
        Recipe currentRecipe = addOperation.getRecipe();
        String titleRecipe = currentRecipe.getTitle();
        // Evaluamos si podemos añadir
        if (this.log.add(addOperation)){
            // Añadimos la receta al tree map de recetas donde se conservara su orden
            this.recipes.add(addOperation.getRecipe());
            // Evaluamos si el tombstone contiene el timestamp de la receta para removerla posteriormente
            if(tombstones.contains(addOperation.getTimestamp())) {
                this.recipes.remove(titleRecipe);
                this.tombstones.remove(addOperation.getTimestamp());
            }
        }
    }
}
```

- **removeOperation(RemoveOperation removeOperation)**

Tal y como se ha comentado en el primer punto, he eliminado la función genérica que añade o elimina una operación dividiendo esta en dos métodos específicos que son usados por las clases TSAESessionOriginatorSide y TSAESessionPartnerSide.

En este caso el método recibe un objeto RemoveOperation y comprueba que la marca de tiempo de esta no se haya sido registrada en el tomstones, es decir, que no se haya borrado ya anteriormente, y en caso de que esta marca coincida, se encarga de borrarla, en caso de que no se encuentre esta marca de tiempo, el método una vez eliminada la receta añade al objeto tomstone la marca de tiempo del objeto eliminado.

```
public synchronized void removeOperation(RemoveOperation removeOperation){
    //TODO Phase 4.1
    // synchronized para regular el uso de threads sobre el conjunto de datos y operaciones
    synchronized(tombstones) {
        // Comprobar si podemos añadir la operación al log
        if(this.log.add(removeOperation)) {
            // Validamos que el título de la receta no sea null
            if (this.recipes.get(removeOperation.getRecipeTitle()) != null) {
                // Eliminamos la receta del treemap (se conservará el orden)
                this.recipes.remove(removeOperation.getRecipeTitle());
                // Si el tombstone contiene el timestamp de la receta procedemos a eliminarlo
                if(tombstones.contains(removeOperation.getRecipeTimestamp())){
                    // Finalmente eliminamos la operación
                    this.tombstones.remove(removeOperation.getTimestamp());
                }
            }else {
                // Si el tombstone no contiene el timestamp de la receta procedemos a añadirlo
                if(!tombstones.contains(removeOperation.getRecipeTimestamp())) {
                    this.tombstones.add(removeOperation.getRecipeTimestamp());
                }
            }
        }
    }
}
```

- **Clases TSAESessionOriginatorSide y TSAESessionPartnerSide**

El dividir el método AddRemoveOperation de la clase ServerData en dos métodos específicos que se encargan de añadir y de borrar operaciones implica que las dos clases encargadas de realizar las operaciones en la sesión TSAE hayan sido modificadas ligeramente para que la solución pueda ser efectiva.

El cambio realizado es muy pequeño e implica evaluar el tipo de operación a realizar antes de realizar la llamada a la función correspondiente.

Imagen de la modificación realizada en la clase TSAESessionOriginatorSide:

```
synchronized (serverData) {
    //Phase 4.1
    for (MessageOperation operation : listOperations) {
        // Revisamos si el tipo de operación es de eliminar o añadir
        if (operation.getOperation().getType() == OperationType.ADD) {
            // Añadimos la operación
            serverData.addOperation((AddOperation) operation.getOperation());
        } else {
            // Eliminamos la operación
            serverData.removeOperation((RemoveOperation) operation.getOperation());
        }
    }

    serverData.getSummary().updateMax(messageAER.getSummary());
    serverData.getAck().updateMax(messageAER.getAck());
    serverData.getLog().purgeLog(serverData.getAck());
}
```

Imagen de la modificación realizada en la clase TSAESessionPartnerSide:

```
synchronized (serverData) {  
    //Phase 4.1  
    // Recorremos la lista de operaciones  
    for (MessageOperation operation : listOperations) {  
        // Revisamos si la operacion a realizar es de tipo añadir o eliminar  
        if (operation.getOperation().getType() == OperationType.ADD) {  
            // Añadimos la operacion  
            serverData.addOperation((AddOperation) operation.getOperation());  
        } else {  
            // Removemos la operacion  
            serverData.removeOperation((RemoveOperation) operation.getOperation());  
        }  
    }  
  
    serverData.getSummary().updateMax(messageAER.getSummary());  
    serverData.getAck().updateMax(messageAER.getAck());  
    serverData.getLog().purgeLog(serverData.getAck());  
}
```



## 3. Test

### 3.1 Test en entorno local

#### 3.1.1 Test de funcionamiento del proyecto con la implementación del proyecto en Phase4

Se ha realizado el test marcado por el enunciado y que validan que se añaden las recetas y que se purgan los logs:

##### Test1 – start.sh

- **Instrucción para el test:** `./start.sh 20004 15 --logResults -path ./results`
- **Descripción:** el test ejecuta el programa java y trata de realizar las operaciones TSAE, añadiendo las recetas en los nodos participantes, en este caso 15. Entre las operaciones que se realizan están las de borrado de las recetas
- **Resultado:** se puede observar el resultado de varias interacciones con la aplicación en la imagen del terminal.

```

Server0127.0.1.1:35005: Server0127.0.1.1:35000: 2
Server0127.0.1.1:35011: 4
Server0127.0.1.1:35010: 1
Server0127.0.1.1:35002: 3
Server0127.0.1.1:35012: 4
Server0127.0.1.1:35001: 4
Server0127.0.1.1:35012: 3
Server0127.0.1.1:35004: 5
Server0127.0.1.1:35003: 1
Server0127.0.1.1:35014: 5
Server0127.0.1.1:35006: 2
Server0127.0.1.1:35005: 3
Server0127.0.1.1:35006: 2
Server0127.0.1.1:35007: 3
Server0127.0.1.1:35009: 1
Server0127.0.1.1:35008: Server0127.0.1.1:35000: 2
Server0127.0.1.1:35011: 3
Server0127.0.1.1:35010: 1
Server0127.0.1.1:35002: 2
Server0127.0.1.1:35013: 2
Server0127.0.1.1:35001: 4
Server0127.0.1.1:35012: 1
Server0127.0.1.1:35004: 4
Server0127.0.1.1:35003: 1
Server0127.0.1.1:35014: 2
Server0127.0.1.1:35006: 1
Server0127.0.1.1:35005: 2
Server0127.0.1.1:35008: 2
Server0127.0.1.1:35007: 1
Server0127.0.1.1:35009: 1
Server0127.0.1.1:35007: Server0127.0.1.1:35000: 2
Server0127.0.1.1:35011: 4
Server0127.0.1.1:35010: 1
Server0127.0.1.1:35002: 3
Server0127.0.1.1:35013: 4
Server0127.0.1.1:35001: 4
Server0127.0.1.1:35012: 3
Server0127.0.1.1:35004: 5
Server0127.0.1.1:35003: 1
Server0127.0.1.1:35014: 5
Server0127.0.1.1:35006: 2
Server0127.0.1.1:35005: 3
Server0127.0.1.1:35008: 2
Server0127.0.1.1:35007: 3
Server0127.0.1.1:35009: 1
Server0127.0.1.1:35009: Server0127.0.1.1:35000: 2
Server0127.0.1.1:35011: 4
Server0127.0.1.1:35010: 1
Server0127.0.1.1:35002: 3
Server0127.0.1.1:35013: 4
Server0127.0.1.1:35001: 4
Server0127.0.1.1:35012: 3
Server0127.0.1.1:35004: 5
Server0127.0.1.1:35003: 1
Server0127.0.1.1:35014: 5
Server0127.0.1.1:35006: 2
Server0127.0.1.1:35005: 3
Server0127.0.1.1:35008: 2
Server0127.0.1.1:35007: 3
Server0127.0.1.1:35009: 1
24-05-2021 18:15:13:216 TSAE_Server@127.0.1.1:35012 [INFO] : END
kepa@belkoain:~/Dokumentuak/Keperena/UOC/SD/PRA2/PR - Phases/2021p_SD--students--kepa--Phase4/scripts$ (main)

```

**Conclusión:** la aplicación realiza las operaciones y logra realizar el protocolo TSAE de forma correcta, además del purgado de los logs, logra realizar la eliminación de las recetas en todos los nodos de manera exitosa.

### 3.2 Test en DSLab

Una vez realizadas las pruebas en el entorno local, se han realizado los test en la herramienta DSLab (<https://sd.uoc.edu/dslab>). Para realizar el test se ha seguido la serie de instrucciones que indica el enunciado:

1. Crear un proyecto y cargar las ficheros que contienen las clases afectadas por la Phase4 en el proyecto, al igual que en la fase 2 y 3:
2. Crear un experimento que ejecute el proyecto
3. Verificar los resultados

The screenshot displays the DSLab web interface. At the top, there is a header with the UOC logo and the text 'DSLab'. Below the header, a navigation bar shows the user 'Kepa Sarasola Bengoetxea (STUDENT)' and the classroom '202\_75\_644\_02'. The main content area is titled 'Mostrar Submission' and shows details for a submission with ID 1.725. The submission is for a task named 'Fase 4' under the project 'PRA2 - Fase 4 - 0526'. The submission date is '17:37:41 26-05-2021' and the state is 'FINISHED'. The result date is '23:47:58 26-05-2021'. The submission was successful, as indicated by a checkmark. The result summary states 'Correct. Results are equal Nodes converged at the iteration 1'. There are links to 'View result details' and 'View logs'. A footer at the bottom of the page reads 'Universitat Oberta de Catalunya - 2020'.

UOC DSLab

User: Kepa Sarasola Bengoetxea (STUDENT) Classroom: 202\_75\_644\_02 Help Mattermost Logout Laboratorio Sistemas distribuidos aula 2

Principal Submissions

Mostrar Submission

Submission with id "1.725" is running. The duration of the execution will depend on the parameters of the execution. In case of an error, this execution will last a maximum of 22 minutes.

Id 1.725

Group 911402191

Users Kepa Sarasola Bengoetxea (ksarasola@uoc.edu)

Assignment Lab session 2

Task Fase 4

Project PRA2 - Fase 4 - 0526

Submission Date 17:37:41 26-05-2021

State FINISHED

Result Date 23:47:58 26-05-2021

Success ✓

Result Summary Correct. Results are equal Nodes converged at the iteration 1

Result View result details

Logs View logs

Universitat Oberta de Catalunya - 2020

## 4. Conclusiones Phase4.1

### 4.1 Dificultades

En esta fase, la dificultad ha estado en descubrir que hay que lanzar una operación del tipo `RemoveOperation` con el objetivo de que todo el sistema sea consistente y se llegue al resultado esperado, es decir, que se logre eliminar las recetas no solo en el nodo que recibe la orden sino en todos los nodos del sistema.

Además se ha incorporado el trabajo a realizar con el objeto *tombstones* cuyo objetivo es el de llevar un registro de las marcas de tiempo de los objetos que se han ido eliminando a lo largo de la sesión TSAE. El tener en cuenta este objeto ha tenido como consecuencia la reestructuración y reescritura del código de la clase `ServerData` que se encarga de realizar las operaciones de añadir o eliminar un objeto, separando este método en dos nuevos métodos específicos, uno para añadir objetos y otro para eliminarlos y que en cuanto al tratamiento del objeto *tombstones* se refiere tienen diferentes que hacer.

Para completar esta fase, también he tenido que modificar el método *equals(Object obj)* de la clase `Log`, desarrollado anteriormente en la Phase1 y que en las anteriores fases evaluadas no ha dado ningún problema al realizar los test, tanto en local como en DSLab.