



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Servicios y Aplicaciones Distribuidas Seminarios

Josué Gutiérrez Durán
Eduardo de la Paz Gonzalez
Sergio García de la Iglesia

16 de enero de 2019

Índice

1. Introducción	3
2. Seminario 4	3
2.1. Actividad 1	3
2.2. Actividad 2	4
2.3. Actividad 3	8
3. Seminario 5	10
3.1. Actividad 1	10
3.2. Actividad 2	11
4. Seminario 6 - 7	13
4.1. Actividad 1	13
4.2. Actividad 2	13
5. Seminario 8	16
5.1. Actividad 1	16
5.2. Actividad 2	16
5.3. Actividad 3	16
5.4. Actividad 4	17
5.5. Actividad 5	17
5.6. Actividad 6	18
5.7. Actividad 7	18
6. Seminario 9	19
6.1. Actividad 1	19
6.2. Actividad 2	20
6.3. Actividad 3	21
6.4. Actividad 4	22
7. Seminario 10	23
7.1. Actividad 1	23

2.2. Actividad 2

Se realizaron las siguientes modificaciones:

Broadcast a message to connected users when someone connects or disconnects

En el fichero index.js se añadió el siguiente código:

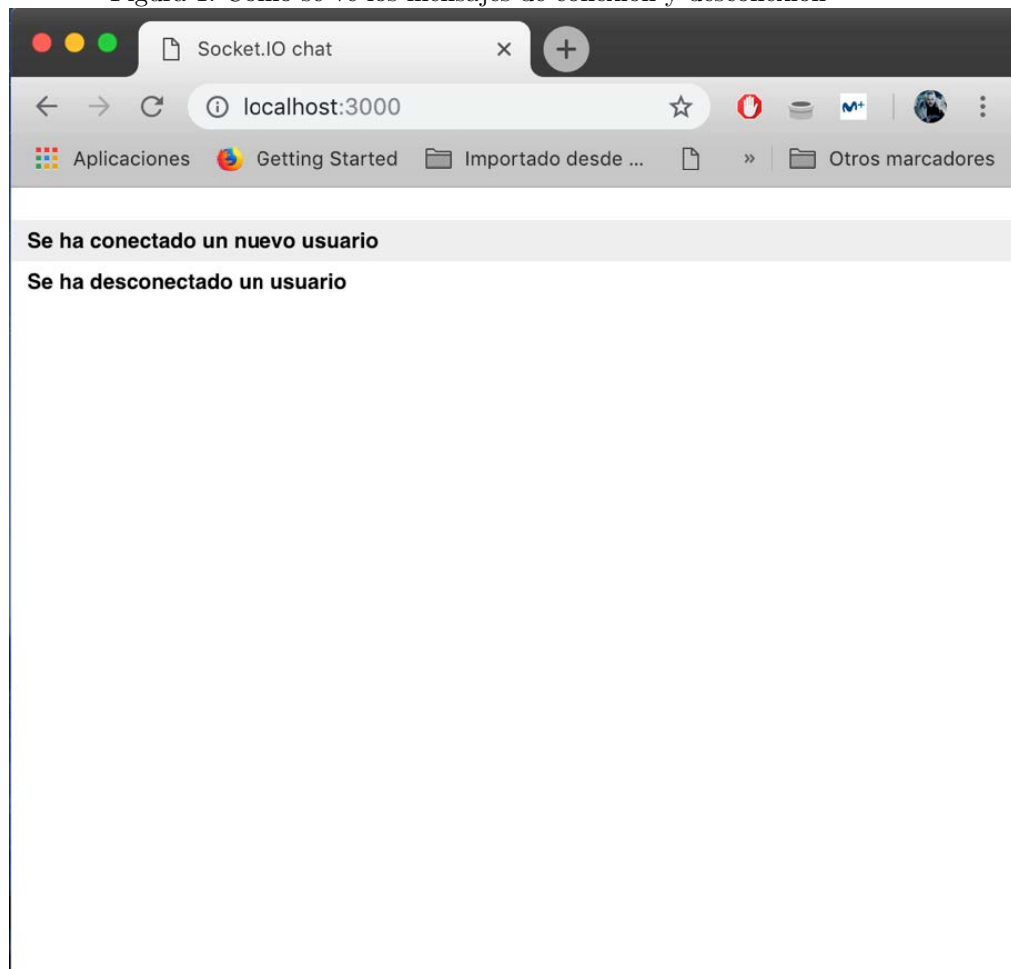
```
io.on('connection', (socket) => {  
  console.log('a user connected');  
  socket.broadcast.emit('user connection', 'Se ha conectado un nuevo usuario');  
  
  socket.on('disconnect', () => {  
    console.log('user disconnected');  
    io.sockets.emit('user connection', 'Se ha desconectado un usuario');  
  });  
});
```

En caso de detectar una conexión, se manda al resto de sockets la notificación. En caso de que sea una desconexión, se procede de la misma manera.

En el fichero index.html se añadió el código necesario para manejar la notificación, de tal forma que se añade al elemento de los mensajes uno con la información de si se ha conectado o desconectado un usuario.

```
socket.on('user connection', (info) => {  
  var line = $('<strong></strong>').text(info);  
  var li = $('<li></li>').append(line);  
  
  $('#messages').append(li);  
});
```

Figura 1: Como se ve los mensajes de conexión y desconexión



Add support for nicknames

En el fichero index.js se añadió el siguiente código:

```
socket.on('chat message', (msg) => {  
  socket.broadcast.emit('chat message', msg);  
});
```

Simplemente una vez recibe el servidor una notificación de un mensaje, la propaga al resto de usuarios.

En el fichero index.html se añadió el código necesario para poder manipular la posibilidad de establecer un nick.

Cuando un usuario inicia la aplicación, le aparece un mensaje en el cual se le pide que introduzca un nick. Una vez se introduce el nick, el usuario ya puede utilizar el chat.

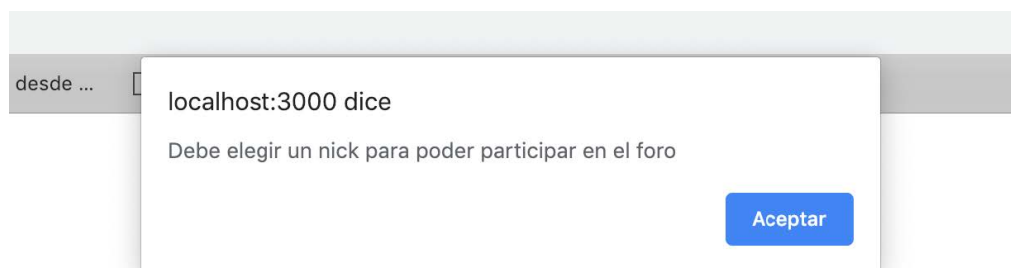
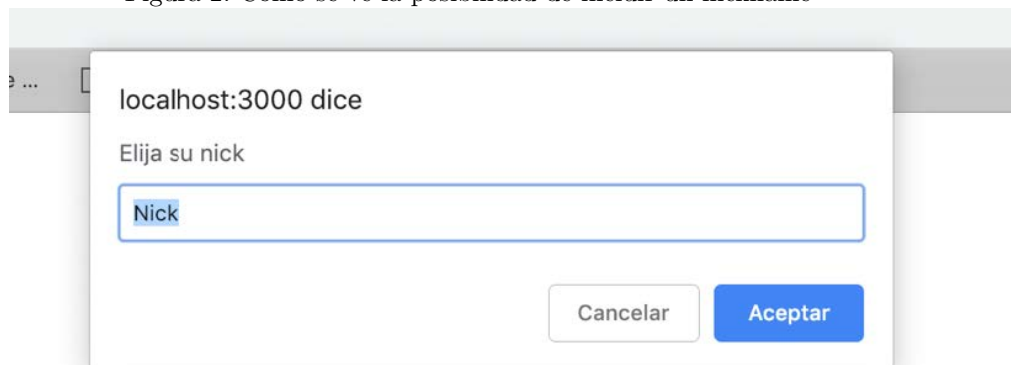


Figura 2: Como se ve la posibilidad de incluir un nickname



Don't send the same message to the user that sent it himself. Instead, append the message directly as soon as he presses enter.

En vez de mandar el mensaje al propio usuario también, lo que se hace es añadir el mensaje directamente. (index.html)

```
var form = function() {  
  // Lo copiamos directamente para él  
  var line = $('#<strong></strong>').text($('#nick').val() + ': ');  
  var li = $('#<li></li>').append(line).append($('#m').val());  
  $('#messages').append(li);  
  
  var message = {nick: $('#nick').val(), m: $('#m').val()}  
  
  socket.emit('chat message', message);  
  $('#m').val('');  
  
  return false;  
};
```

```
$('#form').on('submit', form);
```

Figura 3: Ejemplo de como se ve el nickname con los mensajes

Sergio: HooLaa LooKoo

Pepe: Qué pasó xaval

2.3. Actividad 3

Add “{user} is typing” functionality.

En el fichero index.js se añadió el siguiente código, mediante el cual si al servidor le llegaba una notificación de que un usuario estaba escribiendo la propagaba al resto de usuarios.

```
socket.on('is typing', (nick) => {  
  socket.broadcast.emit('is typing', nick + ' is typing...');  
});
```

En el fichero index.html se añadió un nuevo elemento (*status*) en la parte superior la cual mostrará si algún usuario se encuentra escribiendo.

```
<div>  
  <p id='statusContainer'><i text-align: center; id='status'></i></p>  
</div>
```

Junto a esto, se añadió el código gracias al cual se comunica con el servidor para especificar que se está escribiendo:

```
var keypressFunction = function () {  
  var nick = $('#nick').val();  
  socket.emit('is typing', nick);  
}  
  
/**  
 * Eventos  
 **/  
  
$('#form').on('submit', form);  
$('#m').on('keypress', keypressFunction);
```

Y en caso de que seas otro usuario que va a recibir la notificación, se añadió el código que actualiza el elemento *status*.


```
socket.on('is typing', (info) => {
  $('#status').text(info);
});
```

Una vez se envía el mensaje y se actualiza cada uno de los clientes con el respectivo mensaje, también se actualiza el elemento **status** poniéndolo vacío.

Figura 4: Cómo se ve la funcionalidad is typing

Sergio is typing...

Show who's online.

Para poder mostrar quien se encuentra online en el chat, se creo un array en el servidor el cual almacena el Nick de los usuarios que se encuentran activos. Mediante sockets, se manda esta lista al html para que muestre el nick de los usuarios conectados.

Figura 5: Cómo se ve la funcionalidad de quién está online

Están conectados: Nick

Add private message.

Para poder implementar los mensajes privados, se añadió un nuevo campo justo encima del campo del mensaje en el cual se puede especificar el nombre del usuario al que se le quiere enviar un mensaje privado. Es importante señalar, que este campo está acompañado de un checkbox que es el que determina si va a ser un mensaje privado o no (en el caso que no esté activado, se desactiva la posibilidad de el nick de la persona a la que se le quiere enviar el mensaje y se determina que el mensaje es público).

Figura 6: Cómo se ve la funcionalidad de mensaje privado

<p><i>Están conectados: Sergio, Edu</i></p> <p>Se ha conectado Edu</p> <p>Sergio (privado a Edu): Hola Edu</p> <p>Edu (privado): Hola amigo</p> <p>Edu: Señores qué tal</p>	<p><i>Están conectados: Sergio, Edu</i></p> <p>Se ha conectado Sergio</p> <p>Sergio (privado): Hola Edu</p> <p>Edu (privado a Sergio): Hola amigo</p> <p>Edu: Señores qué tal</p>
---	---

3. Seminario 5

3.1. Actividad 1

En nuestra solución como bien comenta el enunciado, contamos con 3 tipos de elementos: clientes, brokers y workers. Los workers deberán estar conectados entre sí para poder propagar la información y tenerla actualizada. Los brokers sirven de enlace entre clientes y workers. Por último, tenemos los clientes, los cuales solicitan información al sistema.

Estos elementos se interconectan de la siguiente manera:

Clientes Utilizarán sockets ZMQ del tipo request/responder para conectarse con los brokers. Los clientes serán el socket request que harán peticiones esperando la respuesta de los brokers.

Brokers Utilizarán sockets ZMQ del tipo req/res para conectarse con los clientes. Los brokers serán el socket responder que responderá las peticiones que les efectúen los clientes.

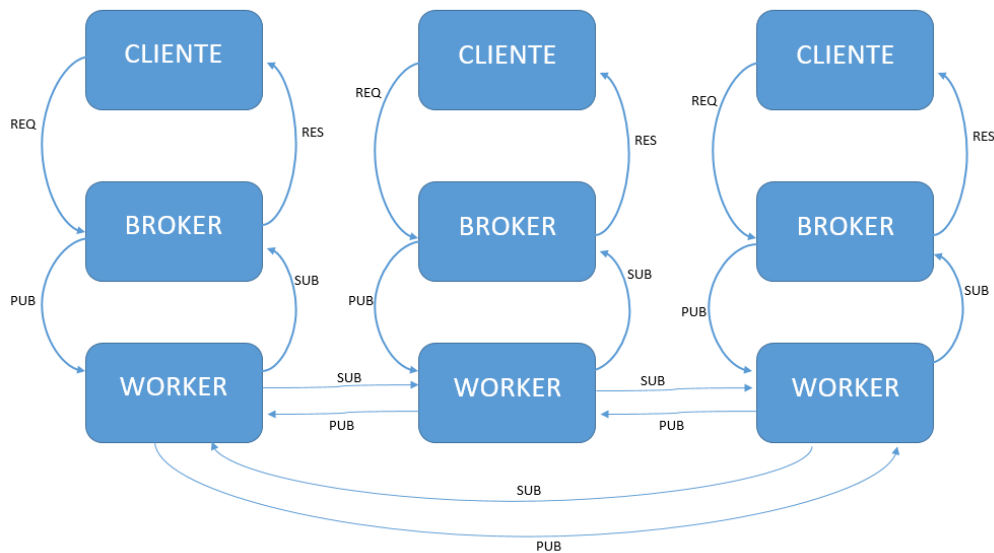
Además, utilizarán sockets ZMQ del tipo pub/sub para interactuar con los workers, este socket será del tipo subscriber y obtendrá la información publicada por los workers.

Workers Utilizarán sockets ZMQ del tipo pub/sub para conectarse con los brokers. Estos sockets publicarán la información en unos canales determinados en los cuales los brokers estarán suscritos.

Además, para las conexiones entre workers, cada uno deberá tener un socket publisher para mandar la información que ha recibido a los otros workers, y un socket subscriber para poder recibir información de los otros workers.

3.2. Actividad 2

Para realizar la implementación de este proyecto realizamos el siguiente diagrama de funcionamiento.



El cliente debe de tener un código REQ para poder realizar peticiones a su broker mientras que este debe de implementar un RES para poder responderle al cliente que empezó la petición.

```
var zmq      = require('zmq')
    , requester = zmq.socket('req');

requester.connect('tcp://localhost:5559');
var replyNbr = 0;
requester.on('message', function(msg) {
    console.log('got reply', replyNbr, msg.toString());
    replyNbr += 1;
});

for (var i = 0; i < 10; ++i) {
    requester.send("Hello");
}
```

Además, los brokers deben de tener implementado un PUB/SUB para comunicarse con los respectivos Workers.

```
var zmq = require('zmq')
var publisher = zmq.socket('pub')

publisher.bind('tcp://*:5563', function(err) {
  if(err)
    console.log(err)
  else
    console.log('Listening on 5563...')
})

setInterval(function() {
  publisher.send(["A", "We do not want to see this"]);
  publisher.send("B", zmq.ZMQ_SNDMORE);
  publisher.send("We would like to see this");
}, 1000);
```

Por ultimo, los Workers deben de tener implementado un PUB/SUB adicional para poder comunicarse entre ellos, y así, propagar a los clientes los cambios que se realizan en otros Workers.

```
var zmq = require('zmq')
var subscriber = zmq.socket('sub')

subscriber.on('message', function() {
  var msg = [];
  Array.prototype.slice.call(arguments).forEach(function(arg) {
    msg.push(arg.toString());
  });

  console.log(msg);
})

subscriber.connect('tcp://localhost:5563')
subscriber.subscribe('B')
```

4. Seminario 6 - 7

4.1. Actividad 1

Se realiza el paso de callbacks a promesas.

```
function retrieve(file, name, cb){
  cb(file[name]);
}

function processEntry(name, cb){
  if(rolodex[name]){
    cb(rolodex[name]);
  }else {
    retrieve(rolodexFile, name, function(val){
      rolodex[name]=val;
      cb(val);
    });
  }
}

function test() {
  for (var n of testName){
    console.log('processing', n);
    processEntry(n, function (res){
      console.log('processed', n);
    });
  }
}
```

CALLBACKS

```
function retrieve(file, name, cb){
  cb(file[name]);
}

function processEntry(name, cb){
  var p = new Promise(function(res){
    if(rolodex[name]){
      res(rolodex[name]);
    }else {
      retrieve(rolodexFile, name, function(value) {
        rolodex[name]=value;
        res(value);
      });
    }
  });
  return p;
}

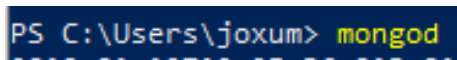
function test() {
  testName.forEach(function(name){
    console.log('processing', name);
    processEntry(name)
      .then(function (res){
        console.log('processed', name);
      });
  });
}
```

PROMESAS

4.2. Actividad 2

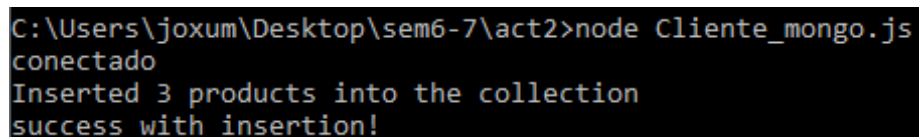
Para esta actividad es necesario ejecutar mongo por lo que lanzamos una terminal con el comando

```
# mongod
```



Y posteriormente ejecutamos el fichero Cliente_mongo.js para llenar una base de datos de ejemplo:

```
# npm install
# node Cliente\_mongo.js
```



Tras esto, ya tendremos la base de datos lista para ejecutar la Actividad 2. Como ya instalamos las dependencias, solo tendremos que ejecutar el proyecto

```
# node servidor-https.js
```

```
C:\Users\joxum\Desktop\sem6-7\act2>node servidor_https.js
Servidor web escuchando en el puerto 3000
```

Si entramos en la dirección localhost:3000 podremos ver el funcionamiento de la actividad.



Apareciendo una lista de productos que tenemos en el almacén y una lista de productos que tenemos en nuestro carrito de la compra. Además tenemos botones para añadir productos al carrito desde el almacén y para poder eliminar productos del carrito, devolviéndolos así al almacén.

Si queremos agregar un producto al carrito el cual no tiene stock en el almacén, no nos dejara agregarlo, imprimiendo también un mensaje de error en la terminal donde se lanzo el servidor web.

```
Servidor web escuchando en el puerto 3000
ERROR: No hay stock del producto palos
```

Para esta actividad se realizan 3 métodos principales en el archivo carrito-Compra.js

Un método de listado, uno de añadir items al carrito de compra y un ultimo método para eliminar items del carrito.

Los tres métodos han sido implementados siguiendo una estructura de promesas.

```
// AÑADIR CARRITO
app.get("/addToCart", function(req, res) {
  return new Promise(function(resolve, reject) {
    mongo.connect(url, function(err, db) {
      var o_id = new mongo.ObjectID(req.query.id.replace(/["]+/g, ''));
      var collection = db.collection('products');

      collection.findOne({'_id': o_id}, function (err, item) {

        if (item.stock > 0) {
          shoppingCart.push(item);
        }
      });
    });
  });
});
```

5. Seminario 8

5.1. Actividad 1

1. Estricto: No, porque la lectura de las variables tras una escritura no leen el valor de dicha escritura.
Secuencial: No, P2 lee el valor 3 antes del 2.
Casual: Si, $1 \rightarrow 3$ y se cumple en todas.
FIFO: Si.
Cache: No.
Procesador: No, cumple FIFO pero no Cache.
2. Estricto: No, porque la lectura de las variables tras una escritura no leen el valor de dicha escritura.
Secuencial: No, P3 no sigue el mismo orden que P4.
Causal: No, ya que no es FIFO.
FIFO: No.
Caché: Sí.
Procesador: No, cumple Caché pero no FIFO.

5.2. Actividad 2

1. Estricto: No, porque la lectura de las variables tras una escritura no leen el valor de dicha escritura.
Secuencial: No, P2 lee R(y)2 antes que R(x)1.
Causal: Sí, las escrituras son concurrentes.
FIFO: Sí.
Caché: No, P2 lee R(x)1 antes que R(x)3.
Procesador: No.
2. Estricto: No, porque la lectura de las variables tras una escritura no leen el valor de dicha escritura.
Secuencial: No, P3 no sigue el mismo orden que P2.
Causal: No, P3 lee R(x)1 después que R(x)3.
FIFO: Sí.
Caché: No, P3 no lee X1 primero.
Procesador: No, cumple FIFO pero no Caché.

5.3. Actividad 3

1. Estricto: Sí.
Secuencial: Si.
Causal: Si.
FIFO: Sí.
Caché: No, P4 no lee x=1 antes que x=2 y P3 no lee x=2 antes que x=3.
Procesador: No, cumple FIFO pero no Caché.

- #### 5.4. Actividad 4

- ### 5.5. Actividad 5

- | | | | | | | | | | | | | |
|----|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| P1 | W(x)1 | | | W(y)3 | | | | | | | | |
| P2 | | W(x)2 | W(y)4 | | | | | | | | | |
| P3 | | | | | R(x)1 | | R(y)3 | | R(x)2 | | R(y)4 | |
| P4 | | | | | | R(y)3 | | R(x)1 | | R(y)4 | | R(x)2 |

- | | | | | | | | | | | | | |
|----|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|--|
| P1 | W(x)1 | | | W(y)3 | | | | | | | | |
| P2 | | W(x)2 | W(y)4 | | | | | | | | | |
| P3 | | | | R(x)1 | | R(y)3 | | R(x)2 | | R(y)4 | | |
| P4 | | | | | R(x)2 | | R(y)4 | | R(x)1 | | R(y)3 | |

5.6. Actividad 6

1. Procesador pero no Casual

P1	W(x)1								
P2		R(x)1	W(y)2						
P3				R(y)2	W(z)3				
P4						R(y)3	R(y)2	R(x)1	

2. Casual pero no Procesador

P1	W(x)1								
P2		R(x)1	W(x)2	W(x)3					
P3					R(x)1	R(x)2		R(x)3	
P4							R(x)1		R(x)3

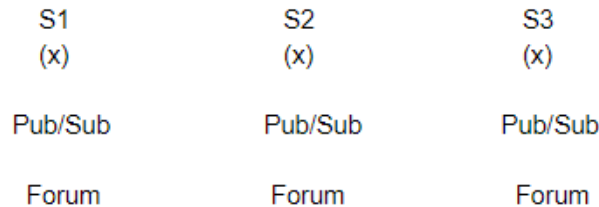
5.7. Actividad 7

1. Si es secuencial, siempre va a ser Cache.
2. Cache, pero no secuencial.

P1	W(x)1		W(y)2							
P2		W(x)2		W(y)3						
P3					R(x)1		R(y)2		R(x)2	R(y)3
P4						R(y)2		R(x)1		R(y)3

6. Seminario 9

6.1. Actividad 1



Debido a que nuestros servidores tienen implementado el modelo Pub/Sub, los servidores están suscritos a la información que le publica a su servidor forum correspondiente. A su vez, los servidores forum publican los datos a cada servidor dmserver.

La consistencia de este modelo está determinada por una única variable, la cual es leída y modificada por todos los servidores. Por lo tanto, cada vez que un servidor forum publica información se podría soportar un modelo estricto de consistencia si los servidores forum no pudiesen publicar otra cosa hasta que no recibiesen confirmación de los dmserver. De esta manera se podrá soportar la lectura de los datos por parte de todos los dmserver de manera estricta.

En caso de no poseer un sistema de confirmación cada vez que el forum publica datos, los sistemas soportados serían algo más relajados ya que no se podría controlar estrictamente en qué orden leen el valor de la variable cada dmserver. En este caso, un proceso forum puede escribir información más de una vez antes de que un dmserver lea el valor de x. Aquí, los modelos soportados no podrían ser ni estricto ni secuencial. El modelo causal se cumpliría si cada proceso dmserver leyera los datos que han cambiado otros procesos obligatoriamente antes de escribir sus datos. De esta manera se respetaría la propiedad transitiva entre procesos.

Por último, los modelos FIFO y caché también podrían ser soportados. En el caso de FIFO, todas las escrituras generadas por un mismo proceso de deben de leer en el mismo orden por los demás procesos. Por lo tanto, este modelo podrá ser soportado si disponemos, por ejemplo, de un middleware intermedio que bloquee la variable x a la hora de realizar una escritura en otros servidores y propague las escrituras en un orden determinado a los otros servidores.

6.2. Actividad 2

N1		N2		N3
Forum		Forum		Forum
S1(x)	N?W(x)/N?R(x)	S2(x)	N?W(x)/N?R(x)	S3(x)

Debido a que nuestros servidores tienen implementado el modelo Pub/Sub, los servidores están suscritos a la información que le publica cada cliente su forum y dmserver, pero también a lo que otros dmserver publican.

La consistencia de este modelo está determinada por una única variable, la cual es leída y modificada por todos los servidores. Por lo tanto, cada vez que un servidor forum publica información se podría soportar un modelo estricto de consistencia si los servidores forum no pudiesen publicar otra cosa hasta que no recibiesen confirmación de los dmserver. De esta manera se podrá soportar la lectura de los datos por parte de todos los dmserver de manera estricta y propagarse hacia arriba en el resto de dmserver hacia los clientes.

En caso de no poseer un sistema de confirmación cada vez que el forum publica datos, los sistemas soportados serían algo más relajados ya que no se podría controlar estrictamente en qué orden leen el valor de la variable cada dmserver. En este caso, un proceso forum puede escribir información más de una vez antes de que un dmserver lea el valor de x y que el valor sea propagado. Aquí, los modelos soportados no podrían ser ni estricto ni secuencial. El modelo causal se cumpliría si cada proceso dmserver leyera los datos que han cambiado otros procesos obligatoriamente antes de escribir sus datos. De esta manera se respetaría la propiedad transitiva entre procesos.

Por último, los modelos FIFO y caché también podrían ser soportados. En el caso de FIFO, todas las escrituras generadas por un mismo proceso de deben de leer en el mismo orden por los demás procesos. Por lo tanto, este modelo podrá ser soportado si disponemos, por ejemplo, de un middleware intermedio que bloquee la variable x a la hora de realizar una escritura en otros servidores y propague las escrituras en un orden determinado a los otros servidores y sus clientes.

6.3. Actividad 3

S1 (x)(y)(z)	S2 (x)(y)(z)	S3 (x)(y)(z)
Pub/Sub	Pub/Sub	Pub/Sub
Forum	Forum	Forum

Debido a que nuestros servidores tienen implementado el modelo Pub/Sub, los servidores están suscritos a la información que le publica a su servidor forum correspondiente. A su vez, los servidores forum publican los datos a cada servidor dmserver.

La consistencia de este modelo está determinada por tres variable, las cuales pueden ser leídas y modificada por todos los servidores simultáneamente. Por lo tanto, cada vez que un servidor forum publica información se podría soportar un modelo estricto de consistencia si los servidores forum no pudiesen publicar otra cosa en esa misma variable hasta que no recibiesen confirmación de los dmserver. De esta manera se podrá soportar la lectura de los datos por parte de todos los dmserver de manera estricta.

En caso de no poseer un sistema de confirmación cada vez que el forum publica datos, los sistemas soportados serían algo más relajados ya que no se podría controlar estrictamente en qué orden leen el valor de cada variable en cada dmserver. En este caso, un proceso forum puede escribir información más de una vez antes de que un dmserver lea el valor de una misma variable. Aquí, los modelos soportados no podrían ser ni estricto ni secuencial. El modelo causal se cumpliría si cada proceso dmserver leyera los datos de una misma variable que han cambiado otros procesos obligatoriamente antes de escribir sus datos. De esta manera se respetaría la propiedad transitiva entre procesos.

Por último, los modelos FIFO y caché también podrían ser soportados. En el caso de FIFO, todas las escrituras generadas por un mismo proceso en una misma variable de deben de leer esa variable en el mismo orden por los demás procesos. Por lo tanto, este modelo podrá ser soportado si disponemos, por ejemplo, de un middleware intermedio que bloquee las variables a la hora de realizar una escritura en otros servidores y propague las escrituras en un orden determinado a los otros servidores.

6.4. Actividad 4

El considerar separar los datos en diferentes variable tiene sentido a la hora de realizar un sistema distribuido, ya que permite la lectura y escritura distribuida y simultánea entre variables. Un proceso puede estar escribiendo en una variable y otro proceso realizar una escritura simultánea en una variable diferente sin que esto suponga un conflicto.

A la hora de realizar distintas implementaciones, como ya se ha justificado anteriormente, es interesante considerar un sistema Pub/Sub que propague los cambios hacia el resto de servidores que pueden hacer uso del dato. Al mismo tiempo, debemos de integrar un middleware que permita el interbloqueo de variables si un proceso está haciendo uso de esta, o al menos, que gestione de manera inequívoca la información para evitar inconsistencias en la misma.

7. Seminario 10

7.1. Actividad 1

En esta actividad se prueba el funcionamiento del sistema blockchain, ejecutando así dicho sistema, generando el paquete génesis y generando los siguientes bloques de la serie.

```
# npm install
# node main.js
```

```
C:\Users\joxum\Desktop\sem10>npm install
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN naivechain@1.0.0 No description
npm WARN naivechain@1.0.0 No repository field.
npm WARN naivechain@1.0.0 No license field.

added 68 packages in 3.811s

C:\Users\joxum\Desktop\sem10>node main.js
listening websocket p2p port on: 6001
listening http on port: 3001
```

```
localhost:3001/blocks
[{"index":0,"previousHash":"","timestamp":1465154705,"data":"my genesis block!!","hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7"}]
```

Blockchain genera una cadena de bloques relacionados, en nuestra actividad incluiremos el índice, marca de tiempo, datos, hash y hash anterior.

El hash del bloque anterior debe encontrarse en el bloque para poder preservar la integridad de la cadena, comprobando así la secuencia y si ha sido alterada en algún momento



En nuestra actividad generamos un hash SHA-256 para poder mantener la integridad de los datos

Para generar un bloque nuevo, debemos conocer el hash anterior para poder almacenarlo en el bloque generado. El almacenaje de bloques los haremos mediante una matriz.

```
localhost:3001/mineB
POST localhost:3001/mineBlock
```

```
localhost:3001/blocks
[{"index":0,"previousHash":"","timestamp":1465154705,"data":"my genesis block!!","hash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7"},
{"index":1,"previousHash":"816534932c2b7154836da6afc367695e6337db8a921823784c14378abed4f7d7","timestamp":1547382831.399,"hash":"a0ea2a1ba94a7d7b11eac178857d687814cca15362c863f822500397c474a6"}]
```

Para calcular la integridad de bloques, debemos comprobar la integridad de hash y hash anteriores almacenados a lo largo de la cadena. Si algún hash de bloques, no coincide con el hash anterior almacenado significa que algún bloque de la cadena ha sido alterado y por tanto no debemos aceptar la cadena.

Ante un conflicto de consistencia, debemos escoger la cadena más ya que esta cadena tiene nuevos nodos en la cadena de bloques.

