




# Tema 8

# Websockets

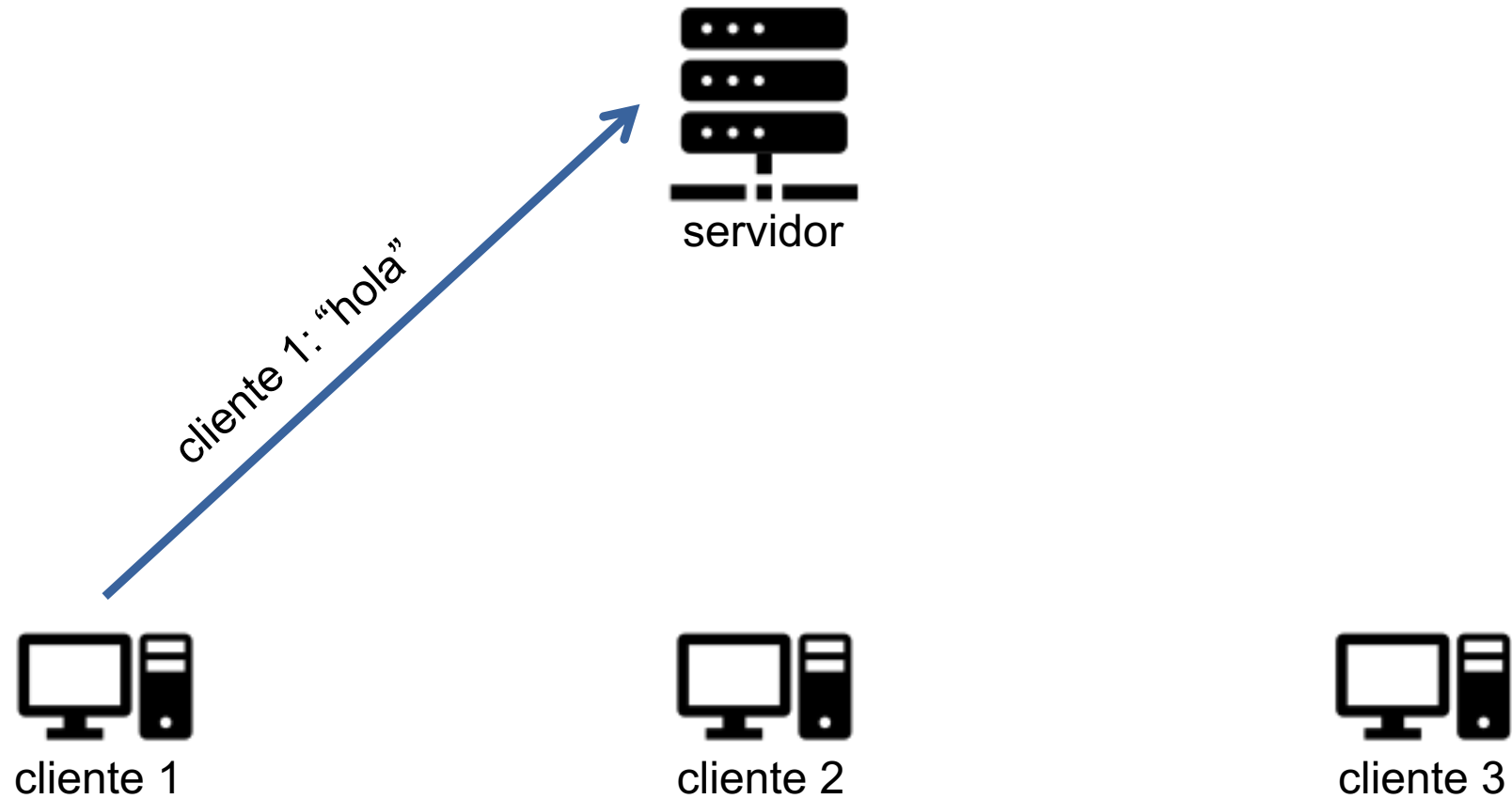
Plataformas de Software Empresariales  
Grado en Ingeniería Informática de Servicios y Aplicaciones  
Curso 2021 /2022

# Introducción

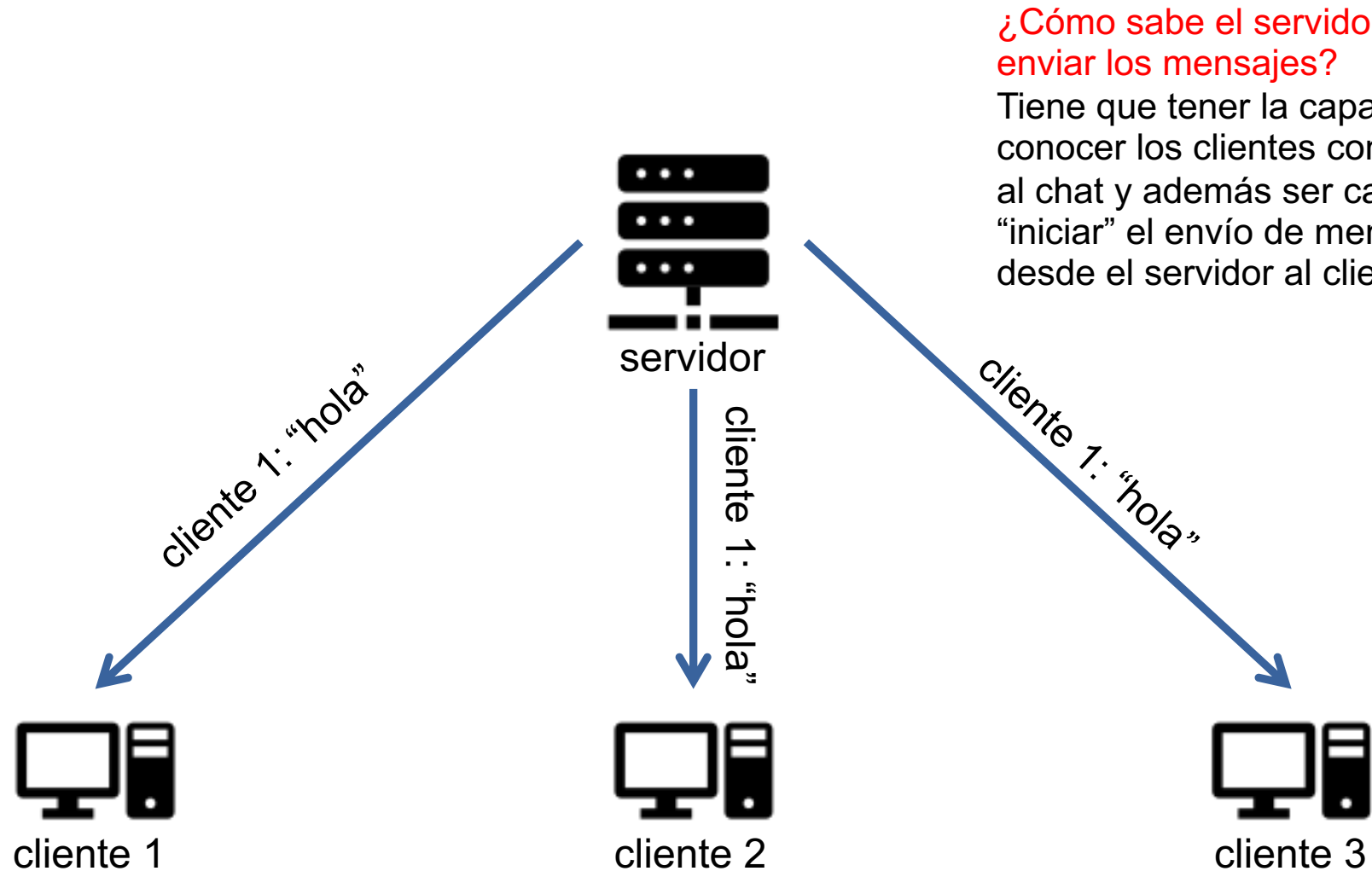
## ► Objetivos

- Conocer el funcionamiento del API Java para Websockets
  - Construir un servidor usando anotaciones
  - Construir un cliente usando anotaciones
  - Construir un cliente usando JavaScript
- 

# Introducción (ej: chat)



# Introducción (ej: chat)



¿Cómo sabe el servidor a quién enviar los mensajes?

Tiene que tener la capacidad de conocer los clientes conectados al chat y además ser capaz de "iniciar" el envío de mensajes desde el servidor al cliente

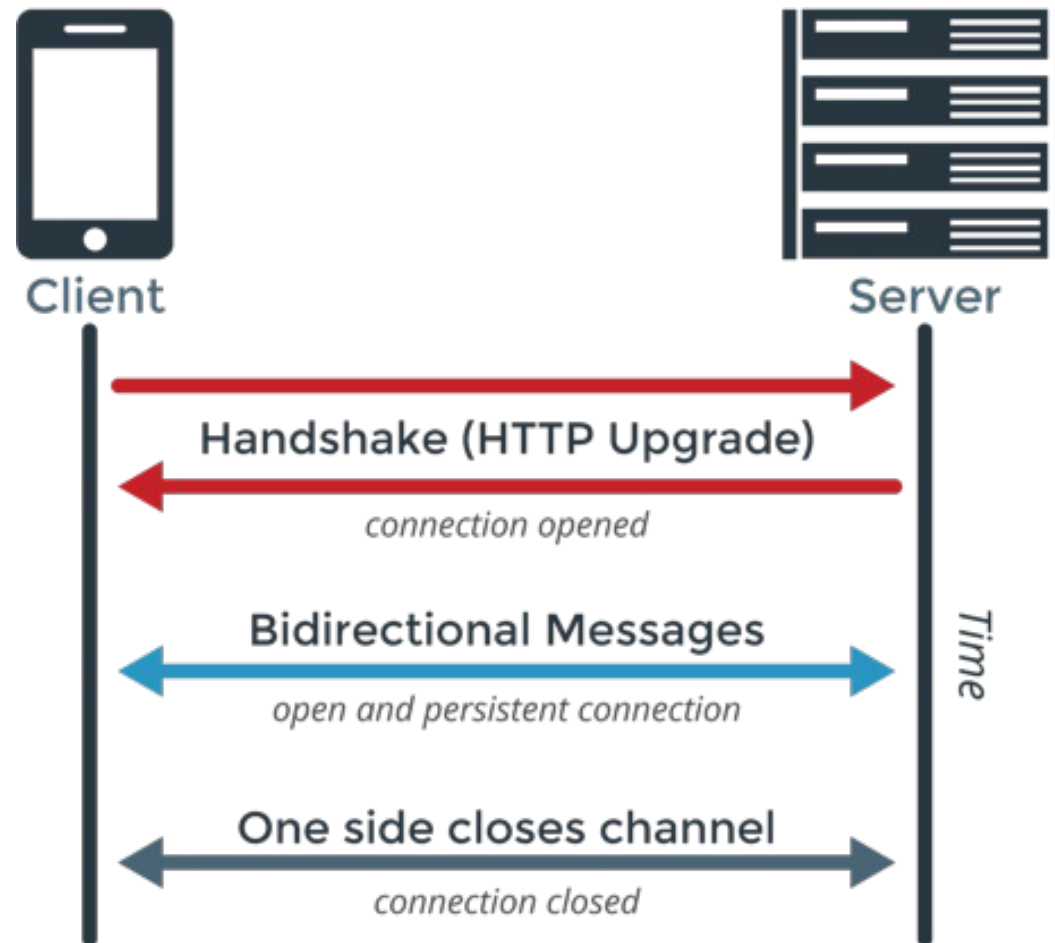
# Introducción

- ▶ En muchas aplicaciones es necesario que los datos se actualicen sin intervención del usuario
  - Pensad por ejemplo en un Chat, en un juego online, etc...
- ▶ No es práctico que el cliente esté constantemente haciendo peticiones al servidor para ver si tiene nuevos datos a actualizar
  - Ni desde el punto de vista de la sobrecarga, ni del ancho de banda, ni del manejo de la aplicación en “tiempo real”
- ▶ Lo mejor es crear un canal de comunicación y que sea el propio servidor el que envíe la información al cliente cuando sea necesario
  - Pero, como ya hemos comentado para explicar la necesidad de los servicios web, los puertos del servidor están limitados
- ▶ El API de Websockets nos permite trabajar sobre el puerto HTTP
- ▶ HTML5 y Java EE desde la versión 7 ya incluyen soporte nativo para Websockets.

# Websockets

- ▶ Protocolo de comunicación bidireccional y full-duplex a través de una única conexión TCP
  - Bidireccional: cliente puede enviar un mensaje al servidor y viceversa
  - Full-duplex: cliente y servidor pueden enviar mensajes independientemente del otro de forma simultanea
- ▶ El protocolo de Websockets define un proceso inicial de handshake (por el que se establece la conexión) y un mecanismo de envío de mensajes
- ▶ Una vez que el handshake se ha hecho, los mensajes se pueden enviar de manera indistinta
- ▶ La comunicación es simétrica, con dos excepciones:
  - El cliente inicia la conexión con el servidor que esté escuchando una petición WebSocket
  - El cliente se conecta al servidor usando una URI. El servidor escucha todas las peticiones de conexión de los clientes en la misma URI

# Websockets

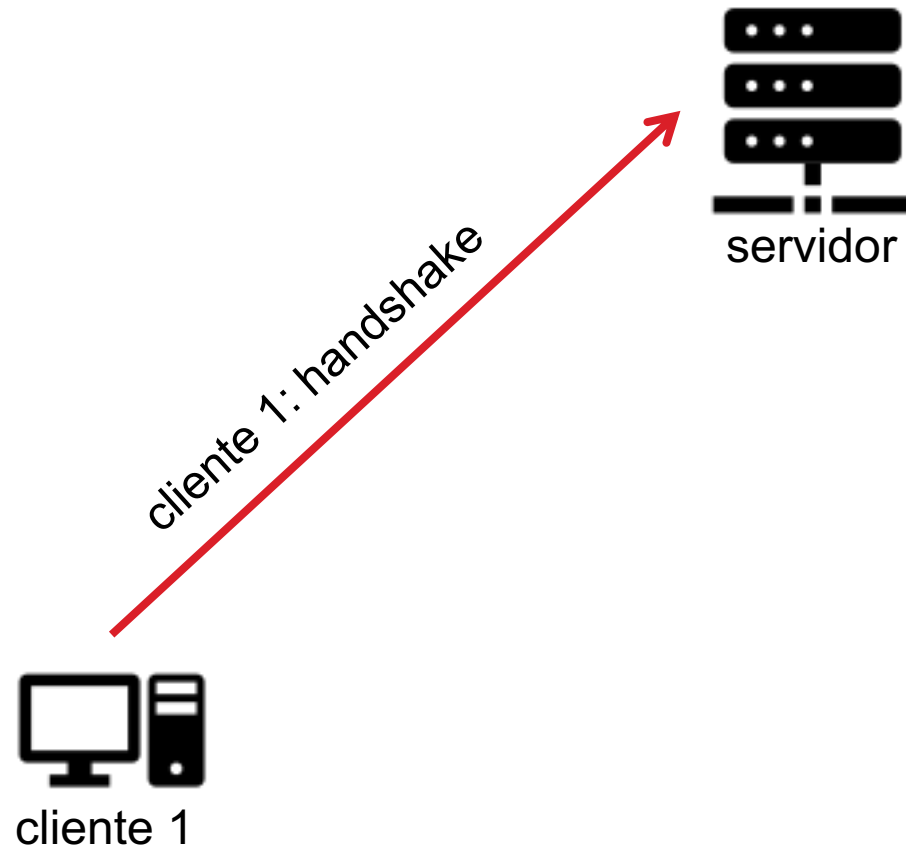


# Websockets

- ▶ Después del handshake, tanto cliente como servidor son considerados “peers”
- ▶ Cada peer puede transmitir datos en unidades conceptuales llamadas mensajes (message)

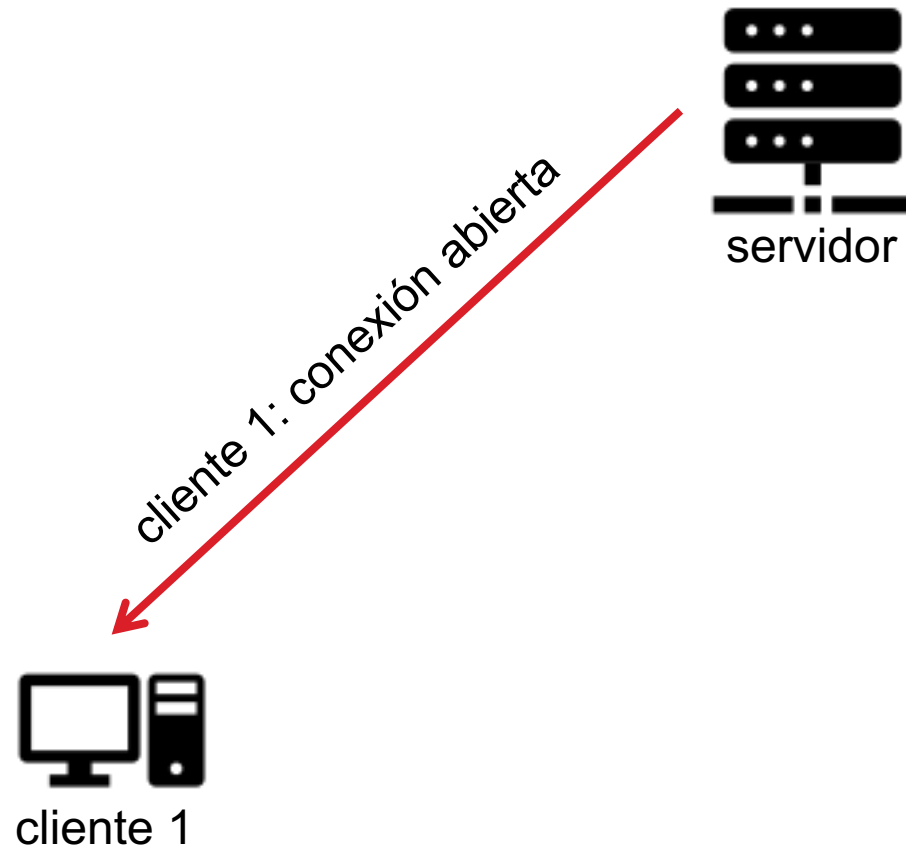


# Websockets (ej: chat)

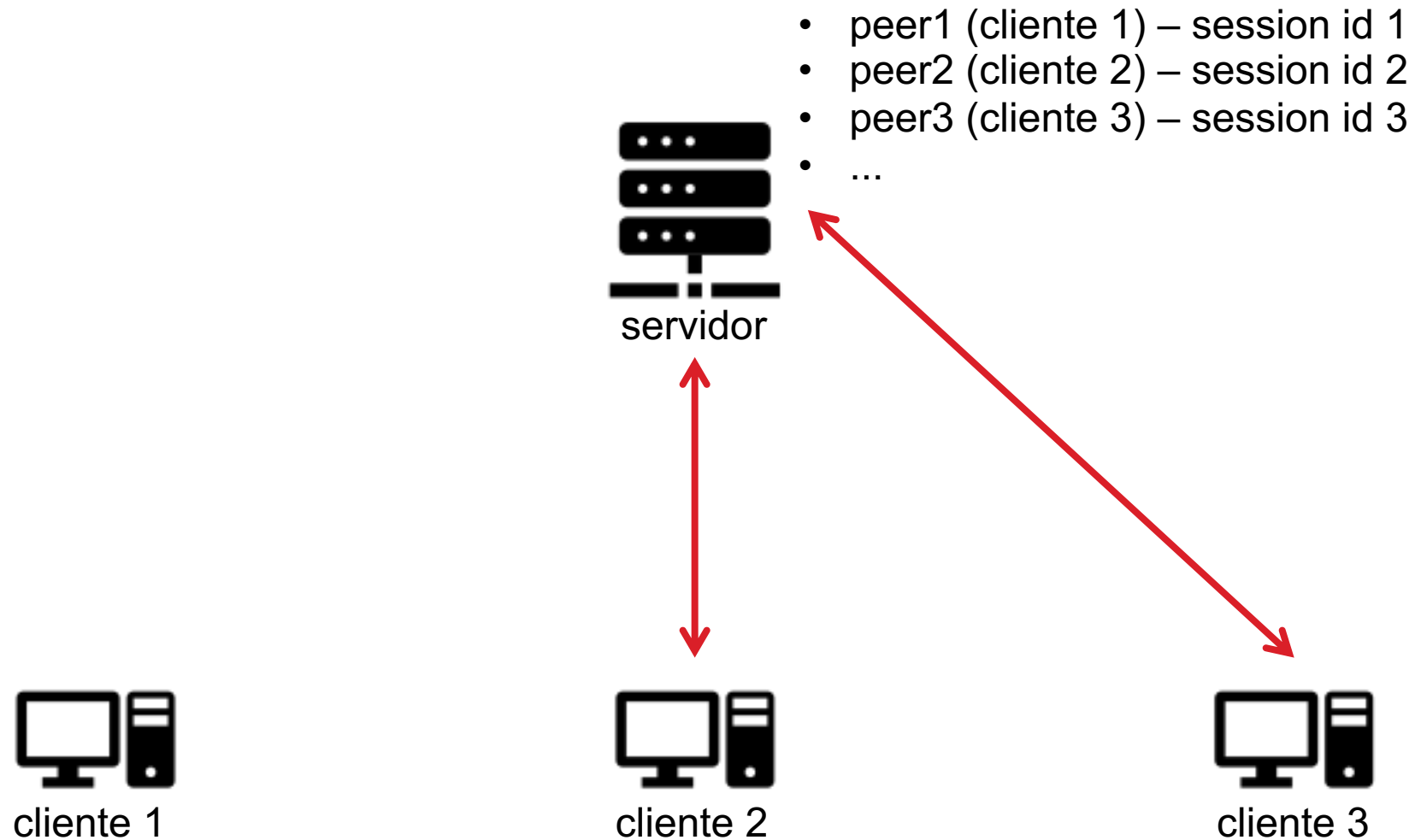


# Websockets (ej: chat)

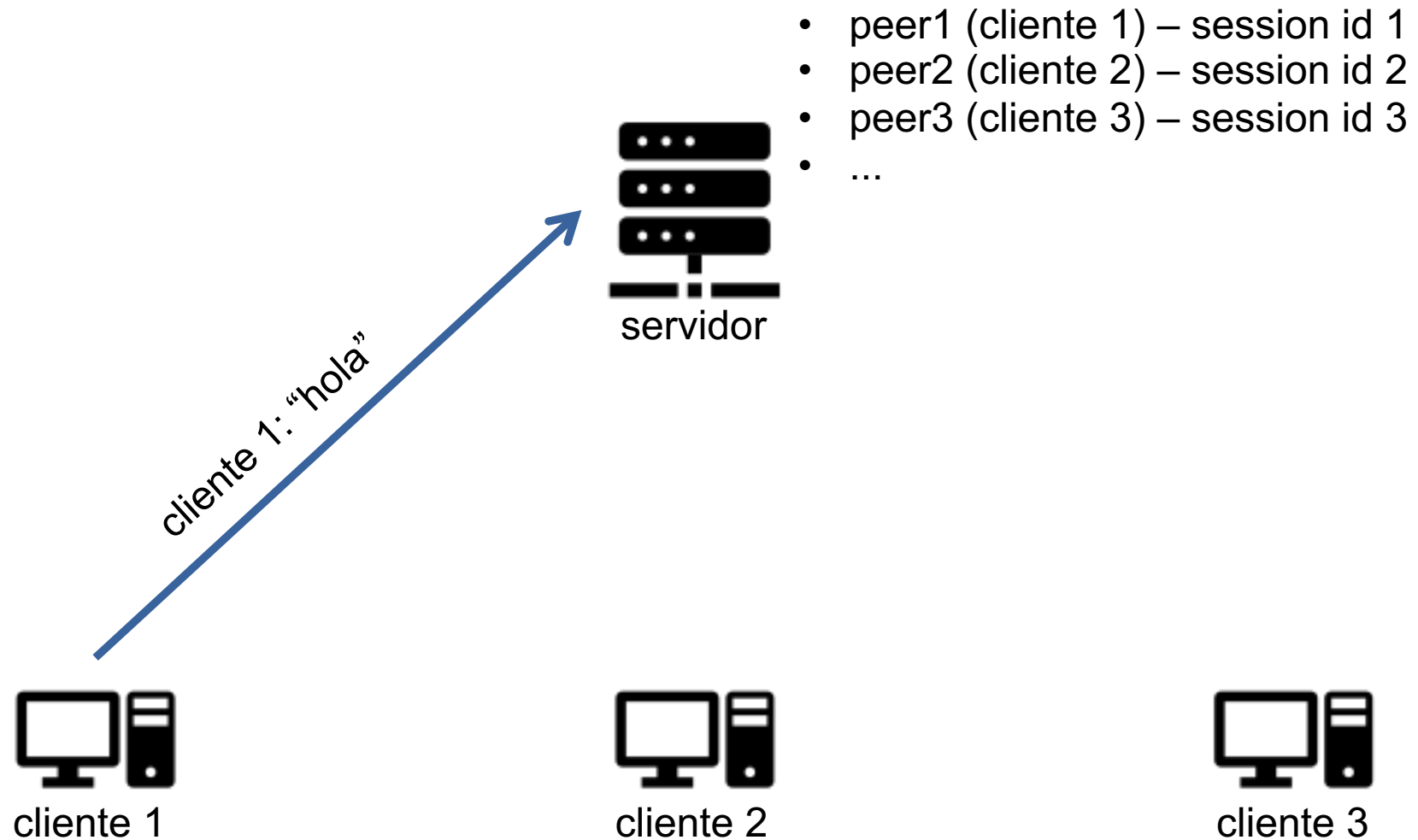
- peer1 (cliente 1) – session id 1



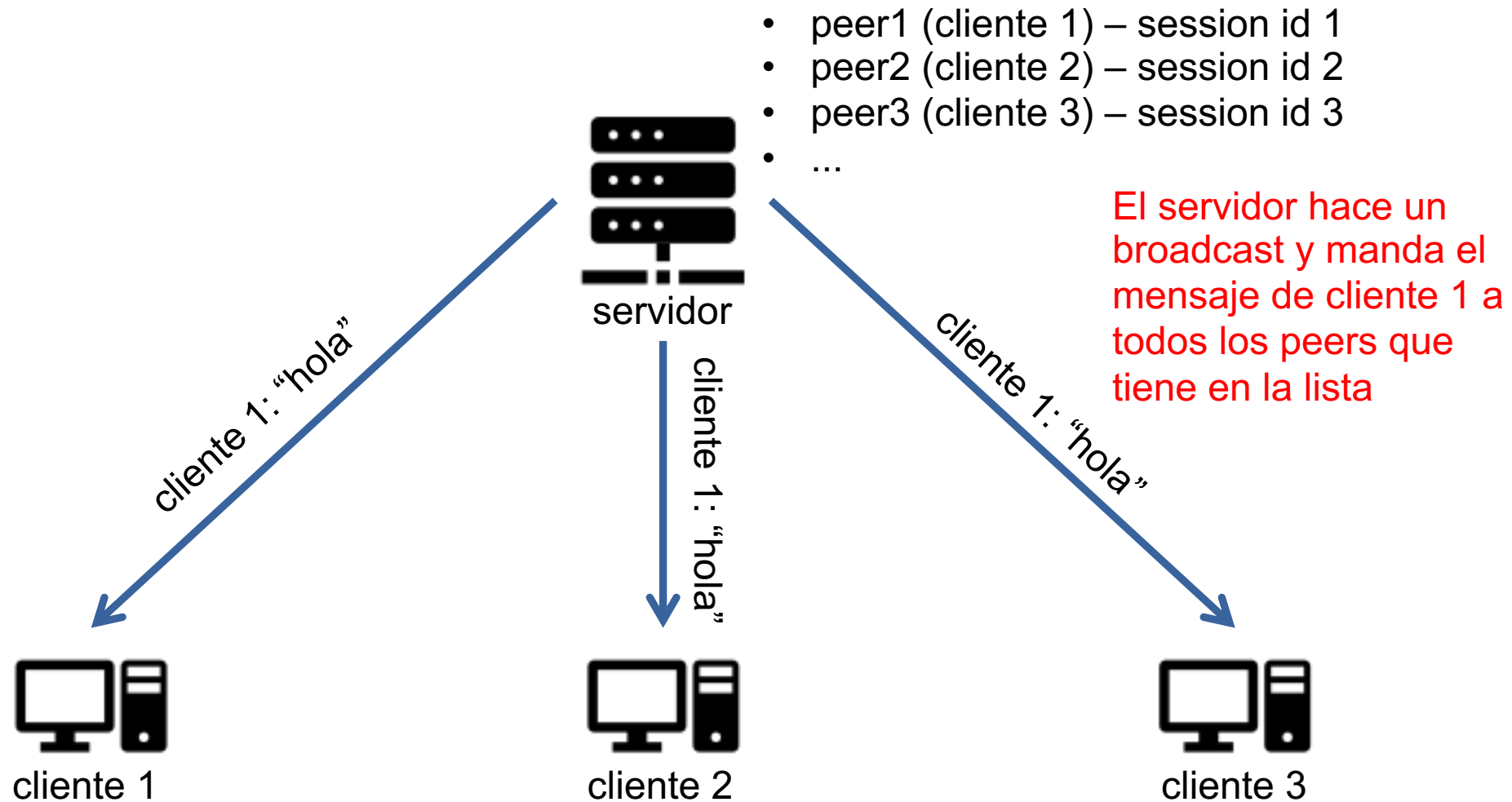
# Websockets (ej: chat)



# Websockets (ej: chat)



# Websockets (ej: chat)



# Websockets en Java EE

- ▶ El API Java para WebSockets permite
  - **Crear clientes Websocket y servidores (endpoint) usando anotaciones (nosotros lo haremos de esta manera para el servidor)**
  - Crear **clientes** Websocket y **servidores** (endpoint) de manera **programática**
  - **Crear y consumir mensajes Websocket de texto, binarios, y de control**
  - Iniciar e interceptar eventos del ciclo de vida de los Websockets
  - Configurar y manejar sesiones Websocket
  - Especificar el funcionamiento de la aplicación Websocket dentro del modelo de seguridad Java EE

# Servidor endpoint

- ▶ Se puede convertir una clase Java en un servidor endpoint de Websockets utilizando la anotación `@ServerEndpoint`

```
@ServerEndpoint("/chat")
public class ChatServer {
    @OnMessage
    public String receiveMessage(String message) {
        //...
    }
}
```

# Servidor endpoint

- ▶ La anotación `@ServerEndpoint` declara la clase como un endpoint Websocket publicado en la URI especificada en el valor de la anotación (“/chat”)
- ▶ `@OnMessage` declara el método que recibe los mensajes entrantes de Websocket.
- ▶ Este método puede recibir mensajes de texto, binarios y de tipo ping/pong (mensaje de control Websocket) y puede tener distintos tipos parámetros. Ej (para texto):
  - `public void receiveMessage(String s)`: recibe el mensaje completo en el string `s`
  - `public void receiveMessage(int i)`: recibe el mensaje completo convertido en un tipo concreto (en este caso un entero)
  - `public void receiveMessage(String message, boolean last)`: recibe el mensaje en partes. El booleano se hace true cuando se reciba la última parte
  - Etc, etc, etc.



# Servidor endpoint

- ▶ También puede tener un parámetro de tipo sesión, que indique la sesión concreta entre un cliente y un servidor
- ▶ Por ejemplo, para que el servidor responda al cliente lo podemos hacer de la siguiente manera:

```
public void receiveMessage(String message, Session session) {  
    session.getBasicRemote().sendText(...);  
}
```

- ▶ En principio, el método puede retornar un void, es decir, el cliente no espera respuesta del servidor, aunque también podría devolver un String, clase Java, etc...

# Servidor endpoint

- ▶ El atributo `maxMessageSize` se puede utilizar para definir el tamaño máximo del mensaje (en bytes) que el método será capaz de procesar. Ej:

```
@OnMessage(maxMessageSize=6)
public void receiveMessage(String s)
{
    //...
}
```

- ▶ Si el mensaje recibido tiene más de 6 bytes, entonces se reporta un error y se cierra la conexión.
- ▶ Los mensajes de error se pueden procesar dentro del método anotado como `@OnError`
- ▶ El valor por defecto es `-1`, que significa que no hay máximo

# Servidor endpoint

- ▶ `@OnOpen` se utiliza para anotar el método que se llama cuando se recibe una nueva conexión desde un cliente
- ▶ `@OnClose` se utiliza para anotar el método que se llama cuando se cierra una conexión desde un cliente
- ▶ `@OnError` se utiliza para anotar el método que se llama cuando se recibe un error
- ▶ Estos métodos pueden tener varios parámetros:
  - `Session`
  - `EndpointConfig` para `@OnOpen`
  - `CloseReason` para `@OnClose`
  - Etc...

# Cliente endpoint

- ▶ Se puede convertir una clase Java en un cliente WebSocket usando la anotación `@ClientEndpoint`

```
@ClientEndpoint  
public class MyClientEndpoint {  
    //...  
}
```

# Cliente endpoint

- ▶ La conexión entre el cliente y el servidor se realiza utilizando ContainerProvider:

```
WebSocketContainer container = ContainerProvider.getWebSocketContainer();  
String uri = "ws://localhost:8080/myApp/chat";  
container.connectToServer(MyClient.class, URI.create(uri));
```


- ▶ WebSocketContainer es capaz de gestionar todo el proceso de handshake de manera automática
- ▶ El proceso queda bloqueado hasta que se ha establecido la conexión o hasta que retorna error

# Cliente endpoint

- ▶ Los mensajes entrantes desde el servidor se pueden recibir en cualquier método Java anotado con `@OnMessage`. Ej:

```
@OnMessage  
public void processMessage(String message, Session session) {  
    //...  
}
```

# Cliente endpoint

- ▶ La anotación `@OnOpen` llama a su método cuando una nueva conexión se establece con el servidor
  - ▶ La anotación `@OnClose` llama a su método cuando la conexión se termina
  - ▶ La anotación `@OnError` llama a su método cuando hay un error en la conexión
- 

# Cliente endpoint

- ▶ Podemos enviar mensajes desde el cliente hasta otro endpoint. Ej:

```
@OnOpen
public void onOpen(Session session) {
    try {
        session.getBasicRemote().sendText("Nombre");
    } catch (IOException ex) {
        //...
    }
}
```

- ▶ En este ejemplo, cuando el usuario se conecta al chat y abre la sesión, se envía el nombre del usuario al servidor



# Cliente JavaScript

- ▶ Podemos llamar a un endpoint Websocket usando el API para JavaScript
- ▶ El API nos permite conectarnos a un endpoint Websocket especificando la URL y una serie opcional de subprotocolos:

```
var websocket = new WebSocket("ws://localhost:8080/myApp/chat");
```

# Cliente JavaScript

- ▶ El API de JavaScript define manejadores de eventos que se invocan para distintos métodos del ciclo de vida:
  - onopen – se llama cuando se inicia un conexión
  - onerror – se llama cuando se recibe un error durante la comunicación
  - onclose – se llama cuando se termina la conexión

```
websocket.onopen = function(evt) {  
    //...  
}  
websocket.onerror = function(evt) {  
    //...  
}  
websocket.onclose = function(evt) {  
    //...  
}
```


# Cliente JavaScript

- ▶ Los mensajes se reciben a través del manejador de eventos onmessage (los datos están en evt.data):

```
websocket.onmessage = function(evt) {  
    console.log("message received: " + evt.data);  
}
```

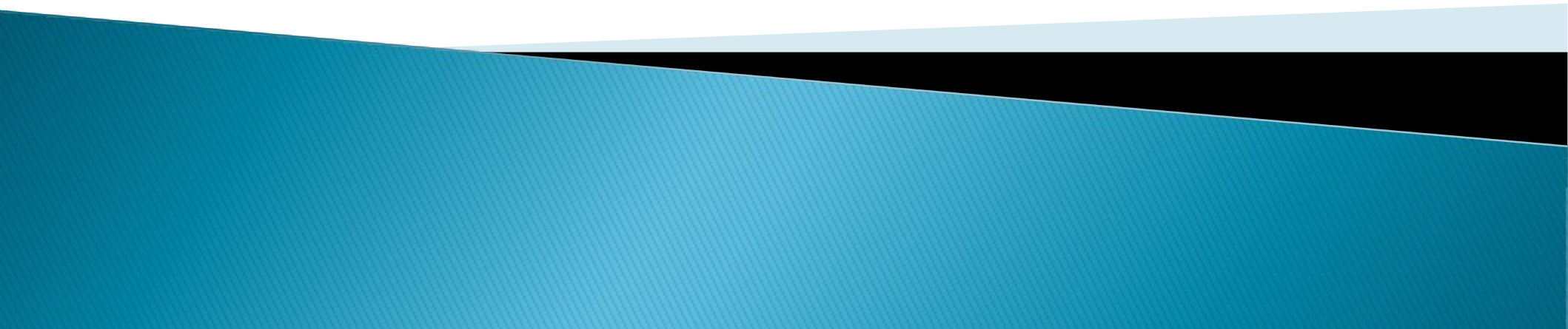
- ▶ Los datos en formato texto o binario se pueden enviar utilizando los métodos send:

```
websocket.send(myField.value);
```



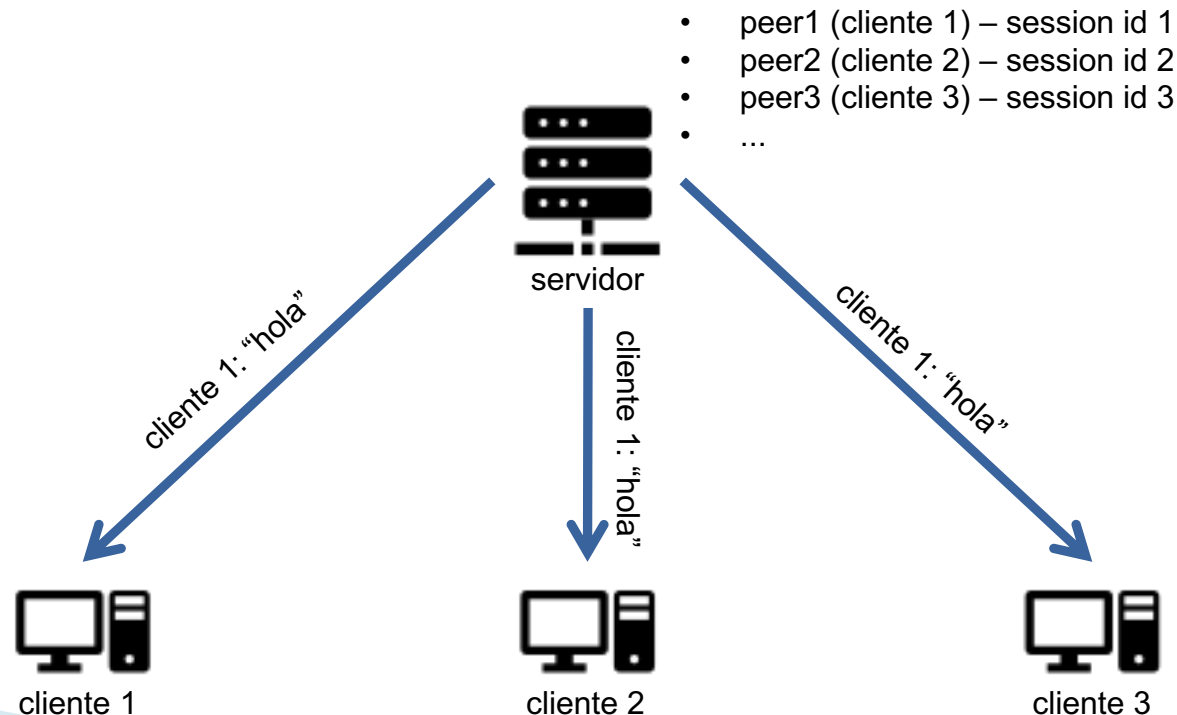
# Proyecto de desarrollo

Vamos a añadir un chat de usuarios

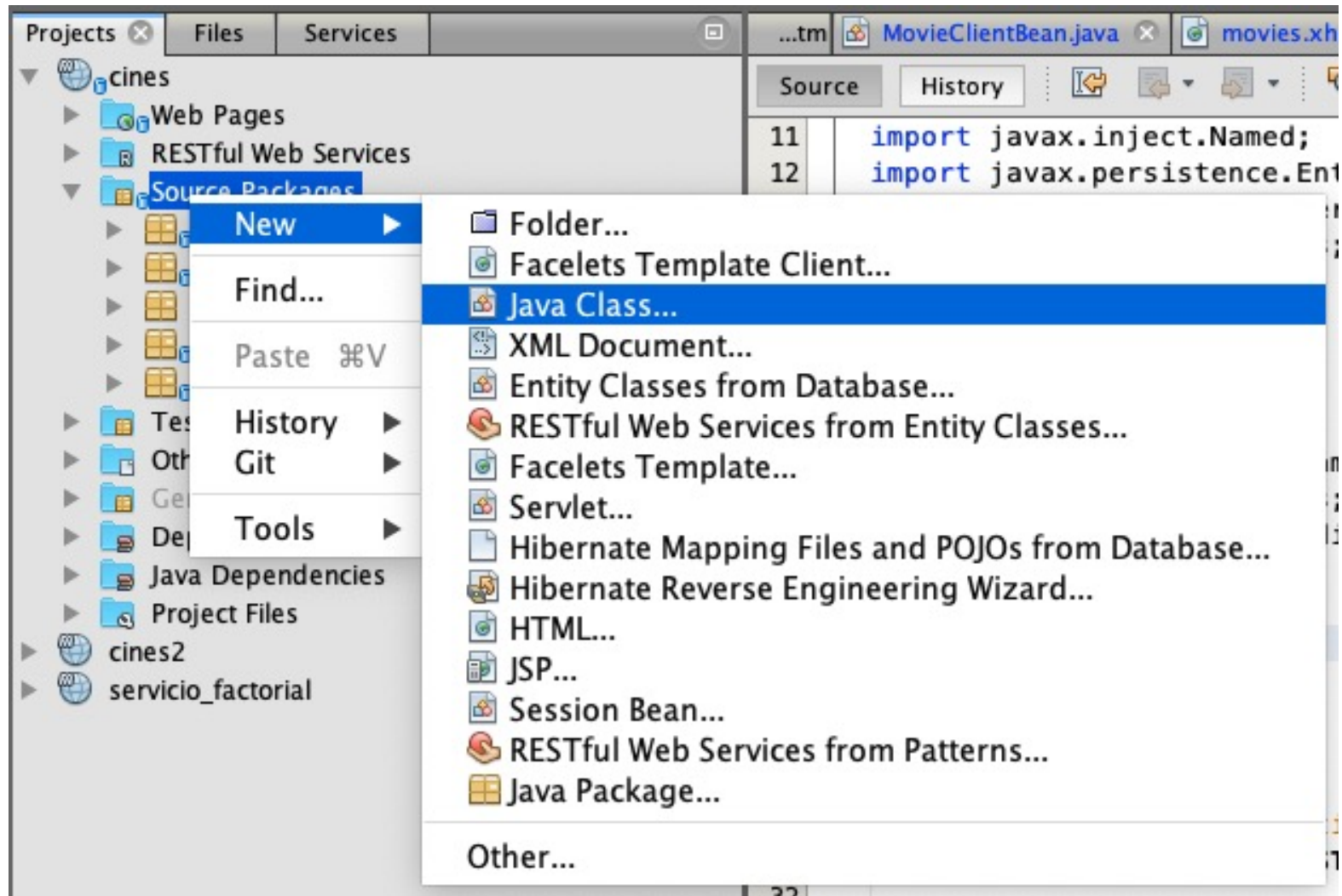


# Servidor

- ▶ Creamos la clase para el servidor de chat y lo ponemos dentro de un paquete nuevo (chat):
  - Botón derecho en “Source Packages”, seleccionamos New/Java Class, le damos el nombre “ChatServer” y el nombre de paquete correspondiente (en mi caso “com.anibal.cines.chat”)



# Servidor



# Servidor

**New Java Class**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Class Name: ChatServer

Project: cines

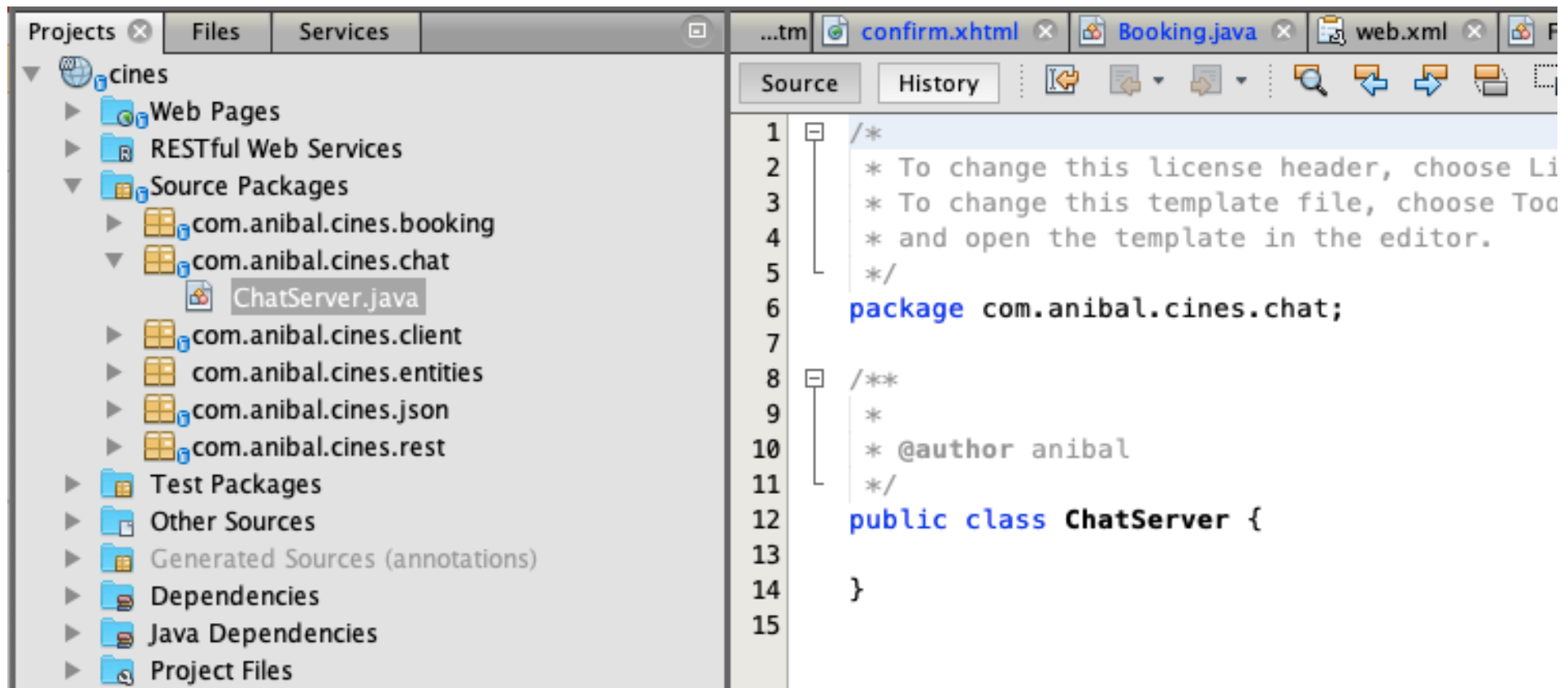
Location: Source Packages

Package: com.anibal.cines.chat

Created File: objects/cines/cines/cines/src/main/java/com/anibal/cines/chat/ChatServer.java

Help < Back Next > **Finish** Cancel

# Servidor





# Servidor

- ▶ Añadimos la anotación del “ServerEndpoint” y los métodos del servidor:

```
@ServerEndpoint("/websocket")
public class ChatServer {

    private static final Set<Session> peers = Collections.synchronizedSet(new HashSet<Session>());

    @OnOpen
    public void onOpen(Session peer) {
        peers.add(peer);
    }

    @OnClose
    public void onClose(Session peer) {
        peers.remove(peer);
    }

    @OnMessage
    public void message(String message, Session client) throws IOException, EncodeException {
        for (Session peer : peers) {
            peer.getBasicRemote().sendText(message);
        }
    }
}
```

# Servidor

## ▶ @ServerEndpoint

- Indica que la clase va a ser un WebSocket endpoint
- El valor indica la URI donde se va a publicar

## ▶ @OnOpen y @OnClose

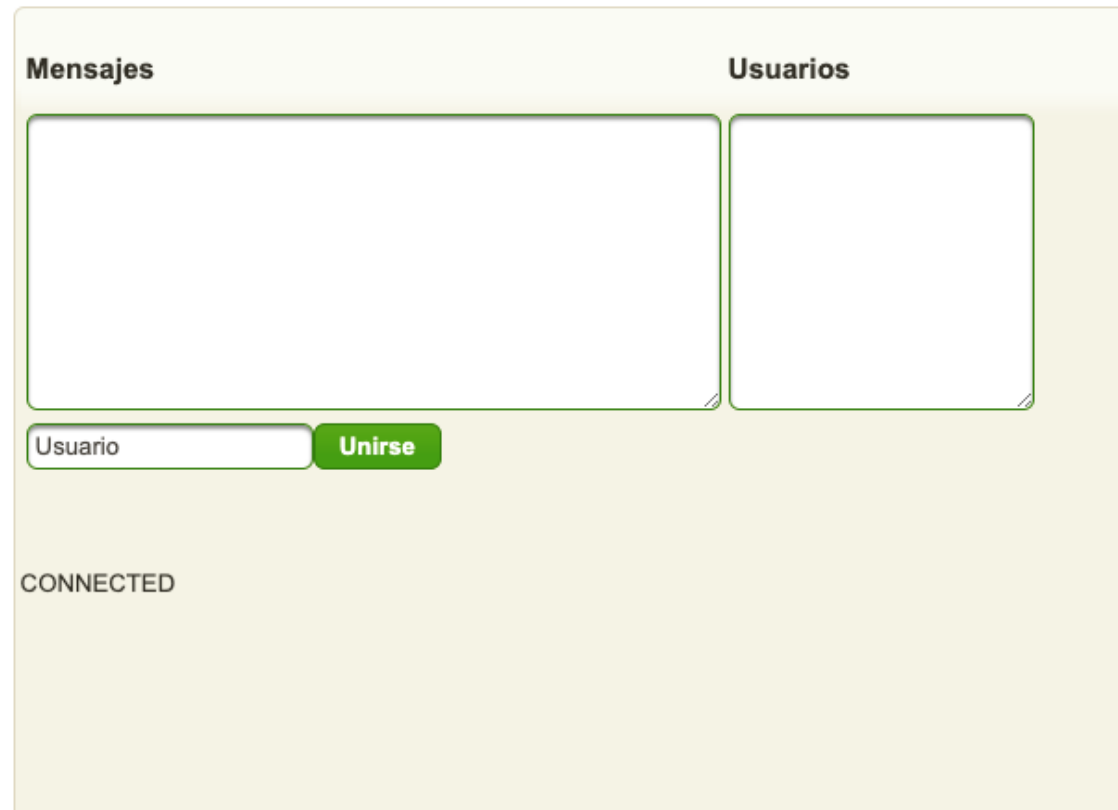
- Indican los métodos a los que se tiene que llamar cuando se abre y cuando se cierra la sesión WebSocket
- El parámetro peer define el cliente solicitando la apertura o el cierre de la conexión

## ▶ @OnMessage

- Indica el método que recibe el mensaje WebSocket entrante
- El parámetro message es el mensaje recibido
- El parámetro client es el cliente que ha enviado el mensaje
- El método envía cada mensaje recibido a todos los clientes conectados (peers)

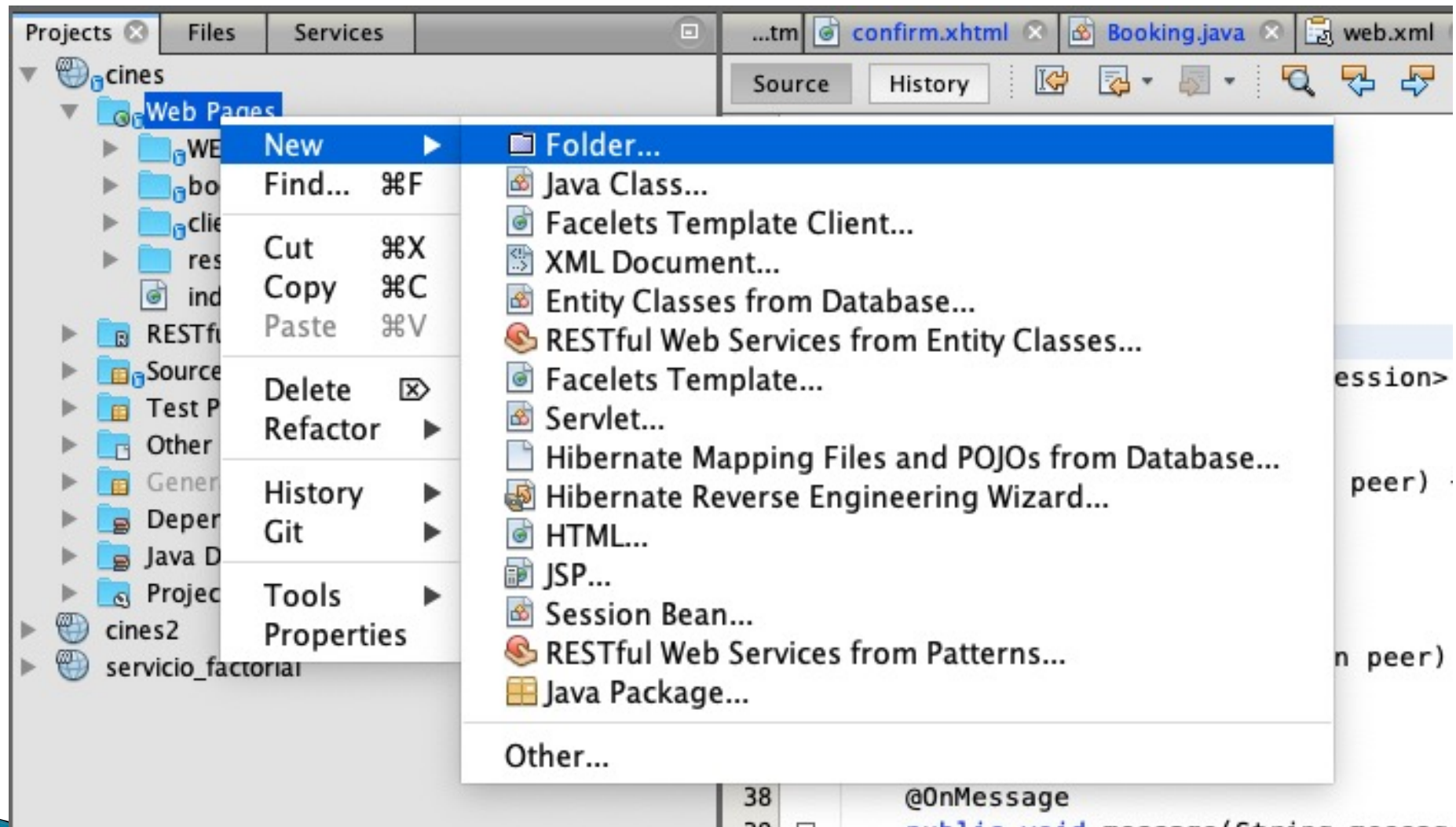
# Chatroom

- ▶ Creamos un nuevo directorio dentro de las páginas web llamado “chat”
  - En “Web Pages”, seleccionar New/Folder, y poner nombre “chat”

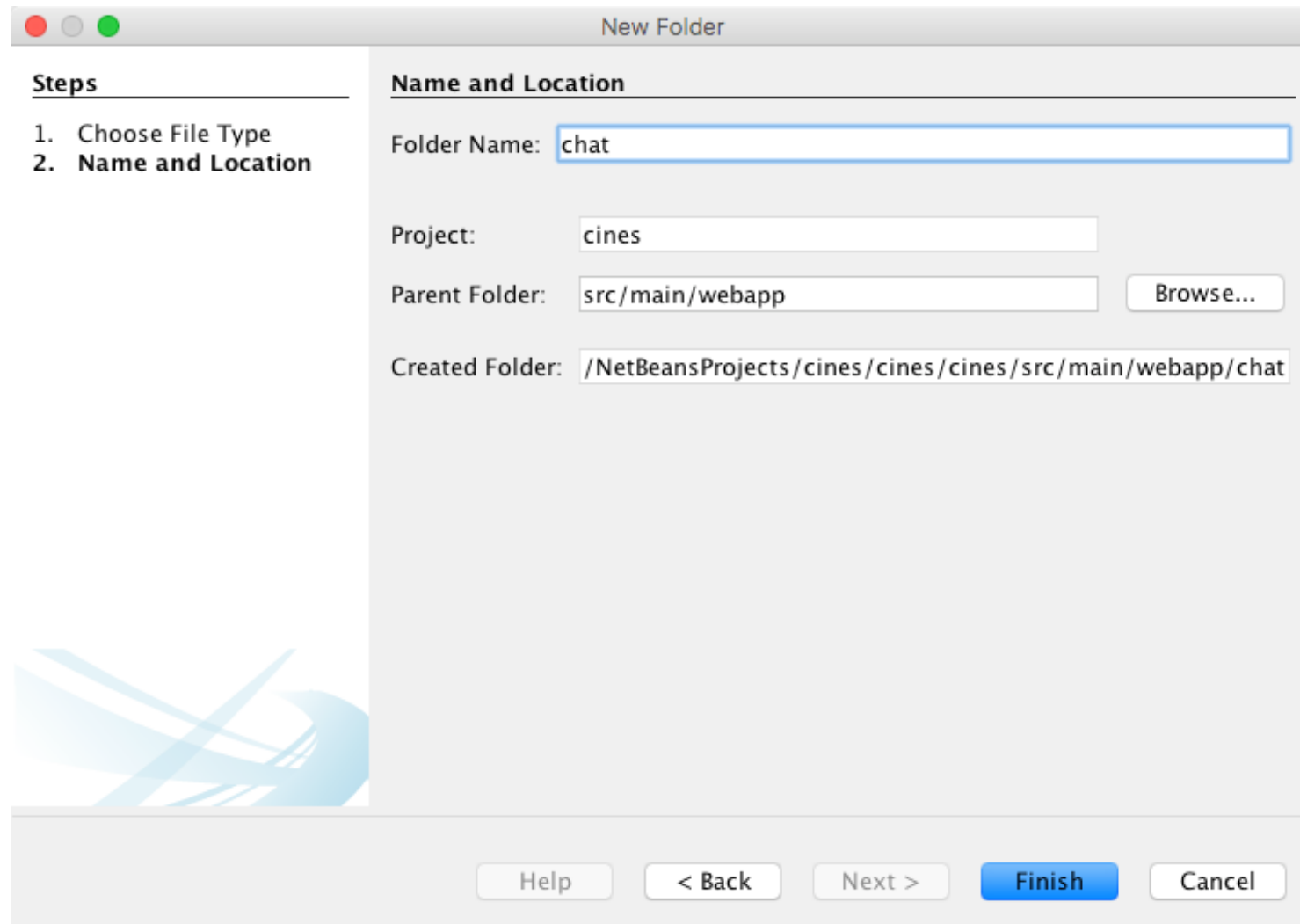


The image shows a web chatroom interface with a light beige background. At the top, there are two columns: "Mensajes" (Messages) on the left and "Usuarios" (Users) on the right. Below these columns are two large, empty rectangular boxes with green borders. At the bottom left, there is a text input field labeled "Usuario" and a green button labeled "Unirse" (Join). Below the input field and button, the word "CONNECTED" is displayed in a small, dark font.

# Chatroom



# Chatroom



The image shows a 'New Folder' dialog box from the NetBeans IDE. The window has a title bar with standard macOS window controls (red, yellow, green buttons) and the title 'New Folder'. On the left side, there is a 'Steps' panel with two steps: '1. Choose File Type' and '2. Name and Location', with the second step being the active one. The main area is titled 'Name and Location' and contains several input fields: 'Folder Name' with the text 'chat', 'Project' with the text 'cines', 'Parent Folder' with the text 'src/main/webapp' and a 'Browse...' button to its right, and 'Created Folder' showing the full path '/NetBeansProjects/cines/cines/cines/src/main/webapp/chat'. At the bottom of the dialog, there are five buttons: 'Help', '< Back', 'Next >', 'Finish' (which is highlighted in blue), and 'Cancel'.

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

Folder Name:

Project:

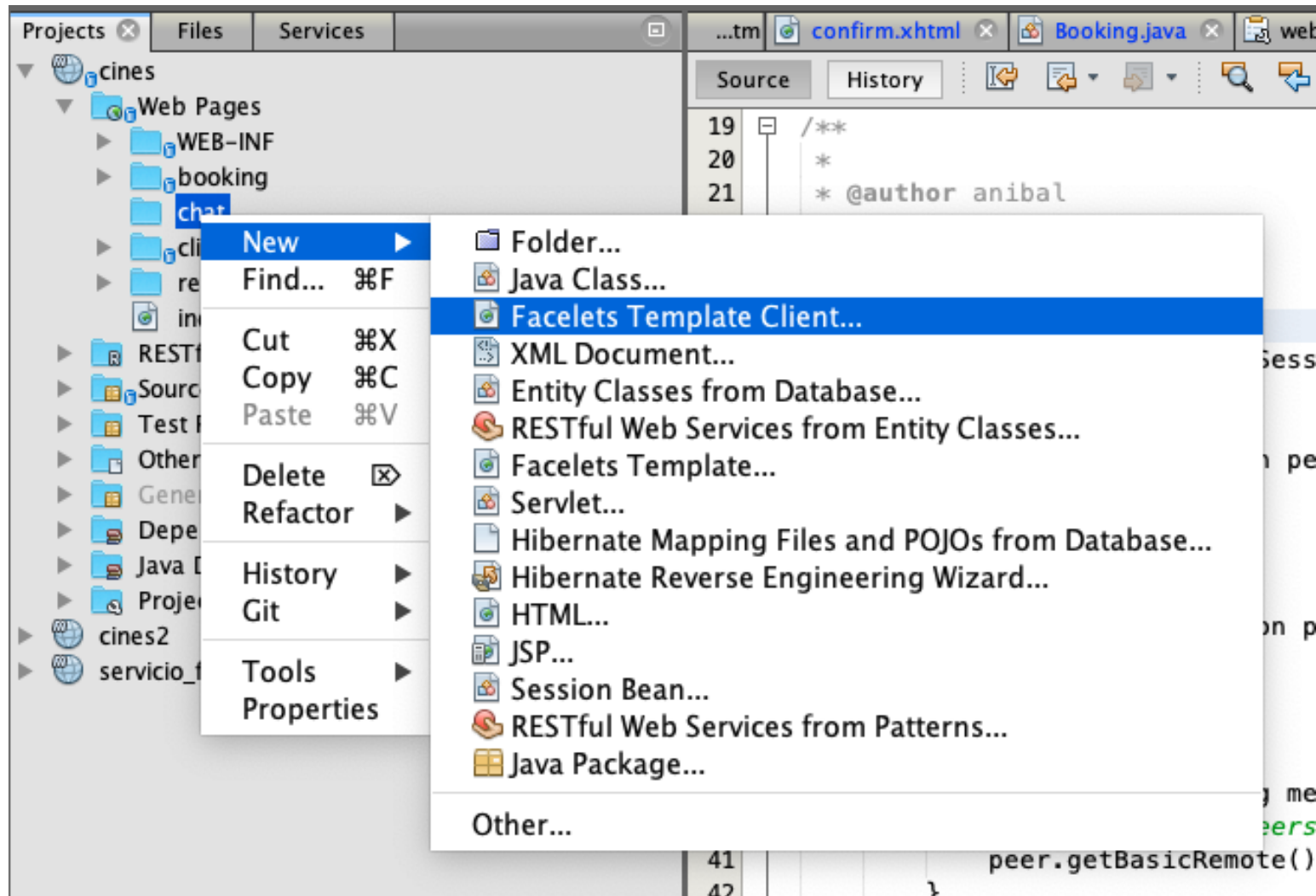
Parent Folder:

Created Folder:

# Chatroom

- ▶ Creamos una nueva página (Facelets Template Client) llamada “chatroom”
- ▶ Esta página permitirá a los usuarios conectarse/desconectarse al chat, ver los usuarios que están conectados, enviar un mensaje, y ver todos los mensajes que se han enviado

# Chatroom



# Chatroom

New Facelets Template Client

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

File Name:

Project:

Folder:

Created File:

Template:

Generated Root Tag: ☒ `<html>`  
☐ `<ui:composition>`

Sections To Generate:

<input checked="" type="checkbox"/>	content
-------------------------------------	---------



# Chatroom

- ▶ Añadir lo siguiente a chatroom.xhtml:

```
<form action="">
  <table>
    <tr><td>
      <h3><p:outputLabel value="Mensajes" /></h3>
      <p:inputTextarea readonly="true" id="chatlog" rows="10" cols="50" autoResize="false" />
    </td>
    <td>
      <h3><p:outputLabel value="Usuarios" /></h3>
      <p:inputTextarea readonly="true" id="users" rows="10" cols="20" autoResize="false" />
    </td>
  </tr>
  <tr>
    <td colspan="2">
      <p:inputText id="texto" value="Usuario"/>
      <p:commandButton id="unirse" onclick="join();" value="Unirse"/>
      <p:commandButton id="enviar" onclick="send_message();" value="Enviar" style="visibility:hidden"/><p/>
      <p:commandButton id="desconectar" onclick="disconnect();" value="Desconectarse" style="visibility: hidden"/>
    </td>
  </tr>
</table>
</form>
<div id="output"></div>
<script language="javascript" type="text/javascript"
  src="{facesContext.externalContext.requestContextPath}/chat/websocket.js">
</script>
```

The image shows a visual mockup of the chatroom interface. It consists of a light yellow background. At the top, there are two labels: 'Mensajes' on the left and 'Usuarios' on the right. Below each label is a large, empty rectangular text area. Underneath these text areas, there is a horizontal input field with the placeholder text 'Usuario' and a green button with the text 'Unirse'. At the bottom of the interface, the word 'CONNECTED' is displayed in a small, dark font.

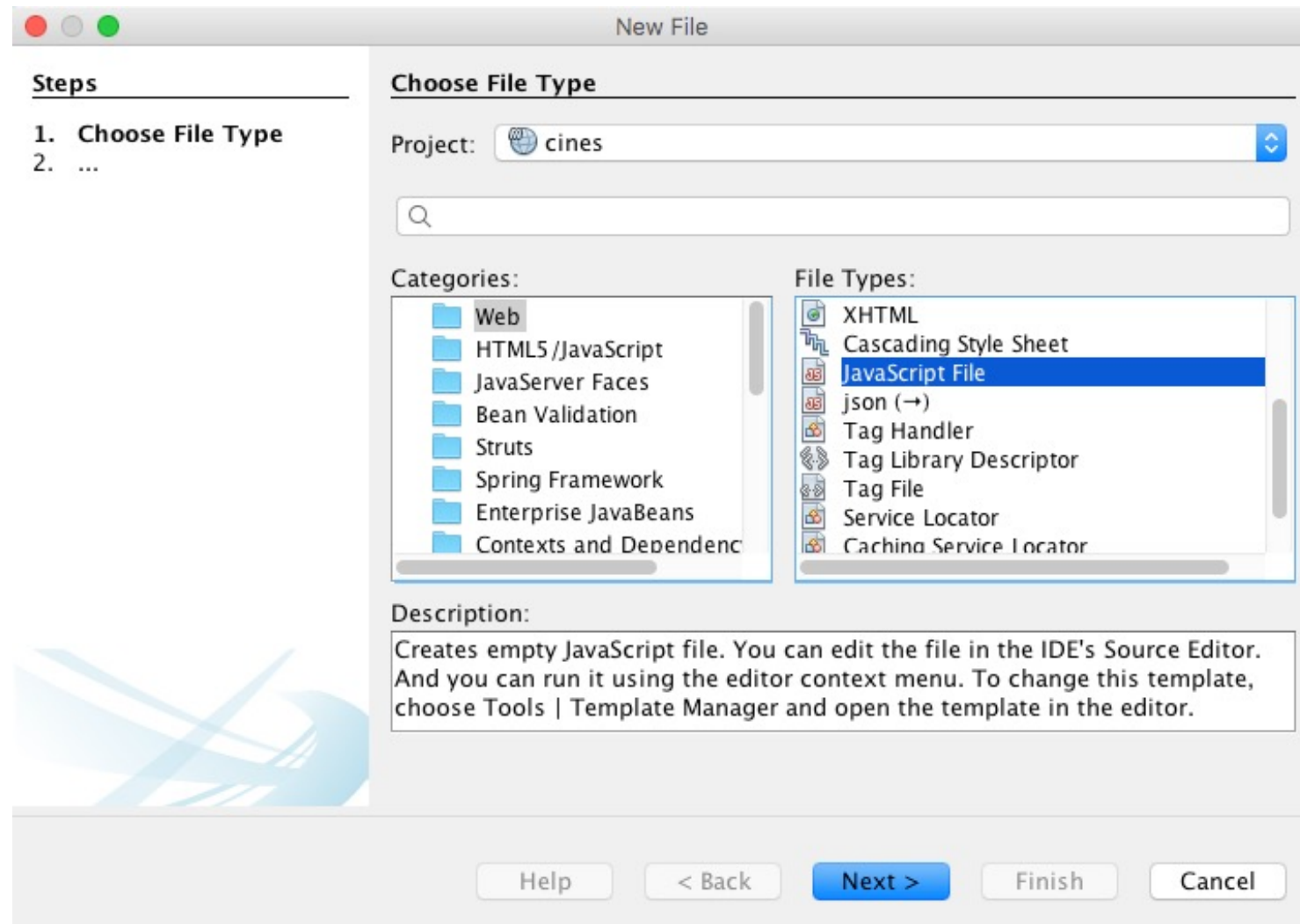
# Chatroom

- ▶ Creamos dos áreas de texto, una para el log de chats y otra para la lista de usuarios conectados
- ▶ El cuadro de texto se usa para que el usuario introduzca su nombre y para que escriba ahí los mensajes a enviar
- ▶ Haciendo click en el botón “Conectarse” se coge el valor del campo de texto con el nombre del usuario y haciendo click en el botón “Enviar” se envía el contenido del campo de texto como mensaje de chat
- ▶ Los métodos javascript correspondientes se invocan cuando el usuario pulsa los botones
- ▶ `<div id="output"></div>` se utiliza para que veamos los mensajes de estado (para comprobar su funcionamiento, no se utilizaría en un chat real)

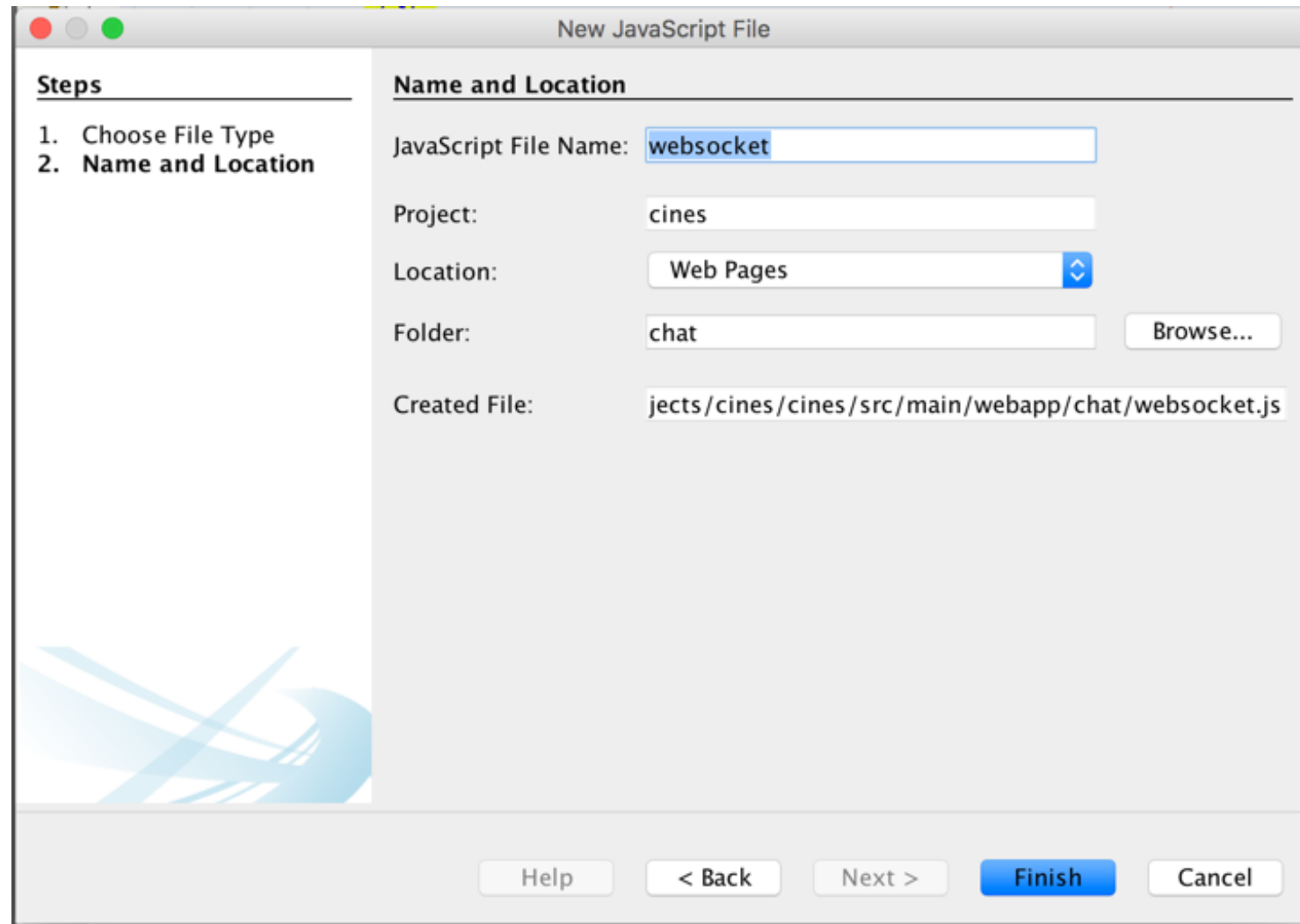
# Cliente JavaScript

- ▶ Creamos el fichero JavaScript websocket.js
  - Botón derecho en “chat”, luego New/Other/Web/JavaScript File.
  - Le daremos el nombre “websocket” y pulsamos en terminar

# Cliente JavaScript



# Cliente JavaScript



**New JavaScript File**

**Steps**

1. Choose File Type
2. **Name and Location**

**Name and Location**

JavaScript File Name:

Project:

Location:

Folder:

Created File:

# Cliente JavaScript

- ▶ Añadimos lo siguiente...
- ▶ Conexión Websocket y variables:

```
var wsUri = 'ws://' + document.location.host  
          + document.location.pathname.substr(0,  
document.location.pathname.indexOf("/faces")) + '/websocket';
```

```
console.log(wsUri);  
var websocket = new WebSocket(wsUri); //Inicializa el websocket  
var textField = document.getElementById("texto");  
var users = document.getElementById("users");  
var chatlog = document.getElementById("chatlog");  
var output = document.getElementById("output");  
var username;
```

The screenshot shows a web application interface with a light yellow background. At the top, there are two labels: "Mensajes" on the left and "Usuarios" on the right. Below "Mensajes" is a large, empty rectangular box with a green border. Below "Usuarios" is a smaller, empty rectangular box with a green border. At the bottom left, there is a text input field labeled "Usuario" and a green button labeled "Unirse". At the bottom center, the text "CONNECTED" is displayed.

# Cliente JavaScript

- Funciones:

```
function join() {  
    username = textField.value;  
    websocket.send(username + " joined");  
    document.getElementById("unirse").style.setProperty("visibility", "hidden");  
    document.getElementById("enviar").style.removeProperty("visibility");  
    document.getElementById("desconectar").style.removeProperty("visibility");  
}  
function send_message() {  
    websocket.send(username + ": " + textField.value);  
}  
function disconnect() {  
    websocket.close();  
    document.getElementById("unirse").style.setProperty("visibility", "hidden");  
    document.getElementById("enviar").style.setProperty("visibility", "hidden");  
    document.getElementById("desconectar").style.setProperty("visibility", "hidden");  
}
```

The screenshot shows a web interface with a light yellow background. At the top, there are two columns: 'Mensajes' on the left and 'Usuarios' on the right, each with a large empty rectangular box. Below these boxes is a text input field labeled 'Usuario' and a green button labeled 'Unirse'. At the bottom of the interface, the text 'CONNECTED' is displayed.

# Cliente JavaScript

► Listeners:

```
websocket.onopen = function (evt) {
    writeToScreen("CONNECTED");
};

websocket.onclose = function (evt) {
    writeToScreen("DISCONNECTED");
};

websocket.onmessage = function (evt) {
    writeToScreen("RECEIVED: " + evt.data);
    if (evt.data.indexOf("joined") !== -1) {
        users.innerHTML += evt.data.substring(0, evt.data.indexOf("joined")) + "\n";
    } else {
        chatlog.innerHTML += evt.data + "\n";
    }
};

websocket.onerror = function (evt) {
    writeToScreen('<span style="color: red;">ERROR:</span> ' + evt.data);
};

function writeToScreen(message) {
    var pre = document.createElement("p");
    pre.style.wordWrap = "break-word";
    pre.innerHTML = message;
    output.appendChild(pre);
}
```



# Cliente JavaScript

- ▶ WebSocket(...) inicializa el Websocket
- ▶ Los manejadores de eventos del ciclo de vida se registran usando los mensajes onXXX
- ▶ Los listeners son los siguientes:
  - onopen – la conexión Websocket se ha iniciado
  - onmessage – mensaje Websocket recibido
  - onerror – ha ocurrido un error durante la comunicación
  - onclose – la conexión Websocket se ha terminado
- ▶ La función writeToScreen se utiliza para escribir el estado en el navegador
- ▶ La función join envía un mensaje al endpoint de que un usuario se ha unido al chat, el endpoint envía el mensaje a todos los clientes conectados
- ▶ La función send\_message une el nombre de usuario y el valor del campo de texto y procede de igual manera que la función join
- ▶ Fijaros que el método onMessage toma la decisión de si es un usuario de chat que se une o un mensaje. En cada caso actualiza un campo de texto u otro

# Actualizamos el menú

- ▶ Por último, en el menú (en template.xhtml) ponemos el link para la página de chat:

```
<p:menuitem value="Chat" outcome="/chat/chatroom.xhtml" />
```

# Resultado

**Cines Luz de Castilla - Segovia**

**Menú**

[Inicio](#)  
[Booking](#)  
[Info películas](#)  
[Chat](#)

**Mensajes**

**Usuarios**

Usuario

Unirse

CONNECTED

# Resultado

**Cines Luz de Castilla - Segovia**

Menú

[Inicio](#)  
[Booking](#)  
[Info películas](#)  
[Chat](#)

Mensajes

Usuario 1

Enviar

Desconectarse

CONNECTED

RECEIVED: Usuario 1 joined

Usuarios

Usuario 1

# Resultado

Cines Luz de Castilla - Segovia

Menú

[Inicio](#)

[Booking](#)

[Info películas](#)

[Chat](#)

Mensajes

Usuario 2: Hola usuario 1

Usuario 1

Enviar

Desconectarse

CONNECTED

RECEIVED: Usuario 1 joined

RECEIVED: Usuario 2 joined

RECEIVED: Usuario 2: Hola usuario 1

Usuarios

Usuario 1  
Usuario 2

# Resultado

Cines Luz de Castilla - Segovia

Menú

[Inicio](#)

[Booking](#)

[Info películas](#)

[Chat](#)

Mensajes

Usuario 2: Hola usuario 1  
Usuario 3: Hola usuarios 1 y 2

Usuarios

Usuario 1  
Usuario 2  
Usuario 3

Usuario 1

Enviar

Desconectarse

CONNECTED

RECEIVED: Usuario 1 joined

RECEIVED: Usuario 2 joined

RECEIVED: Usuario 2: Hola usuario 1

RECEIVED: Usuario 3 joined

RECEIVED: Usuario 3: Hola usuarios 1 y 2

**¿PREGUNTAS?**