10 : Web Sockets

IT4206 – Enterprise Application Development

Level II - Semester 4

© e-Learning Centre, UCSC

Overview

- This topic will discuss the WebSockets and develop simple WebSocket using Java.

© e-Learning Centre, UCSC

Intended Learning Outcomes

- At the end of this lesson, you will be able to;
  - Explain the behaviour of WebSockets.
  - Compare WebSockets with HTTP.
  - Implement a WebSocket.

© e-Learning Centre, UCSC

List of sub topics

10.1 Developing a WebSocket server endpoint

10.2 Developing WebSocket clients

© e-Learning Centre, UCSC

WebSockets

- Client can establish a lightweight connection with the server.
- WebSockets facilitates a bi-directional communication.
- Servers can push data to the connected clients when only needed.
- Once the connection is established, the client and the server can send simple information between each other without reconnecting.
- The connection can be destroyed when client or server decided to do so.

© e-Learning Centre, UCSC

Why WebSockets?

- Consider an auction web site as an example.
  - Auction items are posted in the web site.
  - Users can place their bids on items within the given time.
  - The item will be sold to the highest bidder.
  - Therefore, the users need to view the current bid all the time.
  - Lets assume that the site need to refresh the price every 2 seconds.
    - For 1 minute there will be 30 updates for a single user.
    - If the header contains 512 bytes, for 1 minute, 15Kb of data need to transferred as header information.
    - But if there are only 15 bids and the data contains 64 bytes, total will be 0.9 Kb.
  - With WebSockets once the connection is established, only the information will be send, without the repeated data (header).

© e-Learning Centre, UCSC

HTTP vs WebSockets

- HTTP
  - Unidirectional - Clients can send request to the server and server responds.
  - HTTP is a stateless protocol and use TCP to guarantee the delivery of data.
  - When client send a HTTP request, a TCP connection opens and after sending the respond the connection closes. If there are 10 requests, 10 TCP connections needed.
  - For each HTTP request, HTTP header information need to be communicated.
- WebSocket
  - Bidirectional - Full duplex communication.
  - It is a stateful protocol. Connection between client and server stays util the it is terminated by one of the parties.

© e-Learning Centre, UCSC

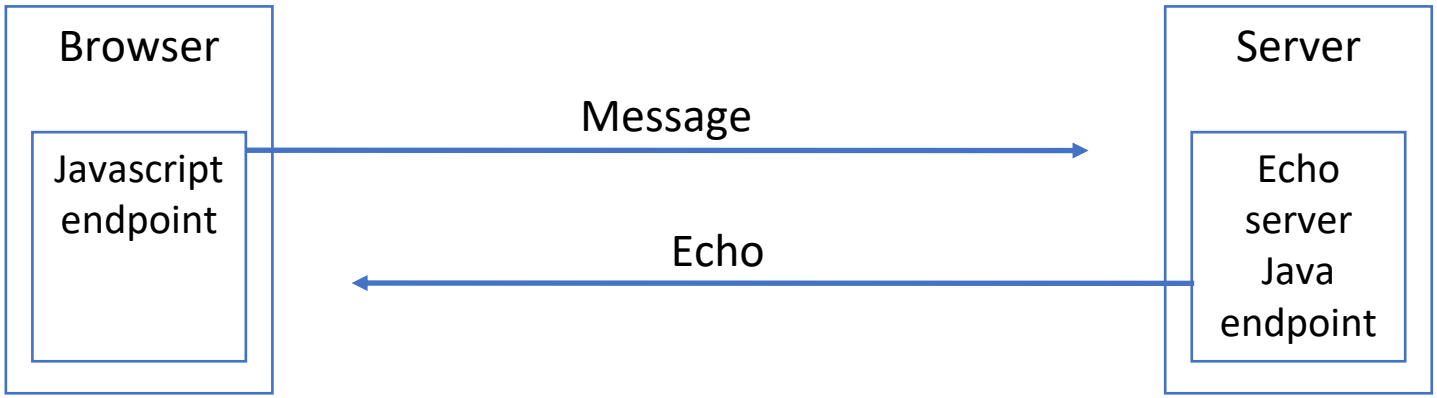
WebSocket Protocol

- WebSocket protocol is a network protocol which enables two connected peers to have full duplex communication over a single TCP connection.
- Connection establishment
  - Connection initiator will send a specially formulated HTTP request with the URL of the WebSocket endpoint which it wants to connect.
  - This is the ***opening handshake***.
  - If the server accept the request, it will generate a specially formulated HTTP response called the ***opening handshake response*** and send to the client.
- Connection termination
  - Either party will decide to terminate the connection or external factors like timeouts or problems with physical network will terminate the connection.

© e-Learning Centre, UCSC

Lets Create a Simple WebSocket

- Lets create a simple echo server, which echos the message send by the client.
- For this we will use a Java endpoint as the server and Javascript endpoint as the client.



© e-Learning Centre, UCSC

Lets Create a Simple WebSocket : WebSocket server endpoint

- Create the WebSocket endpoint
  - We can use the Tomcat server to host the application.
  - Create a new project in the Tomcat's webapp directory as "echoserver"
  - Create a Java class file as "EchoServer.java" inside the "/echoserver/src" as we did in topic 7 (Servlets)

```
import javax.websocket.server.ServerEndpoint;
import javax.websocket.OnMessage;

@ServerEndpoint("/echo")
public class EchoServer {
     @OnMessage
     public String echo(String incomingMessage) {
            return "I am Echo Server. I got this (" + incomingMessage + "), let me send it back !";
          }
}
```

© e-Learning Centre, UCSC

Lets Create a Simple WebSocket : WebSocket server endpoint

- @ServerEndpoint annotation inform the Java platform that the class is going to be a WebSocket endpoint. Parameter is the relative URI (Uniform Resource Identifier)
- @OnMessage annotation is used to handle incoming message.
- Compile the Java class.

javac -cp <path to>/websocket-api.jar EchoServer.java -d ../WEB-INF/classes

```
import javax.websocket.server.ServerEndpoint;
import javax.websocket.OnMessage;

@ServerEndpoint("/echo")
public class EchoServer {
     @OnMessage
     public String echo(String incomingMessage) {
            return "I am Echo Server. I got this (" + incomingMessage + "), let me send it back !";
          }
}
```

© e-Learning Centre, UCSC

Lets Create a Simple WebSocket : WebSocket Client

- Creating the WebSocket client using HTML and JavaScript.

```
<html>
    <head>
        <title>Web Socket Client</title>
        <script language="javascript" type="text/javascript">
            var webSocket;
            function init() {
                display = document.getElementById("display");
              }
            function echoMessage() {
                var uri = "ws://localhost:8080/echoserver/echo";
                webSocket = new WebSocket(uri);
                webSocket.onopen = function (val) {
                    sendMessage(myMessage.value);
                };
                webSocket.onmessage = function (val) {
                    show("Received message: <span style='color: blue;'>" + val.data +"</span>");
                    webSocket.close();
                };
                webSocket.onerror = function (val) {
                    show('<span style="color: red;">Connection Error</span> ');
                    webSocket.close();
                };
              }
            function sendMessage(message) {
                webSocket.send(message);
                show("Sent Message: " + message);
              }
            function show(message) {
                var pre = document.createElement("p");
                pre.style.wordWrap = "break-word";
                pre.innerHTML = message;
                display.appendChild(pre);
              }
              window.addEventListener("load", init, false);
        </script>
    </head>
```

© e-Learning Centre, UCSC

Lets Create a Simple WebSocket : WebSocket Client

- Creating the WebSocket client using HTML and JavaScript.

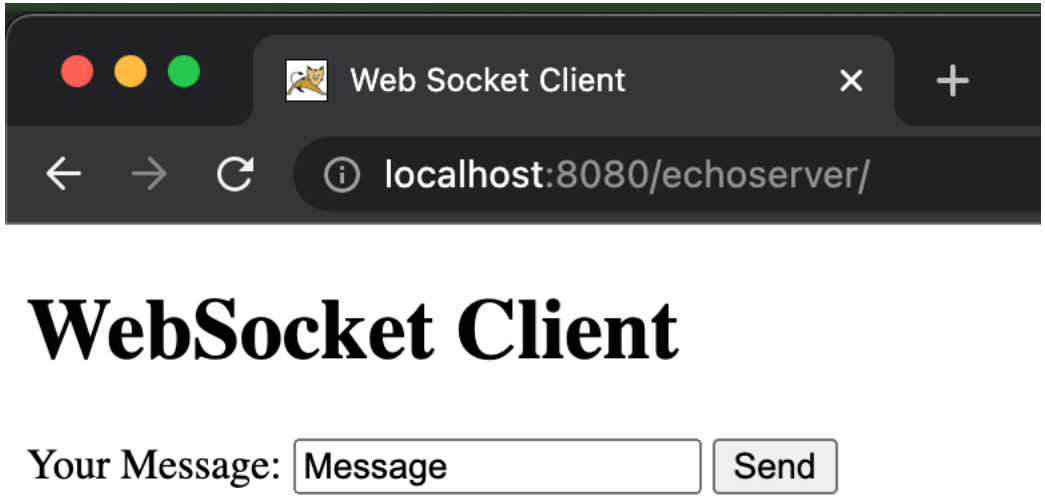
Code continue…

```
    <body>
        <h1>WebSocket Client</h1>
        <form action="">
            Your Message: <input id="myMessage" name="message" value="Message" type="text">
            <input onclick="echoMessage()" value="Send" type="button">
        </form>
        <div id="display"></div>
    </body>
</html>
```

© e-Learning Centre, UCSC

Lets Create a Simple WebSocket : WebSocket Client

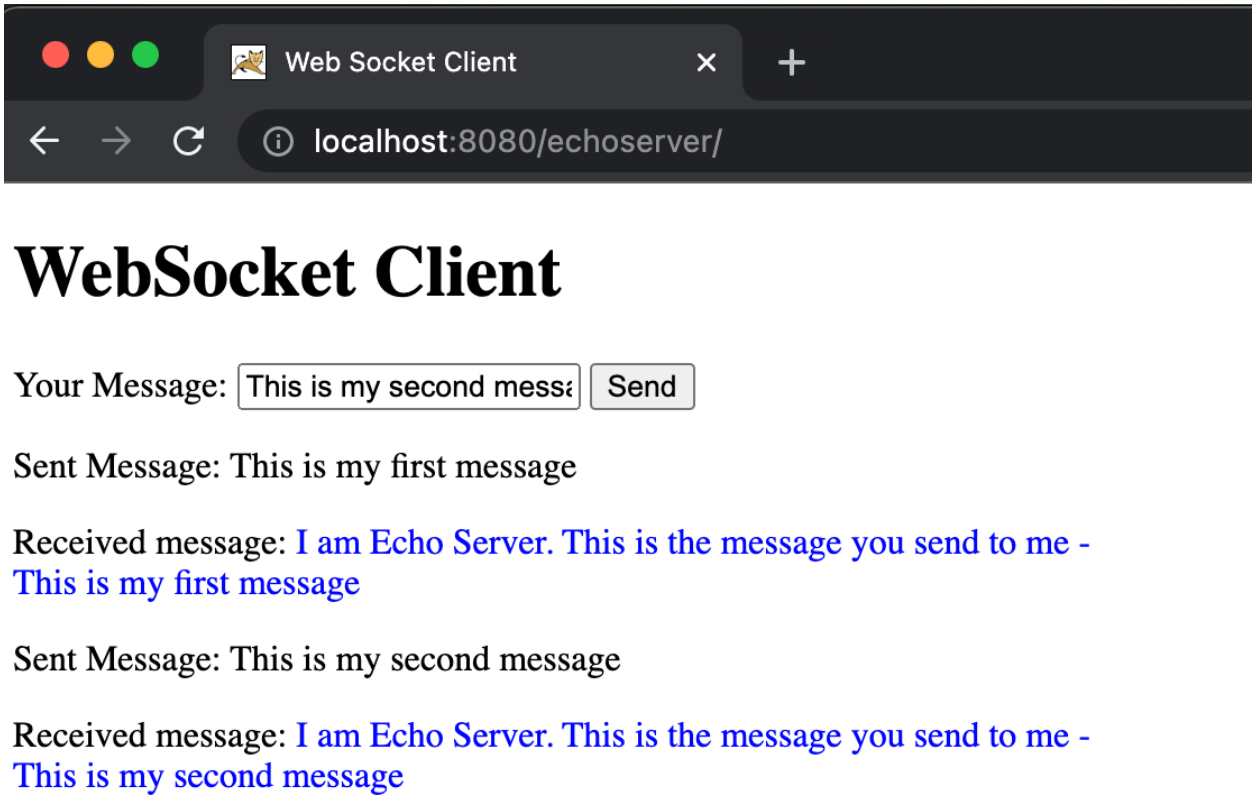
- Start the Tomcat server and goto : http://localhost:8080/echoserver/



© e-Learning Centre, UCSC

Lets Create a Simple WebSocket : WebSocket Client

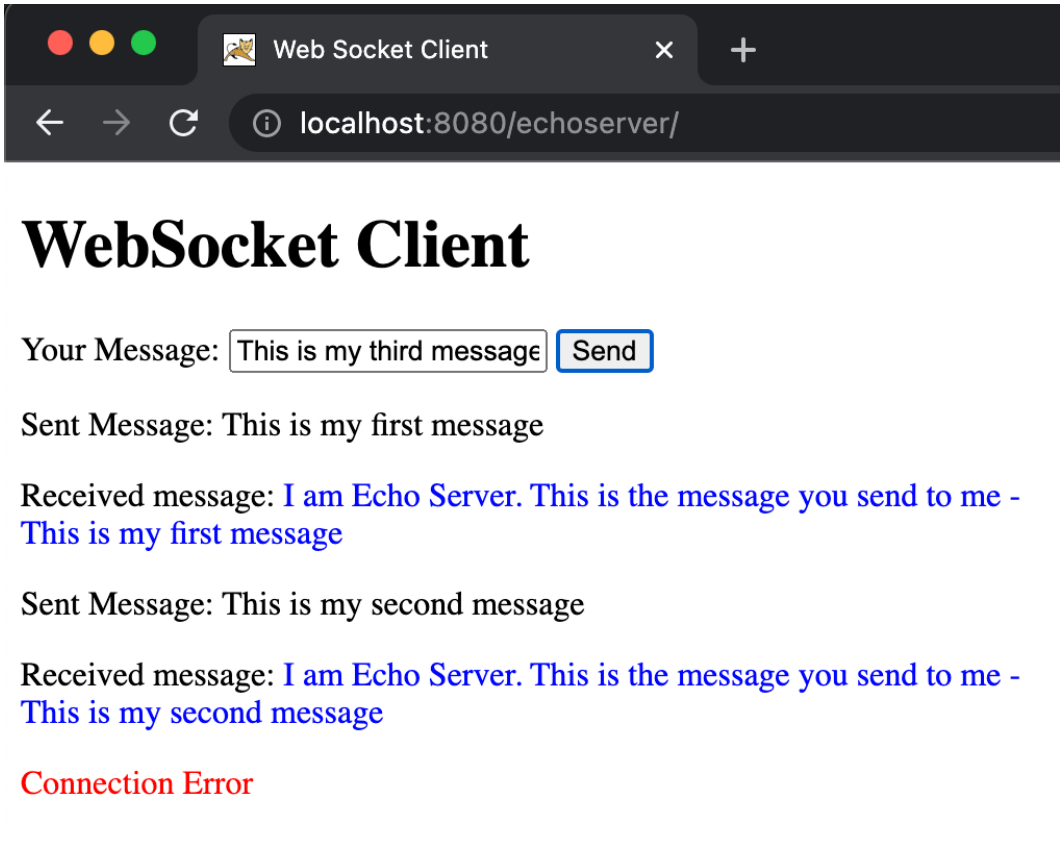
- Send messages



© e-Learning Centre, UCSC

Lets Create a Simple WebSocket : WebSocket Client

- Shutdown the server and try again



© e-Learning Centre, UCSC

Accepting the Connection

- When initiating the WebSocket, first the HTTP request and responds occurs.
- This is the *WebSocket opening handshake*.

```
Hypertext Transfer Protocol
  GET /echoserver/echo HTTP/1.1\r\n
    > [Expert Info (Chat/Sequence): GET /echoserver/echo HTTP/1.1\r\n]
      Request Method: GET
      Request URI: /echoserver/echo
      Request Version: HTTP/1.1
    Host: localhost:8080\r\n
    Connection: Upgrade\r\n
    Pragma: no-cache\r\n
    Cache-Control: no-cache\r\n
```

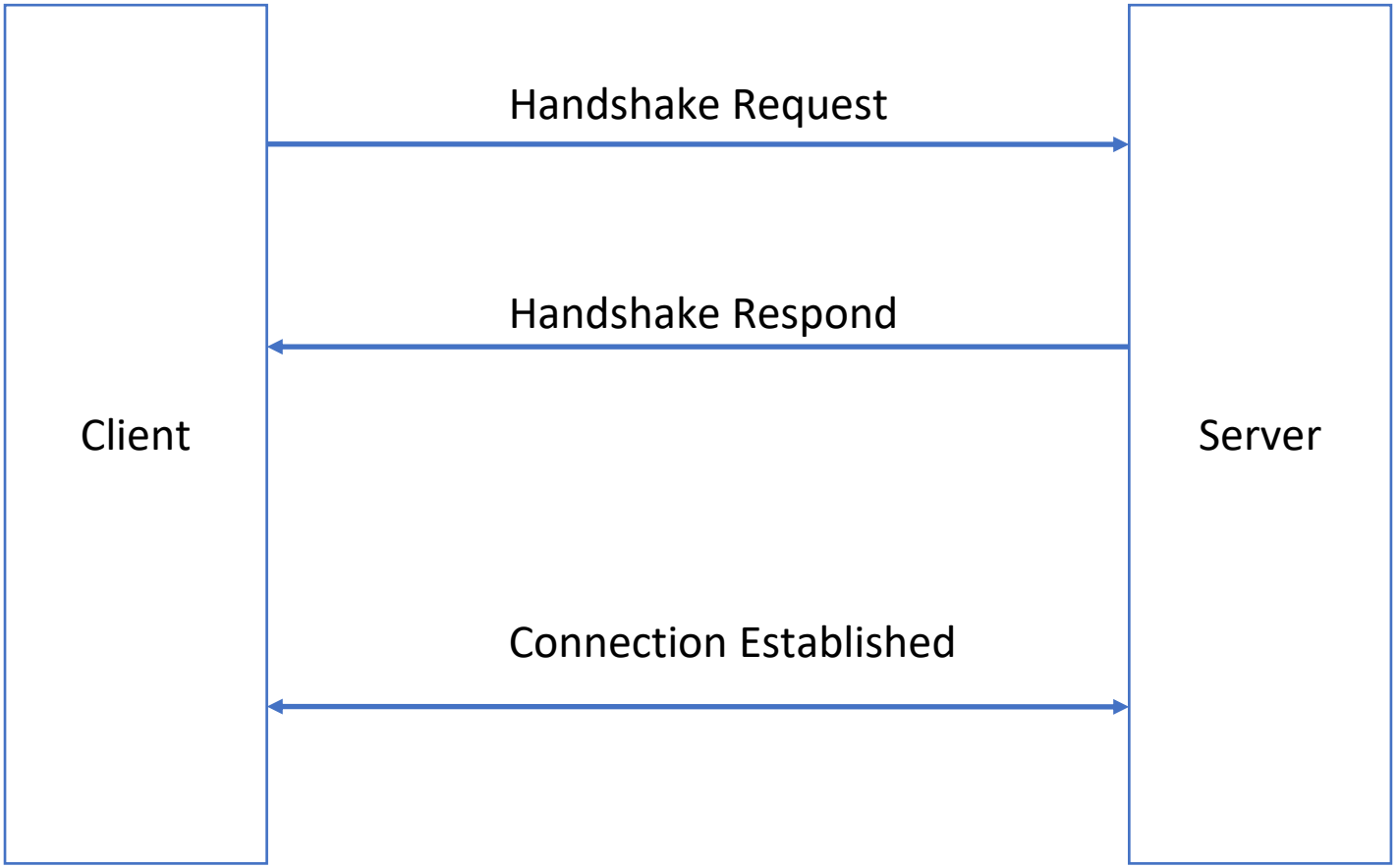
Request

Respond

```
Hypertext Transfer Protocol
  HTTP/1.1 101 \r\n
    > [Expert Info (Chat/Sequence): HTTP/1.1 101 \r\n]
      Response Version: HTTP/1.1
      Status Code: 101
      [Status Code Description: Switching Protocols]
    Upgrade: websocket\r\n
    Connection: upgrade\r\n
    Sec-WebSocket-Accept: SnQ1NjKad2Pl0RsISoVy3h3ZyaI=\r\n
    Sec-WebSocket-Extensions: permessage-deflate;client_max_window_bits=15\r\n
```

© e-Learning Centre, UCSC

Accepting the Connection



© e-Learning Centre, UCSC

WebSocket Messaging

- After *WebSocket opening handshake,* TCP connection established.
- WebSocket protocol define messaging protocol on top of TCP connection.
- Different WebSocket protocol frames sent forward and backward over the TCP connection.

© e-Learning Centre, UCSC

WebSocket Messaging

```
WebSocket
    1... .... = Fin: True
    .100 .... = Reserved: 0x4
    .1.. .... = Per-Message Compressed: True
    .... 0001 = Opcode: Text (1)
    1... .... = Mask: True
    .001 1001 = Payload length: 25
    Masking-Key: 976e96e7
    Masked payload
    Payload
Line-based text data (1 lines)
    This is my first message
```

Message send

Message received

```
WebSocket
    1... .... = Fin: True
    .100 .... = Reserved: 0x4
    .1.. .... = Per-Message Compressed: True
    .... 0001 = Opcode: Text (1)
    0... .... = Mask: False
    .100 0110 = Payload length: 70
    Payload
Line-based text data (1 lines)
    I am Echo Server. This is the message you send to me - <br>This is my first message
```

© e-Learning Centre, UCSC

WebSocket Lifecycle

- In servlets, there is a simple request and response interactions. It is totally independent of the next interactions.
- But in WebSocket protocol defines a longer lived dedicated TCP connection between the client and the server.
- There are two types of frames in the WebSocket protocol
  - Control frames - Used to transmit control messages such as *close frame* which used to inform that the sender is going to close the connection. *Ping* and *Pong* frames are used to check health of the connection.
  - Data frames - used to transmit application data.

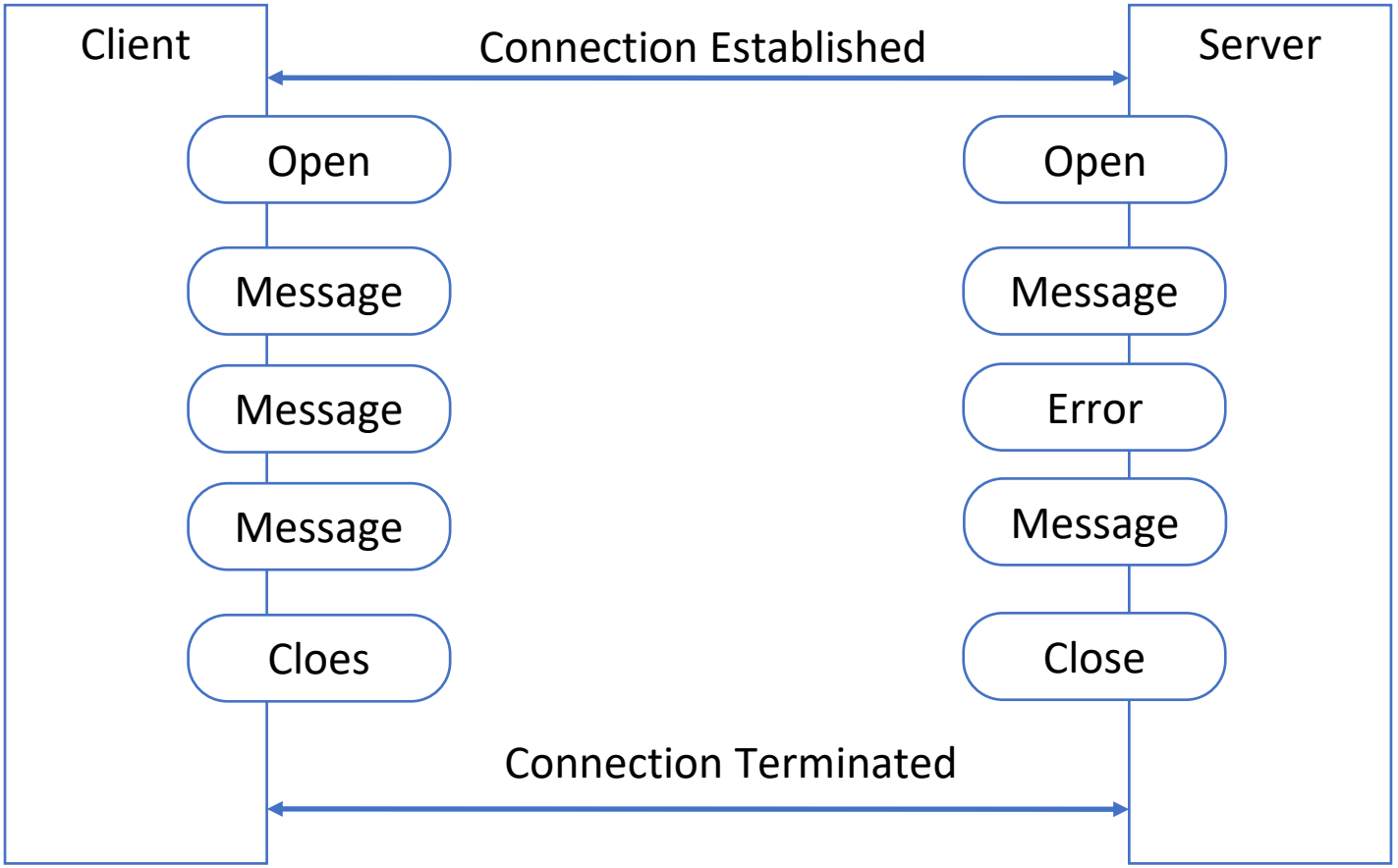
© e-Learning Centre, UCSC

WebSocket Lifecycle

- Open event - notify that the connection to the other end of the WebSocket session is established. After getting the open notification, end points can start communication.
- Message event - Receiving a message from other end of the connection.
- Error event - this occur if there is an error in the transmitted data or in the connection.
- Close event - if either party of the connection wish to end the communication, this event is generated.

© e-Learning Centre, UCSC

WebSocket Lifecycle



© e-Learning Centre, UCSC

Annotations

- @OnOpen - used to annotate the function should called when a new connection is established to the particular endpoint.
- @OnMessage - used to annotate the functions which handle incoming messages.
- @OnError - used to annotate a method which handle errors.
- @OnClose - used to annotate the closing function.

Read More : https://docs.oracle.com/javaee/7/tutorial/websocket004.htm

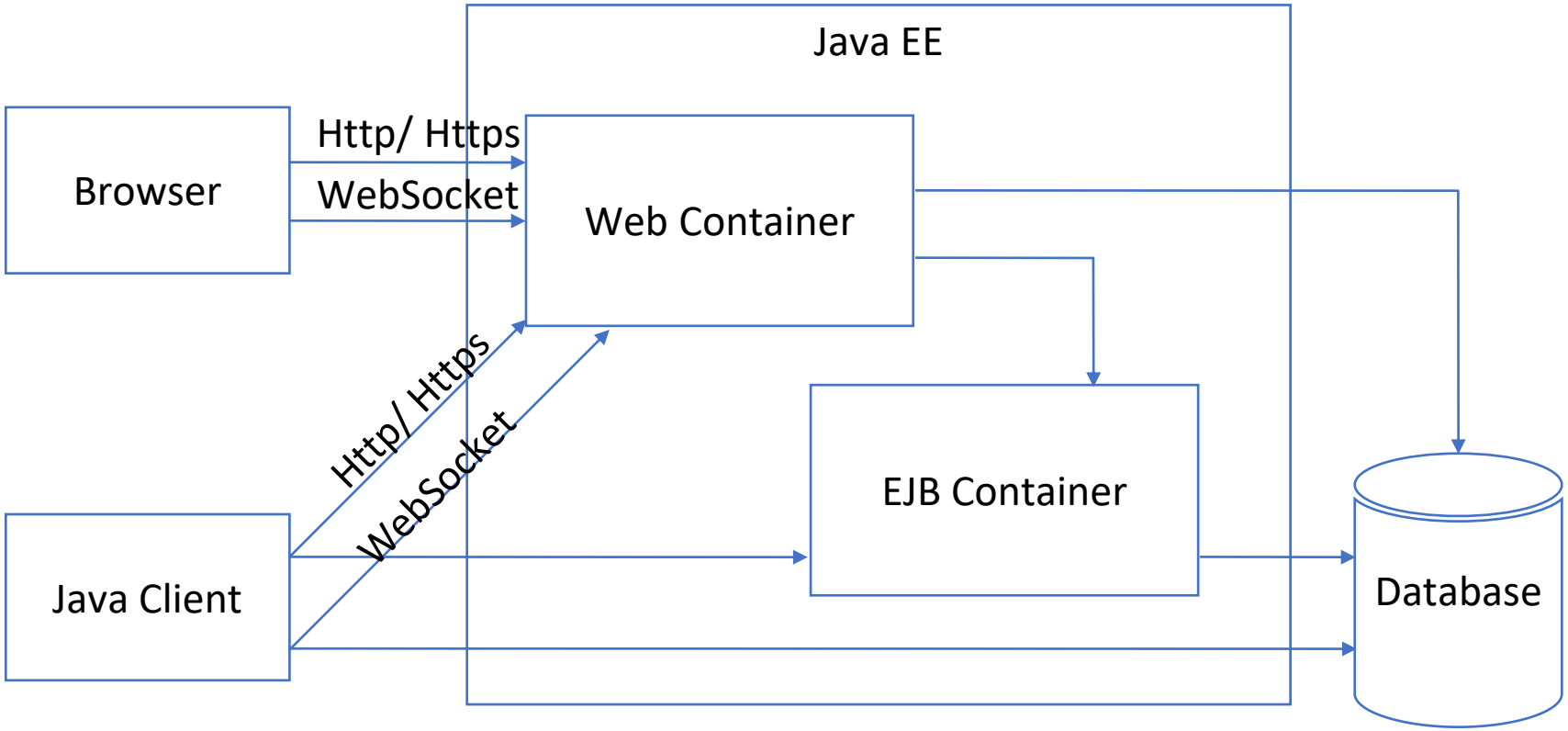
© e-Learning Centre, UCSC

Java WebSockets in the Java EE

- The Java EE platform has two server-side containers
  - Web container - hosts *Servlets, JavaServer Pages, JavaServer Faces, WebService endpoints, WebSocket endpoints*
  - EJB container - hosts *Enterprise JavaBeans*
- Users access these Java EE applications through a browser client, or Java or JavaFX rich client application.
- Browser interactions are occurred over HTTP and WebSocket connections.

© e-Learning Centre, UCSC

Java WebSockets in the Java EE



© e-Learning Centre, UCSC

Java WebSockets in the Java EE

- WebSocket endpoint place in the web container.
- It want to share application state with other web components in the same web application, whether it be application state associated with one client, or application state shared across all clients of the application.
- Once a WebSocket endpoint can share data with other web components and interact with EJB components, it is truly part of the wider Java EE application that contains it.

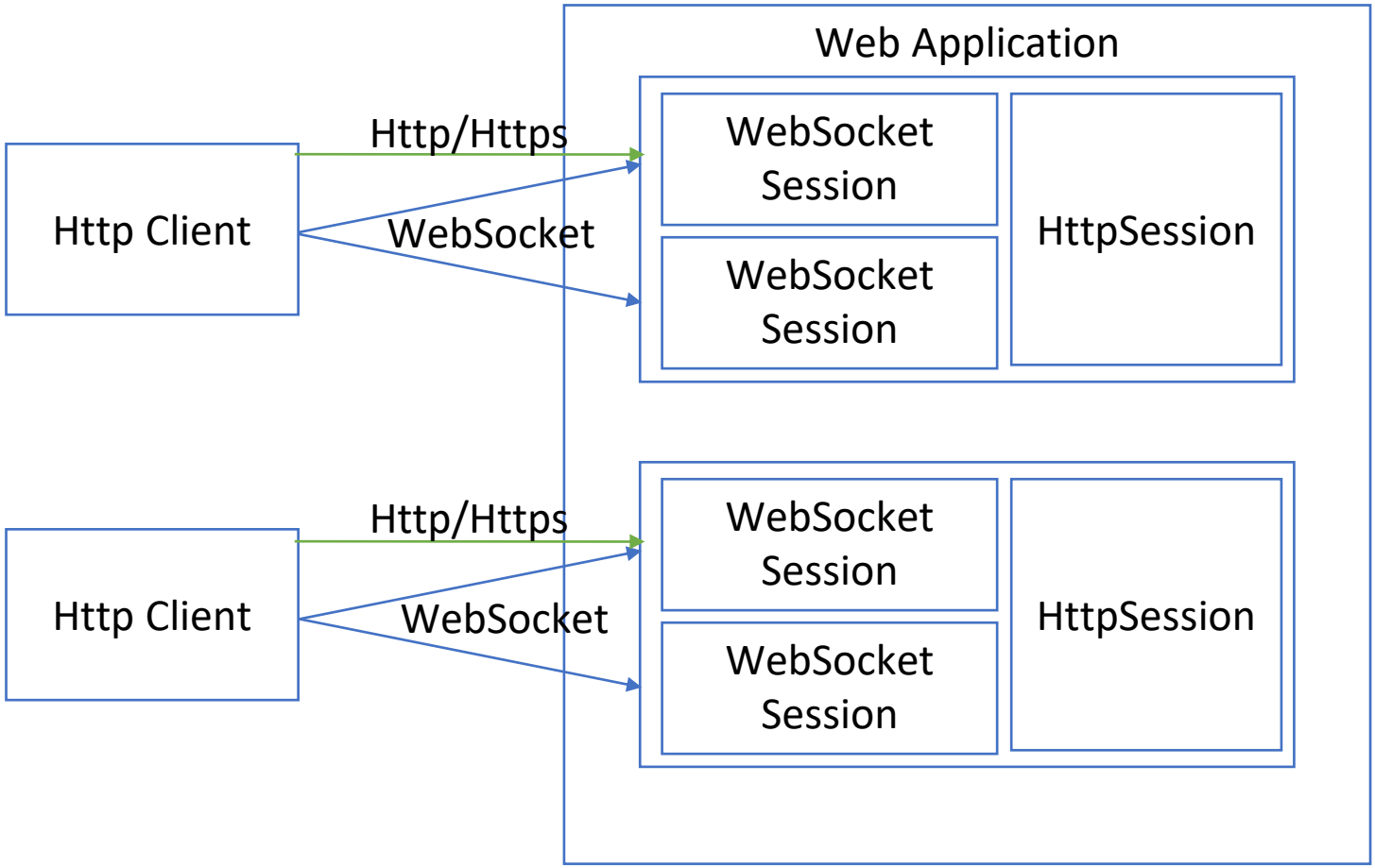
© e-Learning Centre, UCSC

Share Web Application Status

- *HttpSession* object represents a sequence of HTTP interactions from the same HTTP client to a web application.
- *WebSocket* connections begin with the *opening handshake*, and because the opening handshake is an HTTP interaction, each *WebSocket* Session object is related to the *HttpSession* object that was initiated or already in place during the opening handshake that established it.
  - Assume that you have a web application with a web page having two JavaScript WebSocket client endpoints and Java WebSocket server endpoints.
  - Each time when a user downloaded the web page and caused the WebSockets to connect, it would have an HttpSession associated with two separate WebSocket Sessions.

© e-Learning Centre, UCSC

Share Web Application Status



© e-Learning Centre, UCSC

Share Web Application Status

- *HttpSession* object holds a dictionary that developers may use to hold application data.
- Because there is one *HttpSession* instance per user of the application, this application data is associated with the user.
- All Java Servlet and Java Servlet–based technologies have access to the *HttpSession* object, so they are able to share application state associated with a particular user.
- Java *WebSocket* endpoints can get access to the *HttpSession* object associated with the *WebSocket Sessions* in order to share application data associated with a particular user with other web components.

© e-Learning Centre, UCSC