Webontwikkeling 4

Reverse Ajax

Elke Steegmans

AJAX

- Sending data from the client to the server
- HTTP protocol is not suitable for AJAX because
  - the server wishes to push information to the client
  - the client and the server wish to hold a long running conversation with many small messages transferred back and forth

Reverse AJAX

- Is being able to send data from the server to the client
- The goal is to let the server push information to the client

Patterns

- AJAX
  - Polling
- Reverse AJAX
  - Long polling
  - Push

Polling

- Problems
  - very wasteful on network resources
  - involves latency

Long Polling

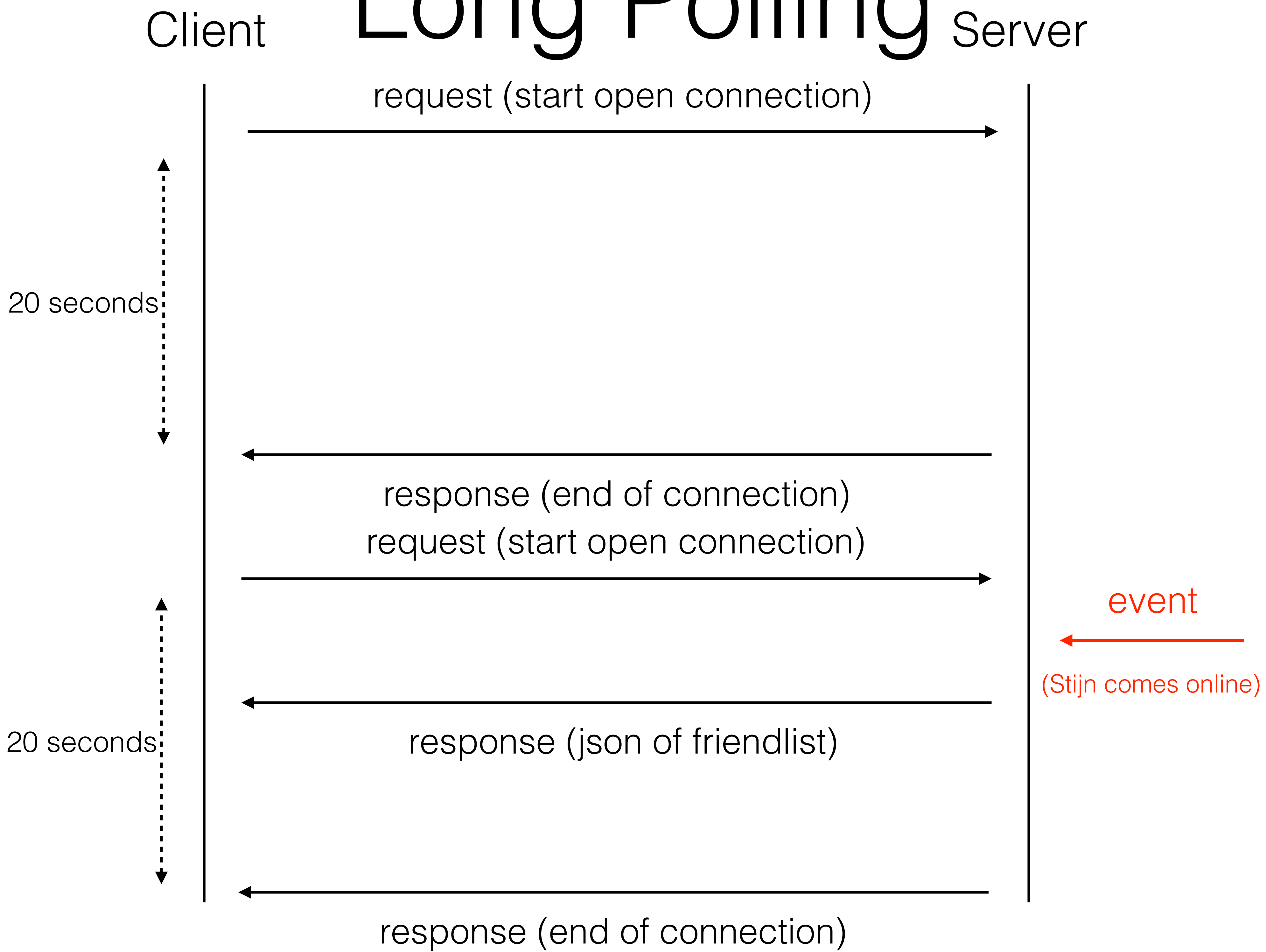
- Is a combination of

  server push

  and

  client pull

Long Polling


Client
Server
request (start open connection)
20 seconds
response (end of connection)
request (start open connection)
event
(Stijn comes online)
20 seconds
response (json of friendlist)
response (end of connection)

Long Polling

- **COMET**
  - client polls the server with a standard HTTP request
    - if the server has a notification for the client
      - => it will send an HTTP response
    - if the server has no notification
      - => it will block the connection until a notification is available, at which point an HTTP response will be returned

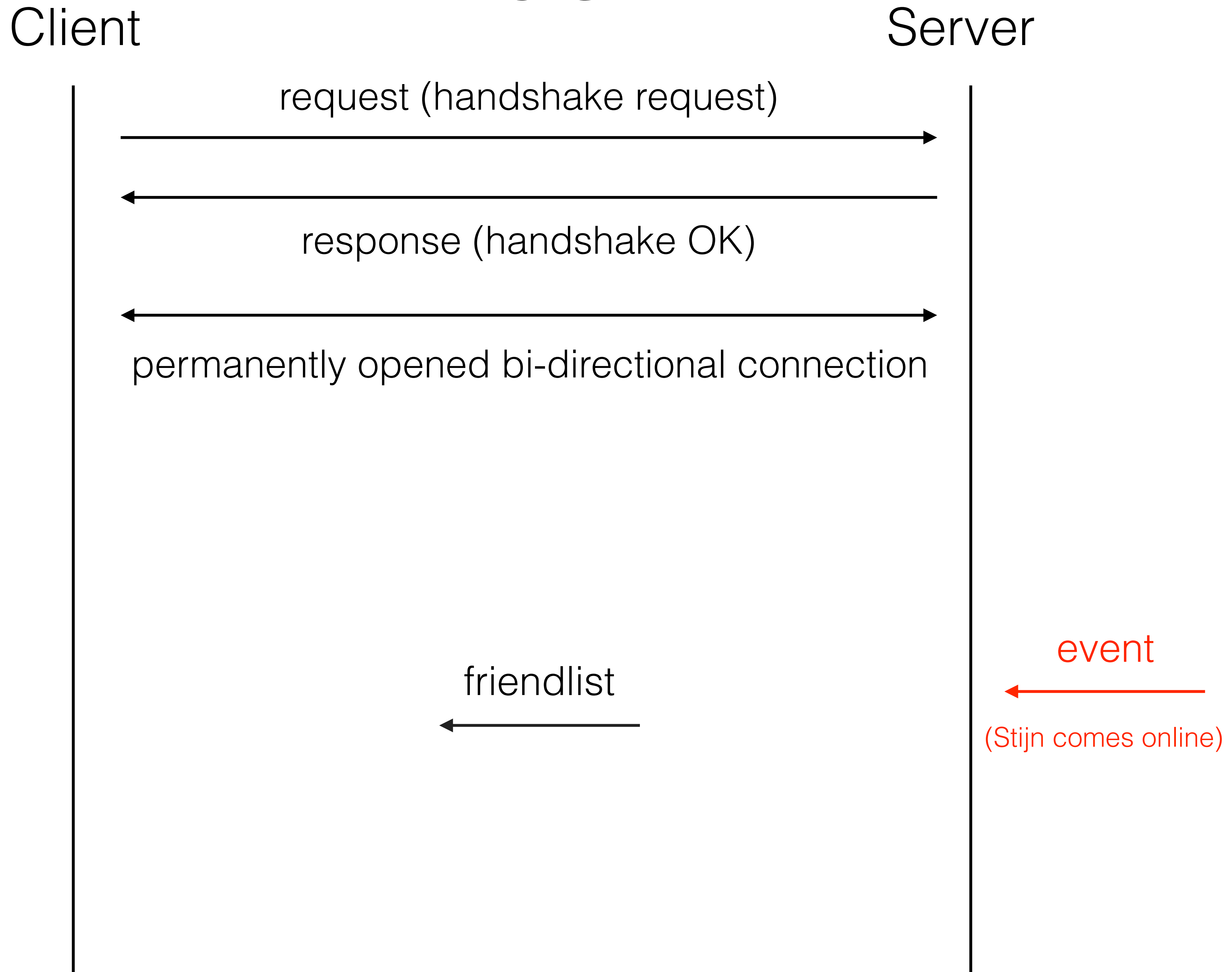
Long Polling

- Problems
  - number of active threads are often limited on the server
  - server gets overloaded
    - solution: server park
  - clients sit behind firewalls, and firewalls are often suspicious for connections that remain open for significant periods of time

Push

- Wouldn't it be great if the server could wake up one morning and send its data to clients who are willing to listen without some sort of pre established connection?
    - Welcome to the world of **push** technology
        - **Web Sockets** is an implementation of push

Push


Client
Server
request (handshake request)
response (handshake OK)
permanently opened bi-directional connection
friendlist
event
(Stijn comes online)

Using HTTP

- Polling and long polling carry the overhead of HTTP, which doesn't make them well suited for low latency applications.
  - Think multiplayer first person shooter games in the browser or any other online game with a realtime component.

Web Sockets

- is a HTML5 API
- is a protocol which allows for communication between the client and the server/endpoint using a single TCP connection
    - the protocol is full-duplex (allows for simultaneous two-way communication)
    - it’s header is much smaller than that of a HTTP header, allowing for more efficient communication even over small packets of data

Web Sockets

- API for real-time, bi-directional communication between the client and server using a TCP based protocol.
- Connections are full duplex: it is possible to send and receive data simultaneously on the same connection.

Web Socket Life Cycle

1. Client sends the Server a handshake request in the form of a HTTP upgrade header with data about the WebSocket it’s attempting to connect to.

2. The Server responds to the request with another HTTP header, this is the last time a HTTP header gets used in the WebSocket connection. If the handshake was successful, the server sends a HTTP header telling the client it’s switching to the WebSocket protocol.

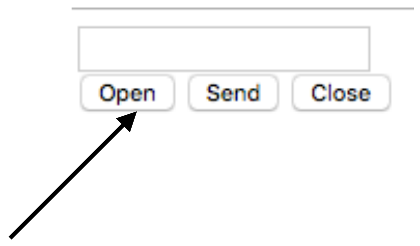
3. Now a constant connection is opened and the client and server can send any number of messages to each other until the connection is closed. These messages only have about 2 bytes of overhead.

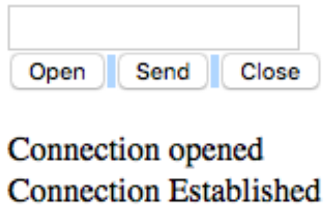
Web Socket API

- http://www.w3.org/TR/2011/WD-websockets-20110419/

Demo - Step 1

User 1





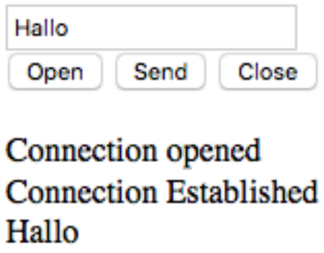
Connection opened
Connection Established

```html
<html>
    <head>
        <title>Chat Chamber</title>
   </head>
   <body>
       <div>
            <button type="button" onclick="openSocket();" >Open</button>
       </div>
       <script type="text/javascript">
            var webSocket;
            var messages = document.getElementById("messages");

            function openSocket(){
               webSocket = new WebSocket("ws://localhost:8080/GWT_Ajax_Example_Chat_Push/echo");
               webSocket.onopen = function(event){
                    writeResponse("Connection opened")
               };
               webSocket.onmessage = function(event){
                    writeResponse(event.data);
               };
               webSocket.onclose = function(event){
                    writeResponse("Connection closed");
               };
            }
           function writeResponse(text){
                messages.innerHTML += "<br/>" + text;
            }
        </script>

    </body>
</html>
```

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

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


    @OnOpen
     public void onOpen(Session session){
         System.out.println(session.getId() + " has opened a connection");
         sendMessageToAll("User has connected");
         try {
             session.getBasicRemote().sendText("Connection Established");
         } catch (IOException ex) {
             ex.printStackTrace();
         }
         sessions.add(session);
     }

    private void sendMessageToAll(String message){
         for(Session s : sessions){
             try {
                 s.getBasicRemote().sendText(message);
             } catch (IOException ex) {
                 ex.printStackTrace();
             }
         }
     }
 }
```

Demo - Step 2

User 1



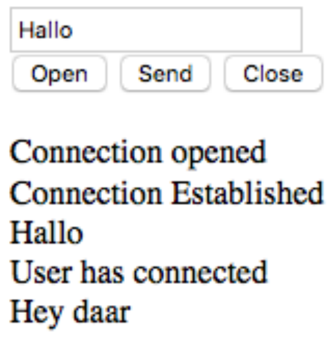
Connection opened
Connection Established
Hallo

```
<html>
    <head>
        <title>Chat Chamber</title>
   </head>
    <body>
        <div>
            <input type="text" id="messageinput"/>
        </div>
        <div>
           <button type="button" onclick="send();" >Send</button>
       </div>
        <div id="messages"></div>

        <script type="text/javascript">
            var webSocket;
            var messages = document.getElementById("messages");

            function openSocket(){
               webSocket.onmessage = function(event){
                    writeResponse(event.data);
                };
            }

            function send(){
                var text = document.getElementById("messageinput").value;
                webSocket.send(text);
            }

           function writeResponse(text){
                messages.innerHTML += "<br/>" + text;
            }
        </script>

    </body>
</html>
```

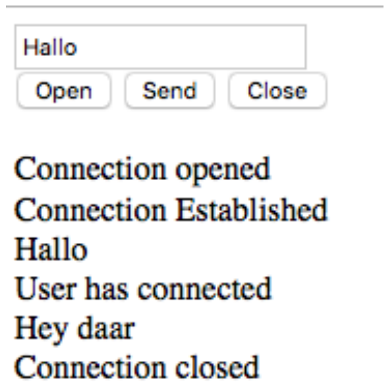
```java
@ServerEndpoint("/echo")
public class ChatServer {

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


   @OnMessage
   public void onMessage(String message, Session session){
      System.out.println("Message from " + session.getId() + ": " + message);
     sendMessageToAll(message);
   }

   private void sendMessageToAll(String message){
        for(Session s : sessions){
            try {
                s.getBasicRemote().sendText(message);
            } catch (IOException ex) {
                ex.printStackTrace();
            }
        }
    }
 }
```

Demo - Step 3

User 1

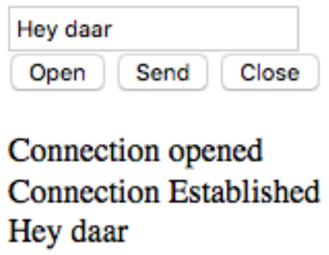


```
Connection opened
Connection Established
Hallo
User has connected
Hey daar
```



```
Connection opened
Connection Established
Hallo
User has connected
Hey daar
Connection closed
```

User 2



```
Connection opened
Connection Established
Hey daar
```

```
<html>
    <head>
        <title>Chat Chamber</title>
   </head>
    <body>
        <div>
           <button type="button" onclick="closeSocket();" >Close</button>
        </div>
       <script type="text/javascript">
            var webSocket;

            function openSocket(){
               webSocket.onclose = function(event){
                    writeResponse("Connection closed");
                };
            }

           function closeSocket(){
                webSocket.close();
            }

            function writeResponse(text){
                messages.innerHTML += "<br/>" + text;
            }
        </script>

    </body>
</html>
```

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

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


    @OnClose
    public void onClose(Session session){
      System.out.println("Chat " +session.getId()+"
      has ended");
      sessions.remove(session);
    }

}
```

Referenties

- https://blog.idrsolutions.com/2013/12/websockets-an-introduction/
- http://www.ibm.com/developerworks/library/wa-reverseajax2/
- http://www.html5rocks.com/en/tutorials/websockets/basics/
- http://code.tutsplus.com/tutorials/start-using-html5-websockets-today--net-13270