UNIVERSITÀ DI PISA

AI-DII

Hands-on session 7: WebSockets

Distributed Systems and Middleware Technologies

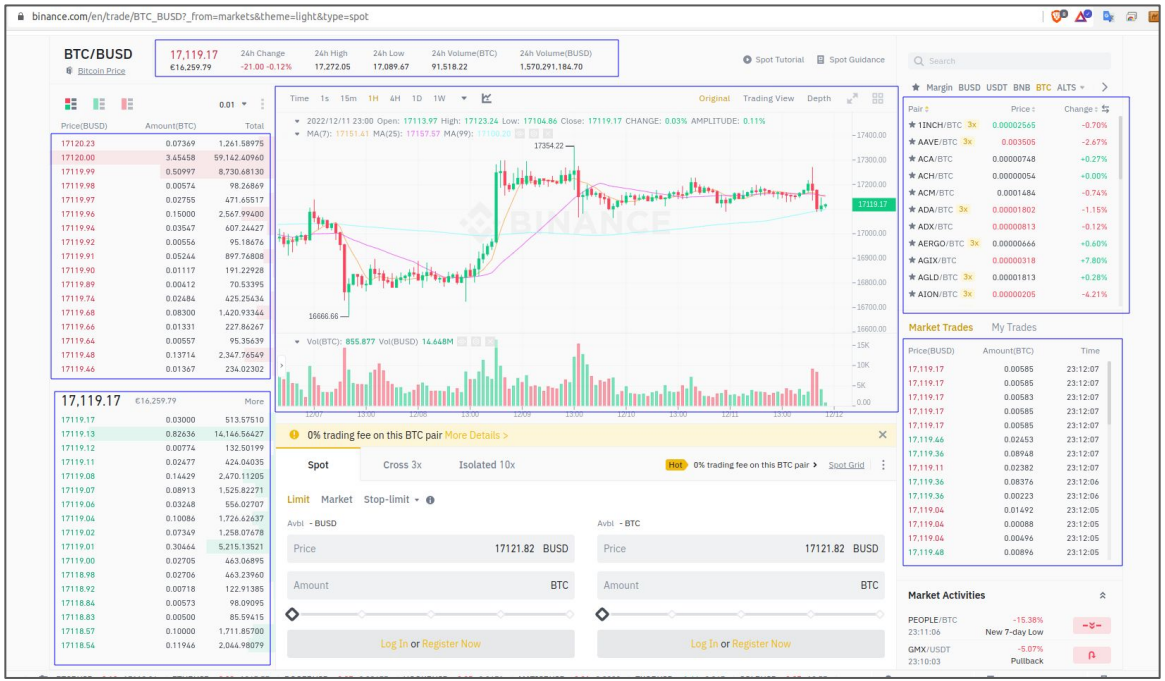
2023/2024

By José Luis Corcuera Bárcena

Agenda

1. Strategies for getting updates from the server
2. WebSockets definition
3. WebSockets - architecture
4. WebSockets client side
5. WebSockets server side
6. Exercises

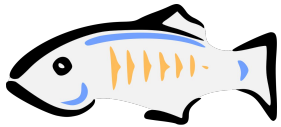
Strategies for getting updates from the server (1)



How can these sections of that web page update their content without refreshing the entire page?

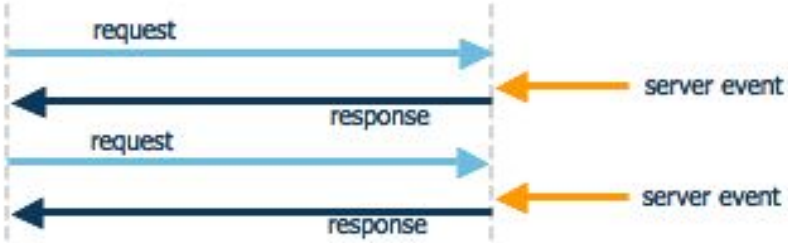
Website: https://www.binance.com/en/trade/BTC_BUSD?_from=markets&theme=light&type=spot

Strategies for getting updates from the server (2)



```
00:00:00 C-> Is the cake ready?
00:00:01 S-> No, wait.
00:00:01 C-> Is the cake ready?
00:00:02 S-> No, wait.
00:00:02 C-> Is the cake ready?
00:00:03 S-> Yeah. Have some lad.
00:00:03 C-> Is the other cake ready?
```

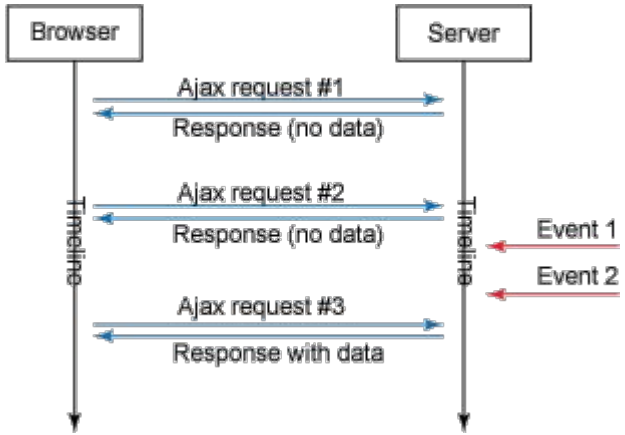
HTTP Short Polling: A more straightforward approach. A large number of requests are processed as they arrive at the server, resulting in a large amount of traffic (uses resources, but frees them as soon as the response is sent back).



HTTP Long Polling: One request is sent to the server, and the client awaits the response. The server keeps the request open until new data arrives (it is unresolved and resources are blocked). When a server event occurs, you are notified immediately. More complex, requiring more server resources.

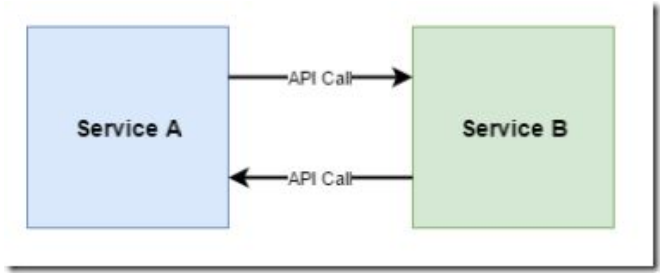
Image from: https://shyamapadabatabyal.wordpress.com/

Strategies for getting updates from the server (3)



HTTP Periodic Polling: There is a set amount of time between two requests. Polling has been improved and managed. Increase the time between two requests to reduce server consumption. However, if you need to be notified immediately when a server event occurs, this is not a good option.

Image from: https://medium.com/platform-engineer/web-api-design-35df8167460



Webhooks: In recent years, an increasing number of services have made it possible to configure "webhooks," which notify you when something interesting has occurred. You must provide a "callback" URI if you want to use a webhook, and the service providing the webhook will make an HTTP request to that URI whenever the event of interest occurs.

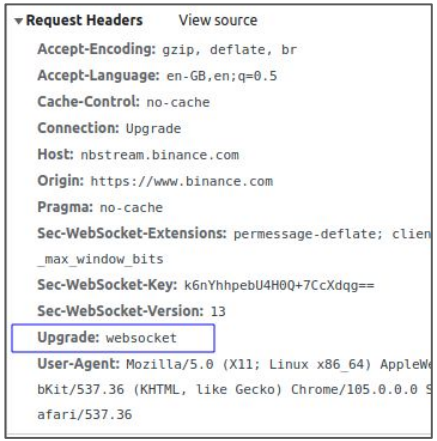
Image from: https://www.markheath.net/post/basic-introduction-webhooks

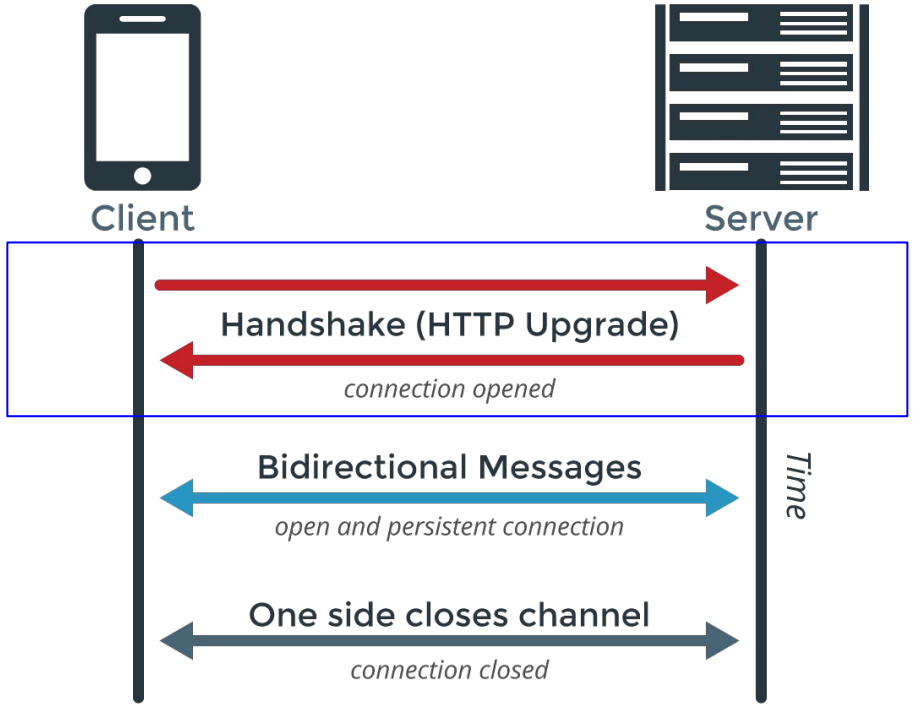
WebSockets definition

HTTP was created for the World Wide Web and has since been used by browsers, it had limitations. When you made a request, such as downloading HTML or an image, a port/socket was opened, data was transferred, and the port/socket was closed. **The opening and closing generate overhead**, which is inefficient for some applications, particularly those that require rapid responses, real-time interactions, or data streams to be displayed.

In 2011, the WebSocket protocol was standardized, allowing people to use the very flexible WebSocket protocol for transferring data to and from servers from the browser, as well as Peer-to-Peer (P2P), or direct communication between browsers. **In contrast to HTTP, the socket connected to the server remains "open" for communication. This means that data can be "pushed" to the browser in real time and from the browser to the server.**

WebSockets architecture

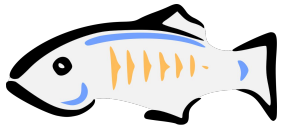




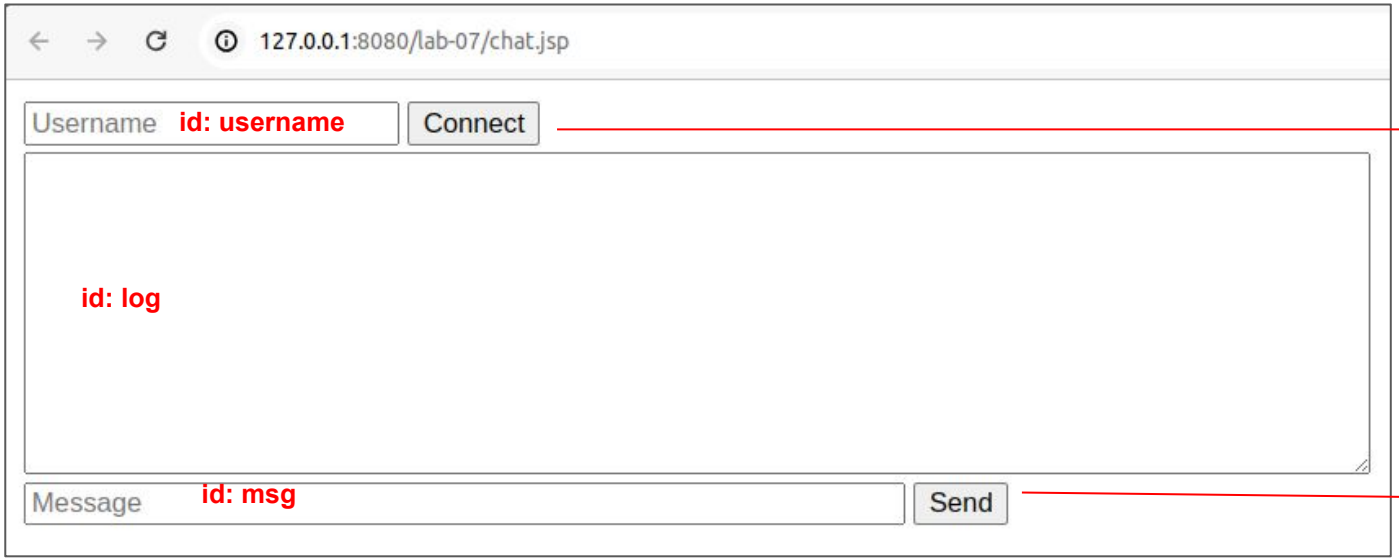
Upgrade attribute is sent in the Request Header to invite the server to switch to one of the listed protocols.

Image from: https://www.pubnub.com/blog/websockets-vs-rest-api-understanding-the-difference/

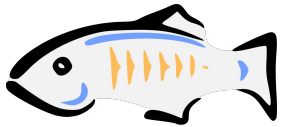
WebSockets client side (1)



Our chat window is show next with their javascript function.



WebSockets client side (2)



Javascript function: connect

```
<script type="application/javascript" >
 var ws;
 function connect() {
   var username = document.getElementById("username").value;
   var host = document.location.host;
   var pathname = "${pageContext.request.contextPath}";

   const url = "ws://" + host  + pathname + "/chat/" + username;
   alert('url: ' + url);
   ws = new WebSocket(url);

   ws.onmessage = function(event) {
     var log = document.getElementById("log");
     console.log(event.data);
     var message = JSON.parse(event.data);
     log.innerHTML += message.from + " : " + message.content + "\n";
   };
 }
```

Getting the Context Path by using JSP - Expression Language (EL)

Initializing ws variable with a WebSocket.

When a message is received, it is appended to the log text area. Note that the event.data variable is a string representation of a JSON object. This JSON object has the following attributes:

- **from**
- **content**

WebSockets client side (3)

Javascript function: send

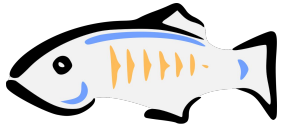
```
function send() {
  var content = document.getElementById("msg").value;
  var json = JSON.stringify({
    "content":content
  });
  ws.send(json);
}
```

```
</script>
```

A string representation of a JSON object is sent to the server.

Somehow this JSON string representation have to be converted into a Java Object…

WebSockets server side (1)



In order to convert incoming and outgoing messages, we define a decoder and encoder. **Decoders** specifies a list of classes that can be used to decode incoming messages to the WebSocket endpoint. These classes implement the Decoder interface. Instead **encoders** can be used to encode outgoing messages from the WebSocket endpoint. These classes implement the Encoder interface.

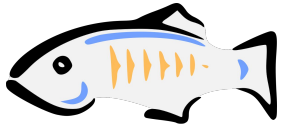
```java
public class MessageDTOEncoder implements Encoder.Text<MessageDTO>{
    1 usage
    private static Gson gson = new Gson();
    @Override
    public String encode(MessageDTO messageDTO) throws EncodeException {
        return gson.toJson(messageDTO);
    }
}
```

```java
public class MessageDTO {
    2 usages
    private String from;
    2 usages
    private String to;
    2 usages
    private String content;
```

```java
public class MessageDTODecoder implements Decoder.Text<MessageDTO>{
    1 usage
    private static Gson gson = new Gson();
    @Override
    public MessageDTO decode(String s) throws DecodeException {
        return gson.fromJson(s, MessageDTO.class);
    }

    @Override
    public boolean willDecode(String s) {
        return s != null && !s.isEmpty();
    }
}
```

WebSockets server side (2)

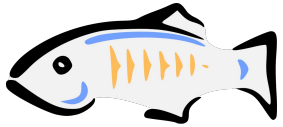


You can use the **@ServerEndpoint** annotation to specify that a class is used as a WebSocket server endpoint. The ChatEndpoint the class is annotated with **@ServerEndpoint** and a URI path of "**/chat/{username}**" for the value attribute was set. Note the path parameter that is enclosed in curly braces at the end of the URI. This allows the endpoint to accept a parameter and it can be retrieved by using the **@PathParam** in the **onOpen** method.

```
@ServerEndpoint(value = "/chat/{username}", decoders = MessageDTODecoder.class, encoders = MessageDTOEncoder.class)
public class ChatEndpoint {
    3 usages
    private static final Set<Session> chatEndpoints = new CopyOnWriteArraySet<Session>();
    3 usages
    private static Map<String, String> users = new HashMap<String, String>();

    @OnOpen
    public void onOpen(Session session, @PathParam("username") String username) throws IOException, EncodeException {
        chatEndpoints.add(session);
        users.put(session.getId(), username);
        MessageDTO message = new MessageDTO();
        message.setFrom(username);
        message.setContent("Connected!");
        broadcast(message);
    }
```

WebSockets server side (3)



```java
@OnMessage
public void onMessage(Session session, MessageDTO message) throws IOException, EncodeException {
    message.setFrom(users.get(session.getId()));
    broadcast(message);
}

@OnClose
public void onClose(Session session) throws IOException, EncodeException {
    chatEndpoints.remove(session);
    MessageDTO message = new MessageDTO();
    message.setFrom(users.get(session.getId()));
    message.setContent("Disconnected!");
    broadcast(message);
}

@OnError
public void onError(Session session, Throwable throwable) {
    // Do error handling here
}
```

When a message arrive, it will be propagated to all the users registered in the chat.

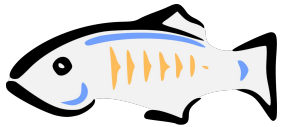
When a user closes the browser, the onClose method is activated and it will notify all other users this event.

WebSockets server side (4)

```
private static void broadcast(MessageDTO message) throws IOException, EncodeException {
    chatEndpoints.forEach(session -> {
        synchronized (session) {
            try {
                session.getBasicRemote()
                        .sendObject(message);
            } catch (IOException | EncodeException e) {
                e.printStackTrace();
            }
        }
    });
}
```

Finally, the broadcast method iterates the list of sessions, and for each session it will send the message received as parameter in this method.

Exercise 02: Chat direct message



Extend the functionality of the chat by adding direct messaging. To do so, a user have to type a message with the following format:

@UserId <message to be sent>

Example:

@CarbonaraIsTheWay are you there?

Only the user identified by @CarbonaraIsTheWay will receive the message.

References

- https://jakarta.ee/specifications/messaging/3.0/jakarta-messaging-spec-3.0.html
- http://devdoc.net/javaxe/JavaEE-7u2/docs/javaee-tutorial/doc/jms-examples002.htm
- https://www.javacodegeeks.com/2013/05/java-ee-7-jms-2-0-with-glassfish-v4.html
- https://stackoverflow.com/questions/7443306/javaee-6-javax-naming-namealreadyboundexception-use-rebind-to-override
- https://medium.com/platform-engineer/web-api-design-35df8167460
- https://blogs.oracle.com/javamagazine/post/how-to-build-applications-with-the-websocket-api-for-java-ee-and-jakarta-ee
- https://jakarta.ee/specifications/websocket/2.0/websocket-spec-2.0.html