WebSocket Server
Lecture Question

Task: Write a Web Socket Server that counts the number of messages it receives

In a package named server, write a class named LectureServer that:

• When created, sets up a web socket server listening for connections on localhost:8080

• Listens for messages of type increment with no data

• Has a method named numberOfMessages that returns (as an Int) the number of times a message of type increment was received
Web App Architecture

View

Controller

Model

Web Front End

Desktop Front End

Web Sockets

Web Socket Server

MySQL Database

Actor System

SQL Statements

Actor Messages

Model

View

Controller

Web Front End

Desktop Front End

Web Sockets

Web Socket Server

MySQL Database

Actor System

Actor Messages
Web App Architecture

View

Controller

Model

Web Front End

Web Sockets

Desktop Front End

Web Socket Server

MySQL Database

Actor System

SQL Statements

Actor Messages
The Problem

• In CSE115 you used HTTP request/responses to build web apps

• If you wanted more data from the server after the page loads, you used AJAX

• Server hosts JSON data at certain end points

• Client makes an AJAX call to retrieve the most current data

• But the server has to wait for a request before sending a response
What if the server wants to send time-sensitive data without waiting for a request?

In CSE115

- Built a chat app using polling
- Client sent AJAX requests at regular intervals
- Only get updates when AJAX request is sent
- Can use long-polling
- Server hangs on poll requests until it has new data to send
Web Sockets

• A newer solution (Standardized in 2011)
• Establishes a lasting connection
• Enables 2-way communication between server and client
• Server can push updates to clients over the web socket without waiting for the client to make a new request
socket.io

- A library build on top of web sockets
- Maintains connections and reconnecting
- Uses message types
  - Similar to actors, except the message type is always a string
- Add listeners to react to different message types
  - Receiving a message is an event
  - Listener code will be called when the event occurs
socket.io Server in Scala

- New library
- Link on the course website
- Dependency included in pom.xml in examples repo
Web Socket Server

• Import from the new library
• Setup and start the server

```scala
import com.corundumstudio.socketio.listener.{ConnectListener, DataListener, DisconnectListener}
import com.corundumstudio.socketio.{AckRequest, Configuration, SocketIOClient, SocketIOServer}
class Server() {
    val config: Configuration = new Configuration {
        setHostname("localhost")
        setPort(8080)
    }
    val server: SocketIOServer = new SocketIOServer(config)
    server.addConnectListener(new ConnectionListener())
    server.addDisconnectListener(new DisconnectionListener())
    server.addEventListener("chat_message", classOf[String], new MessageListener())
    server.addEventListener("stop_server", classOf[Nothing], new StopListener(this))
    server.start()
}
```
Web Socket Server

- Create a configuration object for the server
- This server will run on localhost port 8080

```scala
import com.corundumstudio.socketio.listener.{ConnectListener, DataListener, DisconnectListener}
import com.corundumstudio.socketio.{AckRequest, Configuration, SocketIOClient, SocketIOServer}

class Server() {
  val config: Configuration = new Configuration {
    setHostname("localhost")
    setPort(8080)
  }

  val server: SocketIOServer = new SocketIOServer(config)

  server.addConnectListener(new ConnectionListener())
  server.addDisconnectListener(new DisconnectionListener())
  server.addEventListener("chat_message", classOf[String], new MessageListener())
  server.addEventListener("stop_server", classOf[Nothing], new StopListener(this))

  server.start()
}
```
Web Socket Server

- Create and start the server
- Use the configuration to tell the library how to setup the server
- Call the start() method to start listening for connections

```java
import com.corundumstudio.socketio.listener.{ConnectListener, DataListener, DisconnectListener}
import com.corundumstudio.socketio.{AckRequest, Configuration, SocketIOClient, SocketIOServer}

class Server() {
    val config: Configuration = new Configuration {
        setHostname("localhost")
        setPort(8080)
    }

    val server: SocketIOServer = new SocketIOServer(config)

    server.addConnectListener(new ConnectionListener())
    server.addDisconnectListener(new DisconnectionListener())
    server.addEventListener("chat_message", classOf[String], new MessageListener())
    server.addEventListener("stop_server", classOf[Nothing], new StopListener(this))

    server.start()
}
```
Web Socket Server

- Add listeners to handle different event types
- Connect and disconnect listeners to react to clients connecting and disconnecting
- Event listeners for each different message type received from clients

```java
import com.corundumstudio.socketio.listener.{ConnectListener, DataListener, DisconnectListener}
import com.corundumstudio.socketio.{AckRequest, Configuration, SocketIOClient, SocketIOServer}

class Server() {
  val config: Configuration = new Configuration {
    setHostname("localhost")
    setPort(8080)
  }

  val server: SocketIOServer = new SocketIOServer(config)

  server.addConnectListener(new ConnectionListener())
  server.addDisconnectListener(new DisconnectionListener())
  server.addEventListener("chat_message", classOf[String], new MessageListener())
  server.addEventListener("stop_server", classOf[Nothing], new StopListener(this))

  server.start()
}
```
Web Socket Server

• For connect and disconnect
• Create classes overriding ConnectListener and DisconnectListener
• Implement the onConnect/onDisconnect methods
• These methods take a reference to the sending socket as a parameter
• Can use this reference to send messages to the client
• Usually want to store each reference to send messages later

```scala
server.addConnectListener(new ConnectionListener())
server.addDisconnectListener(new DisconnectionListener())

class ConnectionListener() extends ConnectListener {
  override def onConnect(client: SocketIOClient): Unit = {
    println("Connected: " + client)
  }
}

class DisconnectionListener() extends DisconnectListener {
  override def onDisconnect(socket: SocketIOClient): Unit = {
    println("Disconnected: " + socket)
  }
}
```
Web Socket Server

- To receive messages, specify the message type and the class of the message
- Create classes extending DataListener[message_class]
- For message class we'll use
  - String to receive text data
  - Nothing if it's just a message (Similar to an actor receiving a case object)

```scala
server.addEventListener("chat_message", classOf[String], new MessageListener())
server.addEventListener("stop_server", classOf[Nothing], new StopListener(this))

class MessageListener() extends DataListener[String] {
  override def onData(socket: SocketIOClient, data: String, ackRequest: AckRequest): Unit = {
    println("received message: " + data + " from " + socket)
    socket.sendEvent("ACK", "I received your message of " + data)
  }
}

class StopListener(server: Server) extends DataListener[Nothing] {
  override def onData(socket: SocketIOClient, data: Nothing, ackRequest: AckRequest): Unit = {
    println("stopping server")
    server.server.stop()
    println("safe to stop program")
  }
}
```
Web Socket Server

- The DataListeners must implement onData with parameters:
  - A socket reference. Can be used to lookup a user after storing this reference on connection/registration
  - data with type matching the class of the message. This is the content of the message received
  - AckRequest. Not used in this course

```scala
server.addEventListener("chat_message", classOf[String], new MessageListener())
server.addEventListener("stop_server", classOf[Nothing], new StopListener(this))

class MessageListener() extends DataListener[String] {
  override def onData(socket: SocketIOClient, data: String, ackRequest: AckRequest): Unit = {
    println("received message:" + data + " from" + socket)
    socket.sendEvent("ACK", "I received your message of" + data)
  }
}

class StopListener(server: Server) extends DataListener[Nothing] {
  override def onData(socket: SocketIOClient, data: Nothing, ackRequest: AckRequest): Unit = {
    println("stopping server")
    server.server.stop()
    println("safe to stop program")
  }
}
Web Socket Server

- Use the reference to the Socket to send messages to the client
- Specify the type of the message as a String
- If the message contains data, use a second String

```scala
server.addEventListener("chat_message", classOf[String], new MessageListener())
server.addEventListener("stop_server", classOf[Nothing], new StopListener(this))
```

```scala
class MessageListener() extends DataListener[String] {
  override def onData(socket: SocketIOClient, data: String, ackRequest: AckRequest): Unit = {
    println("received message: "+ data + " from "+ socket)
    socket.sendEvent("ACK", "I received your message of "+ data)
  }
}
```

```scala
class StopListener(server: Server) extends DataListener[Nothing] {
  override def onData(socket: SocketIOClient, data: Nothing, ackRequest: AckRequest): Unit = {
    println("stopping server")
    server.server.stop()
    println("safe to stop program")
  }
}
```
Web Socket Demo
Lecture Question

Task: Write a Web Socket Server that counts the number of messages it receives

In a package named server, write a class named LectureServer that:

• When created, sets up a web socket server listening for connections on localhost:8080

• Listens for messages of type increment with no data

• Has a method named numberOfMessages that returns (as an Int) the number of times a message of type increment was received