WebSocket Clients

Lecture Question

Task: Write a Web Socket Server that echo back to clients the messages they send

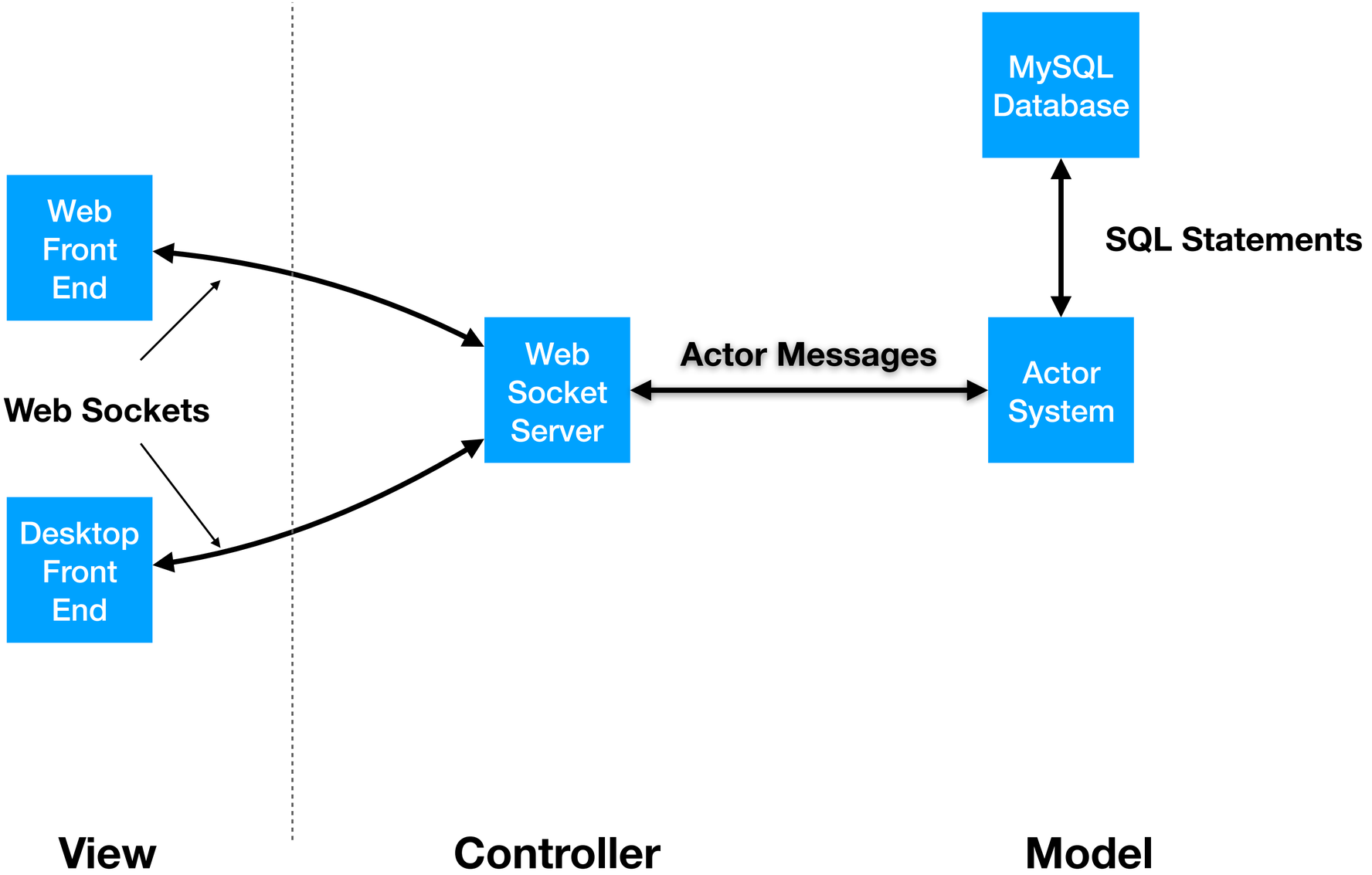
In a package named server, write a class named EchoServer that:

- When created, sets up a web socket server listening for connections on localhost:8080
- Listens for messages of type "send_back" containing a String and send back to the client a message of type "echo" containing the exact string sent by the client

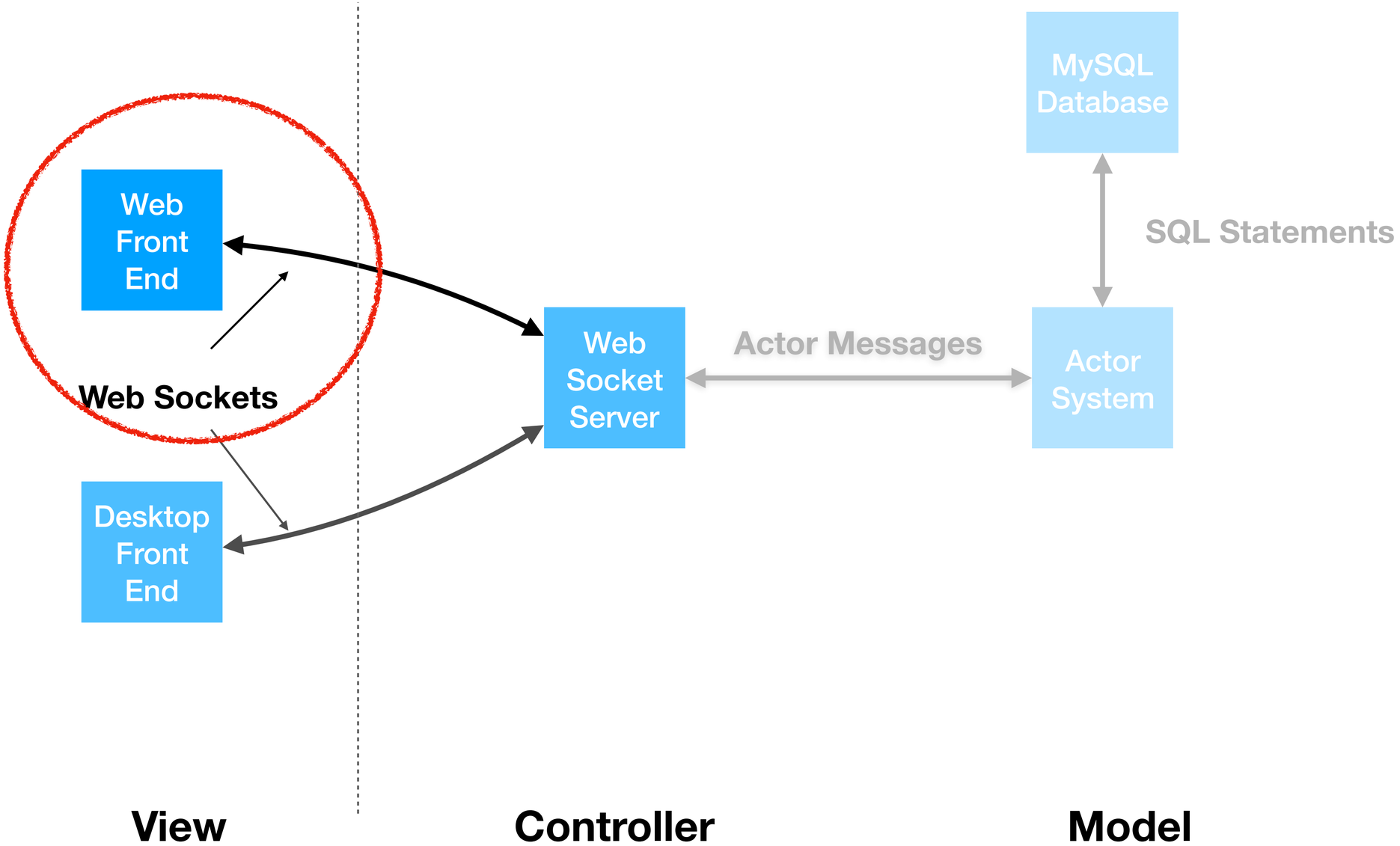
Web Socket Clients

- We've set up a web socket server that will listen for connections and process messages
- Now, let's build a web socket client that will connect to the server

MMO Architecture



MMO Architecture



WebSocket Client - Web

- First, setup the HTML
- Layout and style of the page
  - Could add CSS for more style

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Web Socket Client Example</title>
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.js"></script>
</head>
<body>


<input type="text" id="chat_input"/>
<button id="gold" onclick="sendMessage();">Submit</button>

<div id="display_message"></div>

<script src="WebClient.js"></script>

</body>
</html>
```

WebSocket Client - Web

- Download the socket.io JavaScript client library
- This library contains all the code we'll need to connect to our server

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Web Socket Client Example</title>
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.js"></script>
</head>
<body>


<input type="text" id="chat_input"/>
<button id="gold" onclick="sendMessage();">Submit</button>

<div id="display_message"></div>

<script src="WebClient.js"></script>

</body>
</html>
```

WebSocket Client - Web

- Add elements for the user to enter and send a message
- In JavaScript, we'll implement the sendMessage() function

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Web Socket Client Example</title>
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.js"></script>
</head>
<body>


<input type="text" id="chat_input"/>
<button id="gold" onclick="sendMessage();">Submit</button>

<div id="display_message"></div>

<script src="WebClient.js"></script>

</body>
</html>
```

WebSocket Client - Web

- Download our JavaScript file
- This script runs code to connect to the server as soon as it's downloaded
    - Include this at the end of the body so the page loads before connecting to the server

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Web Socket Client Example</title>
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.js"></script>
</head>
<body>


<input type="text" id="chat_input"/>
<button id="gold" onclick="sendMessage();">Submit</button>

<div id="display_message"></div>

<script src="WebClient.js"></script>

</body>
</html>
```

WebSocket Client - Web

- In WebClient.js
- Call io.connect (from the library) to connect to the server
  - Returns a reference to the created socket

```
const socket = io.connect("http://localhost:8080", {transports: ['websocket']});

socket.on('ACK', function (event) {
    document.getElementById("display_message").innerHTML = event;
});

socket.on('server_stopped', function (event) {
    document.getElementById("display_message").innerHTML = "The server has stopped";
});


function sendMessage() {
    let message = document.getElementById("chat_input").value;
    document.getElementById("chat_input").value = "";
    socket.emit("chat_message", message);
}
```

WebSocket Client - Web

- Define how the socket will react to different message types with the "on" method
- The "on" method takes the message type and a function as arguments
  - Call the function whenever a message of that type is received from the server

```
const socket = io.connect("http://localhost:8080", {transports: ['websocket']});

socket.on('ACK', function (event) {
    document.getElementById("display_message").innerHTML = event;
});

socket.on('server_stopped', function (event) {
    document.getElementById("display_message").innerHTML = "The server has stopped";
});


function sendMessage() {
    let message = document.getElementById("chat_input").value;
    document.getElementById("chat_input").value = "";
    socket.emit("chat_message", message);
}
```

WebSocket Client - Web

- The function should take a parameter which will contain the data of the message if there is any
- We receive an ACK message containing a string which we display on the page (Similar to case class)
- We receive a server_stopped message and inform the user that the server stopped (Similar to case object)

```
const socket = io.connect("http://localhost:8080", {transports: ['websocket']});

socket.on('ACK', function (event) {
    document.getElementById("display_message").innerHTML = event;
});

socket.on('server_stopped', function (event) {
    document.getElementById("display_message").innerHTML = "The server has stopped";
});


function sendMessage() {
    let message = document.getElementById("chat_input").value;
    document.getElementById("chat_input").value = "";
    socket.emit("chat_message", message);
}
```

WebSocket Client - Web

- To send a message, call emit
- Takes the message type and the content of the message, if any
- Can call emit with only message type to send a message with no content (Similar to case object)

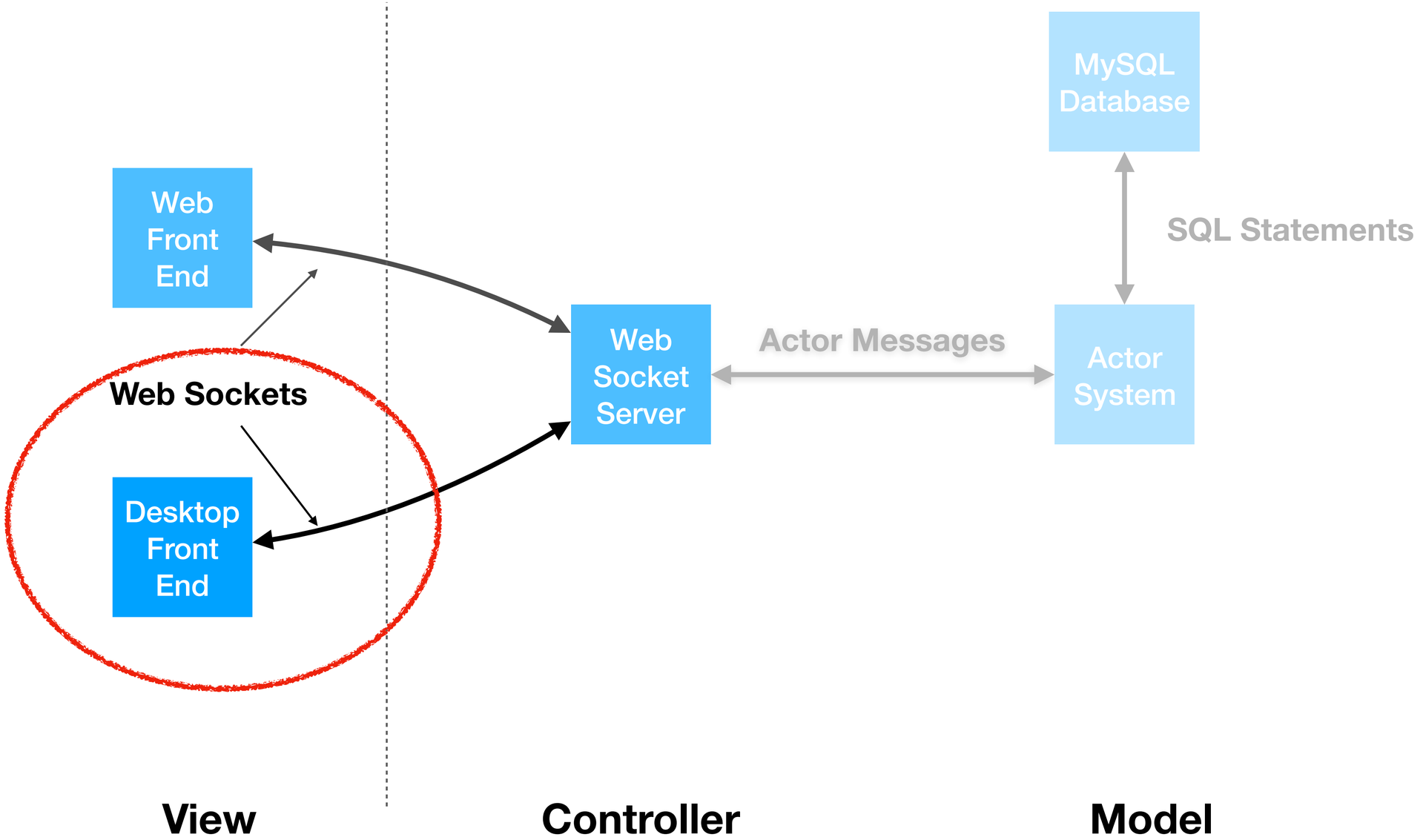
```
const socket = io.connect("http://localhost:8080", {transports: ['websocket']});

socket.on('ACK', function (event) {
    document.getElementById("display_message").innerHTML = event;
});

socket.on('server_stopped', function (event) {
    document.getElementById("display_message").innerHTML = "The server has stopped";
});


function sendMessage() {
    let message = document.getElementById("chat_input").value;
    document.getElementById("chat_input").value = "";
    socket.emit("chat_message", message);
}
```

MMO Architecture



WebSocket Client - Scala

- Another new library!
- We'll use the Scala/Java version of the socket.io client Library
  - Follows the same structure as the web client
- Add to pom.xml and use maven to download
  - Included in examples repo

Web Socket Client - Scala

- Import relavent code from the socket.io library
- Use IO.socket to create a socket
  - Returns a reference to the created socket
- Call connect() to connect to the server

```
import io.socket.client.{IO, Socket}
import io.socket.emitter.Emitter

class ProcessMessageFromServer() extends Emitter.Listener {
  override def call(objects: Object*): Unit = {
    val message = objects.apply(0).toString
    println(message)
  }
}

object SimpleClient{
  def main(args: Array[String]): Unit = {
    val socket: Socket = IO.socket("http://localhost:8080/")
    socket.on("ACK", new ProcessMessageFromServer())

    socket.connect()
    socket.emit("chat_message", "hello")
    socket.close()
  }
}
```

Web Socket Client - Scala

- Call the "on" method to define the behavior for each message type received from the server
  - Takes a message type and an object that extends Emitter.Listener
  - Implement call(Objects*)

```
import io.socket.client.{IO, Socket}
import io.socket.emitter.Emitter

class ProcessMessageFromServer() extends Emitter.Listener {
  override def call(objects: Object*): Unit = {
    val message = objects.apply(0).toString
    println(message)
  }
}

object SimpleClient{
  def main(args: Array[String]): Unit = {
    val socket: Socket = IO.socket("http://localhost:8080/")
    socket.on("ACK", new ProcessMessageFromServer())

    socket.connect()
    socket.emit("chat_message", "hello")
    socket.close()
  }
}
```

Web Socket Client - Scala

- Implement call(Objects*) which is called with the content of the message as an Array (sort of) of Objects
- The library is written in Java and uses Java's Object class
- Object contains a toString method so we access the first element and convert it to a String to process the content of the message
  - If there is no content to the message this will throw an index out of bounds error

```
import io.socket.client.{IO, Socket}
import io.socket.emitter.Emitter

class ProcessMessageFromServer() extends Emitter.Listener {
  override def call(objects: Object*): Unit = {
    val message = objects.apply(0).toString
    println(message)
  }
}

object SimpleClient{
  def main(args: Array[String]): Unit = {
    val socket: Socket = IO.socket("http://localhost:8080/")
    socket.on("ACK", new ProcessMessageFromServer())

    socket.connect()
    socket.emit("chat_message", "hello")
    socket.close()
  }
}
```

Web Socket Client - Scala

- Send messages to the server using the emit method
  - Same syntax as the web version of socket.io

```scala
import io.socket.client.{IO, Socket}
import io.socket.emitter.Emitter

class ProcessMessageFromServer() extends Emitter.Listener {
  override def call(objects: Object*): Unit = {
    val message = objects.apply(0).toString
    println(message)
  }
}

object SimpleClient{
  def main(args: Array[String]): Unit = {
    val socket: Socket = IO.socket("http://localhost:8080/")
    socket.on("ACK", new ProcessMessageFromServer())

    socket.connect()
    socket.emit("chat_message", "hello")
    socket.close()
  }
}
```

Web Socket Client - Scala

- If you need to interact with a ScalaFX GUI when a socket message is received, call Platform.runLater
- Platform.runLater will run your method on the same thread as the GUI
- This allows you to access the GUI elements/variables from your Emitter.Listener

```
class ServerStopped() extends Emitter.Listener {
  override def call(objects: Object*): Unit = {
    Platform.runLater(() => {
      GUIClient.textOutput.text.value = "The server has stopped"
    })
  }
}


object GUIClient extends JFXApp {

  // ...

  socket.on("server_stopped", new ServerStopped)

  // ...

  val textOutput: Label = new Label

  // ...
}
```

Web Socket Client - Scala

- Takes an object extending Runnable with a method named run with no parameters and return type Unit
- Using Scala syntax to condense this inheritance
  - This syntax can be used when extending a trait with a single method
  - Can create your listeners and event handlers with this syntax if you'd prefer

```
class ServerStopped() extends Emitter.Listener {
  override def call(objects: Object*): Unit = {
    Platform.runLater(() => {
      GUIClient.textOutput.text.value = "The server has stopped"
    })
  }
}


object GUIClient extends JFXApp {

  // ...

  socket.on("server_stopped", new ServerStopped)

  // ...

  val textOutput: Label = new Label

  // ...
}
```

WebSocket Demo

Lecture Question

Task: Write a Web Socket Server that echo back to clients the messages they send

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

- When created, sets up a web socket server listening for connections on localhost:8080
- Listens for messages of type "send_back" containing a String and send back to the client a message of type "echo" containing the exact string sent by the client