WebSocket Clients

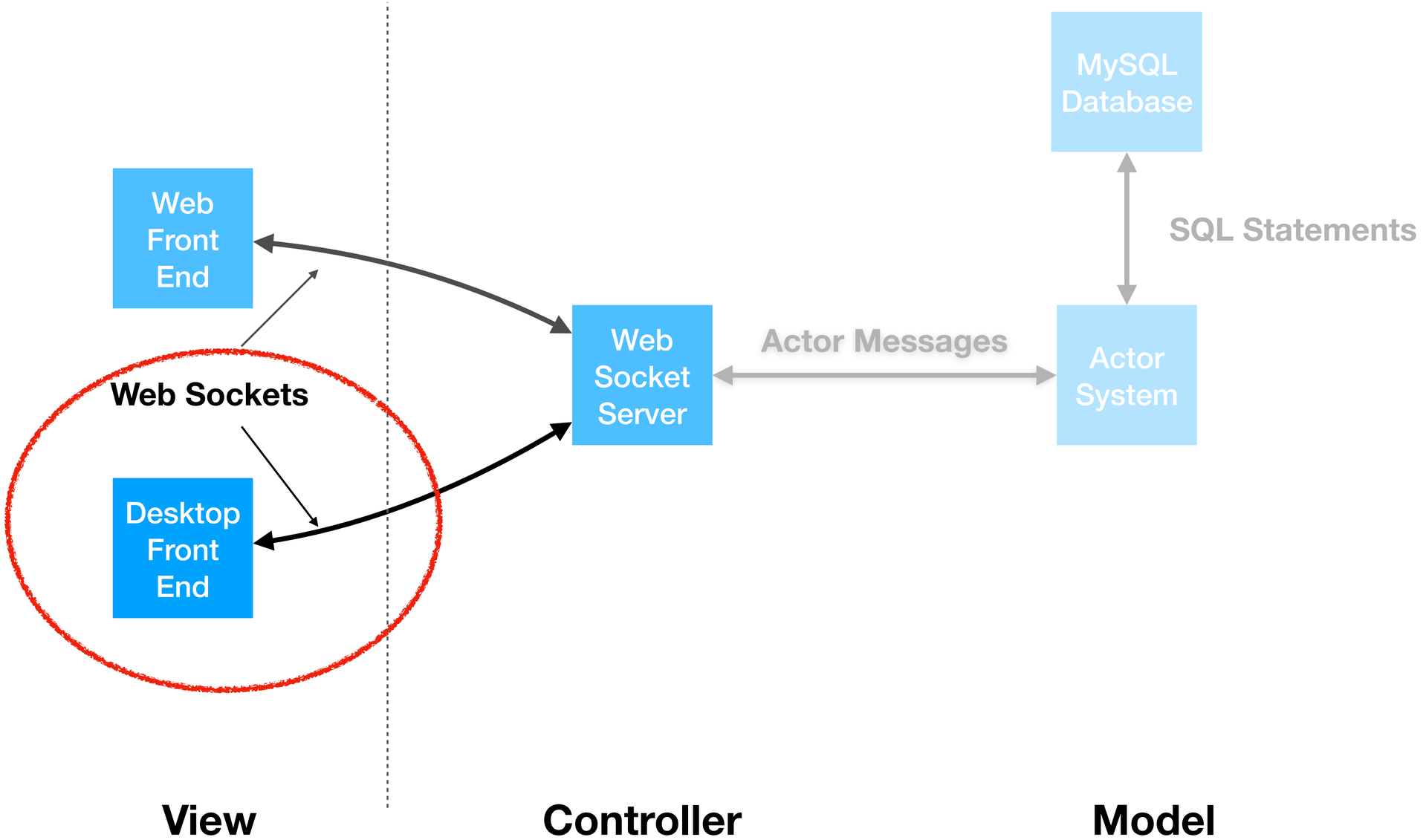
Lecture Question

Task: Write a Web Socket Server that echos 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 sends back to the client a message of type "echo" containing the exact string sent by the client

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(Object*)

```
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

```
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

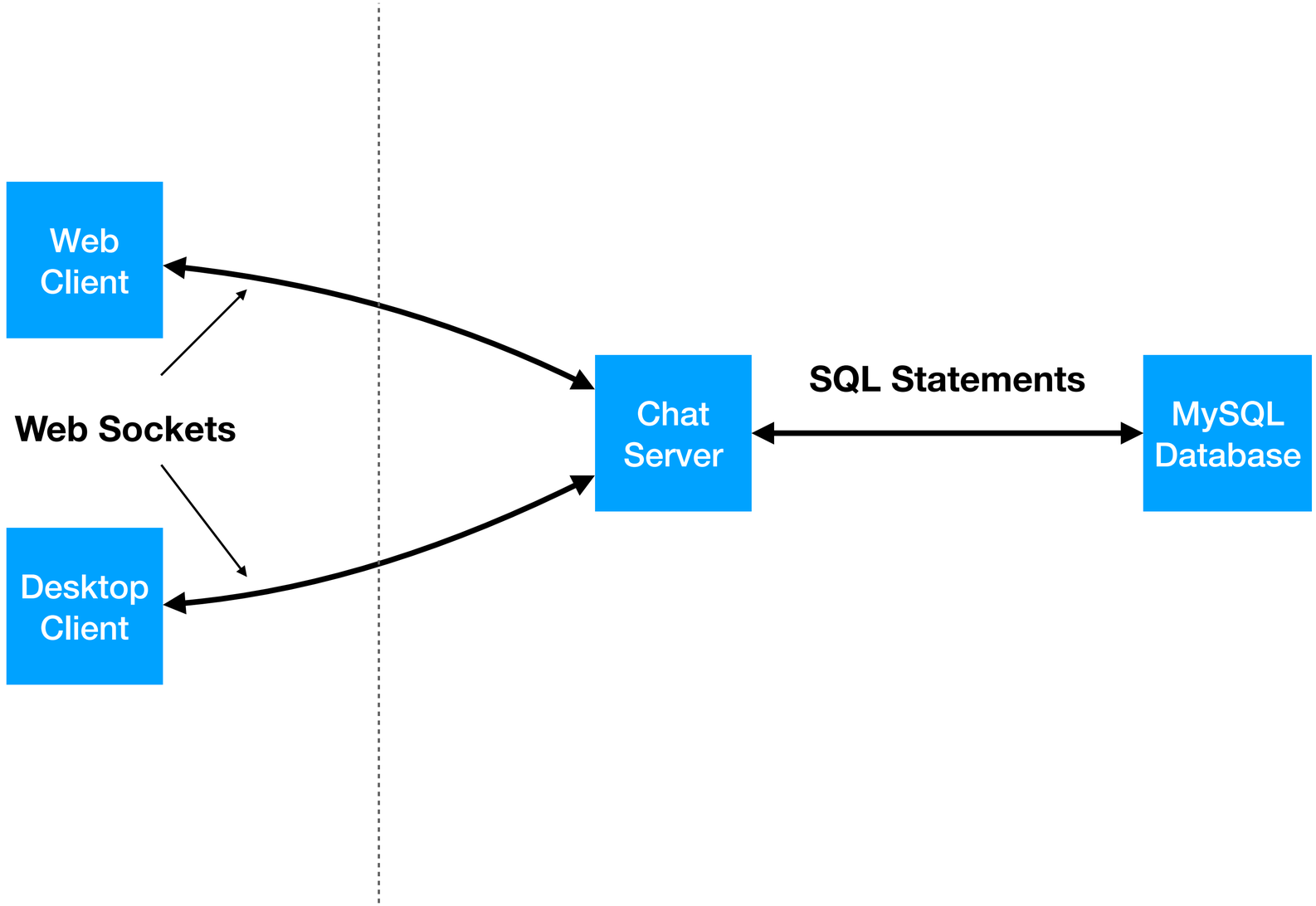
- Takes an object extending Runnable with a method named run with no parameters and return type Unit
- Using Scala syntax to condense this inheritance
  - This syntax can be used when extending a trait with a single method
  - Can create your listeners and event handlers with this syntax if you'd prefer

```
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

  // ...
}
```

Chat Demo

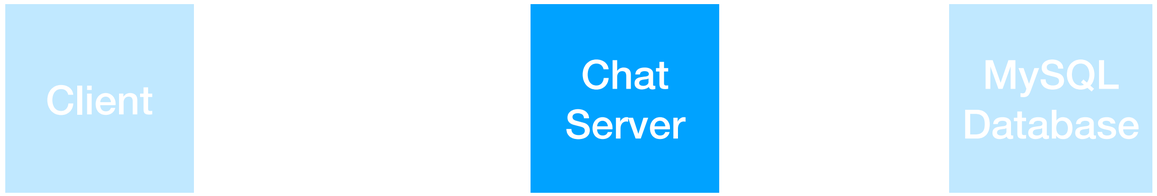
- Let's build a chat app!
  - Code is in the repo
- Users can connect to the chat server
  - Use a web or desktop front end
  - Server doesn't care what type of app a client is using
- All connected users can communicate through text messages

Chat Architecture



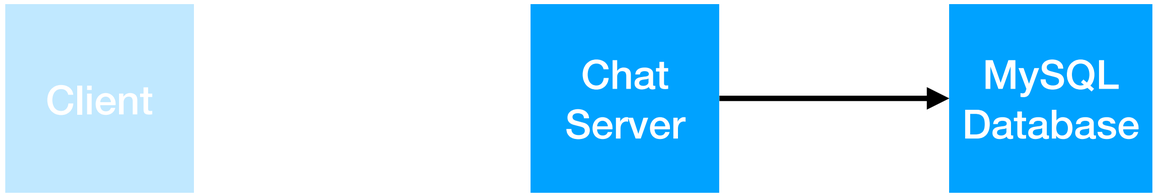
Chat App

- Chat server starts up
- Listens for WebSocket connections on port 8080
- Initialize data structures that will store references to each WebSocket



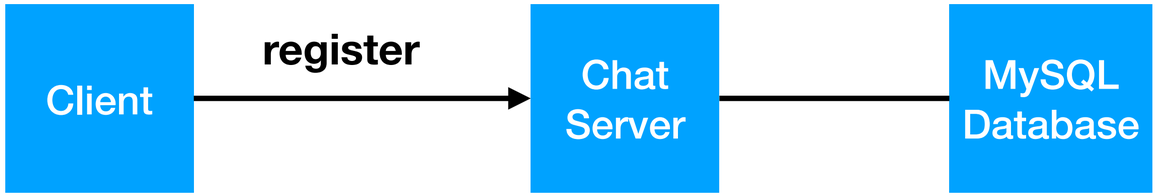
Chat App

- Server connects to a MySQL database to store the chat history
- Communicates via SQL statements
  - MySQL reacts to the event of receiving a statement
- **More details on MySQL in a later lecture**



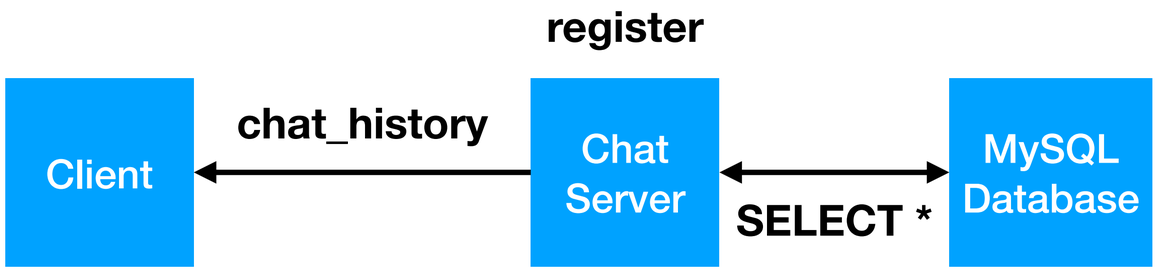
Chat App

- Clients connect to the server using WebSockets
  - Client could be web or desktop
- After the connection is established:
  - Client sends a message of type register containing their username



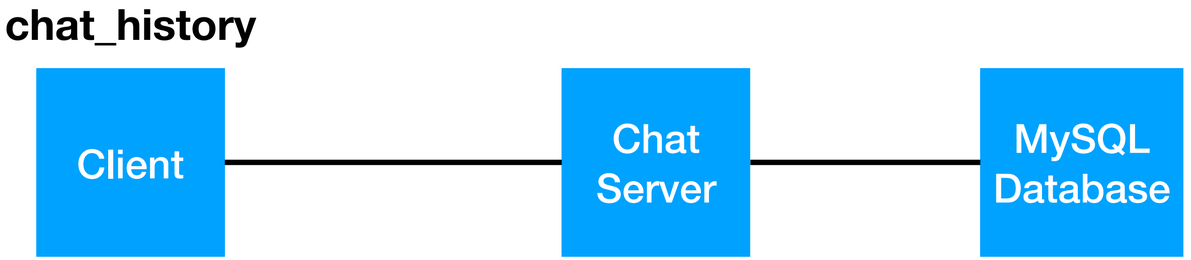
Chat App

- The server receives the register message and reacts to this event
- Adds the new user to the data structures
  - Data structure remembers the username associated with this socket
- Retrieve the chat history from the database and send it to the client



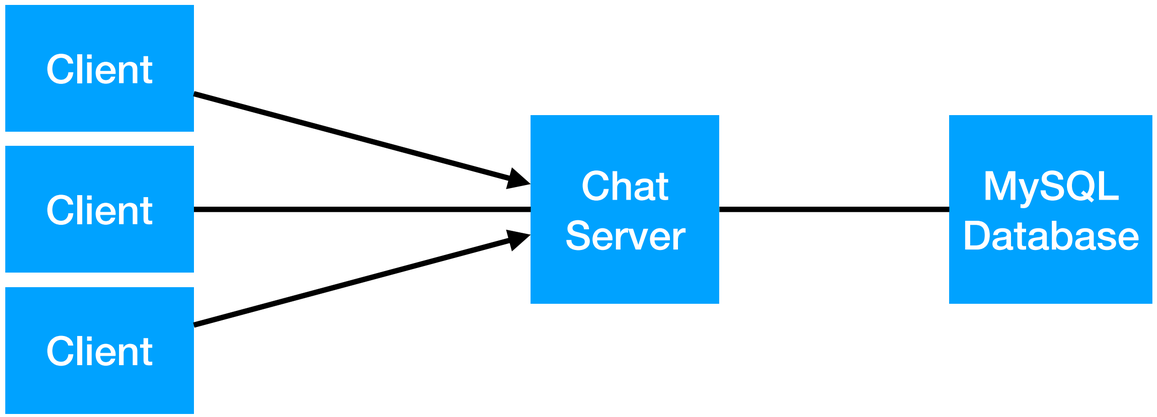
Chat App

- Client reacts to the chat_history message
  - Renders all the content and displays it to the user



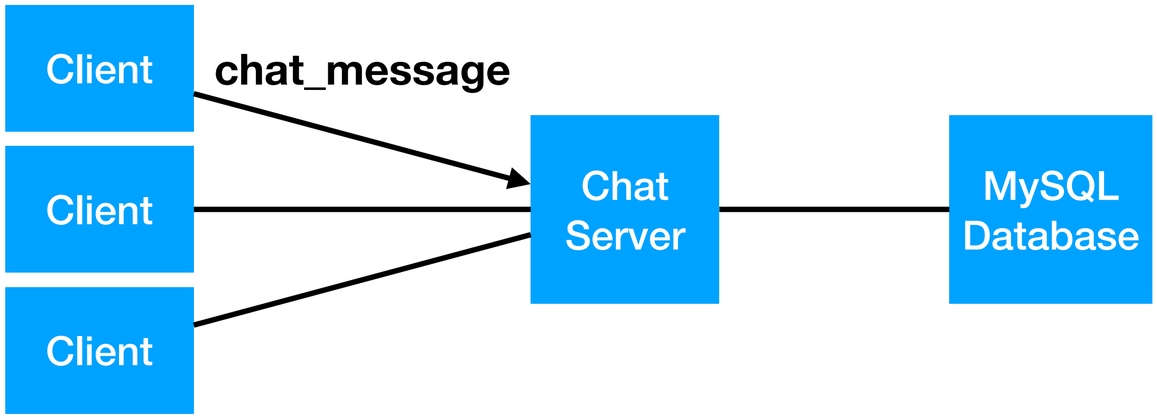
Chat App

- Multiple clients can be connected simultaneously
- Each client sends their username in a register message
- Chat server maps usernames to sockets for all connections



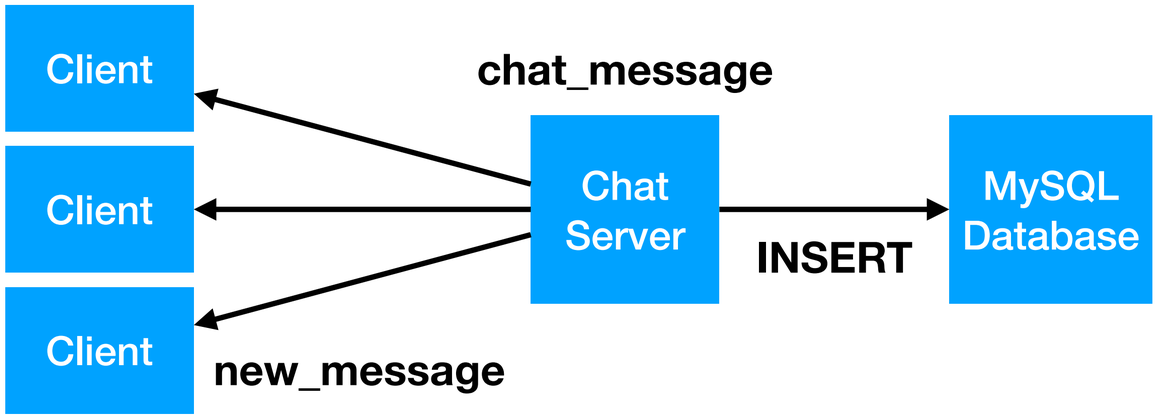
Chat App

- All users can send messages of type chat_message to the server
  - Message is sent when a user sends a message using the GUI
- This message only contains the message (No username)



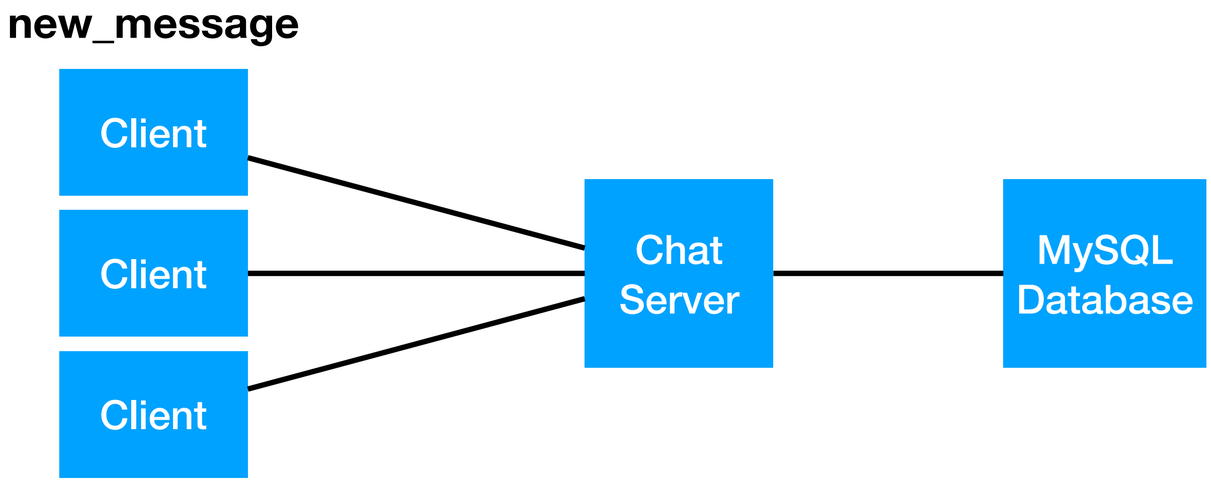
Chat App

- When the server receives a chat_message:
  - Lookup the username for the sending socket
  - Store username/message in the database
  - Send username/message to all connected sockets in a message of type new_message



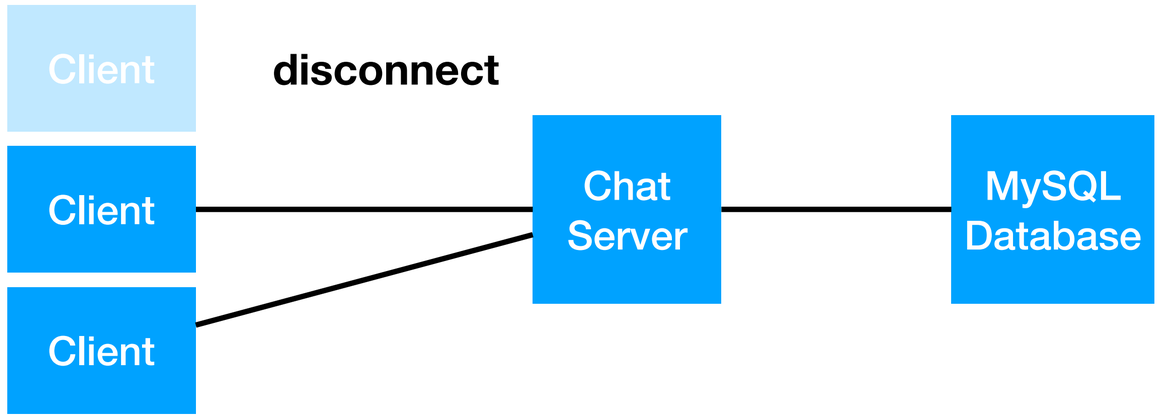
Chat App

- Clients receive the new_message
- Add it to the GUI for the user to read



Chat App

- When a client disconnects the server reacts to the disconnect event
- Remove the user from data structures



To the Code

Lecture Question

Task: Write a Web Socket Server that echos 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 sends back to the client a message of type "echo" containing the exact string sent by the client