

Project Report: 2

this pdf is converted from markdown. To view a rendered markdown click [here](#)

Since the last report, we've been implementing the Java client and we now have a working JavaFX application (scenes are built using `SceneBuilder`)

User Interface

For Project Voxx Java client, we have two main scenes: `Login scene` and `Chatbox scene`



Username

Placeholder Text

Start Chatting



Connected Users



Username Today at 10:30 PM

Lorem ipsum dolor sit amet, consectetur

Connected as
Username

Type message here...

To allow us to switch scenes with ease, we also implemented a utility class called `PrimaryStageManager` that contains a function called `#setScene(String fxml, Consumer<T> controllerConsumer)`. This function allows you to pass in a consumer where the instance of the controller is passed just in case you have to call functions of the controller when the fxml is loaded.

Controller Overview

Currently, the Voxx java client connection is implemented using the classes `ChatController` and `LoginController`.

Login Controller

`LoginController` is responsible for controlling the login screen of the application. When the scene is loaded, and the controller is initialized, it will construct the `ConnectionTask` and run it on a different thread. By default, Voxx will try to connect to the server with the address `localhost` and port `8008` (which is the voxx-server running on local machine). If there is no voxx-server running on localhost, the connection indicator will turn red. To connect to a different server, you can click the red dot and provide the server address and port using the following format: `server_address:port`. The controller will now then try to connect to that new server.

Once connected, we now need to register a new user. To do so, we need to provide a username and click the `start Chatting` button. It will check if the username that is entered is valid. We have designated a valid username to be between 4-7 characters, also allowing for numbers and underscores, however the username cannot start with numbers first. It then sends a request to the server to create a new user with the entered username. If the request is successful, the controller will now call `#setScene()` in the `PrimaryStageManager` to change the scene to the `Chatbox` scene.

Chat Controller

`ChatController` is responsible for controlling the user-interface experience of the application. It handles the display and sending of messages, updating user lists, and connecting to the server to receive updates.

Connection Overview

Currently, the Voxx java client connection is implemented utilizing the class `ReqResClientConnection` and, optionally, `UpdateMessageConnection`.

Request-Response Connection

`ReqResClientConnection` is the main socket connection to the server. This connection allows us to do a synchronous request-response type communication with the server. Sent and received messages are essentially a flatten JSON string but is represented with a `JSONObject` in the client code.

Update Message Connection

`UpdateMessageConnection` is used as a supplementary connection to the server. It is one way connection from the server (server->client) that listens to broadcasted update message from the server.

The client connects to the server via a TCP socket, where it then listens for incoming message from the server. This connection is implemented using a while loop that listens persistently, where as stated above, allows for the client to be updated on changes happening in the server such as a list of current users in the server and when a user leaves the server. `UpdateMessageConnection` implements `Runnable`, which executes as a separate thread and allows for the client to receive updates.

When the `run()` method is called, the client enters a loop and reads messages from the server using object `BufferedReader`. In instances where new messages arrive, it is parsed as a `json` object.

Trying Voxx

Disclaimer: This project is still in development and there are still some issues that might arise when trying out Voxx

A Voxx-server is currently running on `repo.cyr1en.com:8008`. All we need now is the Java client and connect to that server.

Building Voxx-Client

First of all, if you're reading this, that means you have access to the Voxx repo. So the first step is to clone the repo to your local machine. Once done, follow one of the steps below to build the client depending on your preference.

Building executable

This is a platform dependent executable for Voxx and will not open up a console. There is also an optional argument `-PskipInstaller` if you want to skip package Voxx to an installer (Only for Windows for now).

To build the executable

```
./gradlew clean :voxx-client:jpackage -PskipInstaller=true
```

This will build an executable that could be found in `Project-Voxx/voxx-client/build/jpackage/Voxx`

Building distributable

This is the distributable build for Voxx and does not contain any executables (other than the launchers).

To build distributable

```
./gradlew clean :voxx-client:build
```

This will produce a distributable `zip` file located in `Project-Voxx/voxx-client/build`. Unzip the file, go to `bin` and use the corresponding launcher for your OS.

Running your own server

A server build is available here: repo.cyr1en.com

To run the server

```
java -jar -voxx-server-{version}.jar
```

Previous Issues/Changes Made Since PR1

The `UID#asLong()` method was providing a timestamp that was a day ahead of the actual timestamp of when the UID was generate. This was corrected by subtracting our set `TIME_EPOCH`.

Protocol change for sending a chat message:

The prefix `message` for uid and content on the json object `message` as listed in PR1 was determined to be redundant, so it was removed. The protocol listed above for sending a chat message reflects what is current.

Protocol change for update-message:

The key for update-message was changed to `nm`, previously `ns`, for clarity and ease of understanding, where:

nm = new message

nu = new user

ud = user disconnect

Current issues:

Currently, there is a problem with the disconnect if the application is closed prematurely.

In instances where this occurs, the client does not disconnect. To resolve this issue, we may add a check on the server to verify if a client is still connected, and if not, remove automatically.

UML Diagram

The following image contains the UML diagram of all the implemented code that we have so far. The server UML has not changed, however we have added the client UML diagram:

```

classDiagram
    class EventBus {
        <<EventBus>>
        +EventBus()
        +executorServiceSupplier Supplier<ExecutorService>
        +main(String[]) void
        +subscribeListeners(Listener) void
        +iterateAnnotatedFunctions(Listener, RConsumer<Class?T, Method>) void
        +post(T) void
        +unsubscribeListeners(Listener) void
        +executorServiceSupplier Supplier<ExecutorService>
    }

    class Request {
        <<Request>>
        +onRequest(ClientMessageEvent) void
        +requestID String
        +params Map<String, Object>
    }

    class Response {
        <<Response>>
        +body Map<String, Object>
        +id String
        +responseID int
    }

    class Server {
        <<Server>>
        +Server(int, int)
        +clientConnections Array<List<ClientConnection>>
        +eventBus EventBus
        +run() void
        +clientConnections Array<List<ClientConnection>>
        +eventBus EventBus
    }

    class ClientConnection {
        <<ClientConnection>>
        +ClientConnection(Socket, EventBus)
        +in BufferedReader
        +out PrintWriter
        +remoteAddress String
        +isRunning boolean
        +accessToken User
        +close() void
        +sendMessage(String) void
        +run() void
        +clientSocket Socket
        +out PrintWriter
        +remoteAddress String
        +accessToken User?
        +isRunning boolean
        +offLong UID
        +toString() String
        +id int
        +username String
        +id UID
        +username String
    }

    class ClientMessageEvent {
        <<ClientMessageEvent>>
        +ClientMessageEvent(ClientConnection, String)
        +message ClientConnection
        +message String
        +message ClientConnection
        +message String
    }

    class ClientDisconnectEvent {
        <<ClientDisconnectEvent>>
        +ClientDisconnectEvent(ClientConnection)
        +ClientConnection() ClientConnection
    }

    class ClientConnectEvent {
        <<ClientConnectEvent>>
        +ClientConnectEvent(ClientConnection)
        +ClientConnection() ClientConnection
    }

    class Message {
        <<Message>>
        +Message(User, String)
        +content String
        +sender User
        +sender User
        +content String
    }

    class UID {
        <<UID>>
        +UID(long, int)
        +timestamp long
        +id int
        +hashCode() int
        +asLong() long
        +equals(Object) boolean
        +offLong UID
        +toString() String
        +id int
        +username String
        +id UID
        +username String
    }

    class User {
        <<User>>
        +User(UID, String)
        +uid UID
        +username String
        +id UID
        +username String
    }

    class Generator {
        <<Generator>>
        +Generator()
        +generate() UID
    }

    class RequestParser {
        <<RequestParser>>
        +RequestParser()
        +parse(ClientMessageEvent, VoxxServer) Request
    }

    class RequestNum {
        <<RequestNum>>
        +RequestNum(String, Class<Request>)
        +asString() String
        +valueOf(String) RequestNum
        +values() RequestNum[]
        +construct(VoxxServer) Request
    }

    class SendMsg {
        <<SendMsg>>
        +SendMsg(VoxxServer)
        +requestID String
        +params Map<String, Object>
    }

    class UserList {
        <<UserList>>
        +UserList(HashMap)
        +onRequest(ClientMessageEvent) void
    }

    class RegisterUser {
        <<RegisterUser>>
        +RegisterUser(VoxxServer)
        +requestID String
        +params Map<String, Object>
    }

    class ProtocolHandler {
        <<ProtocolHandler>>
        +ProtocolHandler(VoxxServer)
        +registerProtocols()
        +handleDrMessage(ClientMessageEvent) void
        +handleOnDisconnect(ClientDisconnectEvent) void
        +handleOnConnect(ClientConnectEvent) void
    }

    class Launcher {
        <<Launcher>>
        +Launcher()
        +main(String[]) void
    }

    class UserRegistry {
        <<UserRegistry>>
        +UserRegistry()
        +isRegistered(String) boolean
        +registerNewUser(String) User
    }

    EventBus "1" --> "1" Request : requestID
    EventBus "1" --> "1" Response : responseID
    Server "1" --> "1" ClientConnection : clientConnections
    ClientConnection "1" --> "1" ClientMessageEvent : ClientMessageEvent
    ClientConnection "1" --> "1" ClientDisconnectEvent : ClientDisconnectEvent
    ClientConnection "1" --> "1" ClientConnectEvent : ClientConnectEvent
    ClientConnection "1" --> "1" Message : Message
    ClientConnection "1" --> "1" UID : UID
    ClientConnection "1" --> "1" User : User
    ClientConnection "1" --> "1" Generator : Generator
    ClientConnection "1" --> "1" RequestParser : RequestParser
    ClientConnection "1" --> "1" RequestNum : RequestNum
    ClientConnection "1" --> "1" SendMsg : SendMsg
    ClientConnection "1" --> "1" UserList : UserList
    ClientConnection "1" --> "1" RegisterUser : RegisterUser
    ClientConnection "1" --> "1" ProtocolHandler : ProtocolHandler
    ClientConnection "1" --> "1" Launcher : Launcher
    ClientConnection "1" --> "1" UserRegistry : UserRegistry
    Request "1" --> "1" Response : responseID
    Request "1" --> "1" RequestParser : RequestParser
    Request "1" --> "1" RequestNum : RequestNum
    Request "1" --> "1" SendMsg : SendMsg
    Request "1" --> "1" UserList : UserList
    Request "1" --> "1" RegisterUser : RegisterUser
    Request "1" --> "1" ProtocolHandler : ProtocolHandler
    Request "1" --> "1" Launcher : Launcher
    Request "1" --> "1" UserRegistry : UserRegistry
    Response "1" --> "1" Request : requestID
    Response "1" --> "1" RequestParser : RequestParser
    Response "1" --> "1" RequestNum : RequestNum
    Response "1" --> "1" SendMsg : SendMsg
    Response "1" --> "1" UserList : UserList
    Response "1" --> "1" RegisterUser : RegisterUser
    Response "1" --> "1" ProtocolHandler : ProtocolHandler
    Response "1" --> "1" Launcher : Launcher
    Response "1" --> "1" UserRegistry : UserRegistry
    Server "1" --> "1" ClientConnection : clientConnections
    Server "1" --> "1" Request : requestID
    Server "1" --> "1" Response : responseID
    Server "1" --> "1" RequestParser : RequestParser
    Server "1" --> "1" RequestNum : RequestNum
    Server "1" --> "1" SendMsg : SendMsg
    Server "1" --> "1" UserList : UserList
    Server "1" --> "1" RegisterUser : RegisterUser
    Server "1" --> "1" ProtocolHandler : ProtocolHandler
    Server "1" --> "1" Launcher : Launcher
    Server "1" --> "1" UserRegistry : UserRegistry
    ClientConnection "1" --> "1" ClientMessageEvent : ClientMessageEvent
    ClientConnection "1" --> "1" ClientDisconnectEvent : ClientDisconnectEvent
    ClientConnection "1" --> "1" ClientConnectEvent : ClientConnectEvent
    ClientConnection "1" --> "1" Message : Message
    ClientConnection "1" --> "1" UID : UID
    ClientConnection "1" --> "1" User : User
    ClientConnection "1" --> "1" Generator : Generator
    ClientConnection "1" --> "1" RequestParser : RequestParser
    ClientConnection "1" --> "1" RequestNum : RequestNum
    ClientConnection "1" --> "1" SendMsg : SendMsg
    ClientConnection "1" --> "1" UserList : UserList
    ClientConnection "1" --> "1" RegisterUser : RegisterUser
    ClientConnection "1" --> "1" ProtocolHandler : ProtocolHandler
    ClientConnection "1" --> "1" Launcher : Launcher
    ClientConnection "1" --> "1" UserRegistry : UserRegistry
    ClientMessageEvent "1" --> "1" ClientDisconnectEvent : ClientDisconnectEvent
    ClientMessageEvent "1" --> "1" ClientConnectEvent : ClientConnectEvent
    ClientMessageEvent "1" --> "1" Message : Message
    ClientMessageEvent "1" --> "1" UID : UID
    ClientMessageEvent "1" --> "1" User : User
    ClientMessageEvent "1" --> "1" Generator : Generator
    ClientMessageEvent "1" --> "1" RequestParser : RequestParser
    ClientMessageEvent "1" --> "1" RequestNum : RequestNum
    ClientMessageEvent "1" --> "1" SendMsg : SendMsg
    ClientMessageEvent "1" --> "1" UserList : UserList
    ClientMessageEvent "1" --> "1" RegisterUser : RegisterUser
    ClientMessageEvent "1" --> "1" ProtocolHandler : ProtocolHandler
    ClientMessageEvent "1" --> "1" Launcher : Launcher
    ClientMessageEvent "1" --> "1" UserRegistry : UserRegistry
    ClientDisconnectEvent "1" --> "1" ClientConnectEvent : ClientConnectEvent
    ClientDisconnectEvent "1" --> "1" Message : Message
    ClientDisconnectEvent "1" --> "1" UID : UID
    ClientDisconnectEvent "1" --> "1" User : User
    ClientDisconnectEvent "1" --> "1" Generator : Generator
    ClientDisconnectEvent "1" --> "1" RequestParser : RequestParser
    ClientDisconnectEvent "1" --> "1" RequestNum : RequestNum
    ClientDisconnectEvent "1" --> "1" SendMsg : SendMsg
    ClientDisconnectEvent "1" --> "1" UserList : UserList
    ClientDisconnectEvent "1" --> "1" RegisterUser : RegisterUser
    ClientDisconnectEvent "1" --> "1" ProtocolHandler : ProtocolHandler
    ClientDisconnectEvent "1" --> "1" Launcher : Launcher
    ClientDisconnectEvent "1" --> "1" UserRegistry : UserRegistry
    ClientConnectEvent "1" --> "1" Message : Message
    ClientConnectEvent "1" --> "1" UID : UID
    ClientConnectEvent "1" --> "1" User : User
    ClientConnectEvent "1" --> "1" Generator : Generator
    ClientConnectEvent "1" --> "1" RequestParser : RequestParser
    ClientConnectEvent "1" --> "1" RequestNum : RequestNum
    ClientConnectEvent "1" --> "1" SendMsg : SendMsg
    ClientConnectEvent "1" --> "1" UserList : UserList
    ClientConnectEvent "1" --> "1" RegisterUser : RegisterUser
    ClientConnectEvent "1" --> "1" ProtocolHandler : ProtocolHandler
    ClientConnectEvent "1" --> "1" Launcher : Launcher
    ClientConnectEvent "1" --> "1" UserRegistry : UserRegistry
    Message "1" --> "1" UID : UID
    Message "1" --> "1" User : User
    Message "1" --> "1" Generator : Generator
    Message "1" --> "1" RequestParser : RequestParser
    Message "1" --> "1" RequestNum : RequestNum
    Message "1" --> "1" SendMsg : SendMsg
    Message "1" --> "1" UserList : UserList
    Message "1" --> "1" RegisterUser : RegisterUser
    Message "1" --> "1" ProtocolHandler : ProtocolHandler
    Message "1" --> "1" Launcher : Launcher
    Message "1" --> "1" UserRegistry : UserRegistry
    UID "1" --> "1" User : User
    UID "1" --> "1" Generator : Generator
    UID "1" --> "1" RequestParser : RequestParser
    UID "1" --> "1" RequestNum : RequestNum
    UID "1" --> "1" SendMsg : SendMsg
    UID "1" --> "1" UserList : UserList
    UID "1" --> "1" RegisterUser : RegisterUser
    UID "1" --> "1" ProtocolHandler : ProtocolHandler
    UID "1" --> "1" Launcher : Launcher
    UID "1" --> "1" UserRegistry : UserRegistry
    User "1" --> "1" Generator : Generator
    User "1" --> "1" RequestParser : RequestParser
    User "1" --> "1" RequestNum : RequestNum
    User "1" --> "1" SendMsg : SendMsg
    User "1" --> "1" UserList : UserList
    User "1" --> "1" RegisterUser : RegisterUser
    User "1" --> "1" ProtocolHandler : ProtocolHandler
    User "1" --> "1" Launcher : Launcher
    User "1" --> "1" UserRegistry : UserRegistry
    Generator "1" --> "1" RequestParser : RequestParser
    Generator "1" --> "1" RequestNum : RequestNum
    Generator "1" --> "1" SendMsg : SendMsg
    Generator "1" --> "1" UserList : UserList
    Generator "1" --> "1" RegisterUser : RegisterUser
    Generator "1" --> "1" ProtocolHandler : ProtocolHandler
    Generator "1" --> "1" Launcher : Launcher
    Generator "1" --> "1" UserRegistry : UserRegistry
    RequestParser "1" --> "1" RequestNum : RequestNum
    RequestParser "1" --> "1" SendMsg : SendMsg
    RequestParser "1" --> "1" UserList : UserList
    RequestParser "1" --> "1" RegisterUser : RegisterUser
    RequestParser "1" --> "1" ProtocolHandler : ProtocolHandler
    RequestParser "1" --> "1" Launcher : Launcher
    RequestParser "1" --> "1" UserRegistry : UserRegistry
    RequestNum "1" --> "1" SendMsg : SendMsg
    RequestNum "1" --> "1" UserList : UserList
    RequestNum "1" --> "1" RegisterUser : RegisterUser
    RequestNum "1" --> "1" ProtocolHandler : ProtocolHandler
    RequestNum "1" --> "1" Launcher : Launcher
    RequestNum "1" --> "1" UserRegistry : UserRegistry
    SendMsg "1" --> "1" UserList : UserList
    SendMsg "1" --> "1" Register
```

```

classDiagram
    class ReqResClientConnection {
        socket Socket
        reader BufferedReader
        out PrintWriter
        sendRequest(JSONObject) JSONObject
        run() void
        closeConnection() void
        isConnected() boolean
        getReader() BufferedReader
    }
    class UpdateMessageConnection {
        reader BufferedReader
        socket Socket
        onUpdateMessage(Consumer<JSONObject>) void
        getReader() BufferedReader
        run() void
        isConnected() boolean
        onUpdateMessage(Consumer<JSONObject>) void
    }
    class ChatController {
        nameLabel Label
        msgField TextField
        userList ListView<String>
        cBoxScrollPane ScrollPane
        system User
        updateMessageConnection UpdateMessageConnection
        instance VoxxApplication
        chatBox VBox
        removeUserList(String) void
        startTask() void
        initialize() void
        setVoxxApplication(VoxxApplication) void
        onKeyReleased(KeyEvent) void
        getVoxxInstance() VoxxApplication
        addMessage(Message) void
        addUserList(String) void
        updateUserList() void
        setUnamelLabel(String) void
    }
    class VoxxApplication {
        assocUser User
        client ReqResClientConnection
        stageManager PrimaryStageManager
        SERVER_PORT int
        HOST_PORT_REGEX Pattern
        SERVER_HOST String
        start(Stage) void
        setClient(ReqResClientConnection) void
        main(String[]) void
        getClient() ReqResClientConnection
        setAssocUser(User) void
        getStageManager() PrimaryStageManager
        changeHost(String) void
        isConnected() boolean
        getAssocUser() User
    }
    class LoginController {
        F_GREEN Color
        isAttemptingConnect boolean
        warningLabel Label
        unameField TextField
        F_RED Color
        scButton Button
        F_ORANGE Color
        connectionStatus Circle
        voxxApplication VoxxApplication
        UNAME_REGEX Pattern
        connectToServer(boolean) void
        setVoxxApplication(VoxxApplication) void
        onStartChattingPress() void
        onStatusCircleClick() void
        initialize() void
    }
    class PrimaryStageManager {
        primaryStage Stage
        instance VoxxApplication
        setScene(String, Consumer<T>) void
    }
    ReqResClientConnection "1" -- "1" UpdateMessageConnection : client
    UpdateMessageConnection "1" -- "1" ChatController : updateMessageConnection
    ChatController "1" -- "1" VoxxApplication : instance
    VoxxApplication "1" -- "1" LoginController : voxxApplication
    PrimaryStageManager "1" -- "1" VoxxApplication : stageManager
    PrimaryStageManager "1" -- "1" VoxxApplication : instance
  
```