COMP90015 Assignment 2 Report

Name: Duc Trung Nguyen

Username: ductrungn

Student Id: 1069760

1. Problem Context

The objective of this assignment is to implement a shared whiteboard system that supports multiple users to draw simultaneously on a shared canvas. The system provides the following features:

- Client can play as a manager role to create a shared whiteboard.
- Client can play as a user role to join a shared whiteboard.
- Multiple clients can simultaneously draw the line, circle, rectangle, and text on a shared whiteboard.
- Whiteboard manager can perform functionalities such as below:
    - New a shared whiteboard.
    - Open a PNG image and load it to a new shared whiteboard.
    - Save the current whiteboard to the PNG image file.
    - Save As the current whiteboard to the PNG image file.
    - Close the shared whiteboard and shutdown application.
    - Kick a certain user out.

The socket communication, threading, synchronization, Graphic2D technologies are chosen to implement server and client applications.

2. User Interface

2.1. Whiteboard Server

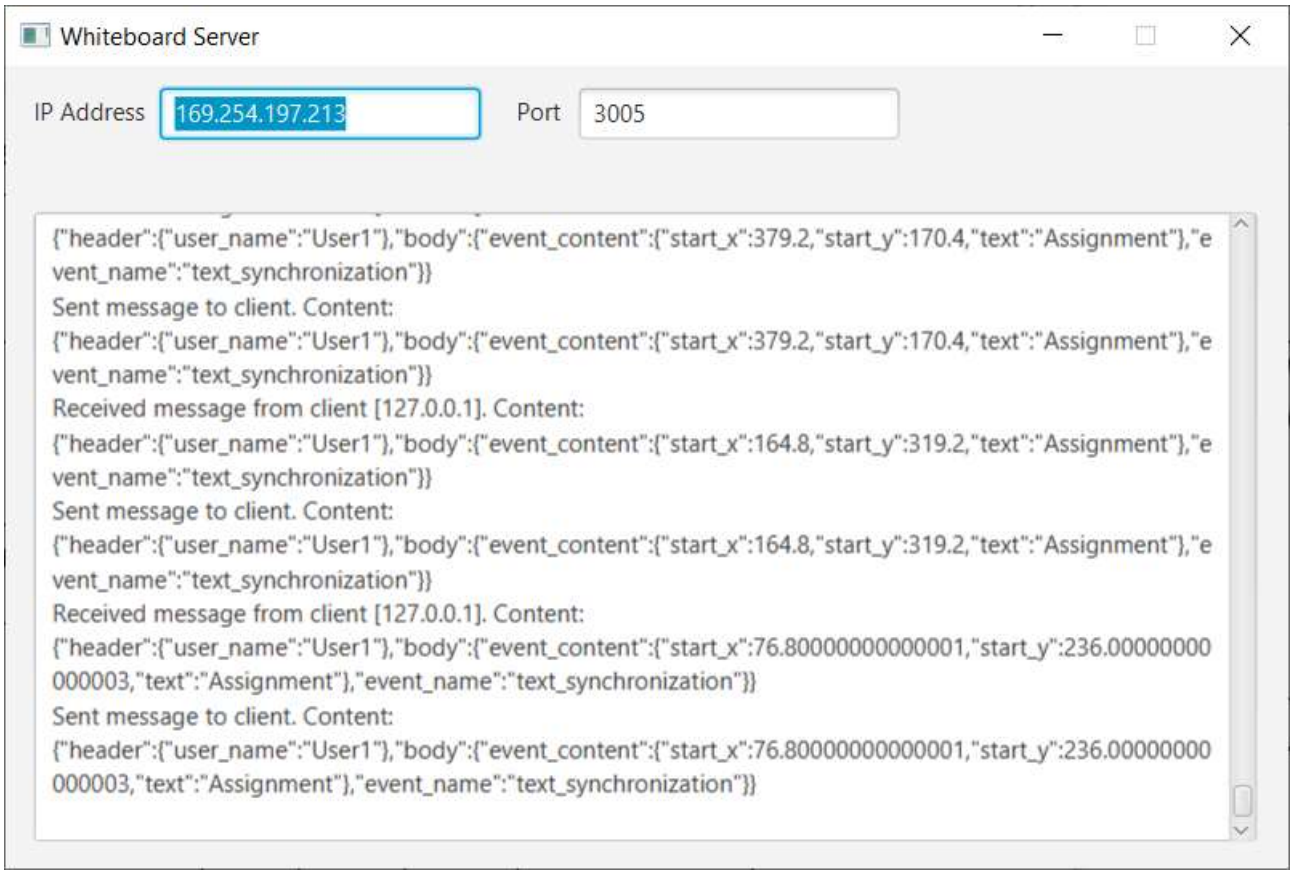

Figure 1 The Whiteboard Server application

- IP Address text filed: show sever computer's IP address.
- Port text field: show port using to make socket communication between server and clients.
- Application Log text area: show all processing logs between server and clients.

2.2. Whiteboard Manager

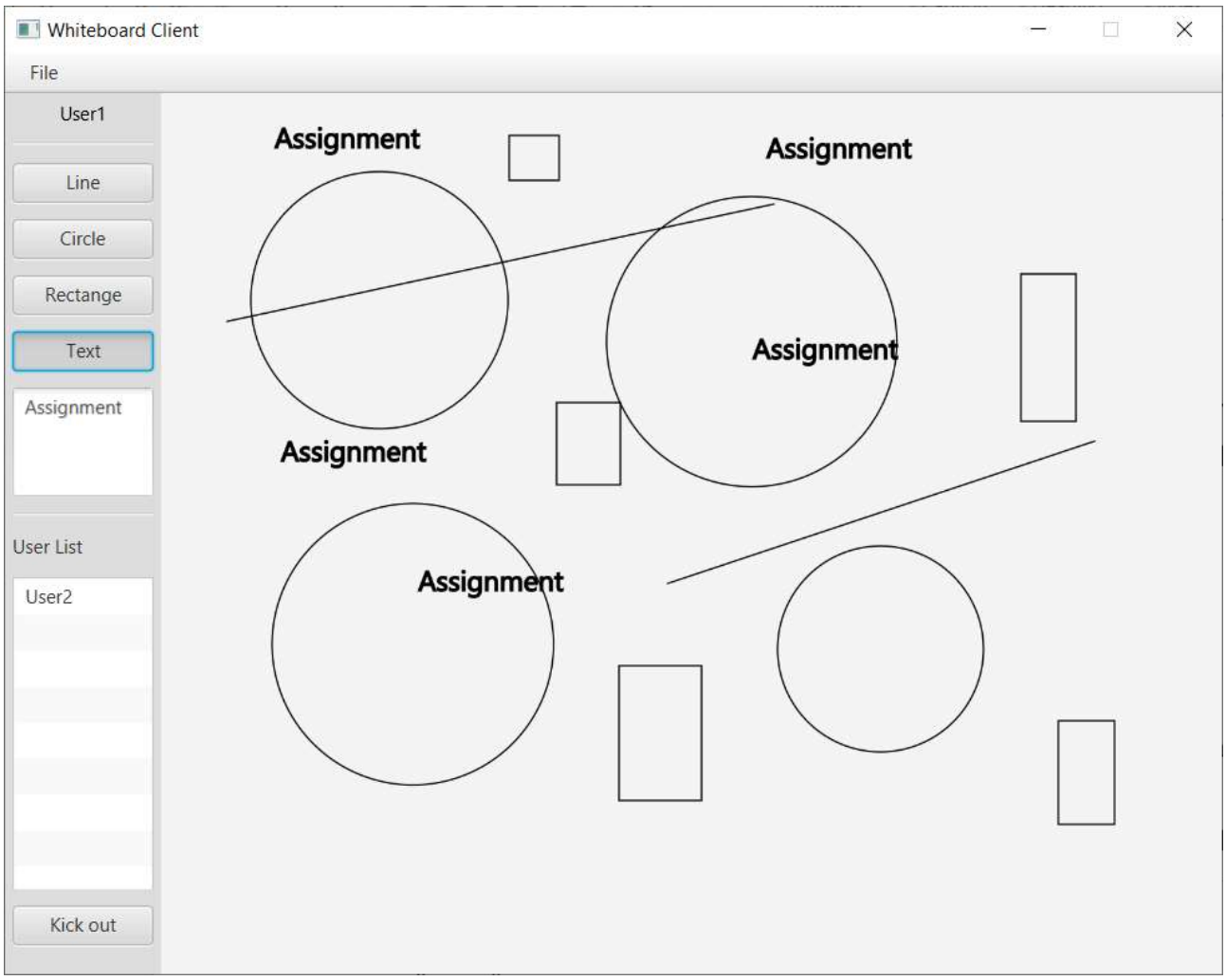

Figure 2 The Whiteboard Manager application

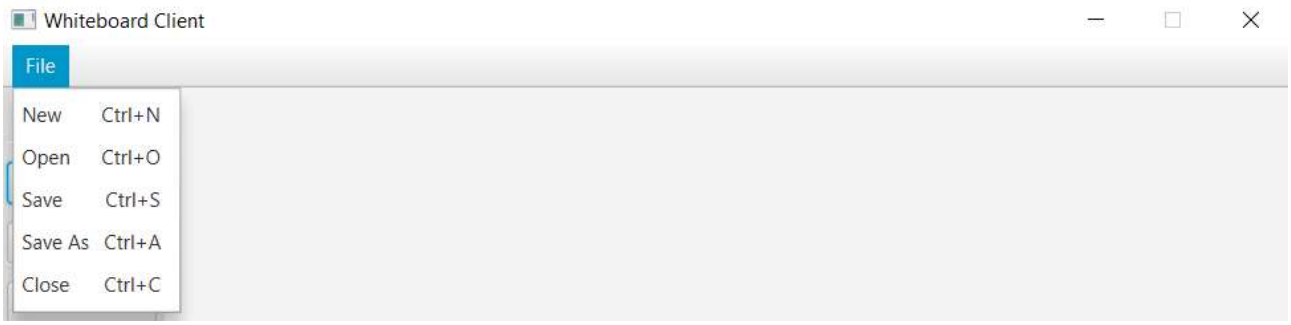

Figure 3 The Whiteboard Manager menu items

- User label: displays the current username. The username must be unique in the system.
- The manager can draw shape to the shared whiteboard from Line, Circle, Rectangle, and Text toggle buttons on toolbar on the left.
- Text area: manager needs to put text here that he/she wants draw text on the share whiteboard.
- User List: displays all approved online users on the shared whiteboard.
- Manager can kick a user out by selecting user on User List, and then click on Kick out button.
- Manger can perform New, Open, Save, Save As operation. He/she also can close the shared whiteboard and shutdown application.

2.3. Whiteboard User

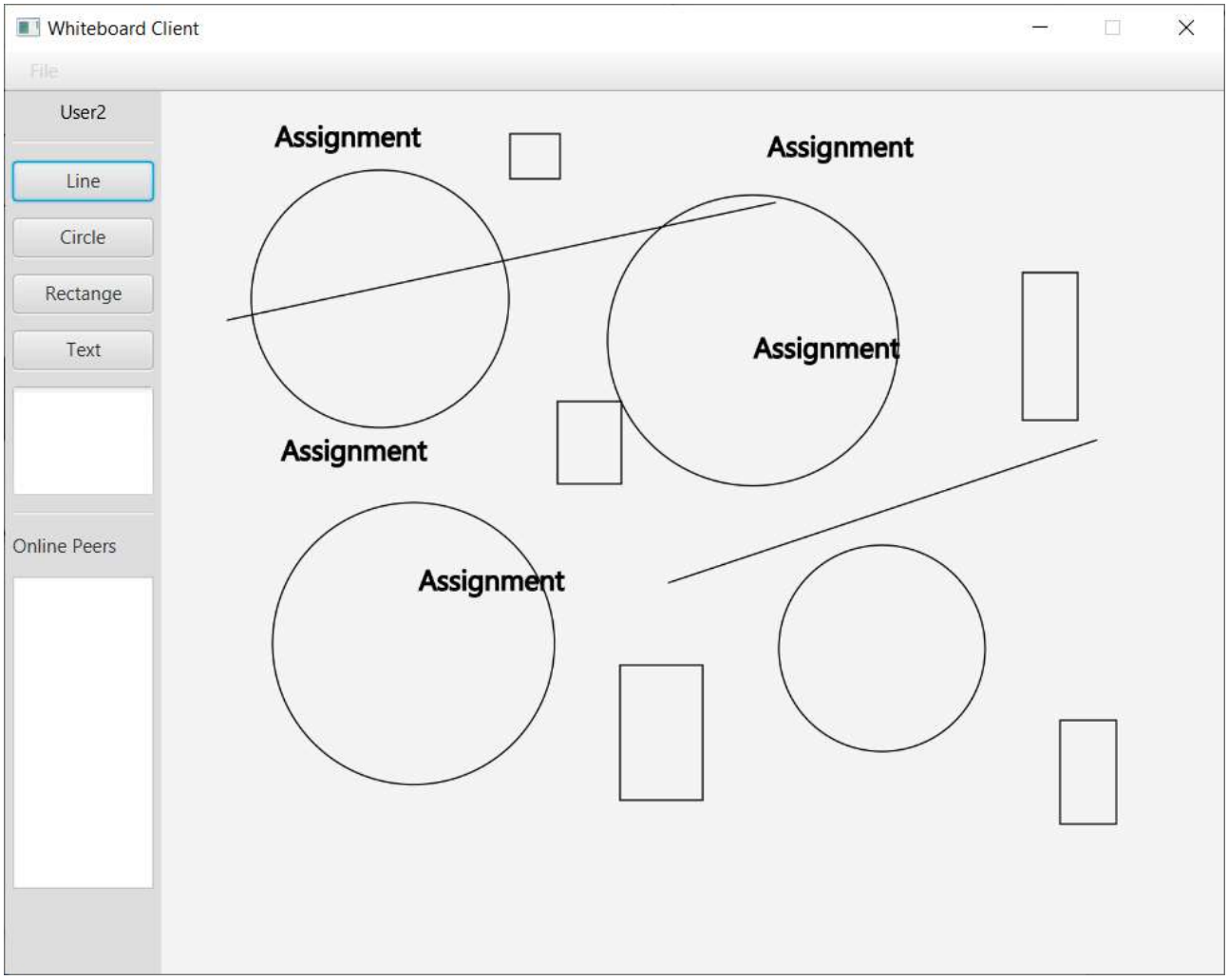

Figure 4 Whiteboard User application

- User label: displays the current username. The username must be unique in the system.
- The user can draw shape to the shared whiteboard from Line, Circle, Rectangle, and Text toggle buttons on toolbar on the left.
- Text area: user needs to put text here that he/she wants draw text on the share whiteboard.
- Online Peers: displays all approved online users on the shared whiteboard.

3. Architecture

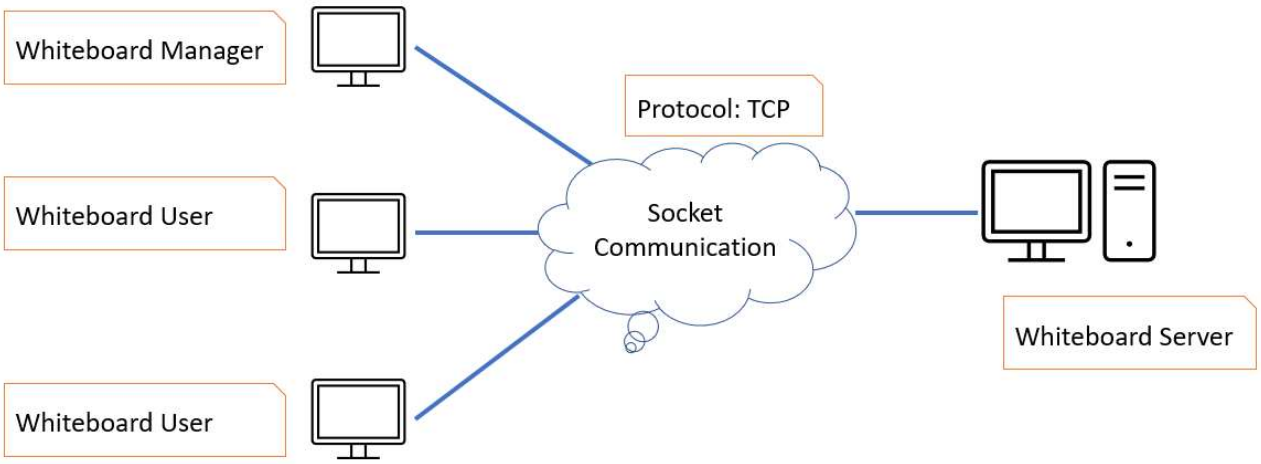

Figure 5 The system architecture

- Server is responsible for making communication and data transferring between whiteboard manager and whiteboard users.
- Server accept and manage all client socket connections.
- Whiteboard Client (Manger and Users) connects to Server to perform draw operations on shared whiteboard.
- The socket communication takes place between Server and Client through TCP protocol.

3.1. Whiteboard Server

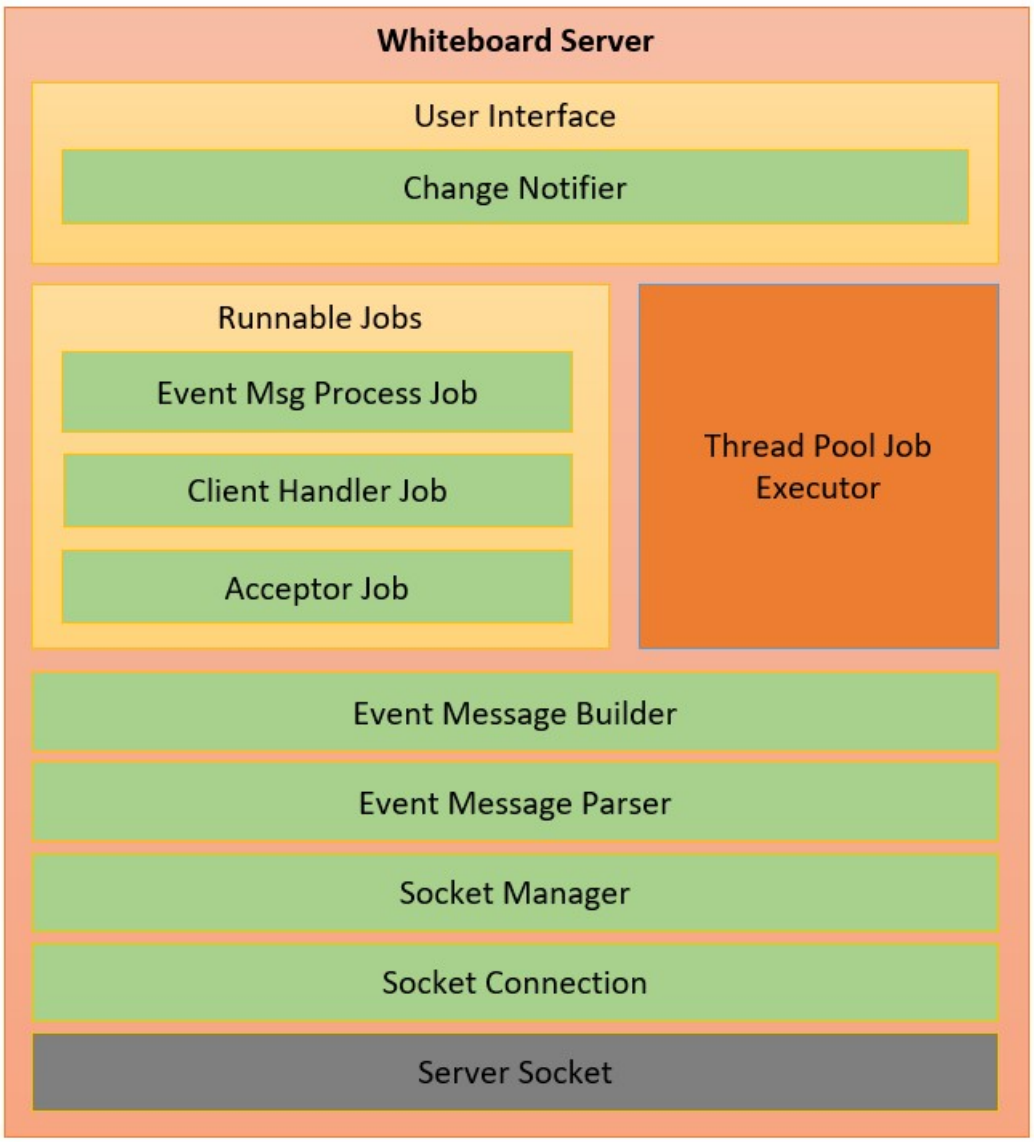

Figure 6 The Whiteboard Server architecture

- **Change Notifier:** is responsible for updating communication messages to the application log.
- **Thread Pool Job Executor**: is responsible for managing allocated threads and executing user defined jobs.
- **Event Msg Process Job**: is responsible for processing all communication event messages between Whiteboard Manager, Whiteboard Users and Whiteboard Server.
- **Client Handler Job**: is responsible for handling the client (Whiteboard Manager or Whiteboard User) socket communication.
- **Acceptor Job**: is responsible for accepting the client (Whiteboard Manager or Whiteboard User) socket connections.
- **Event Message Builder**: is responsible for building all communication event messages between Server and Clients.
- **Event Message Parser**: is responsible for parsing expected values from communication event messages.

- **Socket Manager**: is responsible for managing all client (Whiteboard Manager or Whiteboard User) socket connections.
- **Socket Connection**: is responsible for establishing the socket connection, sending, and receiving communication event messages through the socket input and output streams.

3.2. Whiteboard Client

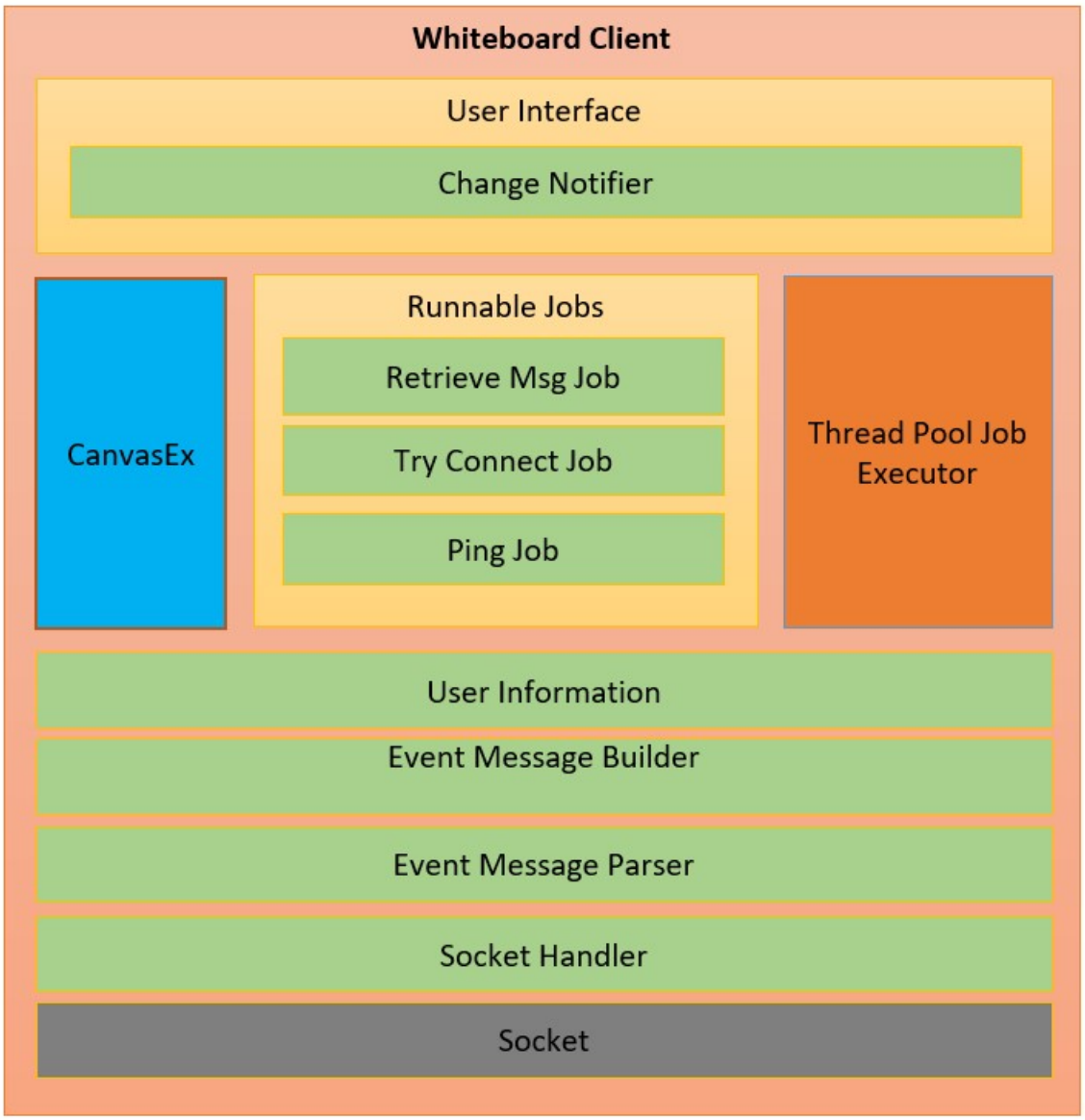

Figure 7 The Whiteboard Client architecture

- **Change Notifier:** is responsible for notifying communication event messages to whiteboard client scene controller.
- **Thread Pool Job Executor**: is responsible for managing allocated threads and executing user defined jobs.
- **CanvasEx**: is responsible for drawing shapes (line, circle, rectangle) and text to canvas of shared whiteboard.
- **Retrieve Msg Job**: is responsible for retrieving the response message from Server.
- **Try Connect Job**: is responsible for re-establishing the socket connection to Server.

- **Ping Job**: is responsible for detecting the socket connection alive status.
- **User Information**: is responsible for managing username and user role.
- **Event Message Builder**: is responsible for building all communication event messages between Server and Clients.
- **Event Message Parser**: is responsible for parsing expected values from communication event messages.
- **Socket Handler**: is responsible for establishing the socket connection, sending, and receiving messages through the socket input and output streams.

4. Class Diagram

4.1 Whiteboard Server

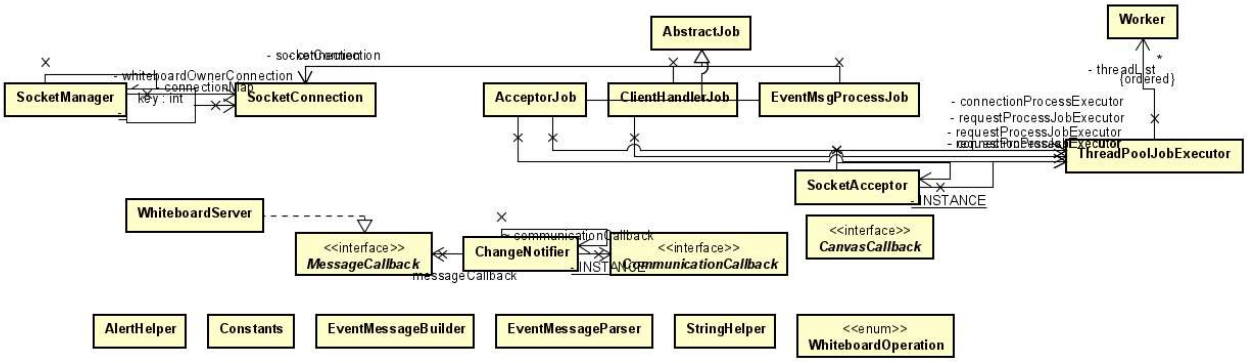

Figure 8 The Whiteboard Server class diagram

4.2 Whiteboard Client

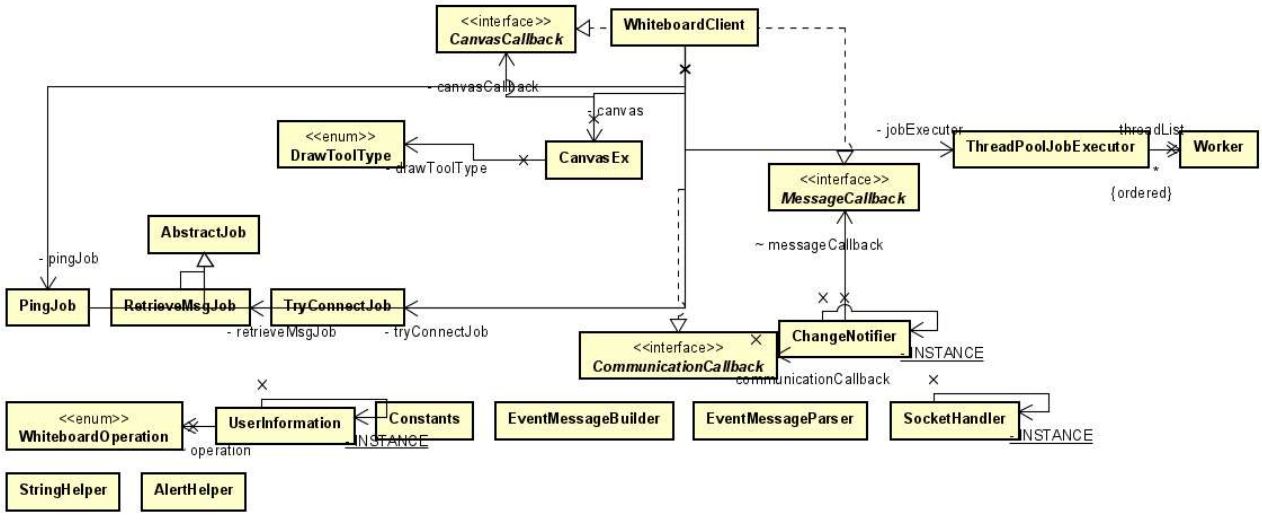

Figure 9 The Whiteboard Client class diagram

5. Communication Event Message Format

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "client_shutdown_notification",
        "event_content": {
            "has_manager_role": "true/false"
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "whiteboard_owner_shutdown_notification",
        "event_content": {
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "ping",
        "event_content": {
            "host_address" : "host name or IP address"
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "handshake_establishment",
        "event_content": {
            "has_manager_role" : "true/false"
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "handshake_ack",
        "event_content": {
            "ack" : "ack value"
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "request_whiteboard_join_approval",
        "event_content": {
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "request_whiteboard_join_approval_ack",
        "event_content": {
            "ack" : "ack value"
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "user_added_notification",
        "event_content": {
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "user_removed_notification",
        "event_content": {
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "manager_kick_user_out",
        "event_content": {
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "all_online_users_synchronization",
        "event_content": {
            "user_name_list" : ["user1, user2, user3, ..."]
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "circle_synchronization",
        "event_content": {
            "center_x" : "value",
            "center_y" : "value",
            "radius" : "value"
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "rectangle_synchronization",
        "event_content": {
            "start_x" : "value",
            "start_y" : "value",
            "width" : "value",
            "height": "value"
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "text_synchronization",
        "event_content": {
            "start_x" : "value",
            "start_y" : "value",
            "text" : "value"
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "whiteboard_cleared_notification",
        "event_content": {
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "whiteboard_new_updated_image_notification",
        "event_content": {
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "whiteboard_synchronization",
        "event_content": {
        }
    }
}
```

```
{
    "header": {
        "user_name" : "{username}"
    },
    "body": {
        "event_name": "whiteboard_synchronization_ack",
        "event_content": {
            "image_as_string" : "value",
            "broadcast_new_image" : "true/false"
        }
    }
}
```

6. Basic Scenario

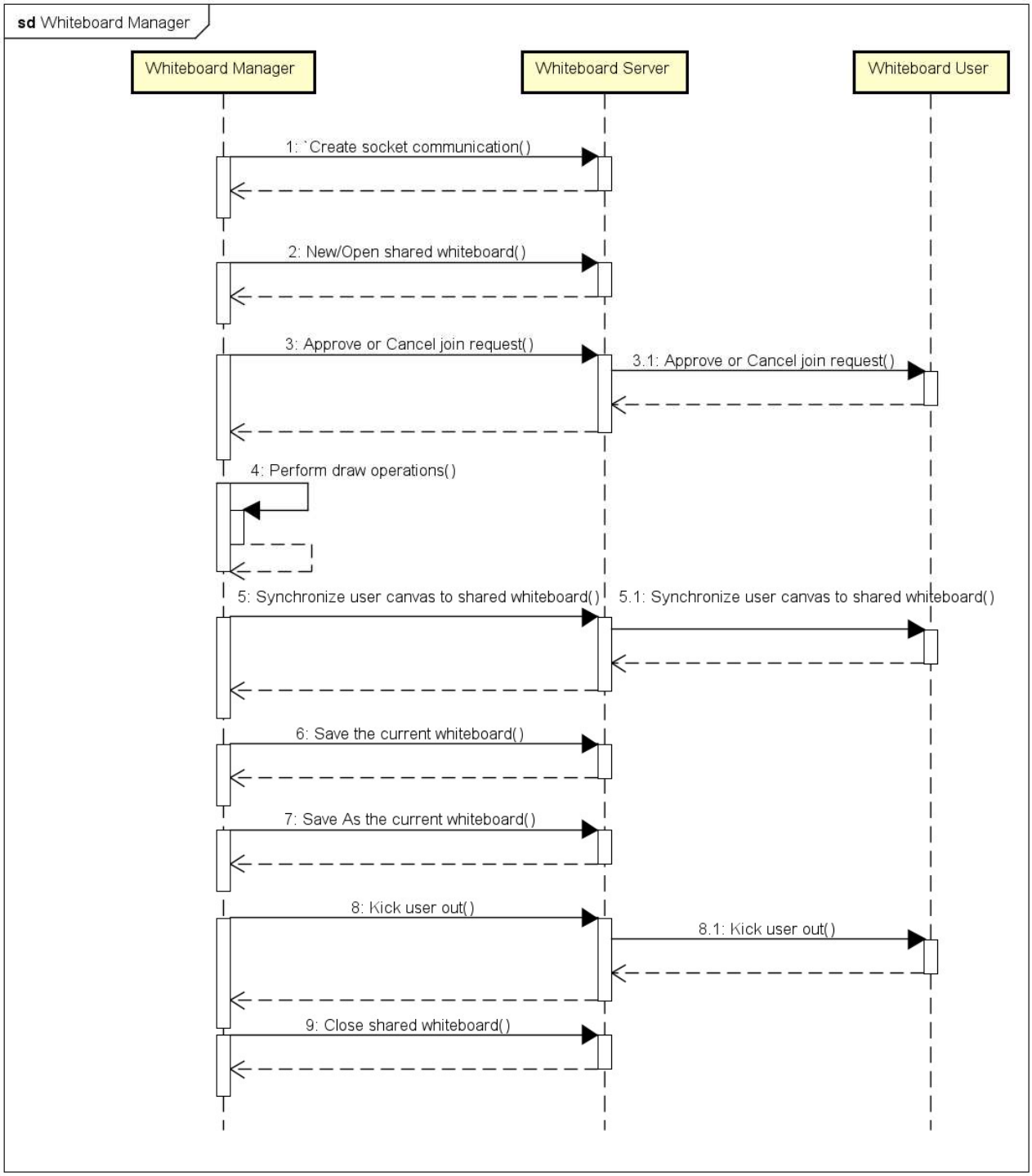

Figure 10 The Whiteboard Manager basic scenario

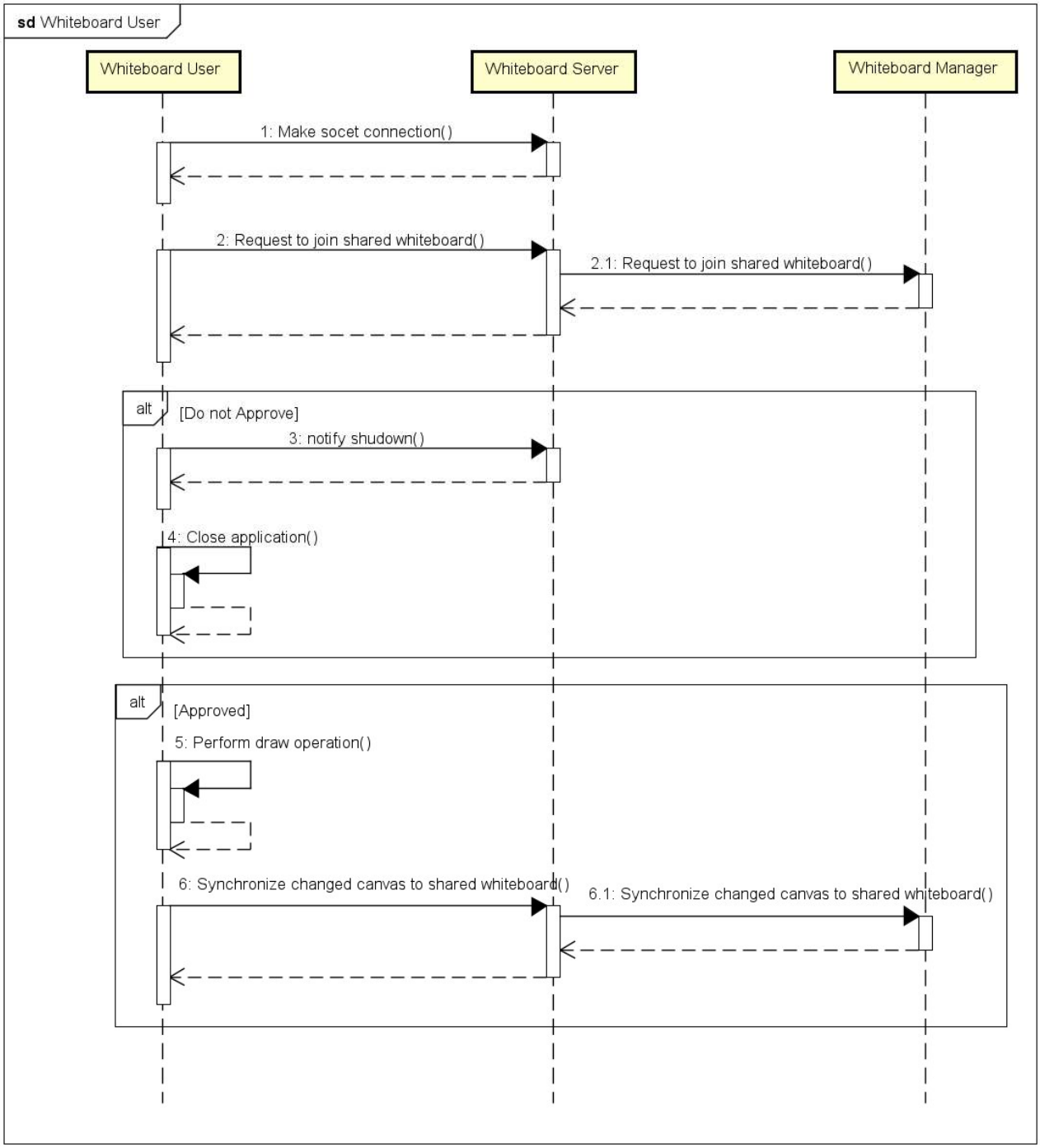

Figure 11 The Whiteboard User basic scenario

7. Basic Features

- The system is carefully designed and coded for readable, extensible, and maintainable abilities.
- Application is built on JavaFx so that it is easy to create, manage, and customize the user interface.
- A user interface is introduced for Whiteboard Server application to display IP address and port using to establish a socket connection from the client-side. Furthermore, all processing logs between Server and Clients are displayed on GUI so that the user can track the workflow as well as run-time errors.
- Whiteboard client can play as Manager or User role in the system. All users can simultaneously draw the line, circle, rectangle, and text on a shared interactive whiteboard.

8. Advance Features

- A whiteboard user with manager role can new a whiteboard, load an image to whiteboard, save/save as the current whiteboard to the PNG image file, close the shared whiteboard and shutdown application. This user also can kick a certain user out.
- The Whiteboard Manger application will show a confirmation dialog to ask manger if he/she wants save changes of the current shared whiteboard to image file before creating new shared whiteboard or opening excited shared whiteboard.
- Design a thread pool job executor to manage threads and processing jobs rather than using Java's built-in thread pool.
- All client requests are queued and dispatched to allocated threads to concurrently process requests. Therefore, Sever GUI application is not freeze even if many Client applications are concurrently sending the request to Server.
- Do not need to launch Whiteboard Server application before launching Whiteboard Client application. The Client application will automatically establish the socket connection when Whiteboard Server is ready.
- The Whiteboard Client application will automatically re-establish the socket connection to Server when the user closes Whiteboard Server application or Whiteboard Server application is interrupted.
- The Whiteboard Client application will send a notification message to Whiteboard Server application when it is shutting down so that Whiteboard Server application has a chance to close corresponding socket connection and clean up all allocated resources for this Whiteboard Client.

9. Deployment

- The release package includes:
    - WhiteboardServer.jar
    - WhiteboardClient.jar
    - json-simple-1.1.jar external library.
- Make sure that JDK 12 is installed in the system and JavaFx 11 libraries folder is configured in the system variable in JAVA_FX_PATH name.
- Copy WhiteboardServer.jar and WhiteboardClient.jar to C:\Demo. Copy json-simple-1.1.jar to C:\Demo\Libs
- Open terminal and run the following commands:
    - **Whiteboard Server**: java --module-path %JAVA_FX_PATH% --add-modules javafx.controls,javafx.fxml -classpath "c:\Demo\;c:\Demo\Libs\json-simple-1.1.jar;%JAVA_FX_PATH%\javafx.base.jar;%JAVA_FX_PATH%\javafx.controls.jar;%JAVA_FX_PATH%\javafx.fxml.jar;%JAVA_FX_PATH%\javafx.graphics.jar;%JAVA_FX_PATH%\javafx.media.jar;%JAVA_FX_PATH%\javafx.swing.jar;%JAVA_FX_PATH%\javafx.web.jar;%JAVA_FX_PATH%\javafx-swt.jar" -jar WhiteboardServer.jar 3005
    - **Whiteboard Manager**: java --module-path %JAVA_FX_PATH% --add-modules javafx.controls,javafx.fxml -classpath "c:\Demo\;c:\Demo\Libs\json-simple-1.1.jar;%JAVA_FX_PATH%\javafx.base.jar;%JAVA_FX_PATH%\javafx.controls.jar;%JAV

A_FX_PATH%\javafx.fxml.jar;%JAVA_FX_PATH%\javafx.graphics.jar;%JAVA_FX_PATH%\javafx.media.jar;%JAVA_FX_PATH%\javafx.swing.jar;%JAVA_FX_PATH%\javafx.web.jar;%JAVA_FX_PATH%\javafx-swt.jar" -jar WhiteboardClient.jar create localhost 3005 User1

- **Whiteboard User**: java --module-path %JAVA_FX_PATH% --add-modules javafx.controls,javafx.fxml -classpath "c:\Demo\;c:\Demo\Libs\json-simple-1.1.jar;%JAVA_FX_PATH%\javafx.base.jar;%JAVA_FX_PATH%\javafx.controls.jar;%JAVA_FX_PATH%\javafx.fxml.jar;%JAVA_FX_PATH%\javafx.graphics.jar;%JAVA_FX_PATH%\javafx.media.jar;%JAVA_FX_PATH%\javafx.swing.jar;%JAVA_FX_PATH%\javafx.web.jar;%JAVA_FX_PATH%\javafx-swt.jar" -jar WhiteboardClient.jar join localhost 3005 User2

10. Conclusion

A solution is provided to demonstrate knowledge about distributed system aspects:

- Client-Server architecture.
- Socket programming through TCP protocol.
- Threading and synchronization.
- Java 2D graphic programming.
- User interface programming.

11. Reference

Buyya, R., Selvi, S.T. and Chu, X., 2009. Object-oriented Programming with Java: Essentials and Applications. Tata McGraw-Hill.