

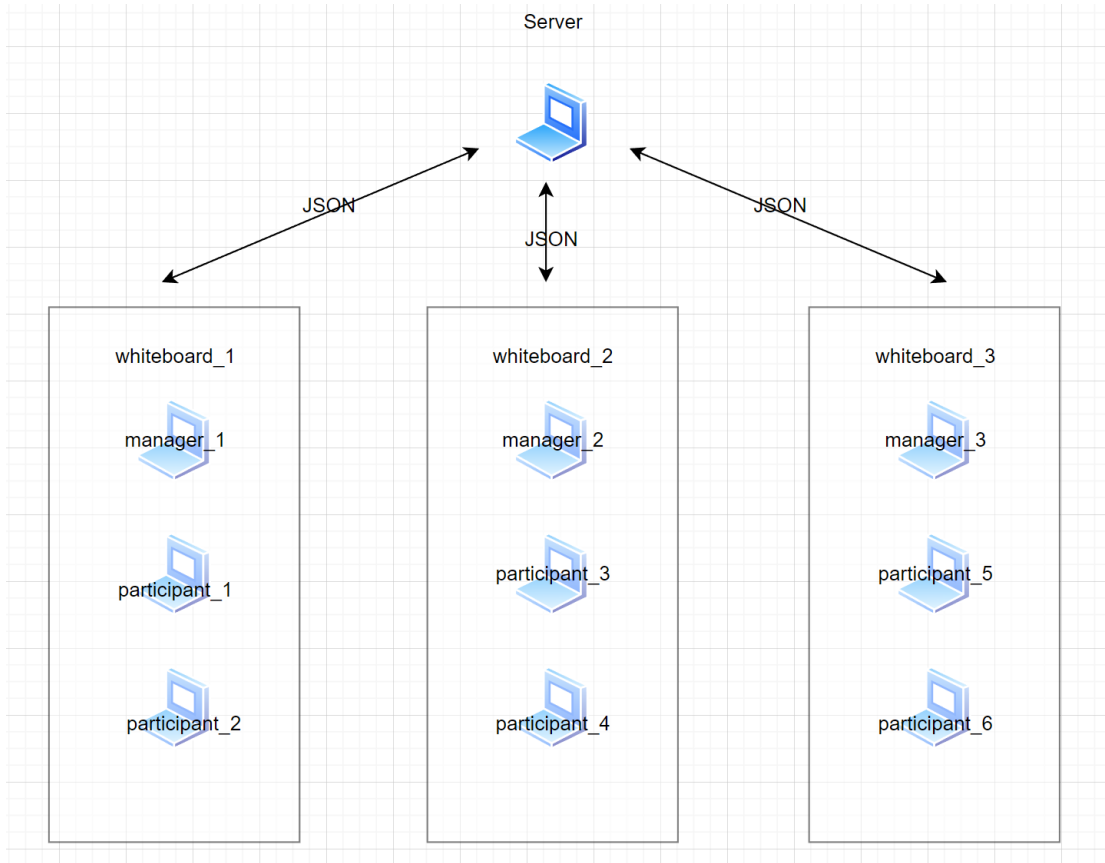


Distributed System and Application

Assignment 02

Student Name	Jiahao Shen
Student ID	1381187
Date	05/01/2023
Subject	COMP90015 Distributed Systems

System architecture



Common

The common module serves as the foundation for the system by providing essential classes and utilities that facilitate user management, shape rendering, and message handling. It is designed to simplify the interaction between different modules, promote code reusability and maintainability, and make it easier to build and extend the functionality of the system.

Server

The server-side of the system architecture consists of a WebSocket server (tyrus-server) that manages connections and communication between clients. It primarily handles user and whiteboard management by maintaining a list of active users and their corresponding whiteboards. The server uses the Gson library for JSON serialization and deserialization, allowing it to efficiently process messages received from clients. Additionally, the server handles various message types, such as creating and joining whiteboards, drawing actions, and chat messages. The server also manages the real-time chat functionality, enabling users to communicate and collaborate effectively. To ensure data consistency, the server maintains caches for shape lists and background images of each whiteboard. Upon a client's disconnection, the server takes care of removing the user from the list and updating the whiteboard participants accordingly.

Create

The Create module serves as the foundation for user-generated content and real-time collaboration within the system. It is responsible for enabling users to create a shared whiteboard and manage its content. This module handles UI interactions, drawing features, and communication with the server. As the primary interface for creating a whiteboard, it allows users to draw shapes, open and save images, and collaborate with others in real-time.

Join

The Join module allows users to join existing shared whiteboards assigned by the system and interact with their content. It is responsible for handling UI interactions, processing of received messages, and communication with the server. As the primary interface for joining a whiteboard, it enables users to access

the assigned whiteboard, contribute to the content of the board, as well as participate in real-time chat and receive updates on the board's status.

Communication protocols

The collaborative whiteboard application utilizes WebSocket technology for real-time, bidirectional communication between clients and the server. This choice ensures low-latency interactions and efficient exchange of data, which is essential for the seamless collaboration experience provided by the system.

The server and clients exchange messages in JSON format, which allows for easy serialization and deserialization of complex data structures. The Gson library is used on both the server and client sides for handling JSON data, streamlining the communication process.

Messages between clients and the server are structured as objects with specific types and content, which helps to simplify message handling and processing. Some of the message types include:

1. **CREATE_PARTICIPANT:** A client sends this message to request to join a whiteboard. The server responds with either `CREATE_PARTICIPANT_ACCEPTED` or `CREATE_PARTICIPANT_REJECTED`.
2. **DRAWN:** A client sends this message to notify the server of new drawing actions. The server broadcasts the message to all other clients connected to the same whiteboard.
3. **CHAT_MESSAGE:** A client sends this message to participate in the real-time chat. The server broadcasts the message to all other clients connected to the same whiteboard.
4. **KICK_PARTICIPANT:** The server sends this message to a specific client when they are removed from the whiteboard.
5. **GET_ALL_WHITEBOARDS and GET_ALL_USERS:** Clients can request a list of all available whiteboards and online users with these messages, respectively.
6. **INIT_WHITEBOARD, NEW_WHITEBOARD, OPEN_WHITEBOARD, SAVE_WHITEBOARD, CLOSE_WHITEBOARD:** These message types handle various whiteboard-related actions such as initializing, creating, opening, saving, and closing whiteboards.
7. **SERVER_SHUTDOWN:** The server sends this message to inform clients that it is shutting down.

In addition to these message types, there are various other messages for handling different actions and maintaining smooth communication between the server and clients. The use of a well-defined communication protocol with structured message types enables efficient and organized interactions between the server and clients, contributing to the overall performance and user experience of the collaborative whiteboard application.

Message formats

In the collaborative whiteboard application, messages exchanged between the clients and the server follow a standardized format to ensure efficient communication and accurate processing of information. These messages are serialized and deserialized using the Gson library, which allows for the easy conversion between JSON data and Java objects.

The messages are structured as objects with specific properties, including:

Type: A property that denotes the message's purpose, such as `DRAWN`, `CHAT_MESSAGE`, or `CREATE_PARTICIPANT`. This property helps both the server and clients identify and process messages accordingly.

Sender: A property that represents the username of the client who sent the message. This information is particularly useful for chat messages and user-specific actions, such as kicking a participant from a whiteboard.

Content: A property that contains the main payload of the message, which can vary depending on the message type. For instance, the content of a `DRAWN` message includes the drawing action details, while the content of a `CHAT_MESSAGE` contains the text of the chat.

Timestamp: A property that records the time at which the message was sent. This information can be useful for ensuring the correct ordering of messages or displaying timestamps in chat messages.

The message format is designed to be flexible and extensible, allowing for the addition of new message types and properties as needed. This standardized format contributes to the system's maintainability, simplifies message handling, and ensures consistent communication between clients and the server.

Here is an example of a JSON message in the DRAWN message format:

```
{
  "type": "DRAWN",
  "content":
  "[{\\"centerX\\":332.6666666666667,\\\"centerY\\":120.0,\\\"radius\\":101.34210488351927,\\\"color\\":\\\"0x000000ff\\\",\\\"type\\":\\\"Circle\\\"}]",
  "sender": "ADMIN",
  "timestamp": {
    "date": {
      "year": 2023,
      "month": 4,
      "day": 27
    },
    "time": {
      "hour": 15,
      "minute": 19,
      "second": 36,
      "nano": 884089300
    }
  }
}
```

In this example, the message has a DRAWN type, a sender username, content containing the drawing action details in JSON format, and a timestamp indicating the time the message was sent.

Design diagrams (class and interaction)

Due to the large size of the image, it cannot be directly displayed in the PDF document like below. Therefore, you may find a link ([class and interaction.png](#)) provided to view the image.



Implementation details

In this section, the key implementation details of the collaborative whiteboard application will be covered, based on the information provided. These aspects play an essential role in the system's functionality and performance.

Synchronization of Whiteboard Content: To maintain a consistent view of the whiteboard across all clients, the server processes drawing actions and forwards them to all connected participants. When a client performs a drawing action, it sends a `DRAWN` message containing the relevant shape details. The server then broadcasts this message to other clients, ensuring that they update their views accordingly.

Base64 Encoded Images: To efficiently transmit images, such as whiteboard backgrounds, the system uses Base64 encoding. This method allows images to be represented as text, making them suitable for transmission over WebSocket connections in JSON messages. When a client sets a new background image, it sends an `OPEN_WHITEBOARD` message with the Base64 encoded image as its content. The server then forwards this message to other clients, allowing them to decode the image and update their views.

Real-time Chat: The chat feature is implemented using the `CHAT_MESSAGE` message type. When a client sends a chat message, it includes the sender's username, message content, and a timestamp. The server then broadcasts this message to all connected participants, allowing them to update their chat interfaces with the new message.

User and Whiteboard Management: The server maintains a list of active users and their assigned whiteboards. When a client requests to join or create a whiteboard, the server processes the request and assigns the client to an appropriate whiteboard. Messages such as `CREATE_PARTICIPANT_ACCEPTED` or `CREATE_PARTICIPANT_REJECTED` are sent to inform clients of the result of their requests.

Kicking Participants: The application allows for the removal of participants from a whiteboard by sending a `KICK_PARTICIPANT` message. This feature is particularly useful for managing user behavior and ensuring a positive collaborative environment.

Server-side Shape and Image Caching: To optimize performance and minimize data transfer, the server maintains caches for shape lists and background images of each whiteboard. This cache helps reduce the overhead of repeatedly sending the same data to clients and ensures that new participants joining a whiteboard receive the current state of the board.

Server Shutdown Handling: To gracefully handle server shutdowns and inform clients of the situation, the `SERVER_SHUTDOWN` message type is used. When the server is about to shut down, it sends this message to all connected clients, allowing them to display an appropriate message to users and close their connections.

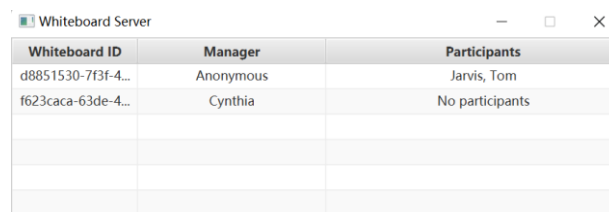
These implementation details highlight some of the core functionalities and techniques employed in the collaborative whiteboard application. By utilizing these strategies, the system is able to provide a seamless and efficient real-time collaboration experience for users.

New innovations

In this section, several new innovations implemented in the collaborative whiteboard application will be discussed. These features enhance the user experience and showcase the flexibility and extensibility of the system.

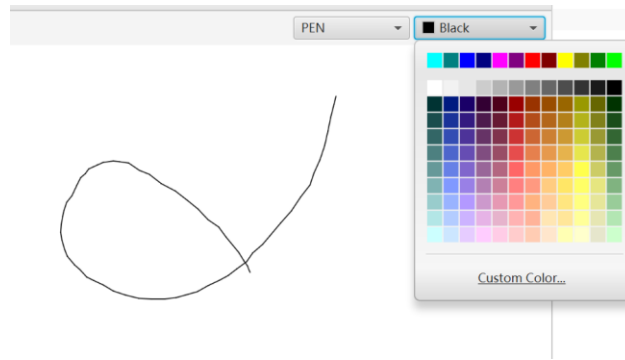
Real-time Monitoring of Whiteboard Information: An additional feature implemented on the server-side is a visualization interface that provides real-time monitoring of whiteboard information. This interface allows administrators to easily track the manager and participants of each whiteboard, ensuring proper oversight and management of the collaborative environment.

Simultaneous Existence of Multiple Shared Whiteboards: The system supports the simultaneous existence of multiple shared whiteboards, allowing for diverse collaboration opportunities. Each shared whiteboard is created by a manager and can have multiple participants. The whiteboards operate independently, without interfering with one another, allowing users to work on different projects concurrently.



Whiteboard ID	Manager	Participants
d8851530-7f3f-4...	Anonymous	Jarvis, Tom
f623caca-63de-4...	Cynthia	No participants

Pen and Custom Color: The pen functionality in the application is implemented by continually adding points and rendering them on the canvas. This innovation goes beyond the initial requirements and offers users a more flexible drawing tool. Furthermore, the application supports a wide range of colors, allowing users to choose from more than 16 predefined colors and even customize their own. This feature enables users to express their creativity and collaborate more effectively.



These new innovations demonstrate the adaptability and potential of the collaborative whiteboard application. By incorporating these additional features, the system provides a more engaging and versatile platform for real-time collaboration and communication.