Whiteboard Protocol

Final Report

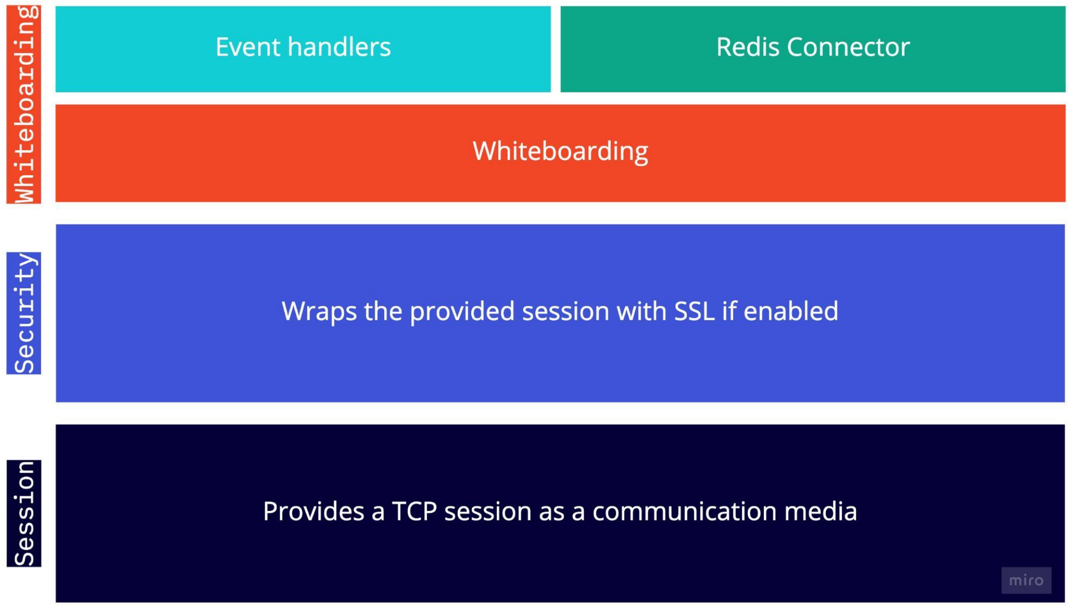
20th March 2022

Group F
Husam Aldeen Alkhafaji (1009987-husamu-aldeen.alkhafaji@aalto.fi)
Sepehr Javid (1011289 - sepehr.javid@aalto.fi)
Bipin Khatiwada (1011292 - bipin.khatiwada@aalto.fi)

Table of Contents

Chapter 1: System architecture

1.1 Server side architecture:


Whiteboarding
Event handlers
Redis Connector
Whiteboarding
Security
Wraps the provided session with SSL if enabled
Session
Provides a TCP session as a communication media
miro

This protocol utilizes a layer based design including three major layers. The bottom layer is responsible for serving and establishing a TCP connection with the clients. The second layer handles the security properties of the protocol which is achieved through the properties of TLS1.3 . The top layer is dedicated to rooms and events handling operations as well as storing the state of each room, user, and event in a redis database.

The Whiteboarding layer consists of three different components. Firstly the base component which is the whiteboarding component and is the core of the entire protocol. This layer provides a function to the session layer including the instructions on how to handle a client. In this handle function all messages received from the client are deserialized into the corresponding event type in the event handler component. This component then executes the event and decides what actions should be carried out based on the event type and ultimately responds to the client with an appropriate message. The event handler also determines whether the server should maintain the connection with the client or the client must be disconnected. In the process of handling an event, the event may require storing data in the redis database for the sake of consistency. Therefore, the event will utilize the redis connector component to achieve this. Ultimately, the event is redistributed, if required , to other room users to maintain the consistency of the board.

There are seven types of events:

1. Room event

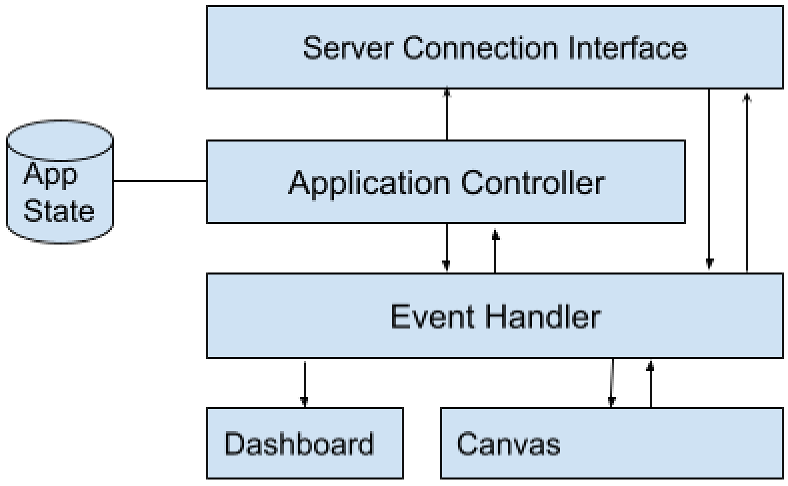
Room event itself is divided into six different actions:

- Create Room
- Request to join
- Leave room
- End room
- Accept join request
- Decline join request

2. Draw
3. Stick Note
4. Image
5. Comment
6. Undo
7. Heartbeat

Event numbers 2 to 6 are considered whiteboard events which have an additional property. Such events have x_coordiante and y_coordinate parameters which are used to position a certain event on the canvas. They are also redistributed to all the users present in the room in order to include the event in their canvas as well. Each whiteboard event holds an action property that determines the required action among those available (Create, Edit, Delete) and the event is later handled based on the action.
Last but not least, the Heartbeat event is a no-op event that is meant to be sent every 30 seconds in order to keep the connection alive.

1.2 Client side architecture:



The “**Canvas**” is the UI element responsible to provide all the whiteboard features and render the contents. “**Dashboard**” provides GUI for room creation and joining.

Similarly, “**Server Connection Interface**” is designed for the user to utilize in order to communicate with the server. This interface holds methods with descriptive names that achieve a specific goal by requesting the server. To request the server, the interface generates a unique id (uuid) and includes it in the server (the uniqueness property is local uniqueness and does not require it to be unique in the server as well). Later the interface creates a promise and returns it to the user so that the user knows if the request was handled successfully or it encountered some errors. The interface also stores the promise locally. Upon receiving a response, the interface finds the promise corresponding to the received uuid and based on the status code, either resolves or rejects the promise and provides the server message.

The “**Event Handler**” is the server that connects to the Canvas providing data needed and calling server whenever any change in canvas is detected. It also listens for the new events from server interface and updates canvas. Before canvas is displayed, it handles all the room creation and joining event similarly using Dashboard.

The “**Application Controller**” is the main service that connects all of these services and controls their creation, joining, listening and destroying. It also manages a central state that stores all the events in local storage.

1.3 Functional requirements

Here, we present functional requirements of this project.

- User can create a new whiteboard session
  a. Each user should be identified by a unique identifier such as UUID
  b. Each user should be able to create a new whiteboard session
  c. Each whiteboard session should have a unique identifier such as UUID
- User can invite the user to the whiteboard session
  a. The invitation link should be the same for all users for a whiteboard session
  b. Each whiteboard session should have different invitation links
- Host can approve or decline the join request
  a. Each user joining the session should be able to add his name
  b. The host should be notified if anyone is waiting to be admitted
  c. The host should be able to approve or decline the request
  d. Participant users should not be able to approve or decline the join request
  e. The approved user should be able to see the current status and contents of the session
  f. The declined user should not be able to see anything
- User can create a new event on the whiteboard (drawing, sticky notes)
  a. User should be able to add freehand drawings on the whiteboard
  b. User should be able to add sticky notes on the whiteboard
  c. Each user should be able to upload an image on the whiteboard
- User can modify an existing event on the whiteboard

a. User should be able to edit any sticky notes on the whiteboard
b. User should be able to erase any free-hand drawing or sticky notes on the whiteboard
c. User can undo the latest event he created on the whiteboard

- Multiple users can create events on the whiteboard at the same time
  a. Multiple users should be able to draw or put sticky notes on the whiteboard
- User can make comments on the image posted on the whiteboard
  a. User should be able to put text as comment on the image
  b. User should be able to attach drawing as comment on the image
- Whiteboard data is consistent or updated for all users
  a. Every user should be able to see the same contents on the whiteboard
- User can save the whiteboard as JPEG or PNG file
  a. Each user can export whiteboard view to an image file (JPEG or PNG format)
- When the host leaves, the session is ended
  a. When the host leaves, all users should be disconnected from the session
  b. Upon disconnection from the session, the user should still be able to export the image.

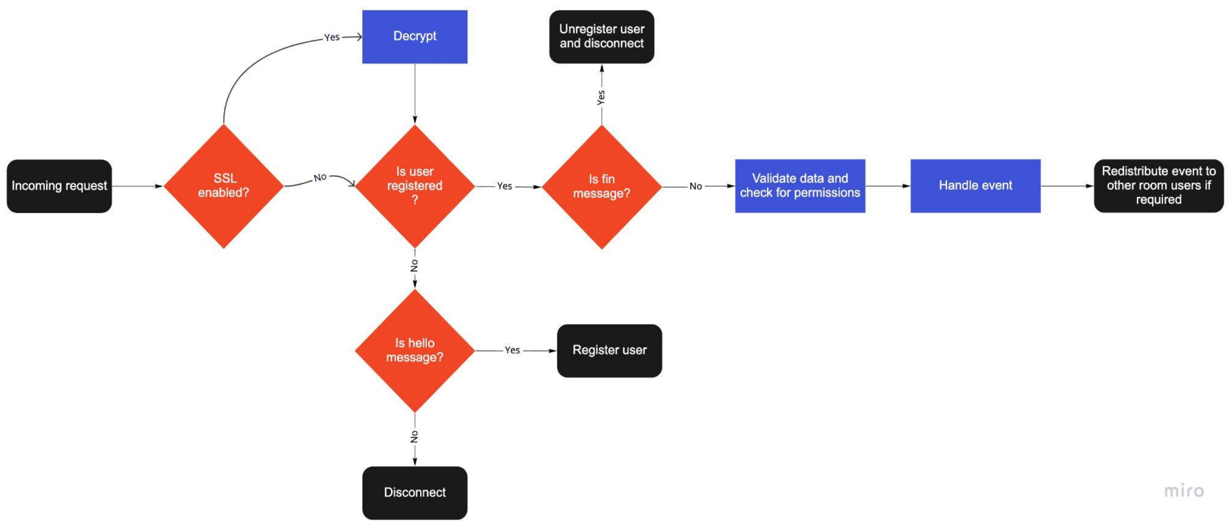
1.4 Non-functional requirements

- A unique identifier is generated for every room
- A unique identifier is generated for every new user.
- Each session should allow at least 25 collaborators.
- New event should be created on the clicked area of the whiteboard
- Users should be able to upload images of at most 7 MB.
- All users should be able to see any change being made within 2 seconds (subject to network connection).
- Export of the whiteboard view should be done client-side.
- Export of the whiteboard to an image should be done within 2 seconds.
- Each session message is fully encrypted if SSL is enabled.

Chapter 2: Design

2.1 State machine

Each message follows the top level state machine to be handled on the server:



2.2 Messages and their formats

2.2.1 From Server to Client

Room Events

General structure is as follows

```
        "status": number,
        "message": string,
        "uuid": string,
        "room_id"*: string,
                  "events"*: list of whiteboard event structure,
                  // * depends on the message.
```

Whiteboard Events

The general response from the server is as follows

```
        "status": number,
        "event": whiteboard event structure,
         // The event is different depending on the action.
```

The whiteboard event structure coming from the server is as follows

```
        "action": number,
        "x_coordinate": number,
                  "y_coordinate": number,
                  "uuid": string,
        "user_id": string,
        "event_id": string,
                // Other attributes depend on the action.
```

2.2.2 From Client to Server

Room Events

General structure is as follows

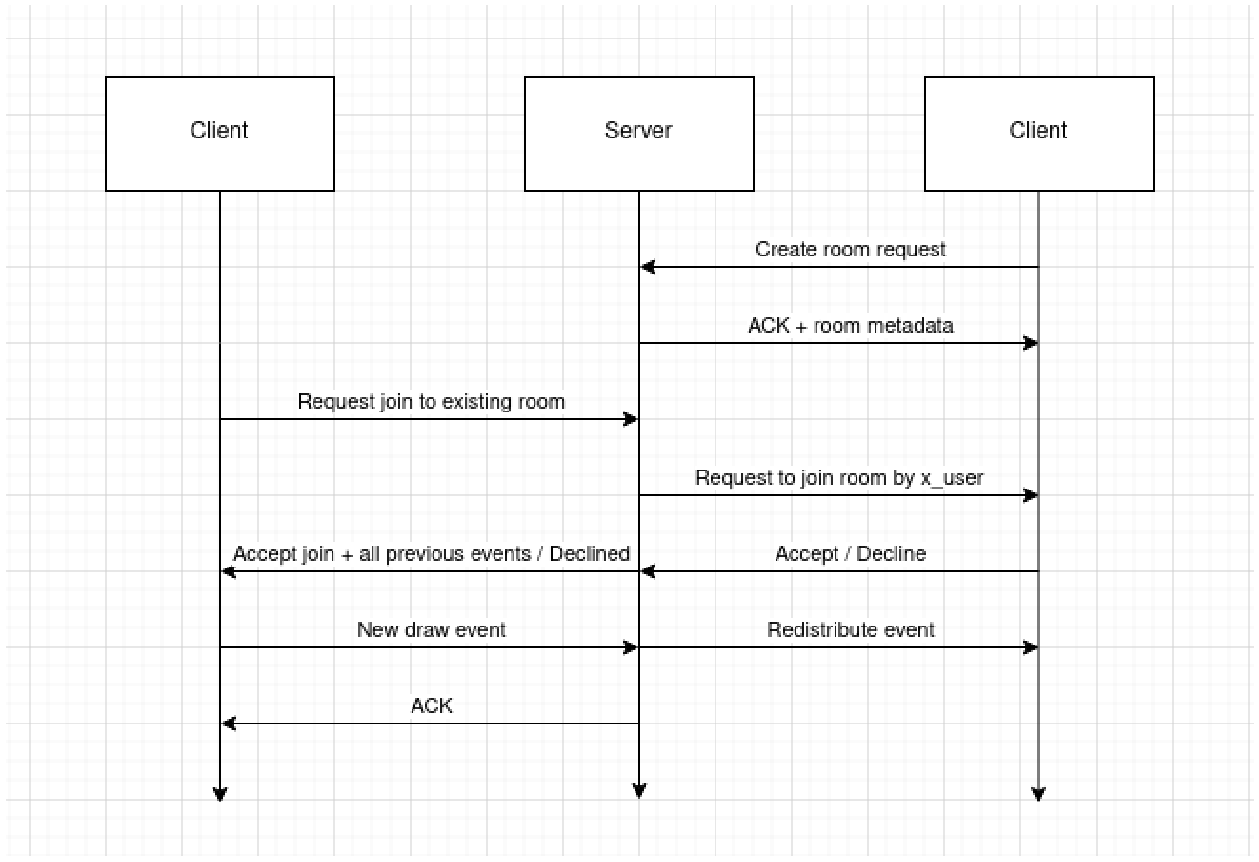
```
        "type": number,
        "room_event_type": number,
        "User_id": string,
        "uuid": string,
        "room_id": string,
                // extra attributes depending on the message.
```

Whiteboard Events

General structure is as follows

```
        "type": number,
        "uuid": string,
        "user_id": string,
        "room_id": string,
        "coordinates": list of lists of numbers,
        "action": number,
        "event_id": string,
                // Extra attributes depending on the message and action.
```

2.3 Message sequence chart



2.4 Comparison

The designed protocol has some advantages and disadvantages compared to the existing whiteboarding protocols:

Advantages:

- Since every action is treated as an event, maintainability of this protocol is notable
- The user id is assumed to be passed from the user application which makes it easier to integrate with other applications

Disadvantages:

- The user id is assumed to be passed from the user application which makes human errors possible when integrating the protocol into an existing application; Causing the protocol to fail
- The protocol lacks appropriate error handling in case of a failure in a connection (but due to the maintainability advantage it is easy to upgrade the protocol to resolve this deficiency)

Chapter 3: Implementation and Evaluation

3.1 Server Implementation

The server follows the async pattern of python instead of utilizing the threading approach. The advantage of this approach is the less performance hit by context switches. In the initial design, our group decided to utilize the threading approach and to assign each user a single thread responsible for that user. However, such a design would have a performance hit once the number of users increases. Alternatively, we chose the async approach so that it only runs the given function in case of a message reception and the entire program still runs on a single thread.

As for the security layer, we incorporated our protocol with an encryption layer which is a wrapper for the session layer that provides an SSL context to the session layer. The information of the SSL context is passed from the upper layer which is elaborated later.

The Whiteboarding layer is the main layer of the implementation. It holds a handler function which handles the client message and performs required actions. It also holds instances of the redis connector component and the encrypted session. The first action required by this layer on startup is to load the config file, validate, and parse the included information. The config file consists of the following information:

```
        "ssl_enabled": true/ false
        "port_number": [port number]
        "interface": "interface name for the server to use e.g. en0"
        "ssl_cert_path": "path to ssl certificate file"
        "Ssl_key_path": "path to ssl private key"
```

Once the config file is loaded, the server starts listening for clients to connect.

In the event component, we have a master event which is the parent of all the events. Each event is handled with the following steps:

1. Checking if the data is valid
2. Checking if the requesting user has permission to perform the request action
3. Perform required action
4. Redistribute the event to others if required

3.2 Client Implementation

We implemented the client interface from scratch using React framework. To generate the canvas, we used HTML3 Canvas and drew the components on top of it. For each events (sticky note, image and drawing), the coordinates were recorded so that the elements could be generated relatively at the same position for all users. This relative spacing is maintained even when multiple users have different canvas size. For drawing, a single stroke is saved as a single event with all of the points forming the arc as the event payload. The experimental setup was done by creating multiple browser instances on the same machine.

For testing**,** we covered the major edge cases while doing the black box testing of our setup. Upon the development period, several bugs and warnings related to React state management, memory leak, and failure to update the canvas during new event were observed and fixed. We covered every possible edge cases that we could think of, mostly on 4 user setup. However, the app is not tested for users accessing the canvas between different networks over the internet.

In our evaluation, the latency is always below 50 milli-seconds. The maximum data one requests support is 10 MB. So the only limitation on the canvas is uploading image above 7 MB. We tested the whiteboard with 23 browser instances as different users in a local network and the average latency for each clients was 70 ms, which is acceptable given the power of the laptop where we run our server.

Chapter 4: Team work

4.1 Teamwork makes the dream work

We followed an agile methodology while developing the application. Tasks were distributed depending on the availability of team members and everyone got to work on all aspects of this project. We met for daily stand ups everyday for around 15 minutes. On top of that, we had longer meetings on weekends where we refined our strategy and try to improve on our design if we encountered flaws.

4.2 Task distribution

Tasks were allocated depending on the availability of each team member, however in general we can easily see that some team members were more interested in working in certain aspects of the application.

Husamu Aldeen Alkhafaji:

- Participated in the design of the protocol
- Programming tasks included backend and frontend session layer (socket communication in session layer, interface for frontend app, whiteboard event structure and data in backend)
- Architecture of frontend
- Documentation
- Report

Sepehr Javid:

- Participated in the design of the protocol
- Programming tasks included backend (encryption layer, room event and stick note event production and handling on server, event types, event redistribution)
- Architecture of backend
- Documentation
- Report

Bipin Khatiwada:

- Participated in the design of the protocol
- Programming tasks included backend and frontend (storage layer with redis, event handling on client, event reception and emission, react GUI)
- Architecture of frontend and react components
- Documentation
- Report
- Testing

Conclusion

We completed all the project requirements including the implementation and demo. The application supports the multi-user shared canvas with consistency of elements in real-time. With features for undo, drag, comment, and different elements this protocol can be implemented in any whiteboard or related use cases. The implementation is done with special consideration for scalability, so new elements and additional features can be added with less effort.