Aalto University
School of Electrical Engineering

ELEC-E7320 - Internet Protocols

# Final report

# Real-Time Interactive Whiteboard App

**Group U**

- Alina Khan 100498130
- Eashin Matubber 1033843
- Manish Kumar 100493449

# Table of Contents

# Chapter 1: System Architecture

## 1.1 Client and Server Architecture

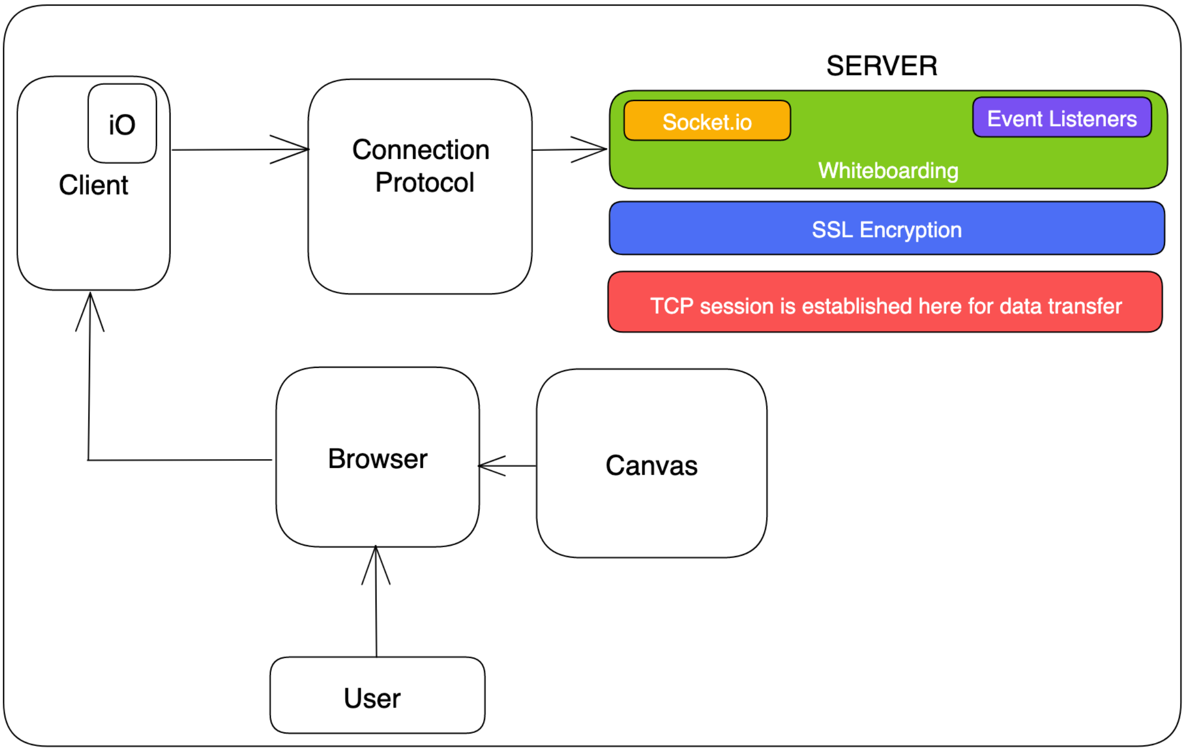


Figure 1: Client & server Architecture

The application is created by setting up an HTTPS server using Node.js and the Express web application framework. We utilise the Socket.io library to handle real-time client and server communication.

This protocol employs a layer-based architecture and has three main layers. The first layer is the base layer, which is in charge of serving clients and connecting to them via TCP. This layer is implemented in the whiteboarding app using the Express framework, which provides a simple and flexible way to create a server and handle HTTP requests. The app creates an instance of the Express app and uses the httpsServer.listen function to listen for incoming HTTPS connections on a specified port. The express.static middleware is also used by the app to serve static files from the public directory.

The security layer is the second layer, and it is in charge of ensuring secure communication between the server and the client. This layer is implemented using the HTTPS protocol and TLSv1.3 encryption. The app creates an HTTPS server with the https module and encrypts the protocol with a locally exported certificate and private key. The server listens on port 443, which is the default port for HTTPS connections. All communication between the client and the server is thus encrypted and secure.

The socket.io library runs in the third and top layers, and the server communicates with the client via WebSockets to send and receive canvas commands.

This layer is implemented in the whiteboarding app using the socket.io library, which allows for real-time bidirectional communication between the server and the client. The io. on function is used by the server-side code to listen for socket connections and handle incoming socket events such as drawing, erasing, and undoing canvas commands.

We prefer a centralised data-sharing model to a distributed model for scalable projects and web applications. The distributed data-sharing model is complex and needs special tools. In contrast, the centralised model usually allows seamless centralised access control, better data and storage processing and, most importantly, easier management. Eventually, a centralised model with multiple replication(s) is the efficient data-sharing model.

## 1.2 Functional Requirements

The functional requirements of this project are presented below:

- Any user can create a whiteboard session
    1. By default, only the session creator has admin/host privileges
    2. There will be only one host identified by a Host ID for each session
    3. Each user will have a distinct identifier, such as a User ID

- The host can approve the requests for joining the session
    1. The host should get a notification if any user is trying to join the session
    2. Admin/Host should be able to accept/decline others to join

- User can create a new event on the whiteboard
    1. User is able to draw freehand on the board
    2. User is able to add sticky notes on the whiteboard
    3. User can upload an image on the whiteboard
    4. User is able to add text as comment on images
    5. User is able to draw on images

- User can change an existing whiteboard event
    1. User can edit any sticky note on the whiteboard
    2. Users can delete sticky notes present on the whiteboard
    3. Users can move the sticky note around the whiteboard
    4. User should be able to erase any freehand drawing/sticky note/image on the whiteboard
    5. User can undo the actions that he/she performed on the whiteboard

- Multiple users will be able to join any session and work simultaneously
    1. The newly joined users are informed of the current view
    2. Multiple users can draw/add sticky note/upload image on the whiteboard

- The data on the whiteboard is consistent and up-to-date for all users
    1. Any action performed by one user syncs in real-time to all other users
    2. The contents of the whiteboard should be consistent for all users

- Users can save the whiteboard as an image
    1. Users can save the current whiteboard view as JPEG or PNG file

- When the host leaves, the session is ended for all other users
    1. Admin/Host is able to end session
    2. Session will be terminated if the admin/host leaves the session
    3. Once the session is terminated, the user should still be able to save/export the whiteboard as an image

## 1.3 Non-Functional Requirements

The non-functional requirements of this project are presented below:

- Every User will have a unique ID
- A low-bandwidth user should have a satisfactory state
- Users should be able to view the whiteboard with low latency
- The session should be secure end-to-end
- Each session should be able to accommodate multiple users (at least 20 users)
- Maximum allowed size to upload an image is 20 MB
- The application should be highly available
- There should be a consistent view of the whiteboard to all the users joined in that session
- There should be no lag in exporting the whiteboard to an image

# Chapter 2: Protocol Design

## 2.1 State Machine

All the event messages are handled by the control centre event handler on the server level and maintain state with any change event coming to the server side event handler. All event messages are forcefully encrypted from the client to the server side, and the final state is emitted and decrypted to the client side.

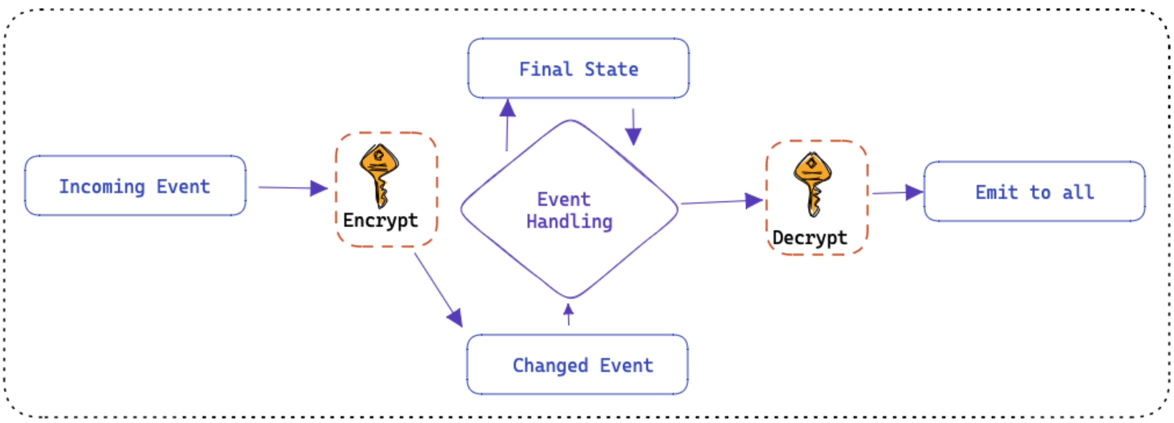


Figure 2: Event state handling

## 2.2 Messages and their formats

### 2.2.1 Server ↔ Client

On the server side, the application maintains all the messages in JSON format and is sent for each event as soon as it is handed to the client in JSON format. Then, between the server and the client, it uses encryption to enforce data security. For example, as below, server-side emitting data in JSON format on the 'stickyNote' event and emitted to all active clients -

```
socket.on('deleteStickyNote', (data) => {
  canvasCommands.push(data);
  users.forEach((user) => {
    user.emit('deleteStickyNote', data);
  });
  host.emit('deleteStickyNote', data);
});
```

### 2.2.2 Client ↔ Server

At client-side messages generated through events are also sent similarly as JSON format, and data types are different from canvas elements. From the client side to the server-side, data communication also uses encryption. For example, in the below snippet, 'mousemove' event data is emitted to the server-side as JSON -

```
canvas.addEventListener('mousemove', (e) => {
  if (mouseDown) {
    let newX = e.clientX;
    let newY = e.clientY;
    if (currentTool === 'pen') {
      ctx.lineTo(newX, newY);
      ctx.stroke();
      io.emit('draw', { type: 'lineTo', x: newX, y: newY });
      x = newX;
      y = newY;
    } else if (currentTool === 'eraser') {
      ctx.clearRect(newX - 5, newY - 25, 30, 40);
      io.emit('erase', { x: newX, y: newY });
    }
  }
});
```

Aalto University
School of Electrical
Engineering

## 2.3 Message Sequence Charts

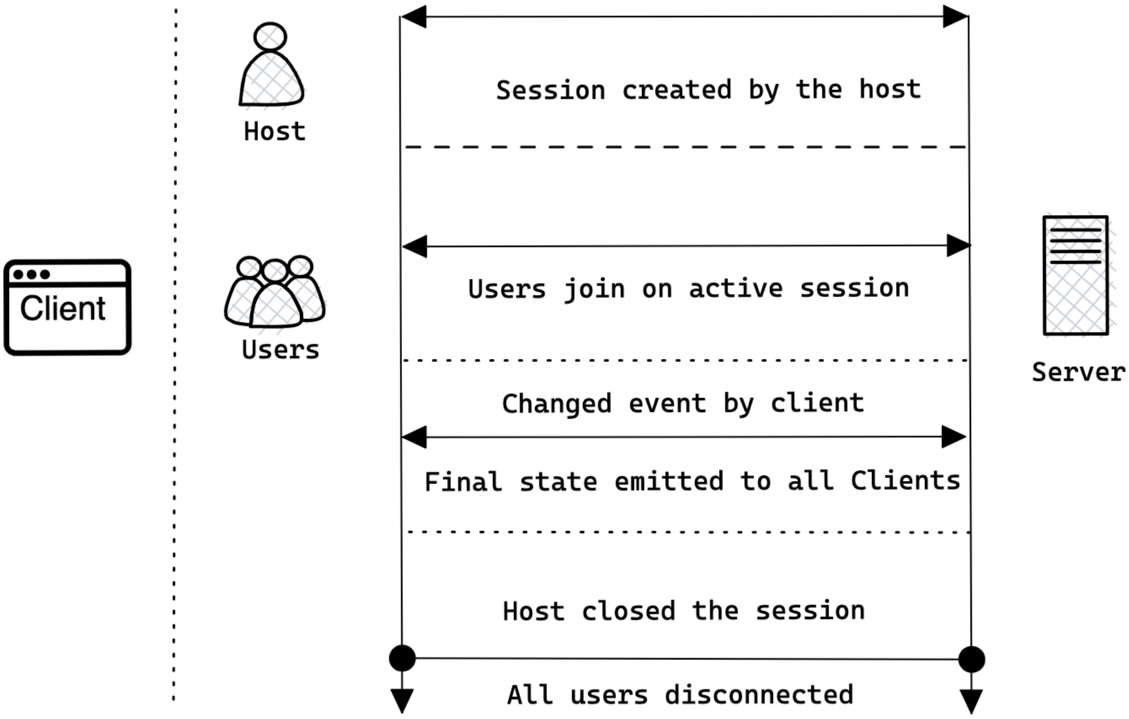


Figure 3: Message sequence chart

## 2.4 Overall Comparison

Using this real-time communication protocol in web applications has both benefits and drawbacks. At this stage, we can assume some of them are compared to similar protocols.

### 2.4.1 Pros

- **Cross-platform compatibility:** It is a consistent protocol for real-time communication, and when all events are saved, tracked and compared with previous events to support real-time functionalities.
- **Easy:** Simple real-time communication in web applications is simple to deploy without requiring additional knowledge of network protocols, as it has handled the most important communication layers and protocols.
- **Reliable:** we have maintained a variety of fallback mechanisms, such as extended polling or WebSockets, to ensure reliable communication over network situations. It also automatically manages reconnection and disconnection events with the detection of role-based privileges.
- **Scalability:** It can be simply deployed in a distributed environment with several servers and supports horizontal scaling.

### 2.4.2 Cons

- **Increased overhead:** Its higher-level abstraction may add additional overhead to the communication channel, which might affect the application's overall performance as it has used many layers and algorithms such as saving and tracking events and comparing them to handle real-time bi-directional communication.

- **Complexity:** It is a sophisticated protocol that comes with a learning curve, especially for different implementations of similar applications.

- **Custom protocol:** On top of other similar protocols, it employs a bespoke protocol that might not work properly with other implementations.

# Chapter 3: Implementation and Evaluation

## 3.1 Server-Side Implementation and Experimental Setup

The code on the server is written in Node.js and runs on socket.io libraries and the express framework. Socket.IO is a JavaScript library that allows two-way communication and is often used to build a client-server web application. In this project, any browser running when the server starts up is the client. Also, socket.io is based on the WebSockets protocol and has an event-driven API that lets messages be sent and received in real-time bi-directionally. When the web server uses the "express" library, the server listens for connections on the PORT variable and serves static files from the public directory. The server then sets up a WebSocket connection to listen for client events, send messages, and keep track of all the canvas commands to broadcast them to the client.

Also, for the security layer, we wrapped the protocol with an encryption layer that wraps the session layer and gives TLSv1.3 encryption to the session layer. First, the server sets up a secure HTTPS connection and locally loads the private key and certificate exported with "mkcert."


Figure 4. Installation and trust of the certificate on the local machine

The certificate, private key, and certificate authority are kept locally in an encrypted vault and can only be viewed after it is decrypted. Also, the machine trusts the private key and the certificate authority locally. After the server verifies the private key and ensures the handshake is established, the session works on port 443. The certificate uses the "SHA-256 with RSA Encryption" signature algorithm, and TLSv1.3, which is the most modern version of SSL, is used by HTTPS for the encryption.

Aalto University
School of Electrical
Engineering

```
Frame 63: 2368 bytes on wire (18944 bits), 2368 bytes captured (18944 bits) on interface lo0, id 0
Null/Loopback
Internet Protocol Version 6, Src: ::1, Dst: ::1
Transmission Control Protocol, Src Port: 443, Dst Port: 56835, Seq: 1, Ack: 518, Len: 2292
Transport Layer Security
 > TLSv1.3 Record Layer: Handshake Protocol: Server Hello
 > TLSv1.3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
 > TLSv1.3 Record Layer: Application Data Protocol: Hypertext Transfer Protocol
 > TLSv1.3 Record Layer: Application Data Protocol: Hypertext Transfer Protocol
 > TLSv1.3 Record Layer: Application Data Protocol: Hypertext Transfer Protocol
 > TLSv1.3 Record Layer: Application Data Protocol: Hypertext Transfer Protocol
```

Figure 5. Breakdown of packet capture on the loopback (lo0) over port 443.

```
0000  1e 00 00 00 60 07 05 00  09 14 06 40 00 00 00 00   ····`··· ···@····
0010  00 00 00 00 00 00 00 00  00 00 00 01 00 00 00 00   ········ ········
0020  00 00 00 00 00 00 00 00  00 00 00 01 01 bb de 03   ········ ········
0030  7a 27 6a 4a a7 d4 99 25  80 18 18 db 09 1c 00 00   z'jJ···% ········
0040  01 01 08 0a e5 45 ac 12  7e 98 e4 66 16 03 03 00   ·····E·· ~··f····
0050  7a 02 00 00 76 03 03 2d  db 67 c3 1f 3f 90 56 51   z···v··– ·g··?·VQ
0060  97 b2 26 25 f0 1b ee d1  5f 35 eb 46 6f 85 28 ef   ··&%···· _5·Fo·(·
0070  f7 09 93 87 75 22 31 20  77 f4 e6 b3 3e 2a 84 5c   ····u"1  w···>*·\
0080  12 98 79 88 26 b5 15 4c  95 68 79 39 ed b0 a6 b8   ··y·&··L ·hy9····
0090  8a 47 20 7a 88 ca bc f1  13 02 00 00 2e 00 2b 00   ·G z···· ····.·+·
00a0  02 03 04 00 33 00 24 00  1d 00 20 08 d4 c2 eb 7d   ····3·$· ·· ····}
00b0  a4 54 2c 86 1a 70 25 87  1f 49 d3 1d f8 71 df b6   ·T,··p%· ·I···q··
00c0  28 20 dd 09 08 a5 59 83  e3 53 2f 14 03 03 00 01   ( ····Y· ·S/·····
00d0  01 17 03 03 00 26 80 84  04 4b d9 81 6a 85 01 fb   ·····&·· ·K··j···
00e0  63 f4 1c 32 01 f1 5f b7  b2 98 cc ab 09 e3 c2 4d   c··2··_· ·······M
00f0  bb 22 9f 55 74 87 1d 5c  55 7d fd 26 17 03 03 06   ·"·Ut··\ U}·&····
```

Figure 6. Snapshot of server ack shows encrypted data over port 443 running on lo0

## 3.2 Client-Side Implementation and Experimental Setup

The client-side implementation is done from scratch using javascript. To generate a working interface and the canvas for freehand drawing, HTML and CSS have been used. The client begins by referencing the canvas element in HTML and settings its dimensions to match the browser window and then picks up localhost:443 to connect to the server. Several variables have been declared that aid in freehand drawing on the canvas. The code tracks and listens for mouse events and responds accordingly; likewise, the code updates the movement of the image across the canvas once it is moved. The event listeners for mouse events on the canvas element, including mousedown, mousemove, and mouseup events, have also been set up along with event listeners such as undo to meet the functionalities of the project.

The experimental setup was done by creating various instances of localhost running on multiple browsers and using a bridged connection to achieve cross-platform testing. During the final evaluation of network latency, the results were very surprising. The average latency was always less than 45 milliseconds. Also, when packets were captured, the average SYN, ACK requests always came under 35 milliseconds.

The entire application was tested under multiple varying networking conditions and on numerous browsers along with testing on cross platforms; overall, the latency was between 45-60 milliseconds which is excellent considering the abstraction leaks.

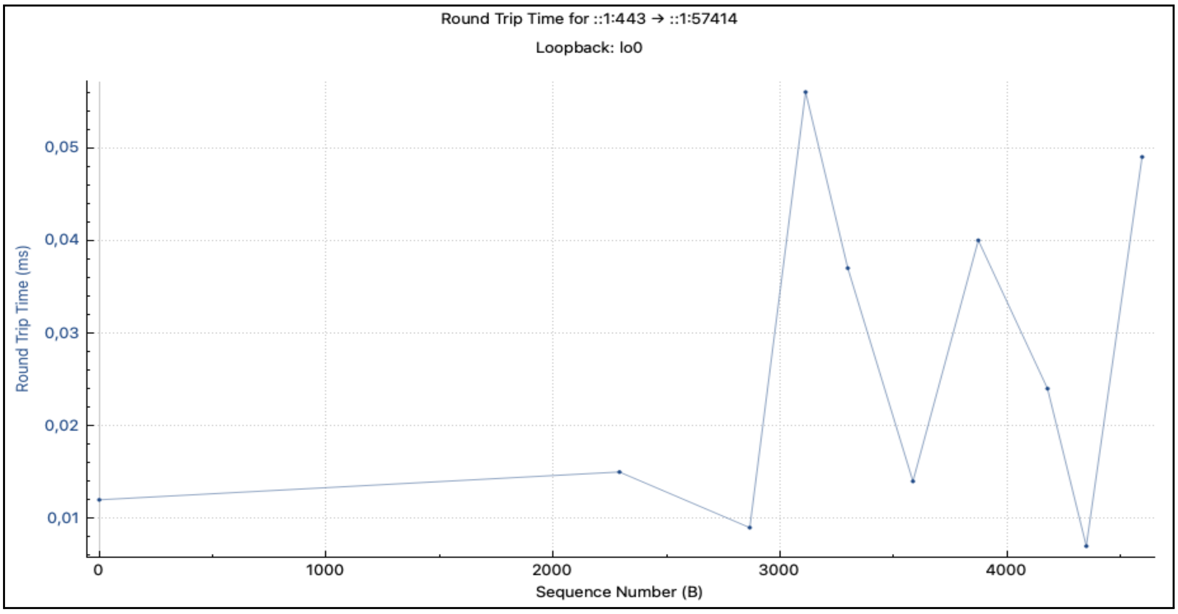


Figure 7. TCP Stream Graph with the parameter Round Trip Time (ms). Tested on lo0 on multiple server refresh(s) with minimal network abstraction leaks.

| Time | Protocol | Info |
|---|---|---|
| 0.032 | TCP | 57415 → 443 [SYN] Seq=0 Win=65535 L |
| 0.032 | TCP | 443 → 57415 [SYN, ACK] Seq=0 Ack=1 |

Figure 7. Average “SYN, ACK” time over port 443. Tested on lo0 on multiple server refresh(s). This also includes the verification and handshake process of the SSL certificate.

# Chapter 4: Team Work

## 4.1 Group-U as a team

To achieve the best of this project, it needed the perspective of multiple brains. Therefore, teamwork is one of the most critical elements to achieving project goals. To achieve this, we followed a proficient and agile development methodology in the process of developing the project. Even before the first checkpoint, we met on-site thrice weekly to discuss the goals and expectations.

After the first checkpoint, we met online regularly to track the progress, communicate and adhere to internal checkpoints. Additionally, Tasks were divided evenly among us so that each of us had a chance to work on every part of the project. Lastly, we did a mock demonstration online before the final one to ensure all the functions worked as expected.

Additionally, we met online and on-site after lectures several times for knowledge transfer sessions about new concepts and terminology we learnt between project meetings.

## 4.2 Task Distribution

The project needed the implementation of multiple functions, and numerous new concepts were introduced during the development phase. Therefore, the workload and tasks were divided equally within the group to keep up with the pace. Moreover, several functionalities, such as encryption and security, had multiple approaches to achieving it. Therefore, the division of tasks also included researching the approach and the best implementation method. In the end, each of us was aware of the precise implementations and functionalities built.

**Eashin Matubber**

- Programming tasks, including developing the backend websocket API and frontend architecture.
- Structuring, designing and implementing the whiteboard functionalities.
- Internal documentation and research.
- Providing knowledge-transfer sessions about new topics and concepts.
- Report(s) and presentations(s).

**Alina Khan**

- Programming tasks, including implementing numerous events, functionalities and frontend architecture.
- Consolidation of individual functionalities in the project to make it operate together.
- Providing knowledge-transfer sessions about new topics and concepts.
- Internal documentation and research.
- Report(s) and presentations(s).

**Manish Kumar**

- Programming tasks, including implementation of functionalities, message/data encryption, network measurement and the implementation of local SSL certificate.
- Contributing to the protocol's design and dockerizing the project out of curiosity.
- Internal documentation and research.
- Providing knowledge-transfer sessions about new topics and concepts.
- Report(s) and presentations(s).

# Conclusion

We achieved most of the project requirements, which were the fundamental blocks of the whiteboard application, such as erasing, freehand drawing, sticky notes, adding images and saving the end canvas as an image. Furthermore, the application has the capability to support multiple users providing a real-time shared consistent view. Additionally, the entire application is SSL encrypted, running on 443 and is dockerized using node:16 base image to simplify it.