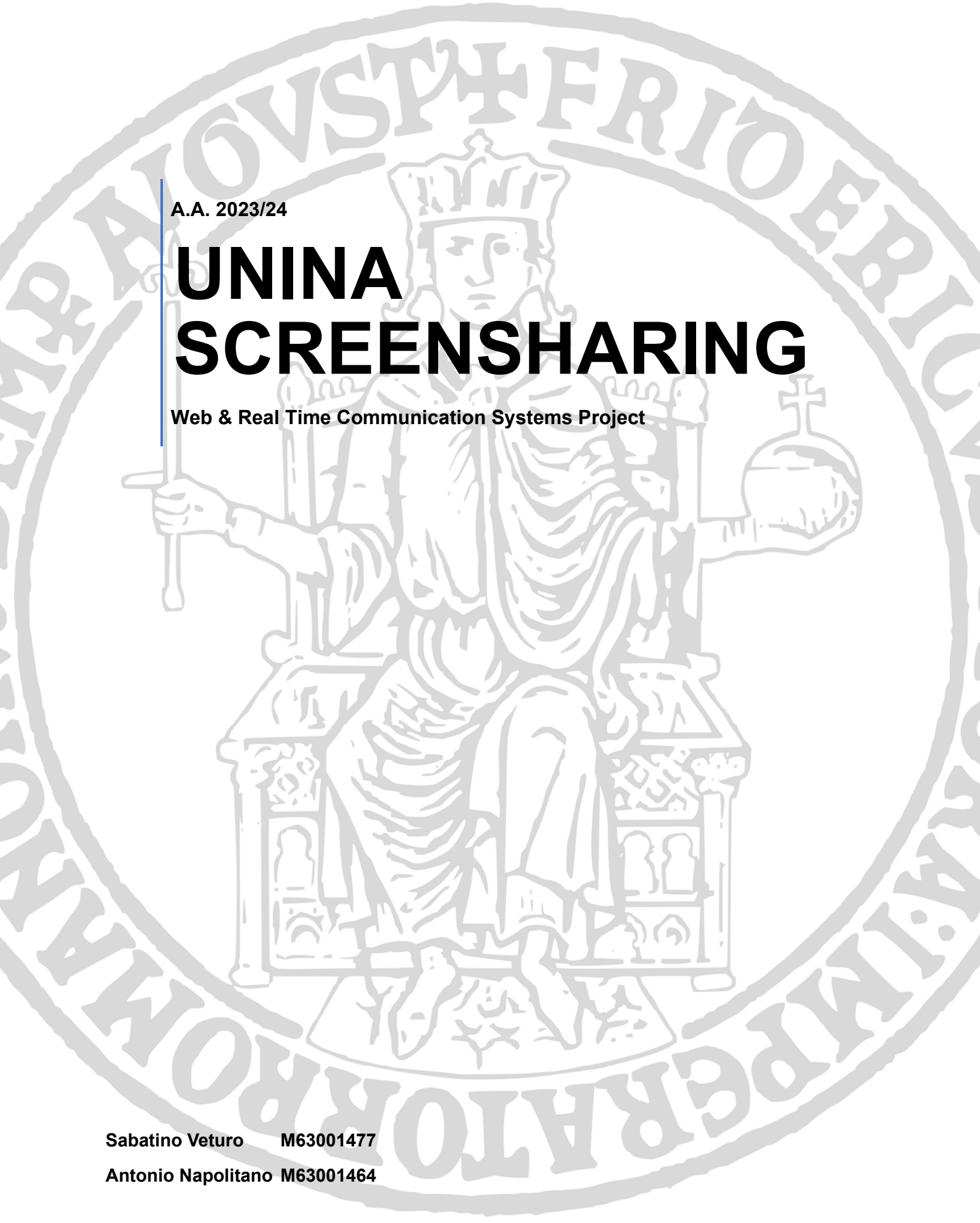


A.A. 2023/24

UNINA SCREENSHARING

Web & Real Time Communication Systems Project

Sabatino Veturo **M63001477**

Antonio Napolitano **M63001464**

Sommario

1. INTRODUZIONE

Il progetto nasce dall'esigenza di voler creare un sistema in grado di permettere la condivisione di più schermi in maniera simultanea, in quanto, sempre più spesso capita di presentare progetti o tenere riunioni da remoto, in collaborazione con colleghi, ed in presenza di più schermi o di più fonti di stream video da dover mostrare.

Tutte queste condizioni rendono complesso l'uso di applicazioni come Teams o Google Meet che richiedono tempi di switch della fonte non ottimizzati per il tipo di comunicazione che si vuole ottenere.

Il progetto Unina ScreenSharing permette la condivisione simultanea di audio e video (fino a sei schermi) oltre all'utilizzo di più *rooms* a cui collegarsi emulando l'organizzazione tipica di uffici ed università.

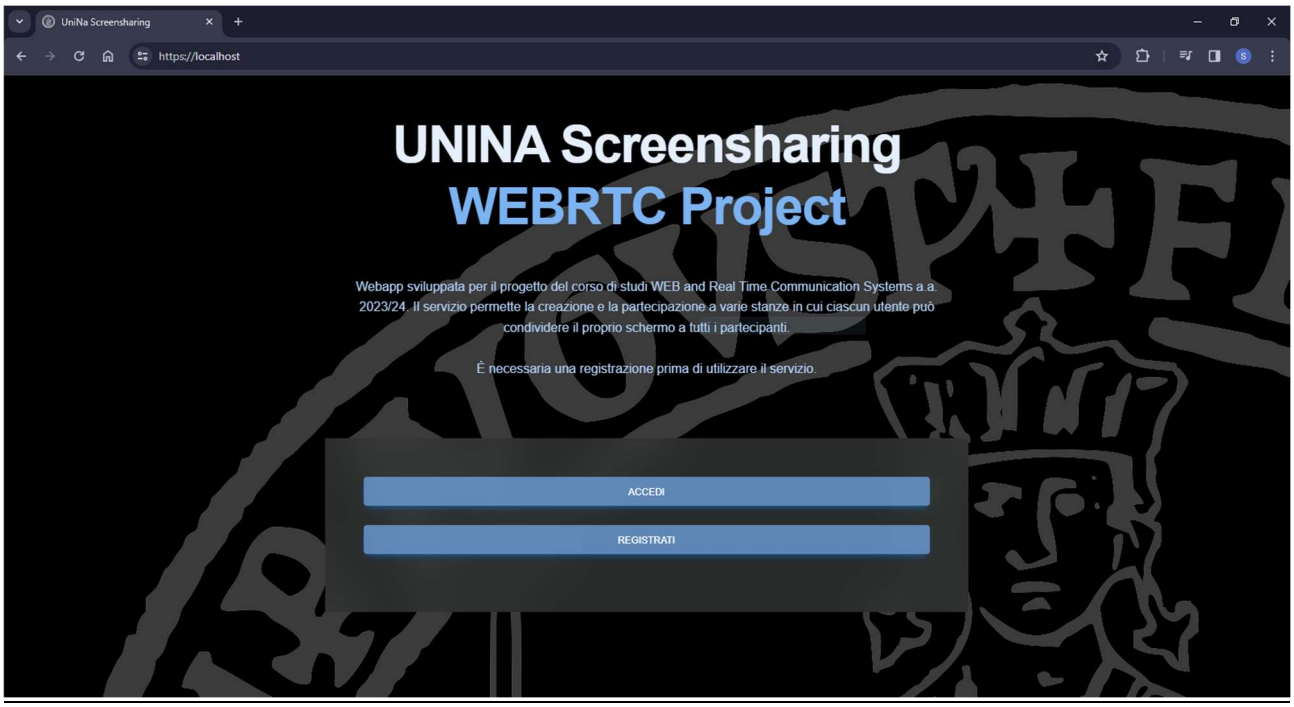


Figura 1 Home page del sito

1.1 Funzionamento applicazione

Unina ScreenSharing si presenta con una home page (figura1) che permette di accedere alle schermate di login e di registrazione al servizio.

Una volta inserite le credenziali, l'utente verrà reindirizzato alla pagina dedicata alle stanze interattive con una panoramica sulle modalità di funzionamento e una serie di bottoni per accedervi.

Figura 2 Creazione rooms

L'interfaccia di selezione delle modalità presenta una serie di possibilità:

- La prima modalità permette di partecipare ad una stanza di default ed è pensata per un accesso veloce.
- La seconda modalità permette all'utente di poter partecipare ad una stanza esistente e di inserire uno username che comparirà sopra il proprio video stream sui browser di tutti i partecipanti.
- L'ultima modalità è in realtà un'opzione che permette di creare una nuova stanza che sarà subito disponibile nella tendina "*Scegli stanza*".

Una volta scelta la modalità di funzionamento si accede al cuore dell'applicazione composto da un'interfaccia contenente sei *box* dedicate agli schermi dei partecipanti alla *room*.

Saranno a disposizione dell'utente una serie di bottoni che permettono l'interruzione dello stream dell'audio, l'interruzione del video e la scelta della finestra da condividere per quanto riguarda la propria box.

Per quanto riguarda il video degli altri partecipanti, sarà possibile osservare statistiche sullo streaming (risoluzione e *bit rate*) oltre a poter mettere in pausa il video o renderlo a schermo intero per una visione più dettagliata.

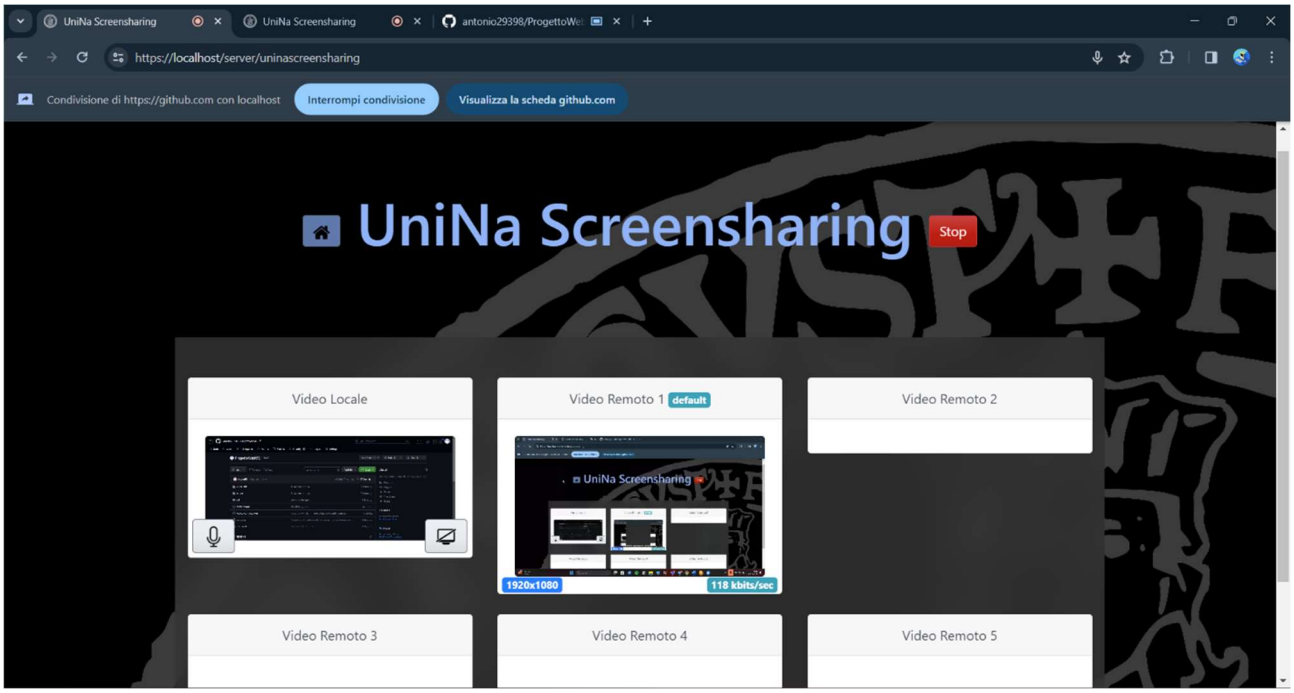


Figura 3 Interfaccia di condivisione schermi

1.2 Strumenti utilizzati

Il progetto è stato realizzato attraverso l'utilizzo di framework noti e linguaggi di programmazione specializzati per il web developing; in particolare è stato applicato il principio dello *"javascript everywhere"* per quanto riguarda lo sviluppo di client e server.

Di seguito si riportano tutti i linguaggi e *framework* utilizzati:

- React
- NodeJs
- Docker
- Linguaggi di scripting
- Sql
- HTML/CSS
- Javascript
- JQuery
- Janus

1.3 Modalità di funzionamento

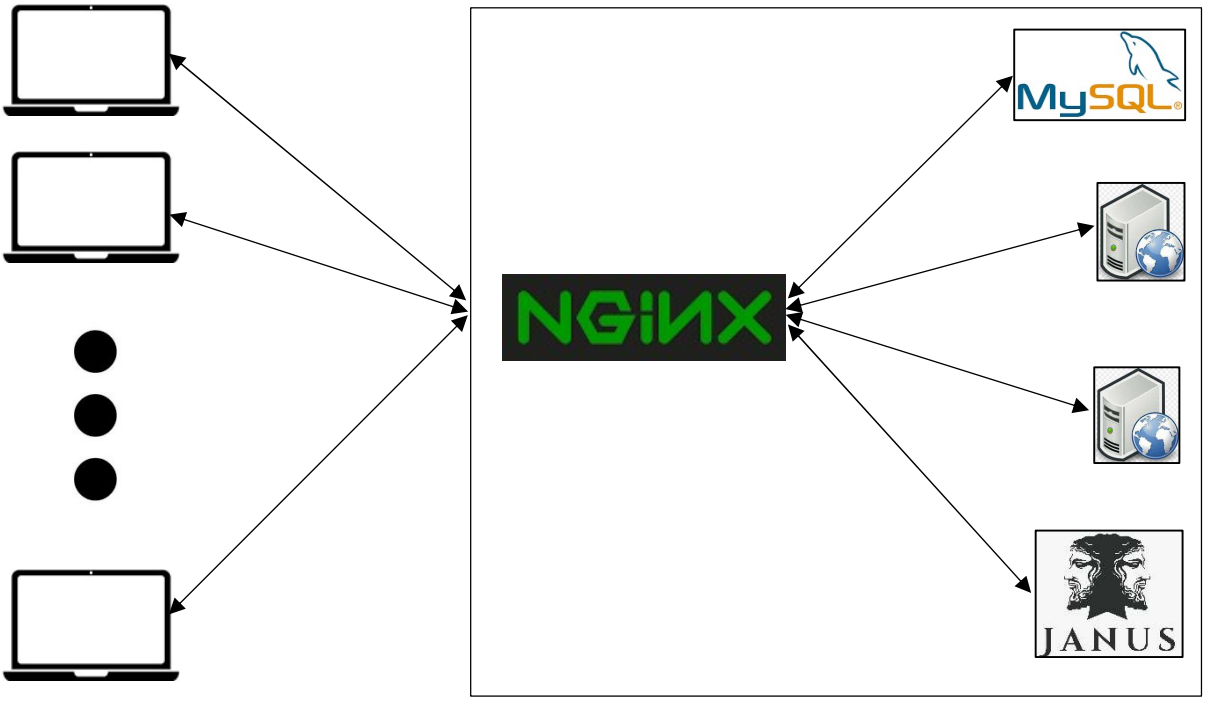


Figura 4 Schema di alto livello

Per la realizzazione dell'applicazione Unina ScreenSharing sono stati utilizzati diversi nodi che assolvono vari compiti fra cui Nginx, che si occupa di smistare le richieste, e Janus che si occupa di gestire la segnalazione.

Il server Janus fornisce di default una demo con un'interfaccia ed alcune funzionalità lato utente molto semplici; lo sviluppo dell'applicazione è partito proprio con la comprensione e ampliamento di tale demo.

In particolare sono state inserite le seguenti modalità *from scratch:*

- Creazione della stanza.
- Collegamento di default.
- Scelta della stanza a cui collegarsi.
-

Nei prossimi capitoli verrà approfondito nello specifico il contenuto di ciascun nodo, i framework, le librerie utilizzate e l'interazione fra ciascuno di essi.

Di seguito si riportano, a titolo esplicativo, il diagramma dei casi d'uso ed un diagramma di flusso relativo all'operazione di accesso alla piattaforma.

Le operazioni descritte nei seguenti grafici verranno approfondite nello specifico nei capitoli successivi.

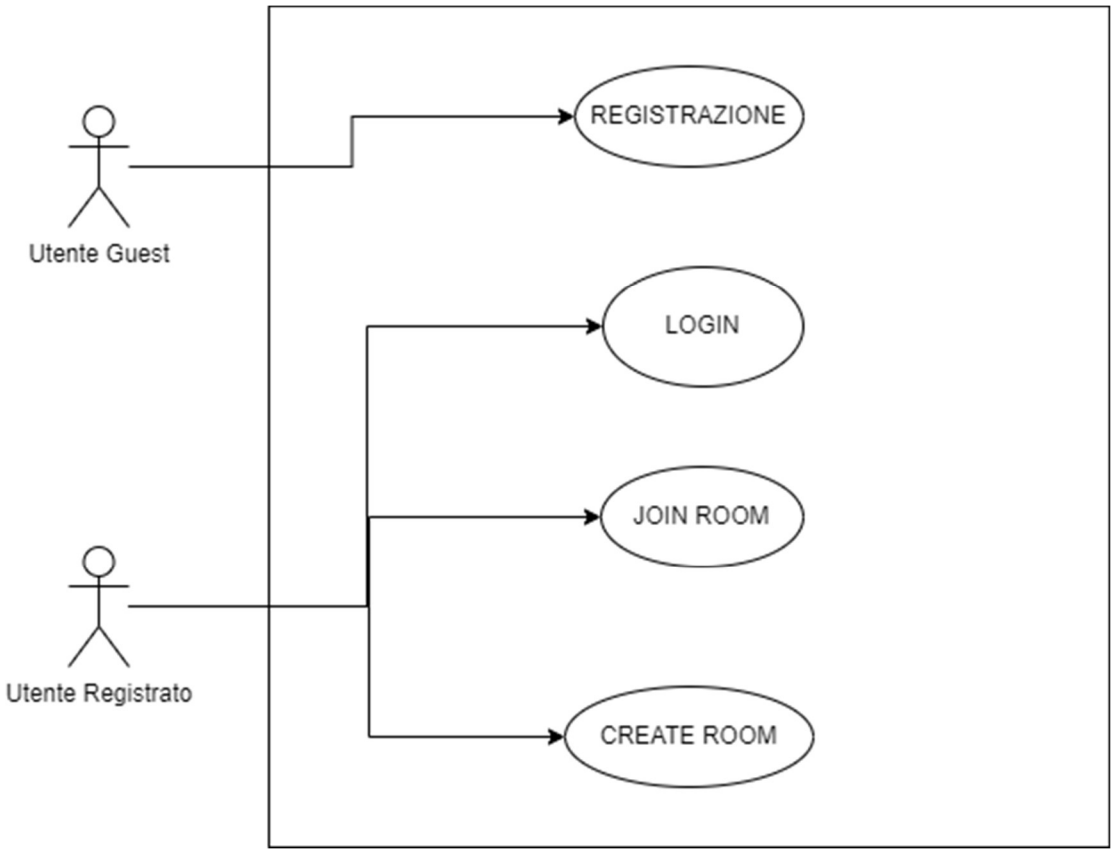


Figura 5 Diagramma dei casi d'uso

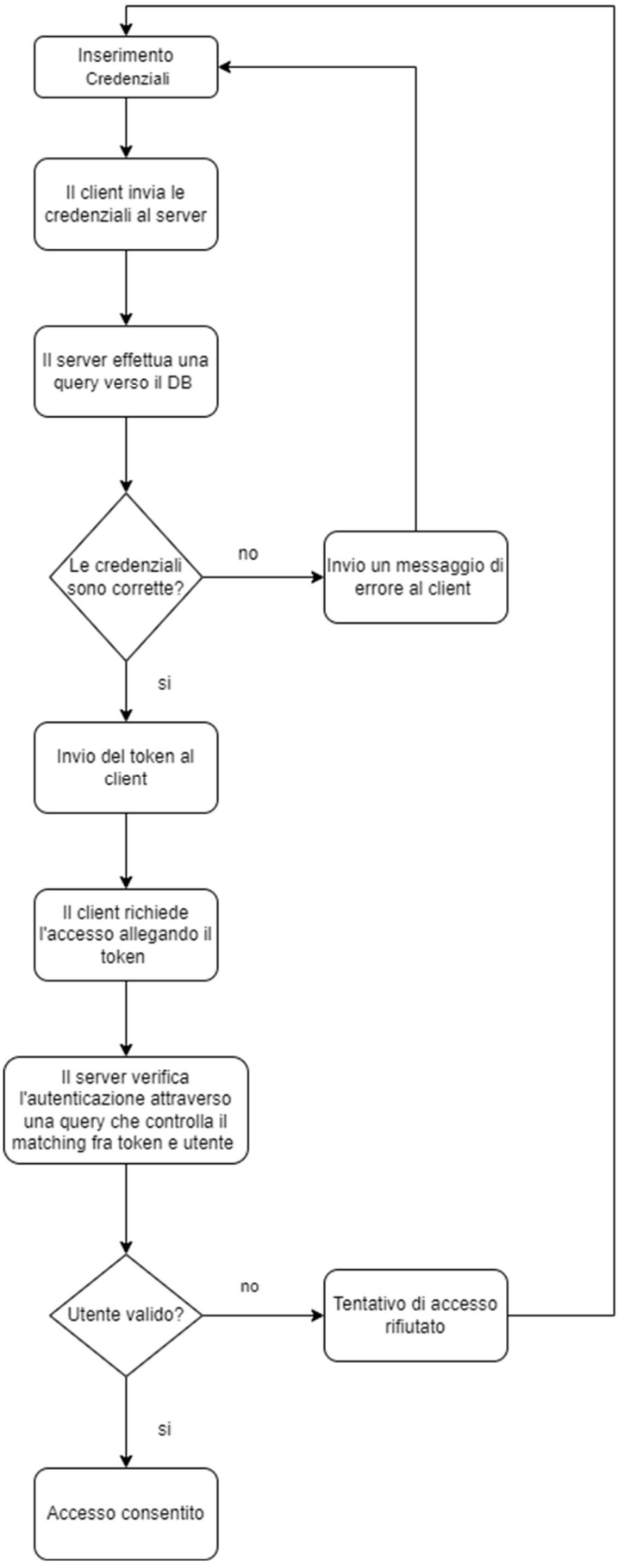


Figura 6 Flowchart di accesso alla piattaforma

2. DOCKER

Si è scelto di utilizzare il software Docker al fine di realizzare un'applicazione che permettesse di eseguire ciascun nodo in un ambiente indipendente senza l'utilizzo di machine virtuali.

L'applicazione è stata resa modulare attraverso l'uso dei seguenti container:

- nginx: Container che ospita il software nginx dedicato al coordinamento delle richieste in maniera sicura.
- Server: Container che ospita i processi operativi del front-end e del back-end.
- Janus: Container che ospita il server di segnalazione con il quale vengono negoziate le informazioni relative al collegamento alle stanze.
- MySQL: Container che ospita il Database MySQL.

La creazione dei container è stata realizzata attraverso l'uso di un *Docker compose* di cui si riporta il codice.

```yaml
version: '3'

services:

  nginx:
    image: "antonio29398/nginxwebrtc:latest"
    container_name : 'nginxCompose'
    ports:
    - "443:443"
    volumes:
      - ./ssl:/etc/nginx/ssl

    restart: always
    networks:
      network_compose:
        ipv4_address: 194.20.1.2

  mysql:
    image: "mysql:latest"
    container_name : 'mysqlCompose'

    environment:
    # Password for root access
      MYSQL_ROOT_PASSWORD: 'password'

    volumes:
      - ./setup.sql:/docker-entrypoint-initdb.d/setup.sql

    restart: always
    networks:
      network_compose:
        ipv4_address: 194.20.1.3
```

```yaml
  janus:
    image: "antonio29398/janus-gateway:v3"

    container_name: 'janusCompose'

    command:  bash -c "/opt/janus/bin/janus "

    ports:
    - "8088:8088"

    restart: always
    networks:
      network_compose:
        ipv4_address: 194.20.1.4


  server:
    image: "antonio29398/ubuntuserver:v5"
    container_name : 'serverCompose'


    volumes:
    - ./:/home/ProgettoWebRTC

    command: bash -c " ./home/ProgettoWebRTC/startup.sh "


    restart: on-failure
    networks:
      network_compose:
        ipv4_address: 194.20.1.5

networks:
  network_compose:
    ipam:
      config:
        - subnet: 194.20.1.1/24
```

Figura 7 Docker compose file

Le *Docker images* utilizzate nel compose sono un misto fra immagini[1] fornite direttamente dalle case produttrici dei software ed immagini personalizzate create ad hoc.

[1] Tutte le immagini utilizzate sono disponibili su Docker hub all'indirizzo - https://hub.docker.com/u/antonio29398.

3. SERVER

Per quanto riguarda il container Server, si è scelto di caricare su un solo container sia il front-end che il back-end in modo tale da semplificare le interazioni fra i due nodi.

I due nodi vengono eseguiti all'interno del container come due processi attraverso l'uso di uno script bash che permette di eseguire in background i processi; in figura 8 possiamo osservare l'esecuzione dei due "server" (è necessario installare il *node package manager*).

```
cd /home/ProgettoWebRTC/client-side
npm install
npm start &
cd /home/ProgettoWebRTC/server
npm install
npm start
```

Figura 8 Startup.sh

3.1 Front-end

Il client side è stato realizzato utilizzando la libreria open source React.

Per la costruzione dei *function component* è stata utilizzata la libreria Bootstrap per la creazione di stili ed applicazioni per il web insieme ad un kit di template *mdb-ui-kit*[2].

Il browsing sul sito è stato realizzato attraverso la libreria *react-router-dom* che ha permesso la navigazione sulle varie *function component* unita ad una serie di chiamate al back-end che hanno permesso di gestire in maniera semplice le richieste al database ed a risorse sensibili.

La procedura di autenticazione al sito comincia con la registrazione che richiede l'inserimento di:

- Nome
- Cognome
- Username
- E-mail
- Password

Sia username che e-mail rappresentano campi unici che, in caso di registrazioni già presenti, rappresentano due condizioni per le quali la registrazione non viene completata con successo.

Per effettuare il login sono richiesti solamente username e password.

[2] Il kit è disponibile all'indirizzo - https://mdbootstrap.com/

3.2 Back-end

Per quanto riguarda il *processo* di back-end, si è scelta un'implementazione in JavaScript, come già menzionato in precedenza, attraverso l'uso di NodeJS (in particolare del framework Express) per gestire l'*hosting* di un server *http*.

Per gestire tutte le richieste al server sono stati mappati solamente i metodi utilizzati per le risorse che vengono accedute tramite la navigazione dall'interfaccia di React mentre per gestire le interazioni con il database sono stati utilizzati i classici metodi che permettono di interfacciarsi con database MySQL.

L'accesso alle risorse viene consentito solo ad alcuni domini attraverso l'uso del middleware *cors (Cross Origin Resource Sharing)* che permette di implementare un sistema di sicurezza che consente di autorizzare le richieste in funzione del richiedente.

```
// Utilizzo il package cors per consentire l'invio di risorse dal
// sito da cui è previsto l'invio del login
var corsOptions = { origin: ['http://localhost:3000', 'https://localhost'], credentials: true}
app.use(cors(corsOptions));
```

Figura 9 Cors

Il resto del back-end ospita tutte le interazioni con il Database.

In figura 10 viene riportata, a titolo di esempio, la query di registrazione.

```
const user = req.body.username;
const pass = req.body.password;
const nome = req.body.nome;
const cognome = req.body.cognome;
const email = req.body.email;

// Collegamento al database
database.connect();

// Chiedo al database di inserire l'utente. Se già esiste l'operazione non viene effettuata.
const insert = 'INSERT ignore INTO utenti.credenziali (username, password, nome, cognome, email) VALUES (?, ?, ?, ?, ?) '

database.query(insert, [user, pass, nome, cognome, email])
    .then((results) => {
        console.log('Risultati della query:', results);

        // In base all'esito della query invio il messaggio corretto al client
        if (results.affectedRows > 0) {

            const token = uuidv4();

            const updatetoken = 'UPDATE utenti.credenziali SET token = ? WHERE username = ?;'
            database.query(updatetoken, [token, user])

            res.send({ message: "utente creato" });

        } else {
            res.send({ message: "utente già esistente" });
        }

    })
    .catch((err) => {
        console.error("Errore durante l'esecuzione della query:", err);
    })
```

Figura 10 Query di registrazione

3.3 Autenticazione

Per quanto riguarda l'autenticazione, è stata dedicata particolare attenzione alla persistenza del login. Si è scelto di utilizzare una combinazione fra cookie di sessione ed un token *uuidb4* che viene creato a tempo di registrazione e scambiato tra client e server come descritto in figura 11.

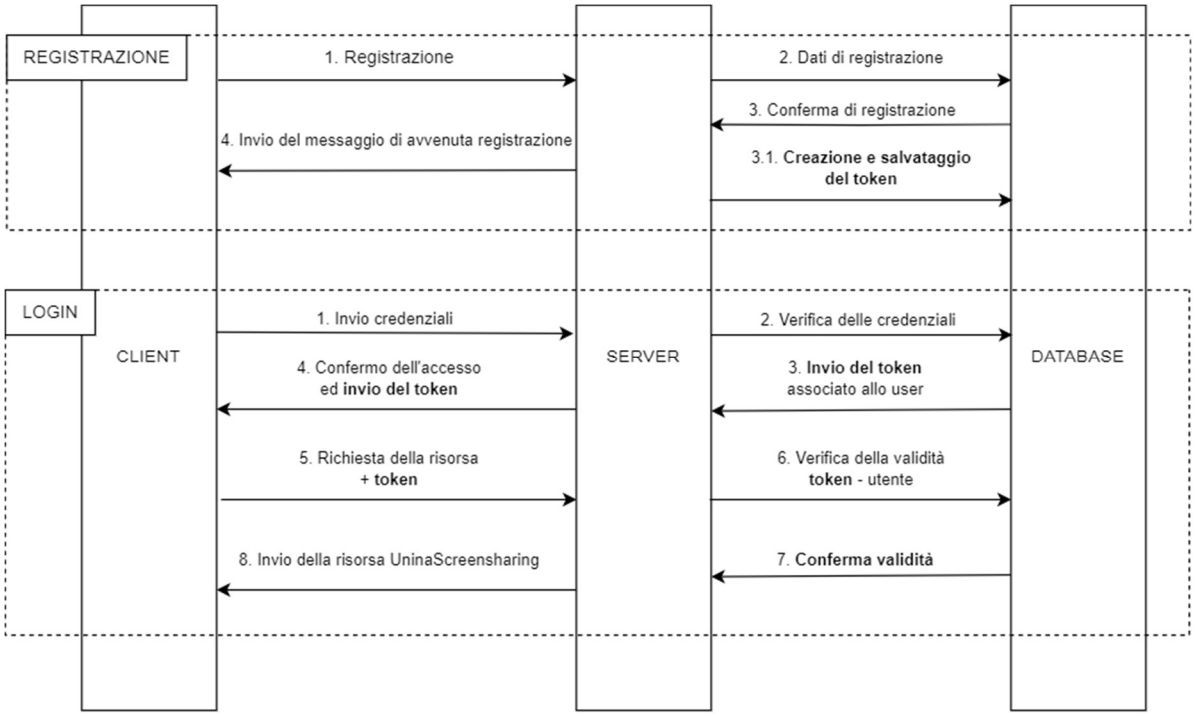


Figura 11 Autenticazione: Gestione del token

Dalla figura 8 è chiaro che questo meccanismo si configura in due fasi:

1. **Sessione**
   La prima fase consiste nella creazione e gestione della *sessione* che viene effettuata attraverso l'uso del middleware fornito dalla libreria *express-session* con cui è possibile salvare, lato server, informazioni su uno specifico collegamento effettuato via client.
   Ai successivi collegamenti di un client, secondo regole specifiche, esso verrà riconosciuto dal server attraverso l'utilizzo di un cookie con cui siamo in grado di garantire un arco di tempo durante il quale un client non ha bisogno di ripetere l'accesso.

```
app.use(session({
    secret: 'pippozzo',
    saveUninitialized: true,
    resave: false,
    rolling: true,
    cookie: {
        name: 'sessionCookie',
        maxAge: 3600000, // Expires in 1h
        httpOnly: false,
        secure: false, // Imposta a true se stai usando HTTPS
    }
}))
```

Figura 12 Configurazione di Express session

La figura 12 mostra le impostazioni utilizzate per la configurazione del middleware che setta la durata della validità del cookie ad un'ora dall'accesso autenticato; un tentativo di accesso diretto alla risorsa che permette l'utilizzo dell'applicazione, senza aver effettuato l'autenticazione, viene rifiutato con una schermata di errore.

2. Token

Il meccanismo dei cookie è molto utile per riconoscere l'utente che si sta collegando sempre dallo stesso client. Durante lo sviluppo si è però palesata una problematica relativa al fatto che più accessi fatti dallo stesso client, con due account diversi, potevano essere in qualche modo manipolati in quanto sarebbe possibile farsi riconoscere dal server utilizzando un cookie, già salvato nella sessione del client, relativo ad un account diverso da quello che sta effettuando l'accesso.

Per risolvere questo problema si è pensato di aggiungere un secondo livello di autenticazione attraverso l'uso di un token che viene creato a tempo di registrazione al servizio.

Il token viene allegato come variabile di sessione nel momento in cui viene richiesta la risorsa dell'applicazione in modo da permettere al server di fare un doppio controllo validando il client che richiede la risorsa con il cookie, che deve risultare valido, e l'utente che si sta autenticando tramite il matching fra token inviato e token dell'utente presente nel database.

```
const verificaAutenticazione = (req, res, next) => {

    console.log("ID della sessione",req.sessionID);
    console.log("Il token è:",req.session.token);
    console.log("Quando entro la sessione è:",req.session);

    if (req.session.token === undefined) {
        // Prosegui con la richiesta se l'utente è autenticato
        res.status(401).sendFile(__dirname + '/resources/no-auth.html');
        console.log("Non autenticato. Token: ", req.session)
    } else {

        const getToken = 'SELECT token FROM credenziali where username = ?'
        let username = req.session.user;

        database.query(getToken, [username])

            // res.send viene fatto nel then perché è l'ultima parte di codice ad essere
            // eseguita, inoltre, siamo sicuri che questa query vada a buon fine
            // (passa già un check su login valido!)
            .then((results) => {

                if (results.length > 0 && results[0].token === req.session.token) {
                    console.log("Autenticato. Token: ", req.session.token);
                    next();
                }
                else {
                    // Rispondi con un errore se non esiste l'utente
                    res.status(401).send('Il token non corrisponde all&apos;utente');
                }

            })
            .catch((err) => {
                console.error("Errore della query token: ", err);
                res.status(500).json({ error: "Errore interno del server",
                message: "Errore durante la verifica dell'autenticazione" });
            })

        console.log("Autenticato. Token: ", req.session.token)

    }
};
```

Figura 13 Verifica autenticazione con Token

Lato back-end è stata importata anche una serie di file template JavaScript, presenti nella documentazione di Janus, che permettono l'interazione tra l'interfaccia grafica, presente nel file salvato sempre lato server chiamato "uninascreensharing.html", e le vere e proprie funzioni di back-end che interagiscono con il server Janus[3].

[3] Per approfondimenti su file citati si rimanda al capitolo 6 relativo a Janus.

4. NGINX

Per come è stato strutturato il progetto e per quanto richiesto dalla documentazione di Janus, tutte le richieste e le interazioni tra i vari container devono supportare il protocollo *http.*

Com'è noto, una delle condizioni necessarie affinché un'applicazione web possa essere considerata sicura è l'implementazione del protocollo *https* in tutte le interazioni che arrivano o vanno verso l'esterno; il modo più semplice per aggiungere tale implementazione è quello di utilizzare *nginx.*

Nginx è un web server open-source ad alte prestazioni che può essere usato anche come *reverse-proxy*; ampiamente utilizzato per la distribuzione di contenuti statici nginx viene utilizzato anche per applicazioni di tipo *load balancing* e per la gestione delle richieste HTTP e HTTPS per applicazioni web.

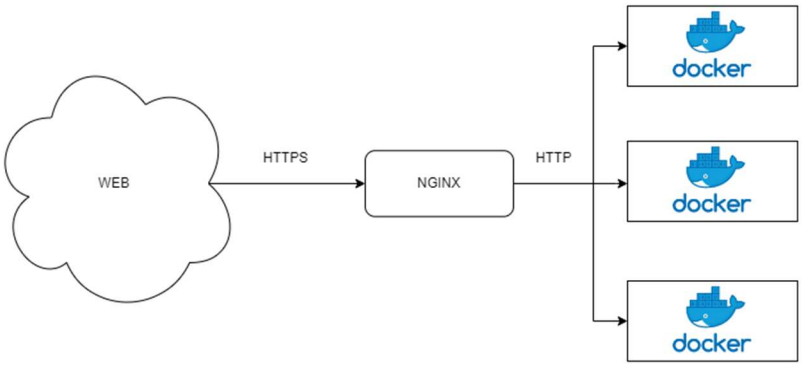


Figura 14 nginx

Come mostra la figura 14, nginx riceve delle richieste dall'esterno in *https* e le inoltra ai vari container in *http*; per tale motivo è stato necessario inserire nella configurazione un certificato ed una chiave SSL.

Di seguito viene riportato il codice di generazione di chiave e certificato:

```
antonio@xpsNaponio: ~/Web
antonio@xpsNaponio:~/WebRtc/ProgettoEsame/ProgettoWebRTC$ openssl req -x509 -nodes -days 365 -ne
wkey rsa:2048 -keyout localhost.key -out localhost.crt -config localhost.conf
```

Figura 15 Creazione certificato e chiave

La configurazione di nginx è molto semplice ed è delegata ad un file “default.conf” che viene usato per configurare l’immagine creata appositamente per questo utilizzo.

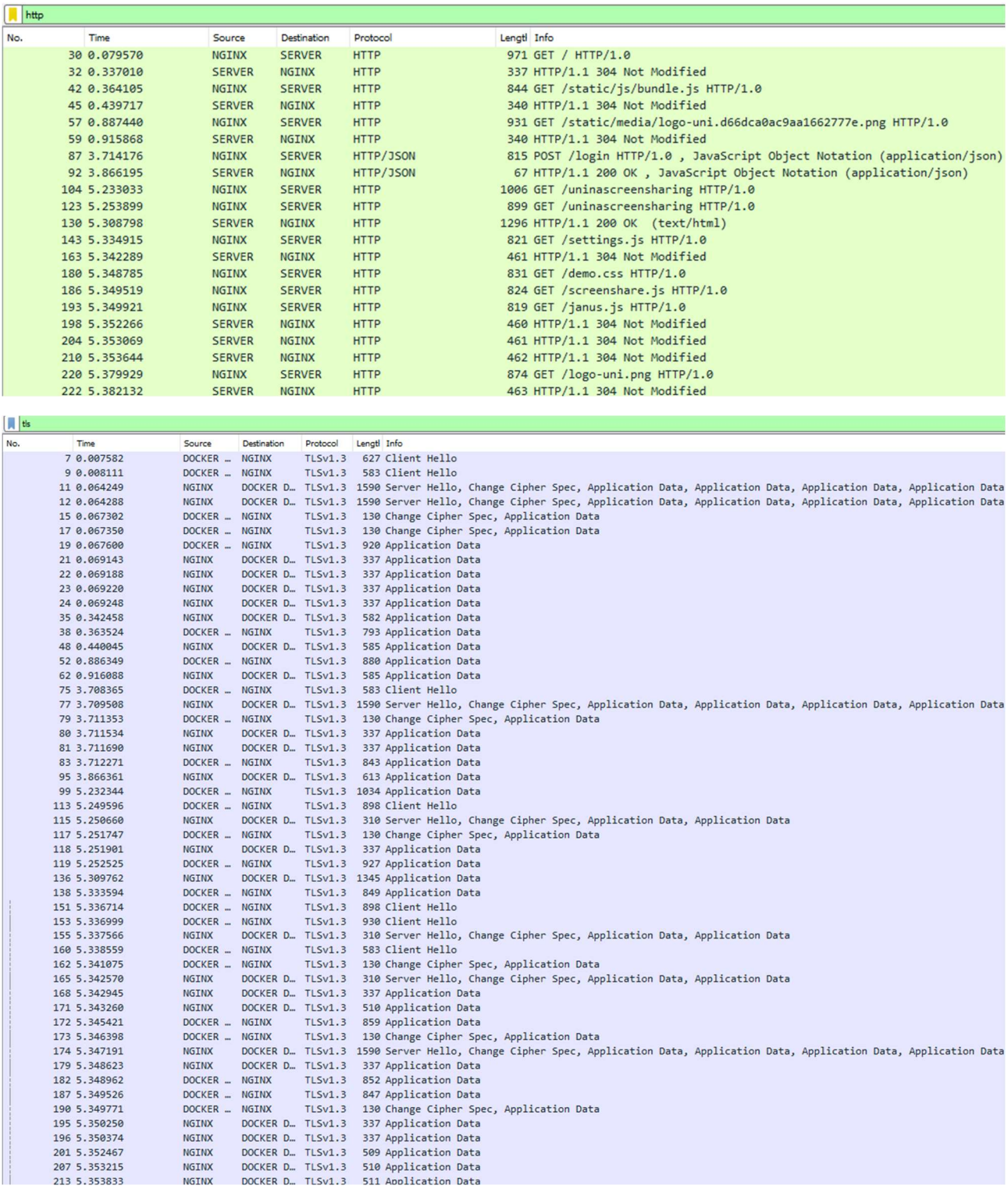
In figura 16 possiamo osservare nel file le configurazioni di:

- Porta su cui è in ascolto il server.
- Percorso dei file di cifratura e protocollo scelto.
- Proxying delle richieste in entrata.

```
server {
    listen 443 ssl default_server;
    server_name localhost;

    ssl_certificate      /etc/nginx/ssl/localhost.crt;
    ssl_certificate_key  /etc/nginx/ssl/localhost.key;
    ssl_protocols        TLSv1.2 TLSv1.3;

    location / {
        proxy_pass http://194.20.1.5:3000/;

    }

    location /server/ {
        proxy_pass http://194.20.1.5:8000/;

    }

    location /janus/info {
        proxy_pass http://194.20.1.4:8088/janus/info;

    }

}
```

Figura 16 File di configurazione

Per controllare l'effettivo funzionamento da *reverse proxy* del server nginx possiamo osservare gli screenshot in figura 17 del file "trace_nginx.pcapng" che mostra due filtri applicati sul trace realizzato tramite comando *tcpdump* sulla console del container nginx. Possiamo osservare come tutte le richieste in *http* siano da/verso nginx a/verso i container interni al *Docker compose* mentre tutte le richieste tls siano da/a nginx a/verso il *Docker deamon*, che rappresenta il collegamento con il mondo esterno, a riprova di come tutte le richieste giungano ai container in maniera sicura.



http

No.	Time	Source	Destination	Protocol	Lengtl	Info
30	0.079570	NGINX	SERVER	HTTP	971	GET / HTTP/1.0
32	0.337010	SERVER	NGINX	HTTP	337	HTTP/1.1 304 Not Modified
42	0.364105	NGINX	SERVER	HTTP	844	GET /static/js/bundle.js HTTP/1.0
45	0.439717	SERVER	NGINX	HTTP	340	HTTP/1.1 304 Not Modified
57	0.887440	NGINX	SERVER	HTTP	931	GET /static/media/logo-uni.d66dca0ac9aa1662777e.png HTTP/1.0
59	0.915868	SERVER	NGINX	HTTP	340	HTTP/1.1 304 Not Modified
87	3.714176	NGINX	SERVER	HTTP/JSON	815	POST /login HTTP/1.0 , JavaScript Object Notation (application/json)
92	3.866195	SERVER	NGINX	HTTP/JSON	67	HTTP/1.1 200 OK , JavaScript Object Notation (application/json)
104	5.233033	NGINX	SERVER	HTTP	1006	GET /uninascreensharing HTTP/1.0
123	5.253899	NGINX	SERVER	HTTP	899	GET /uninascreensharing HTTP/1.0
130	5.308798	SERVER	NGINX	HTTP	1296	HTTP/1.1 200 OK (text/html)
143	5.334915	NGINX	SERVER	HTTP	821	GET /settings.js HTTP/1.0
163	5.342289	SERVER	NGINX	HTTP	461	HTTP/1.1 304 Not Modified
180	5.348785	NGINX	SERVER	HTTP	831	GET /demo.css HTTP/1.0
186	5.349519	NGINX	SERVER	HTTP	824	GET /screenshare.js HTTP/1.0
193	5.349921	NGINX	SERVER	HTTP	819	GET /janus.js HTTP/1.0
198	5.352266	SERVER	NGINX	HTTP	460	HTTP/1.1 304 Not Modified
204	5.353069	SERVER	NGINX	HTTP	461	HTTP/1.1 304 Not Modified
210	5.353644	SERVER	NGINX	HTTP	462	HTTP/1.1 304 Not Modified
220	5.379929	NGINX	SERVER	HTTP	874	GET /logo-uni.png HTTP/1.0
222	5.382132	SERVER	NGINX	HTTP	463	HTTP/1.1 304 Not Modified

tls

No.	Time	Source	Destination	Protocol	Lengtl	Info
7	0.007582	DOCKER …	NGINX	TLSv1.3	627	Client Hello
9	0.008111	DOCKER …	NGINX	TLSv1.3	583	Client Hello
11	0.064249	NGINX	DOCKER D…	TLSv1.3	1590	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data
12	0.064288	NGINX	DOCKER D…	TLSv1.3	1590	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data
15	0.067302	DOCKER …	NGINX	TLSv1.3	130	Change Cipher Spec, Application Data
17	0.067350	DOCKER …	NGINX	TLSv1.3	130	Change Cipher Spec, Application Data
19	0.067600	DOCKER …	NGINX	TLSv1.3	920	Application Data
21	0.069143	NGINX	DOCKER D…	TLSv1.3	337	Application Data
22	0.069188	NGINX	DOCKER D…	TLSv1.3	337	Application Data
23	0.069220	NGINX	DOCKER D…	TLSv1.3	337	Application Data
24	0.069248	NGINX	DOCKER D…	TLSv1.3	337	Application Data
35	0.342458	NGINX	DOCKER D…	TLSv1.3	582	Application Data
38	0.363524	DOCKER …	NGINX	TLSv1.3	793	Application Data
48	0.440045	NGINX	DOCKER D…	TLSv1.3	585	Application Data
52	0.886349	DOCKER …	NGINX	TLSv1.3	880	Application Data
62	0.916088	NGINX	DOCKER D…	TLSv1.3	585	Application Data
75	3.708365	DOCKER …	NGINX	TLSv1.3	583	Client Hello
77	3.709508	NGINX	DOCKER D…	TLSv1.3	1590	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data
79	3.711353	DOCKER …	NGINX	TLSv1.3	130	Change Cipher Spec, Application Data
80	3.711534	NGINX	DOCKER D…	TLSv1.3	337	Application Data
81	3.711690	NGINX	DOCKER D…	TLSv1.3	337	Application Data
83	3.712271	DOCKER …	NGINX	TLSv1.3	843	Application Data
95	3.866361	NGINX	DOCKER D…	TLSv1.3	613	Application Data
99	5.232344	DOCKER …	NGINX	TLSv1.3	1034	Application Data
113	5.249596	DOCKER …	NGINX	TLSv1.3	898	Client Hello
115	5.250660	NGINX	DOCKER D…	TLSv1.3	310	Server Hello, Change Cipher Spec, Application Data, Application Data
117	5.251747	DOCKER …	NGINX	TLSv1.3	130	Change Cipher Spec, Application Data
118	5.251901	NGINX	DOCKER D…	TLSv1.3	337	Application Data
119	5.252525	DOCKER …	NGINX	TLSv1.3	927	Application Data
136	5.309762	NGINX	DOCKER D…	TLSv1.3	1345	Application Data
138	5.333594	DOCKER …	NGINX	TLSv1.3	849	Application Data
151	5.336714	DOCKER …	NGINX	TLSv1.3	898	Client Hello
153	5.336999	DOCKER …	NGINX	TLSv1.3	930	Client Hello
155	5.337566	NGINX	DOCKER D…	TLSv1.3	310	Server Hello, Change Cipher Spec, Application Data, Application Data
160	5.338559	DOCKER …	NGINX	TLSv1.3	583	Client Hello
162	5.341075	DOCKER …	NGINX	TLSv1.3	130	Change Cipher Spec, Application Data
165	5.342570	NGINX	DOCKER D…	TLSv1.3	310	Server Hello, Change Cipher Spec, Application Data, Application Data
168	5.342945	NGINX	DOCKER D…	TLSv1.3	337	Application Data
171	5.343260	NGINX	DOCKER D…	TLSv1.3	510	Application Data
172	5.345421	DOCKER …	NGINX	TLSv1.3	859	Application Data
173	5.346398	DOCKER …	NGINX	TLSv1.3	130	Change Cipher Spec, Application Data
174	5.347191	NGINX	DOCKER D…	TLSv1.3	1590	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data
179	5.348623	NGINX	DOCKER D…	TLSv1.3	337	Application Data
182	5.348962	DOCKER …	NGINX	TLSv1.3	852	Application Data
187	5.349526	DOCKER …	NGINX	TLSv1.3	847	Application Data
190	5.349771	DOCKER …	NGINX	TLSv1.3	130	Change Cipher Spec, Application Data
195	5.350250	NGINX	DOCKER D…	TLSv1.3	337	Application Data
196	5.350374	NGINX	DOCKER D…	TLSv1.3	337	Application Data
201	5.352467	NGINX	DOCKER D…	TLSv1.3	509	Application Data
207	5.353215	NGINX	DOCKER D…	TLSv1.3	510	Application Data
213	5.353833	NGINX	DOCKER D…	TLSv1.3	511	Application Data

Figura 17 Trace wireshark

5. MYSQL

La scelta del database da utilizzare è ricaduta su MySQL che è stato implementato grazie all'utilizzo di una *Docker image* ufficiale fornita direttamente da Oracle.

La configurazione è stata effettuata attraverso i metodi forniti dalla libreria *mysql2* utilizzati all'interno di un file JavaScript presente nei file del server di back-end.

Il setup del database è stato realizzato attraverso il file *setup.sql* che ha permesso un'esecuzione automatica della configurazione iniziale (creazione della table, creazione delle credenziali di accesso e creazione dell'utente admin) all'avvio del *Docker compose*.

```
-- create the databases
CREATE DATABASE IF NOT EXISTS utenti;

-- create the users for each database
CREATE USER 'guest'@'%' IDENTIFIED BY 'prova';
GRANT CREATE, ALTER, INDEX, LOCK TABLES, REFERENCES, UPDATE, DELETE, DROP, SELECT, INSERT ON `utenti`.* TO 'guest'@'%';

FLUSH PRIVILEGES;

CREATE TABLE IF NOT EXISTS utenti.credenziali (
    username VARCHAR(45) PRIMARY KEY NOT NULL,
    password VARCHAR(45) NOT NULL,
    nome VARCHAR(45) NOT NULL,
    cognome VARCHAR(45) NOT NULL,
    email VARCHAR(45) UNIQUE NOT NULL,
    token VARCHAR(45) UNIQUE NOT NULL
);
INSERT INTO utenti.credenziali  (username, password, nome, cognome, email, token)
                                VALUES ('admin', 'admin', 'Ad', 'Min', 'admin@example.com', 'admintoken');
```

Figura 18 Setup mysql

6. JANUS

Per quanto riguarda il container Janus si è scelto di utilizzare un'immagine personalizzata al fine di inserire tutti gli elementi utili ad eseguire la fase di segnalazione e negoziazione come previsto dal protocollo WebRTC.

Tutte le librerie e file aggiunti al Dockerfile (figura 18) sono contenuti all'interno della documentazione di Janus per l'istallazione nel container[4].

```
FROM ubuntu:latest

RUN apt update && apt upgrade -y \
    libmicrohttpd-dev libjansson-dev \
    libssl-dev libsofia-sip-ua-dev libglib2.0-dev \
    libopus-dev libogg-dev libcurl4-openssl-dev liblua5.3-dev \
    libconfig-dev pkg-config libtool automake \
    git wget meson ninja-build nano python3-pip cmake


RUN cd /tmp && \
    wget https://github.com/cisco/libsrtp/archive/v2.2.0.tar.gz && \
    tar xfv v2.2.0.tar.gz && \
    cd libsrtp-2.2.0 && \
    ./configure --prefix=/usr --enable-openssl && \
    make shared_library &&  make install

## ho scaricato con git sul mio pc  la repo di libnice suggerita nel README di janus e poi l'ho copiata nell'immagine
COPY /libnice /tmp/libnice

RUN cd /tmp/libnice && \
    meson --prefix=/usr build && ninja -C build &&  ninja -C build install



# ## installo janus
RUN cd /usr/local/src && \
    git clone https://github.com/meetecho/janus-gateway.git && \
    cd janus-gateway && \
    sh autogen.sh && \
    ./configure --prefix=/opt/janus  && \
    make && \
    make install && \
    make configs
```

Figura 19 Dockerfile janus container

Una delle funzioni aggiunte alla demo fornita all'interno della documentazione è *create room*. Essa permette la creazione di una nuova stanza che potrà essere utilizzata immediatamente ed in maniera indipendente dalle altre.

Questa funzione è essenziale se si vuole ottenere un comportamento simile a quello di un ufficio nel quale i partecipanti possono decidere di dividersi in gruppi discutendo in maniera indipendente del proprio *topic* potendo, all'occorrenza, cambiare stanza utilizzandone il numero come identificativo per riconoscerne l'utilizzo.

[4] Documentazione Janus - https://github.com/meetecho/janus-gateway

In figura 19 viene riportata l'implementazione della funzione in JavaScript.

```
function createRoom() {

    //prendo la stanza selezionata, faccio il parsing perché
    //di default mi viene restituito un "val"
    let input = $("#create").val();

    if (input === "") {
        alert("Inserire un numero di stanza")
        return;
    }

    let myroom = parseInt($("#create").val());

    let list = {
        request: "list"
    };

    let ids = null;
    let rooms = [];
    sfutest.send({
        message: list,
        success: function (result) {
            ids = result["list"]
            $('#arr').empty();
            for (let i in ids) {

                rooms[i] = ids[i]["room"];
            }
        }
    });

    for (let j in rooms) {
        if (rooms[j] === myroom) {
            alert("La stanza esiste già")
            return;
        }
    }

    let create = {
        request: "create",
        room: myroom,
        publishers: 6,
        bitrate: 128000,
        fir_freq: 10,
        record: false,
    };


    $('#DivCreazione').hide();
    $('#modalita3').hide();

    sfutest.send({ message: create });
}
```

Figura 20 Funzione creazione nuova stanza

7. CONCLUSIONE

A conclusione di questa breve documentazione, è possibile affermare che grazie agli strumenti utilizzati è molto semplice sviluppare un sistema di comunicazione che permetta di condividere risorse e, in generale, di mettere in comunicazione più endpoint.

A supporto del server di segnalazione Janus si è scelto di utilizzare il plugin *videoroom* fornito come demo nella documentazione per adempiere all'obiettivo che ci si era prefissati; non rappresenta uno sforzo particolarmente oneroso l'inserimento di funzionalità diverse da questa come, ad esempio, un sistema di live chat o lo sviluppo di una più semplice audio room.

Potenziali semplici aggiornamenti per l'applicazione presentata possono essere:

- Switch contestuale da una stanza all'altra senza interrompere la condivisione.
- Assegnazione di privilegi ad un ipotetico utente host con possibilità di mutare o escludere gli utenti dalla room.
- Schermata di presentazione della stanza con descrizione delle finalità della riunione e elenco dei partecipanti attivi.
- Implementazione di un sistema di partecipazione esclusiva ad una stanza tramite inviti o password.

La fase di user testing non ha evidenziato particolari bug per quanto riguarda la fluidità e la correttezza dell'esecuzione dell'applicazione in tutte le sue funzionalità.