Documentazione

- **Titolo del progetto:**

  SpaceHub: Applicazione web che consente la gestione delle prenotazioni delle aule e lo streaming delle riunioni svolte al loro interno.

- **Nome del corso:**

  Web and Real Time Communication Systems

- **Anno accademico:**

  2025 / 2026

- **Team di sviluppo:**

  Annarita Fabiano (M63001789)

  Giuseppe Maglione (M63001777)

- **Link alla repository:**

  https://github.com/WebRTC-Projects-Unina/Rooms_Reservation

Indice

1. Introduzione

💡 Per evitare ambiguità, nel documento si utilizzeranno le seguenti definizioni per identificare gli utilizzatori del sistema:

1. **Utenti della Piattaforma**
   - *User*: È un utente che ha effettuato il login al sistema. Ha i permessi per consultare la disponibilità delle aule, creare nuove prenotazioni e gestire le proprie.
2. **Ruoli nella Sessione Video**
   - *Host*: È l'User proprietario della prenotazione associata alla stanza. È l'unico abilitato a inizializzare la stanza sul media server ed è l'unico a possedere i permessi di trasmissione e gestione della riunione.
   - *Guest*: È un utente (registrato o non) che accede alla riunione tramite link di invito. Non ha permessi amministrativi sulla stanza.

1.1 Descrizione del progetto

Il progetto consiste nello **sviluppo di un'applicazione web** interattiva pensata per esplorare e mettere in pratica tecnologie di programmazione web e meccanismi di comunicazione real-time nei browser.

L'applicazione, dunque, permette agli utenti di registrarsi, autenticarsi, visualizzare le aule disponibili e prenotarne una per una specifica data e fascia oraria. Al momento dell'ingresso in aula, un sistema di lettura della smart card consente di verificare l'identità dell'utente e autorizzarne l'accesso. Una volta iniziato l'orario della prenotazione, l'utente che ha effettuato la prenotazione può avviare una

riunione online: l’host può condividere schermo e audio e generare un link pubblico che consente anche a utenti non registrati di partecipare. L’applicazione unisce quindi tecniche di sviluppo web con tecnologie WebRTC, offrendo funzionalità interattive e comunicazione real-time.

1.2 Requisiti funzionali

Autenticazione

Il sistema deve permettere all’utente di registrarsi tramite username e password.

- *Soluzione*: Il backend, una volta ricevuti i dati dal frontend, verifica l’unicità dello username, applica hashing della password e salva i dati nel database.

Il sistema deve permettere all’utente di autenticarsi tramite username e password.

- *Soluzione*: Il backend verifica che l’utente esista, confronta la password con l’hash salvato e in caso positivo crea una sessione utente per mantenerne lo stato.

Gestione delle aule

Il sistema deve permettere all’utente di visualizzare le aule disponibili in una specifica data e fascia oraria.

Il sistema deve permettere all’utente di effettuare la prenotazione di un’aula disponibile.

Il sistema deve permettere all’utente di modificare una prenotazione esistente.

Il sistema deve permettere all’utente di eliminare una prenotazione esistente.

- *Soluzione*: Questo viene garantito da una semplice ed efficiente comunicazione tra frontend, backend e database.

Accesso fisico alle aule

Il sistema deve permettere all’utente di autenticarsi fisicamente tramite smart card, attraverso un lettore di carte situato all'ingresso dell’aula.

- *Soluzione*: Il server locale legge i dati dalla carta tramite il lettore, dopodiché li invia al backend, il quale verifica e decide se permettere l’accesso.

Streaming

Il sistema deve permettere all'utente che ha effettuato la prenotazione di avviare lo streaming live della riunione a partire dall'orario di prenotazione.

- *Soluzione*: Si utilizza un plugin Janus dedicato alla videoconferenza.

Il sistema deve generare un link pubblico che consente la visualizzazione dello stream anche a utenti non autenticati.

- *Soluzione*: Il backend genera un link pubblico associato alla specifica riunione.

1.3 Requisiti non funzionali

Bassa latenza (Performance)

Il sistema deve permettere all'host di guardare la riunione trasmessa dal guest con uno scarto di tempo non eccessivo (real-time).

- *Soluzione*: Questo viene garantito dall'uso di protocolli WebRTC.

Attraversamento NAT (Affidabilità e Robustezza)

Il sistema deve essere in grado di stabilire connessioni anche se host e guest si trovano dietro NAT restrittivi.

- *Soluzione*: Si utilizza il protocollo ICE, per il quale è stato configurato il server STUN di Google.

Controllo accessi (Sicurezza)

Il sistema deve permette al solo utente proprietario della prenotazione (host) di inviare stream, gli altri utenti (guest) devono essere limitati alla sola visualizzazione.

- *Soluzione*: L'utente viene considerato host, e quindi abilitato allo stream, soltanto se il suo identificativo coincide con quello presente nella prenotazione.

Protezione pagine (Sicurezza)

Il sistema deve offrire servizi di autenticazione e autorizzazione, in modo da proteggere le pagine per la gestione delle prenotazioni di un utente.

- *Soluzione*: Ogni utente, dopo aver effettuato il login, viene fornito di un identificativo di sessione, da includere in ogni richiesta successiva per poterlo autenticare.

Traffico crittografato (Sicurezza)

Il sistema deve opportunamente proteggere il traffico scambiato con il client.

- *Soluzione*: Il client comunica con il server tramite HTTPS, e con il media server tramite WSS e HTTPS (con certificato self-signed).
- *Motivazione*: L'utente può inviare al server dati sensibili, come nome utente e password.

Autenticazione lettore di carte (Sicurezza)

Il sistema deve verificare l'autenticità delle richieste di accesso alle stanze, accettando solo quelle provenienti da lettori di carte (Smart Card Reader) autorizzati.

- *Soluzione*: Ogni lettore di carte possiede una coppia di chiavi pubblica/privata, grazie alle quali gestisce un meccanismo di firma digitale per i dati inviati al backend.
- *Motivazione***:** Questo impedisce che un attaccante possa costruire richieste ad-hoc da inviare al backend e ottenere l'accesso alla stanza. Senza la chiave privata (che risiede fisicamente nel lettore protetto), l'attaccante non può generare una firma valida e il server ignorerà la richiesta.

Videoconferenza (Scalabilità)

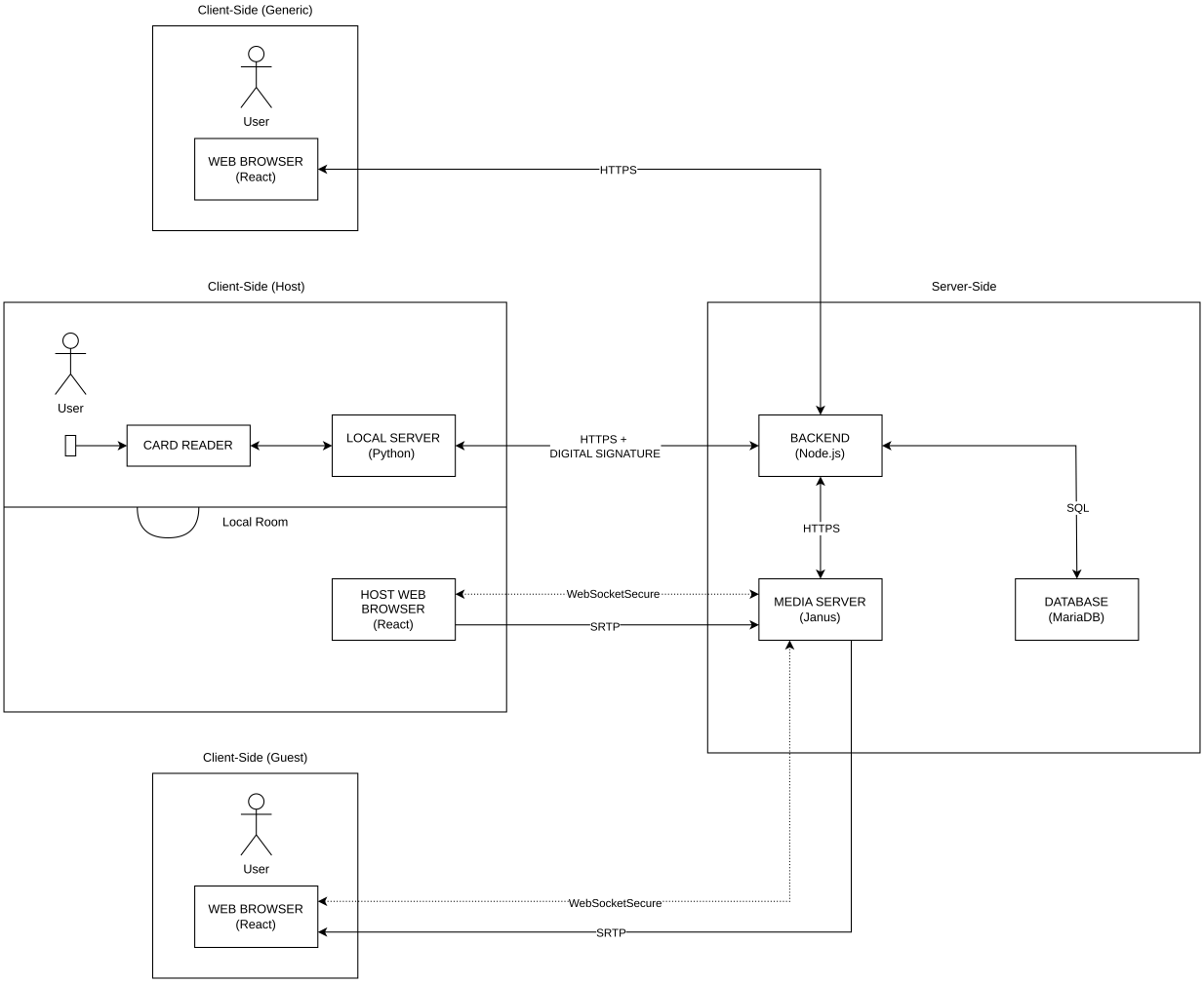
Il sistema deve poter gestire connessioni di un numero elevato di guest, mantenendo una bassa latenza e una qualità video stabile, indipendentemente dal numero di client connessi.

- *Soluzione*: È stata adottata un'architettura basata su un Media Server che distribuisce lo stream dell'host verso i guest, rendendo il flusso dell'host costante indipendentemente dal numero di guest collegati.
- *Motivazione*: Questa scelta supera i limiti delle architetture Peer-to-Peer (mesh), dove l'host dovrebbe generare un flusso in upload distinto per ogni guest.

1.4 Architettura

L'architettura del sistema si basa su cinque componenti fondamentali:

1. **Backend** *(Node.js)*: Routing e logica di business.
2. **Database** *(MariaDB)*: Persistenza dei dati.
3. **Frontend** *(React)*: Interfaccia utente.
4. **Media Server** *(Janus)*: Gestione dei flussi audio e video.
5. **Local Room Server** *(Python)*: Lettura delle smart card e convalida delle richieste di accesso.



Il sistema gestisce tre tipi di flusso dati:

- *Dati Applicativi*: Frontend ↔ Backend ↔ Database (Disponibilità Aule, Prenotazioni, Login, ecc).
- *Dati Media*: Frontend ↔ Media Server (Audio e Video in tempo reale).
- *Dati di Accesso*: Local Room Server ↔ Backend (Autorizza Accesso, Non Autorizzare Accesso).

2. Backend

Il **backend** dell'applicazione è stato sviluppato con *Node.js*. Questo si occupa di:

1. *Offrire una API RESTful*: Espone gli endpoint consumati dal frontend per svolgere operazioni sulle prenotazioni, creare le stanze per lo streaming delle riunioni, verificare la validità delle richieste di accesso del lettore di carte e gestire l'autenticazione.
2. *Gestire Stanze Riunioni*: Svolge un ruolo critico nell'avvio delle riunioni. Poiché il server Janus è stato configurato per richiedere una chiave segreta (`admin_key`) per la creazione della stanza in cui fare lo streaming della riunione, il backend funge da intermediario:
   - Riceve la richiesta di avvio lezione dall'host (frontend).
   - Valida i permessi (controlla che l'utente sia il proprietario della prenotazione).
   - Invia comandi a Janus per istanziare e configurare la stanza del plugin *VideoRoom*.
3. *Servire Assets Statici*: In ambiente di produzione, si occupa di servire i file statici dell'applicazione React compilata.

2.1 Struttura

Il backend è stato sviluppato seguendo il **pattern architetturale** *MVC (Model-View-Controller)*. Questa struttura disaccoppia la logica di gestione delle richieste (*controller*) dalla logica di accesso ai dati (*Model*), permettendo di modificare una parte del sistema senza impattare negativamente sulle altre.

- I **models** rappresentano lo strato a diretto contatto con il database. Astraggono l'accesso ai dati tramite query SQL. Dunque, il resto dell'applicazione non deve sapere come i dati vengono recuperati, ma deve solo chiamare funzioni semantiche.
- Le **routes** fungono da punto di ingresso. Mappano gli URL e i metodi HTTP alle specifiche funzioni dei controller.
- I **controllers** sono la parte operativa che gestisce le richieste HTTP. Ricevono l'input dalle rotte, attuano la logica applicativa e preparano la risposta per il client.

2.2 API Esposte

Il backend **espone una serie di API**, le quali vengono invocate dal frontend per completare le operazioni richieste dall'utente. Le API sono suddivise in varie categorie, sulla base del ruolo che ricoprono.

Auth

Le API di tipo `Auth` si occupano della della sessione dell'utente, aiutando a garantire che solo gli utenti autenticati (e autorizzati) possano accedere a specifiche risorse.

- `POST /auth/login`: Verifica le credenziali dell'utente e ne permette l'accesso al sistema in caso di esito positivo.
- `POST /auth/logout`: Termina la sessione corrente dell'utente.
- `POST /auth/register`: Permette la registrazione di un nuovo utente nel sistema.
- `GET /auth/me`: Verifica lo stato della sessione corrente dell'utente. Ovvero, controlla se l'utente possiede un cookie di sessione valido quando accede a una risorsa.

Non si tratta di endpoint diretti, ma il loro insieme costituisce un *Middleware* utilizzato da altre API.

Bookings

Le API di tipo `Bookings` implementano le operazioni CRUD sulle prenotazioni. La maggior parte di questi endpoint è protetta dal middleware di autenticazione,

garantendo che solo gli utenti autenticati possano prenotare o modificare le proprie risorse.

- `GET /api/aule-disponibili`: Fornisce l'elenco delle aule disponibili nel sistema, sulla base della data e dell'orario indicati da un utente (tramite query string). Non richiede autenticazione.
- `POST /api/crea-prenotazione`: Inserisce una nuova prenotazione nel sistema. Richiede autenticazione.
- `GET /api/prenotazioni`: Recupera lo storico di tutte le prenotazioni di un utente corrente. Richiede autenticazione.
- `GET /api/prenotazioni/:id`: Mostra i dettagli della prenotazione di un utente in base all'identificativo. Richiede autenticazione.
- `PUT /api/prenotazioni/:id`: Modifica i dettagli della prenotazione (esistente) di un utente in base all'identificativo. Richiede autenticazione.
- `DELETE /api/prenotazioni/:id`: Cancella la prenotazione di un utente in base all'identificativo. Richiede autenticazione.

Gli endpoint che eseguono le operazioni CRUD sulle prenotazioni dell'utente, verificano innanzitutto che questo sia il proprietario della prenotazione.

Meetings

La API di tipo `Meetings` si occupa della creazione delle stanze per il plugin *VideoRoom* del server Janus. Dunque, richiede al media server l'allocazione delle risorse necessarie per lo streaming della riunione.

- `POST /api/crea-riunione/:id`: Inizializza una stanza virtuale sul server Janus per la prenotazione in base all'identificativo.

Questo endpoint è protetto dal middleware di autenticazione, in modo da permettere solo a utenti presenti nel sistema la creazione di stanze per lo streaming della riunione. Inoltre, verifica che l'utente che invia la richiesta sia effettivamente il proprietario della prenotazione.

Reader

La API di tipo `Readers` si occupa della recezione di richieste di accesso alle aule effettuate dal server locale presente sul lettore di carte.

- `POST /api/controlla-accesso`: Permette di richiedere l'autorizzazione all'apertura della porta di un'aula.

Questo endpoint non è protetto dal middleware di autenticazione, in quanto i lettori di carte vengono autenticati separatamente tramite un meccanismo di firma digitale (chiave pubblica/privata).

3. Database

L'applicazione utilizza un **database relazionale MariaDB** per la gestione persistente delle informazioni relative agli utenti, alle aule, alle prenotazioni e ai meccanismi di accesso fisico tramite l'uso di smart card. Si è scelto l'utilizzo di un database relazionale per garantire integrità referenziale, coerenza dei dati e supporto a query strutturate.

L'accesso al database avviene esclusivamente tramite backend, il quale funge da livello di astrazione tra il frontend e lo strato di persistenza, evitando accessi diretti, migliorando in questo modo la sicurezza e la manutenibilità.

3.1 Connessione e gestione del pool

Promise

La comunicazione tra backend (Node.js) e database (MariaDB) è realizzata tramite la libreria `mysql2/promise` , che fornisce un'interfaccia basata su *Promise* per l'esecuzione delle operazioni sul database.

Connection pool

Nel file `db.js` è implementata la logica di inizializzazione di un **connection pool**, ovvero di un insieme di connessioni persistenti al database riutilizzabili tra più richieste.

> 💡 Nota: L'uso del pool evita l'apertura e la chiusura continua delle connessioni, migliorando le prestazioni e la scalabilità del sistema.

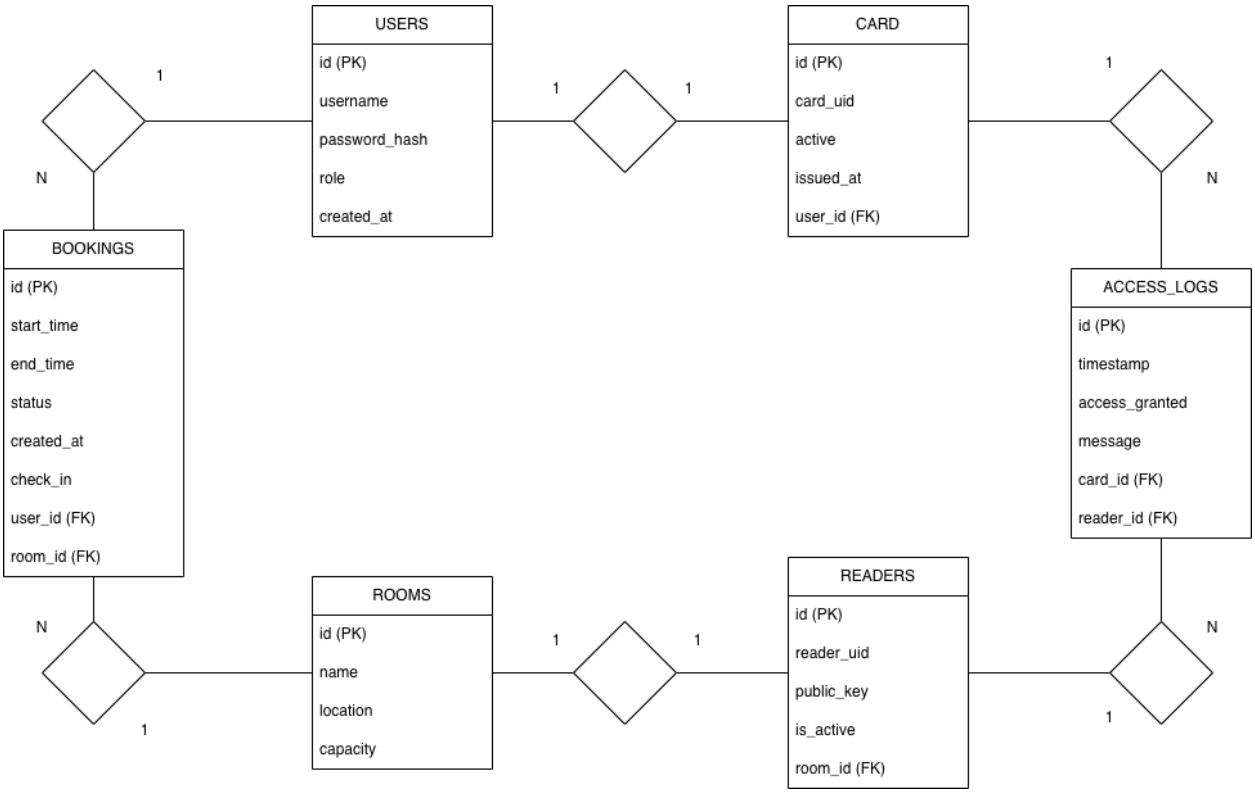
Il pool viene creato una sola volta all'avvio del server tramite la funzione `initDbPool()` ed è condiviso da tutte le componenti del backend.

Parametri di connessione e query

I parametri di connessione (host, porta, credenziali e nome del database) sono configurati tramite **variabili d'ambiente**, rendendo l'applicazione facilmente portabile tra diversi ambienti di esecuzione. È inoltre definita una funzione helper `query(sql, params)` che incapsula l'esecuzione delle query SQL, semplificando l'interazione con lo strato di persistenza.

3.2 Struttura

La **struttura del database** è definita nel file `scheme.sql`, mentre per inizializzare le tabelle necessarie al funzionamento dell'applicazione viene riportato il file `init.sql`. Lo schema di riferimento è il seguente:



Tabelle

- La tabella `users` memorizza le informazioni relative agli utenti registrati.

- La tabella `rooms` contiene i dati relativi alle aule presenti nel sistema, come nome, posizione e capacità massima.
- La tabella `readers` rappresenta i lettori di smart card installati nelle aule.
- La tabella `cards` gestisce le smart card associate agli utenti.
- La tabella `bookings` memorizza le prenotazioni delle aule.
- La tabella `access_logs` registra tutti i tentativi di accesso fisico alle aule tramite smart card.

4. Frontend

4.1 Architettura generale

Il **frontend** dell'applicazione è stato sviluppato utilizzando *React*, una libreria JavaScript per la creazione di interfacce utente basate su componenti.

L'applicazione frontend comunica con il backend tramite *API REST* esposte dal server, utilizzando il protocollo HTTPS e il formato JSON per lo scambio dei dati.

4.2 Gestione delle chiamate API

Per centralizzare e semplificare la comunicazione con il backend è stato creato il file `api.js`, che definisce un insieme di funzioni riutilizzabili per effettuare richieste HTTP.

Sono state implementate funzioni dedicate per ciascun metodo HTTP principale:

- `apiGet` per richieste di lettura (GET).
- `apiPost` per la creazione di nuove risorse (POST).
- `apiPut` per l'aggiornamento di risorse esistenti (PUT).
- `apiDelete` per l'eliminazione di risorse (DELETE).

Tutte le richieste includono l'opzione `credentials: "include"`, necessaria per permettere l'invio dei cookie di sessione e quindi supportare l'autenticazione basata su sessione gestita dal backend.

4.3 Struttura principale dell'applicazione

Il file `main.jsx` rappresenta il punto di ingresso dell'applicazione React e si occupa del rendering del componente principale `App` all'interno del DOM.

Nel file `App.jsx` viene definita la **struttura globale dell'applicazione**, che include:

- Il provider di autenticazione (`AuthProvider`).
- Il sistema di routing tramite `react-router-dom`.
- La barra di navigazione persistente (`Navbar`).

Il routing consente di associare ciascun URL a una specifica pagina dell'applicazione (Home, Login, Aule, Prenotazioni, ecc.), distinguendo tra:

- **Pagine pubbliche**, accessibili a tutti gli utenti.
- **Pagine protette**, accessibili solo agli utenti autenticati.

4.4 Componenti

Protezione delle rotte

L'accesso alle funzionalità riservate (visualizzazione, creazione e modifica delle prenotazioni) è gestito tramite il componente `ProtectedRoute`.

Questo componente:

- Verifica se l'utente è autenticato.
- Reindirizza automaticamente alla pagina di login se l'utente non è autorizzato.

In questo modo, la protezione delle risorse lato frontend risulta centralizzata e coerente con lo stato di autenticazione mantenuto dall'applicazione.

Barra di navigazione

La **barra di navigazione** è implementata tramite il componente React `Navbar` riutilizzabile e visualizzato in modo persistente.

Il contenuto della navbar è dinamico e dipende dallo stato di autenticazione dell'utente:

- Utenti non autenticati visualizzano i pulsanti di login e registrazione.

- Utenti autenticati visualizzano le funzionalità di prenotazione, l'elenco delle proprie prenotazioni e il pulsante di logout.

Questo approccio migliora l'esperienza utente mostrando solo le azioni rilevanti in base al contesto.

4.5 Contesto

Gestione dello stato di autenticazione

L'autenticazione dell'utente è gestita tramite il **Context di React**, implementato nel file `AuthContext.jsx`.

Il contesto mantiene:

- Le informazioni dell'utente autenticato.
- Lo stato di caricamento iniziale.
- Le funzioni di login, logout e registrazione.

Al montaggio dell'applicazione viene effettuata una chiamata al backend per verificare se esiste una sessione attiva (`/auth/me`). Questo meccanismo consente di mantenere l'utente autenticato anche dopo un refresh della pagina, migliorando l'usabilità dell'applicazione.

L'uso del Context permette di condividere lo stato di autenticazione tra tutti i componenti.

4.6 Pagine

Le **pagine principali dell'applicazione** (`Home`, `Login`, `Register`, `Rooms`, `MyBookings`, `CreateBooking`, `EditBooking`, `MeetingGuest`) rappresentano le diverse funzionalità offerte all'utente.

Ogni pagina è implementata come componente React indipendente e utilizza le API fornite dal backend per:

- Autenticazione.
- Visualizzazione delle aule.
- Gestione delle prenotazioni.

- Accesso alle riunioni in streaming.

Questa suddivisione favorisce modularità, leggibilità del codice e facilità di manutenzione. Per quanto riguarda la presentazione grafica, lo **stile delle pagine** è stato separato dalla logica dei componenti React e organizzato in un'apposita cartella `style`. In essa sono contenuti i fogli di stile CSS associati ai vari componenti e alle pagine dell'applicazione, favorendo una migliore modularità del codice, una maggiore leggibilità e una più semplice manutenzione dell'interfaccia utente.

5. Media Server

Per gestire la componente di videoconferenza in tempo reale dell'applicazione è stato utilizzato il server WebRTC *Janus*. All'interno dell'architettura, Janus agisce come **media server**, facendosi carico della complessità legata alla negoziazione delle connessioni, all'attraversamento dei NAT e al trasporto dei flussi audio/video.

5.1 Configurazione

Trasporto sicuro

Per garantire la privacy dei dati e soddisfare i requisiti di sicurezza moderni, Janus è stato configurato per operare esclusivamente su **protocolli crittografati**:

- **WSS (WebSocket Secure)**: Utilizzato per la negoziazione in tempo reale con il client.
- **HTTPS**: Utilizzato per ricevere i comandi amministrativi dal backend.

Tale configurazione prevede l'utilizzo di certificati SSL/TLS validi (nel nostro caso self-signed), indicati all'interno dei file `janus.transport.http.jcfg` e `janus.transport.websockets.jcfg`.

Attraversamento NAT

Poiché il server e i client si trovano spesso dietro firewall o NAT, nel file `janus.jcfg` è stato configurato l'utilizzo del server **STUN** pubblico di *Google*.

💡 Nota: Sebbene l'uso di un server STUN risolva la maggior parte dei casi di connettività, non garantisce il funzionamento dietro NAT Simmetrici o firewall restrittivi. Un'implementazione in produzione richiederebbe l'aggiunta di un server TURN per gestire tali casistiche.

5.2 Plugin VideoRoom

Per il progetto è stato utilizzato il **plugin VideoRoom**, che implementa una logica di videoconferenza basata sul modello publisher/subscriber:

- **Publisher (Host):** Invia i flussi media (schermo e microfono) al server.
- **Subscribers (Guests):** Non inviano media, ma si iscrivono ai flussi disponibili nella stanza per riceverli.

Nel progetto, ad ogni prenotazione è associata una *stanza virtuale* sul server Janus, creata dinamicamente all'avvio della riunione da parte dell'host. Questa viene configurata per permettere al solo proprietario della riunione di agire da publisher.

Al termine della procedura di provisioning per creare la stanza, l'host ottiene un URL di partecipazione condivisibile. Quando i guest navigano su questo indirizzo, il frontend carica la pagina dal file `VideoClassroom.jsx`, che estrae l'ID della stanza dai parametri dell'URL e avvia la negoziazione WebRTC per il `join` alla sessione con ruolo di subscriber.

Sicurezza stanze

Per garantire che le risorse (stanze) del server Janus siano allocate solo agli utenti autorizzati (host), è stata implementata una **strategia di sicurezza** basata su chiave di sicurezza. Il plugin è stato configurato con una chiave (`admin_key`), la quale:

1. È nota solo al backend.
2. È richiesta per operazioni critiche come `create` e `destroy` della stanza.
3. Non è richiesta per operazioni pubbliche come `join`.

In questo modo, le richieste di allocazione delle risorse per lo streaming non partono direttamente dal frontend, ma chiamano una API, passando sempre attraverso l'autenticazione del backend.

5.3 Gestione videoconferenza

Il component del frontend `VideoClassroom.jsx` si occupa della **gestione della videoconferenza**. Questo agisce da interfaccia tra l'utente e la libreria `janus.js`, gestendo l'intero ciclo di vita della connessione WebRTC.

Il componente implementa logiche differenziate in base al `role` dell'utente:

- **Host:** Effettua prima una chiamata API al backend (`/api/crea-riunione`) per garantire il provisioning della stanza. Solo in caso di successo avvia la connessione WSS verso Janus.
- **Guest:** Avvia direttamente la connessione WSS verso Janus per entrare nella stanza.

A gestire l'intera logica basata su ruoli sono le due funzioni:

- `startHostLogic`:
  - Entra nella stanza come publisher.
  - Negozia l'offerta SDP per inviare video (Screen Sharing) e audio (Microfono).
  - Gestisce i controlli locali (es. Mute/Unmute).
- `startGuestLogic`:
  - Entra nella stanza e si mette in ascolto di eventi.
  - Quando rileva un publisher attivo (l'host), esegue la funzione `subscribeToHost`.
  - Crea una risposta SDP da inviare a Janus per ricevere i flussi e li aggancia agli elementi HTML `<video>` e `<audio>` per la riproduzione.

Prima di consentire l'avvio dello streaming, il sistema verifica tramite l'API `/api/prenotazioni/:id` che l'host abbia effettuato fisicamente l'accesso all'aula (flag `check_in` settato a `TRUE` nel database). In assenza di tale conferma, la funzionalità di avvio riunione viene disabilitata.

6. Local Room Server

Il **Local Room Server** è un piccolo server scritto in *Python* in esecuzione sui lettori di smart-card. Il suo compito è semplicemente leggere l'identificativo dell'utente associato alla carta passata sul lettore e richiedere al backend (sfruttando la API `/api/controlla-accesso`) se questo è autorizzato ad accedere o meno all'aula.

6.1 Messaggi con firma digitale

I messaggi scambiati tra il server locale e il backend sfruttano la **firma digitale**, in modo da poter permettere a entrambe le parti di verificare l'autenticità dei messaggi scambiati.

In particolare, ogni lettore possiede una propria coppia di chiavi pubblica/privata e conosce la chiave pubblica del server (hardcoded). Il flusso di scambio dei messaggi tra server locale e backend può essere riassunto come segue.

```
[Local Room Server]
   1. rileva che un utente ha passato la carta sul lettore
   2. legge l'identificativo dell'utente a cui è associata la carta
   3. prepara una richiesta con payload firmato con la chiave privata del lettor
e
   4. invia la richiesta POST all'API esposta dal backend
      ↓
[Backend]
   1. riceve la richiesta dal lettore e ne estrae il payload
   2. verifica che la firma del lettore sia valida
   3. genera una risposta con payload firmato con la chiave privata del server
   4. invia la risposta al lettore
      ↓
[Local Room Server]
   1. riceve la risposta del backend e ne estrae il payload
   2. verifica che la firma del server sia valida
   3. controlla la decisione del backend e apre/non apre la porta di conseguen
za
```

7. Setup

> 💡 Nota: La guida che segue è stata testata su *Arch Linux*, per distribuzioni Linux diverse alcuni dei comandi riportati potrebbero subire delle variazioni.

7.1 Dipendenze

- **Backend**

```
cd /PATH/TO/REPOSITORY/Progetto/backend && npm install
```

- **Database**

```
# --- SETUP DATABASE

# 1. install required packages:
   sudo pacman -S mariadb
   # make sure to make a secure installation
# 2. start mariadb service:
       sudo systemctl start mariadb
# 3. login as root:
       mariadb -u root -p
       # 3.1 create the database:
          CREATE DATABASE DB_NAME CHARACTER SET utf8mb4 COLLATE utf
8mb4_unicode_ci;
   # 3.1 create database user:
          GRANT ALL PRIVILEGES ON DB_NAME.* TO 'USR'@'localhost' IDENTI
FIED BY 'PASS';
        FLUSH PRIVILEGES;
        EXIT;
# 4. upload the schema file:
       mariadb -u USR -p DB_NAME < scheme.sql
# 5. init the database:
```

```
        mariadb -u USR -p DB_NAME < init.sql

# note: it is important to replace the reader's public
#  key inside the init.sql file with the generated one.
```

- **Frontend**

```
cd /PATH/TO/REPOSITORY/Progetto/frontend && npm install
```

- **Media Server**

```
# --- SETUP THE JANUS MEDIA SERVER

git clone https://github.com/meetecho/janus-gateway.git
cd /PATH/TO/REPOSITORY/janus-gateway

sh autogen.sh
./configure --prefix=/opt/janus

make
sudo make install
sudo make configs

# make sure to: enable CORS, setup HTTPS and WSS in janus config files
```

- **Local Room Server**

```
# --- SETUP THE SMART CARD READER

# 1. install required packages:
    sudo pacman -S pcscd-tools
    sudo pacman -S ccid
# 2. start pcscd service:
        sudo systemctl start pcscd
```

```
    # 2.1 add user to pcscd and lp groups:
            sudo usermod -aG pcscd $USER
        sudo usermod -aG lp $USER
# 3. remove kernel modules in conflict with reader:
   sudo modprobe -r pn533_usb
   sudo modprobe -r pn533
   sudo modprobe -r nfc
   # 3.1 to make this changes permanent:
        sudo nano /etc/modprobe.d/blacklist-nfc.conf
        blacklist pn533_usb
        blacklist pn533
        blacklist nfc
# 4. connect and test the reader:
      pcsc_scan    # check if the reader is recognized
```

```
cd /PATH/TO/REPOSITORY/Progetto/smart_card_reader
pip3 -m venv reader-venv
source reader-venv/bin/activate
pip3 install -r requirements.txt
```

7.2 Avvio

Dopo aver scaricato e installato tutte le dipendenze necessarie, per poter avviare il progetto, sono necessari una serie di passaggi.

1. **Clone del repository**

```
git clone https://github.com/giuseppe-maglione/WebRTC
```

2. **Creazione delle variabili di ambiente**

```
# create a .env file in the "Progetto" directory of the repository
# and populate the following content

SESSION_SECRET=YOUR_COOKIE_SESSION_SECRET
```

```
DB_HOST=localhost
DB_PORT=YOUR_DATABASE_PORT
DB_NAME=YOUR_DATABASE_NAME
DB_USER=YOUR_DATABASE_USER
DB_PASS=YOUR_DATABASE_PASS

JANUS_SECRET=YOUR_JANUS_VIDEOROOM_SECRET
```

3. **Avvio backend**

```
cd /PATH/TO/REPOSITORY/Progetto/backend && npm start
npm start
```

4. **Avvio database**

```
sudo systemctl start mariadb
```

5. **Avvio frontend**

```
cd /PATH/TO/REPOSITORY/Progetto/frontend && npm run dev
```

6. **Avvio media server**

```
/opt/janus/bin/janus
```

7. **Avvio local room server**

```
cd /PATH/TO/REPOSITORY/Progetto/smart_card_reader
python3 local_reader_service.py
```

💡 Nota: L’applicazione utilizza dei certificati self-signed, per cui al primo avvio sarà necessario recarsi sugli indirzzi del backend, del frontend e del media server e accettare di voler proseguire con la visualizzazione delle pagine.