SharedPomodoro

Sistemi Distribuiti

Ramzi Gallala - 0001083783 {ramzi.gallala@studi.unibo.it}

Maggio 2025

Indice

Sommario

SharedPomodoro è un'applicazione web progettata per digitalizzare la tecnica del Pomodoro e renderla un'esperienza condivisa e collaborativa.

Gli utenti possono creare una **stanza di lavoro** o unirsi a una stanza esistente per collaborare in tempo reale con altri partecipanti. All'interno della stanza, è possibile **assegnare task a se stessi o agli altri**, favorendo un ambiente di lavoro supportivo e motivante. La tecnica del Pomodoro si basa sull'alternanza tra periodi di lavoro, detti *Pomodori*, e pause regolari. SharedPomodoro applica questo metodo per aiutare gli utenti a migliorare la concentrazione e la produttività, con l'aggiunta di un elemento fondamentale: **il timer è condiviso tra tutti i partecipanti** della stanza, rendendo il lavoro sincronizzato. Il ciclo di lavoro si compone di tre fasi:

- **Pomodoro**: sessione di lavoro, della durata definita a livello di stanza, durante la quale ogni partecipante si dedica ai propri task senza interruzioni.
- **Pausa corta**: breve intervallo dopo ogni Pomodoro, utile per recuperare energie.
- **Pausa lunga**: dopo quattro Pomodori consecutivi, viene attivata una pausa più lunga, pensata per un recupero più completo.

La durata delle fasi può essere **personalizzata dal creatore della stanza o da chi è all'interno**, ma una volta impostati, i tempi sono gli stessi per tutti i membri della stanza.

Inoltre, gli utenti possono visualizzare i propri task completati, tenendo traccia dei progressi e dei risultati raggiunti.

SharedPomodoro si propone come uno strumento semplice ma potente per aumentare la produttività e mantenere alta la concentrazione, sfruttando il sostegno del lavoro di gruppo.

0.1 Goal/Requirements

In questa sezione verranno presentate informazioni dettagliate sugli obiettivi del progetto, sui requisiti necessari e sui risultati attesi.

0.1.1 Quesstions/Answers

Cosa succede all'avvio dell'applicazione?
All'avvio dell'applicazione, l'utente viene accolto dalla schermata di login in cui può scegliere se effettuare il login oppure registrarsi.
La registrazione richiede l'inserimento di nome, cognome, email e una password. Tuttavia, il sistema non consente di registrare un account se l'email è già stata utilizzata da un altro utente.
In caso di login, l'accesso è consentito solo se l'utente è già registrato e le credenziali inserite sono corrette.
È importante notare che un utente può essere connesso da più dispositivi contemporaneamente, pertanto non vi è un limite al numero di sessioni attive per ciascun account.

Cosa succede quando l'utente effettua il login o la registrazione?
Dopo aver completato con successo il login, l'utente viene reindirizzato alla home page dell'applicazione.
Da qui può decidere se creare una nuova stanza, scegliendone liberamente il nome, oppure entrare in una stanza già esistente, se ne conosce il nome.
Nel caso in cui si tenti di creare una stanza con un nome già utilizzato da un'altra stanza attiva, il sistema mostrerà un messaggio di errore.
Inoltre, non è consentito entrare in una stanza in cui si è già presenti, per evitare duplicazioni.

Come funziona la modalità Single Player / Multiplayer?
Quando l'utente crea una stanza, ha la possibilità di avviare una sessione in modalità multiplayer o single player, in base alla scelta di condividere o meno il nome della stanza.

- Se l'utente condivide il nome della stanza con altri, essi potranno accedervi ed effettuare una sessione condivisa (multiplayer).
- Se invece non condivide il nome, l'utente svolgerà la sessione in modalità individuale (single player).

Come funziona la visualizzazione degli utenti connessi?
Gli utenti connessi a una stanza vengono mostrati nella parte inferiore dello schermo, ognuno rappresentato da un cerchio colorato.
Questi cerchi fungono da avatar identificativi. Passando con il mouse sopra uno di essi, appare un tooltip che mostra l'identificativo dell'utente corrispondente. Questo consente di sapere in tempo reale chi è presente all'interno della stanza.

Come funziona la gestione dei tempi?
La gestione dei tempi è visualizzata attraverso un timer centrale, ben visibile nella schermata principale.
All'interno del timer sono presenti i pulsanti *Start*, *Pausa* e *Arresta*, che permettono all'utente di controllare lo scorrere del tempo.
Sopra i pulsanti è indicato lo stato della sessione corrente, che può essere "Pomodoro", "Pausetta" o "Pausa".
Sotto ai pulsanti, invece, si trova il timer vero e proprio, che mostra in tempo reale i minuti e i secondi restanti.

Come funziona la gestione dei task?
La sezione dedicata ai task da completare si trova immediatamente sotto il timer, ed è strutturata per essere intuitiva e facile da usare.
L'utente può aggiungere un nuovo task premendo il pulsante "+", che apre un popup in cui specificare il nome del task e a chi assegnarlo (una singola persona o l'intero gruppo).
I task aggiunti vengono mostrati in una lista più in basso, ognuno accompagnato da un pulsante che consente di eliminarlo dalla lista, quando completato.

Come funziona la configurazione dei tempi?
La configurazione dei tempi è accessibile cliccando sull'icona a forma di clessidra situata nella barra laterale destra.
Una volta aperta, l'utente può scegliere tra valori predefiniti, rappresentati da pulsanti, oppure inserire manualmente una durata personalizzata.
Il sistema verifica che ogni valore inserito sia maggiore di 1 minuto; in caso contrario, viene visualizzato un messaggio di errore.
Questa funzionalità permette di personalizzare la durata delle sessioni Pomodoro e delle pause secondo le preferenze individuali.

Come funziona la visualizzazione dei dati dell'utente?
Nella barra laterale destra è presente un'icona raffigurante una persona.
Cliccando su questa icona, viene visualizzato il nome dell'utente attualmente connesso al sistema.

Cosa succede quando parte il timer?
Quando un utente clicca sul pulsante di avvio del timer, il countdown inizia simultaneamente per tutti i partecipanti presenti nella stessa stanza.
Il tempo visualizzato è lo stesso per ogni utente e l'avanzamento è sincronizzato, garantendo un'esperienza uniforme e coordinata durante le sessioni.

Cosa succede alla fine di ogni sessione?
Al termine del tempo impostato per una sessione, il sistema aggiorna automaticamente lo stato del timer e mostra quale sarà la sessione successiva.
A seconda della sequenza, può trattarsi di una pausa breve, pausa lunga oppure di un nuovo ciclo di lavoro (Pomodoro).

Cosa succede se l'utente abbandona la stanza?
Quando un utente lascia una stanza, possono verificarsi due situazioni:

- Se l'utente era l'unico presente, la stanza rimane attiva ancora per un breve periodo (timeout), e verrà eliminata solo se continuerà a rimanere vuota.
- Se nella stanza sono presenti altri utenti, questi verranno immediatamente aggiornati del cambiamento, e la loro interfaccia rifletterà l'uscita dell'utente in tempo reale.

Cosa succede quando affido un task a un utente?
Quando viene creato un task e assegnato a un utente, il sistema inoltra la richiesta e il task appare nella schermata del destinatario tra quelli da completare.
Nel caso in cui il task venga assegnato all'intero gruppo, esso verrà visualizzato nella lista di tutti i membri della stanza.

Come fa l'utente a uscire dalla stanza?
L'utente può decidere di uscire dalla stanza in due modi:

- Cliccando sull'icona di logout, ritornando alla schermata di login;
- Selezionando il pulsante Home, che consente di tornare alla homepage, in cui è possibile entrare o creare una stanza.

Entrambe le opzioni sono disponibili nella barra laterale.

Come funziona la visualizzazione dei task completati?
Cliccando sull'icona dei task presente nella barra laterale, l'utente accede a una lista dei task completati.
Questi vengono mostrati sotto forma di schede (cards), ciascuna contenente il nome del task e il nome della stanza in cui è stato eseguito.

0.1.2 Scenarios

Quando un utente apre l'applicazione, la prima azione richiesta è l'accesso al sistema, tramite login oppure procedere con la registrazione.

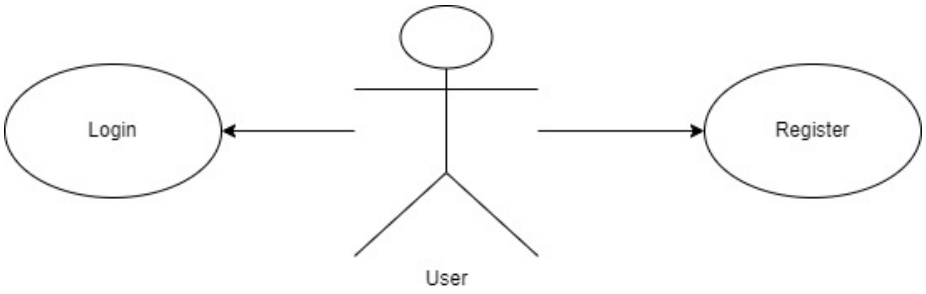

Figura 1: Apertura sito

Una volta effettuato l'accesso, l'utente può scegliere tra due opzioni principali: creare una nuova stanza oppure unirsi a una stanza esistente, in entrambi i casi specificando il nome della stanza. Inoltre, tramite la barra laterale, ha la possibilità, spingendo l'icona, di visualizzare tutti i task completati fino a quel momento.

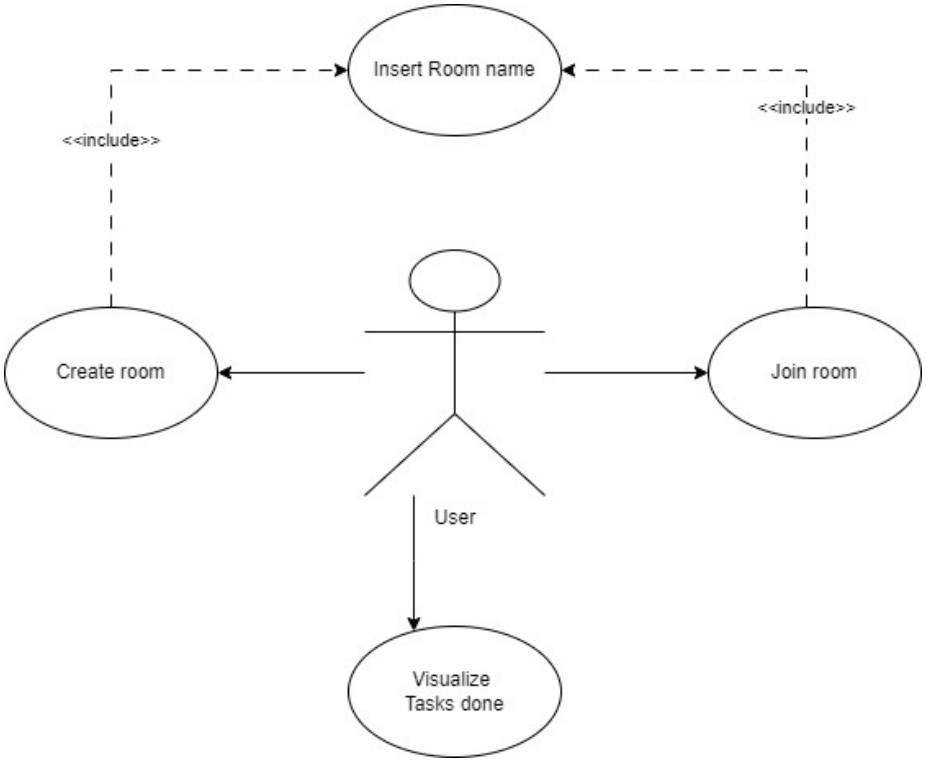

Figura 2: Homepage

Una volta entrato nella stanza, l'utente può visualizzare gli altri partecipanti, i task da completare e, tramite la barra laterale, accedere alla sezione dedicata ai task già completati. Ha inoltre la possibilità di assegnare task a un singolo utente o a tutti i membri della stanza. Per quanto riguarda la gestione del timer condiviso, può avviarlo, metterlo in pausa o arrestarlo. Infine, cliccando sull'apposita icona nella barra laterale, può modificare le impostazioni del tempo.

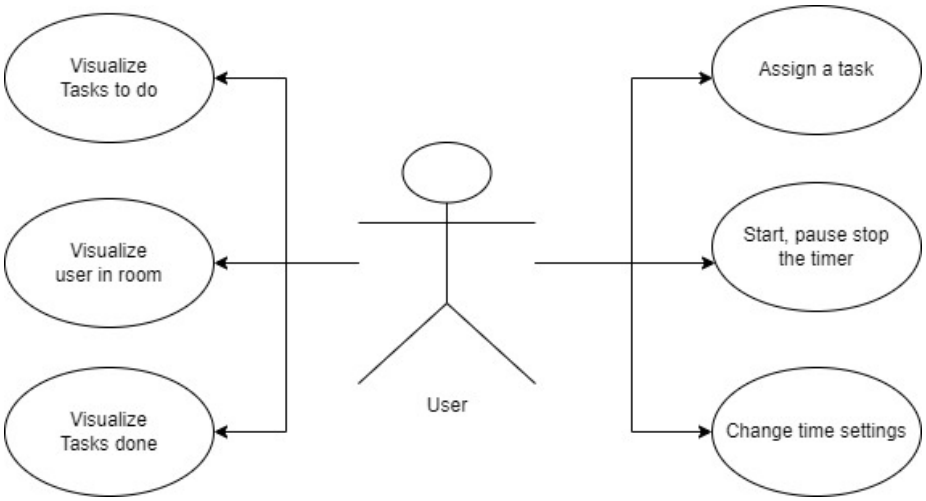

Figura 3: Room

0.2 Requirements Analysis

Requisiti Funzionali

Requisiti Utente

- **Registrazione e login**
  - L'utente deve poter creare un account (registrazione).
  - L'utente deve poter accedere con credenziali (login).
- **Creazione e gestione stanze**
  - L'utente deve poter creare una stanza.
  - L'utente deve poter entrare in una stanza esistente tramite un nome.
  - L'utente deve poter uscire da una stanza.
  - L'utente deve poter visualizzare gli utenti presenti nella stanza.
- **Timer condiviso**
  - L'utente deve poter visualizzare un timer per visualizzare il tempo che trascorre.
  - L'utente deve poter gestire il timer: avviandolo, mettendolo in pausa o stopparlo, per passare alla sessione successiva.
  - L'utente deve poter modificare la durata di lavoro, pausa breve e pausa lunga tra le sessioni.
  - L'utente deve visualizzare il nome della sessione attuale.
- **Assegnazione e gestione task**
  - L'utente deve poter assegnare un task a uno o più utenti all'interno della stanza.
  - L'utente deve poter vedere i task a lui assegnati.
  - L'utente deve poter segnare un task come completato premendo un pulsante "Fatto".
  - L'utente deve poter visualizzare lo storico dei task completati (in una pagina separata).
  - L'utente non deve visualizzare il task una volta completato.

Requisiti di Sistema

- **Gestione degli utenti**
  - Il sistema deve gestire la registrazione, login e logout degli utenti in modo sicuro.

- **Sincronizzazione del timer**
  - Il sistema deve sincronizzare il timer tra tutti gli utenti presenti nella stessa stanza in tempo reale.
  - Il sistema deve garantire che le modifiche al timer vengano propagate a tutti gli utenti.
- **Gestione stanze**
  - Il sistema deve permettere la creazione dinamica di stanze univoche.
  - Il sistema deve gestire l'ingresso/uscita degli utenti dalle stanze.
- **Task management**
  - Il sistema deve permettere la creazione, assegnazione e completamento dei task.
  - Il sistema deve tracciare lo stato dei task (in corso/completato).
  - Il sistema deve salvare lo storico dei task completati in modo persistente.

Requisiti non funzionali

- **Usabilità**: l'interfaccia grafica deve essere user-friendly, ovvero intuitiva, semplice da utilizzare e accessibile anche a utenti non esperti.
- **Affidabilità**: l'applicativo non deve presentare crash o blocchi inaspettati ed essere stabile durante l'intero ciclo di utilizzo.
- **Portabilità**: Il sistema è pienamente compatibile con i browser Chrome, Edge e Opera.
- **Documentazione**: deve essere realizzata una documentazione chiara, completa e aggiornata, che faciliti la comprensione del codice e la risoluzione di eventuali problemi.
- **Manutenibilità**: il codice deve essere strutturato in maniera modulare e leggibile, al fine di agevolare interventi di manutenzione, aggiornamento e refactoring.
- **Performance**: il sistema deve essere reattivo alle azioni dell'utente e in grado di sostenere il carico computazionale necessario per lo svolgimento delle sessioni condivise senza ritardi percepibili.

Suitable technologies

Il sistema si basa principalmente sull'integrazione di **Socket.IO**, utilizzato per gestire in tempo reale le funzionalità interattive dell'applicazione, come la creazione e la gestione delle stanze, il funzionamento del timer condiviso, l'assegnazione e l'aggiornamento dei task. Questo consente una comunicazione continua ed efficiente tra client e server, fondamentale per garantire un'esperienza sincronizzata tra gli utenti.

A supporto di queste funzionalità, vengono utilizzate le **REST API** per la gestione delle operazioni meno frequenti ma essenziali, come la registrazione, il login e la visualizzazione dei task completati.

Per quanto riguarda la persistenza dei dati, il sistema si affida a **MongoDB**, che conserva in modo strutturato e sicuro le informazioni relative agli account degli utenti, alle stanze create e ai task portati a termine, assicurando la continuità delle informazioni anche tra diverse sessioni.

0.3 Design

0.3.1 Application architecture

L'architettura adottata per questo progetto è di tipo client-server. Come verrà illustrato più nel dettaglio nei paragrafi successivi, la logica del sito è gestita quasi completamente dal server. Il client si occupa esclusivamente di operazioni locali. Di seguito è riportato uno schema della struttura implementata.

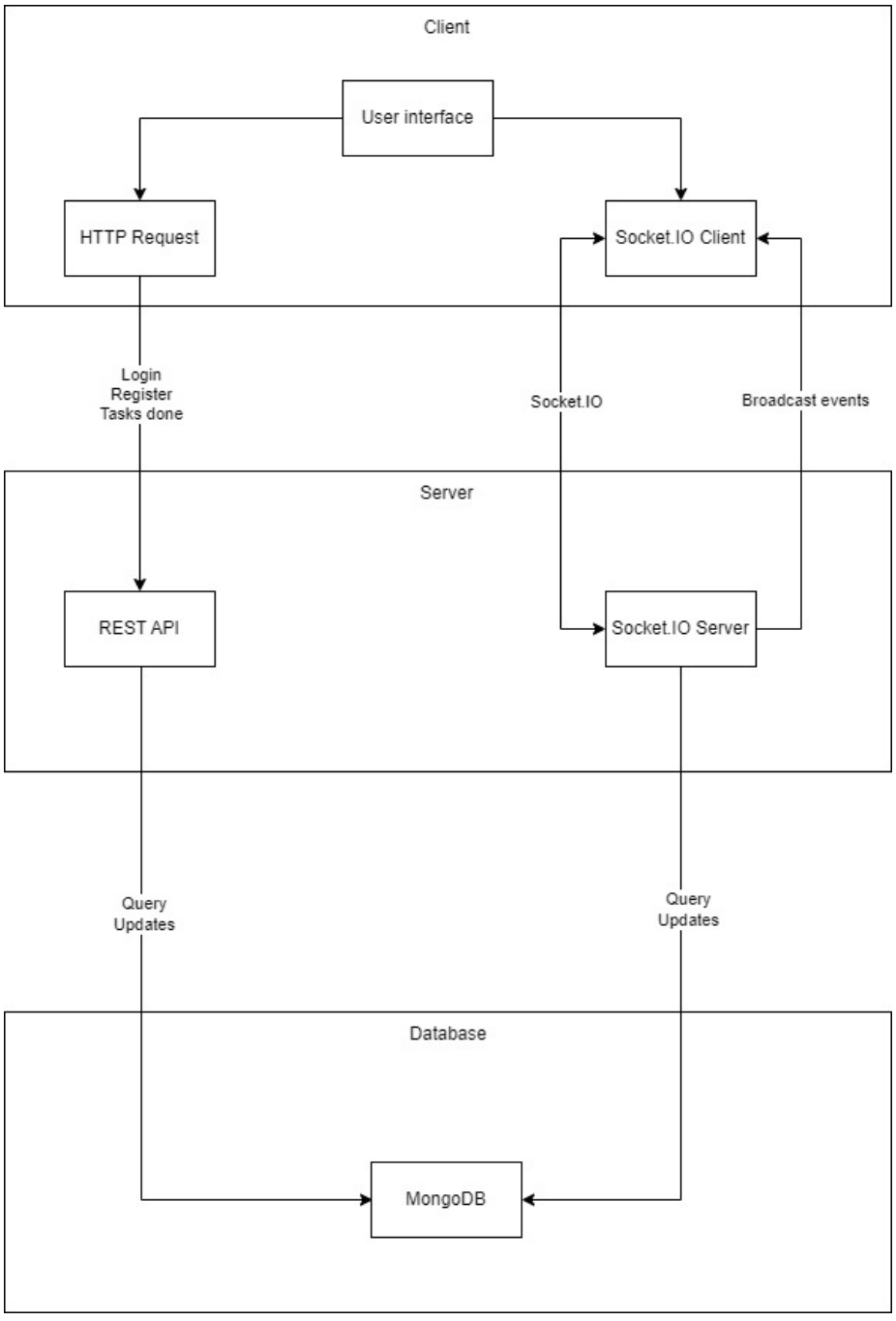

Figura 4: Architettura dell'applicazione

La comunicazione tra client e server è gestita attraverso l'impiego congiunto di **Socket.IO**, per le interazioni in tempo reale, e di **richieste HTTP**, per la gestione delle operazioni sincrone.

Il **client**, tramite l'interfaccia utente, invia richieste HTTP per operazioni come login, registrazione e visualizzazione dei task completati. Le funzionalità legate alla gestione delle stanze — come l'ingresso, le interazioni all'interno di esse e la definizione dei task da completare,... — sono invece gestite tramite **Socket.IO**, che consente una comunicazione bidirezionale in tempo reale con il server.

Dal **lato server**, per ogni client connesso viene creata e mantenuta una socket dedicata, che consente uno scambio di dati continuo e bidirezionale. Le principali modalità di interazione tra client e server sono:

- Il client invia richieste (HTTP o tramite socket) al server.
- Il server elabora le richieste ricevute e restituisce una risposta adeguata.
- In determinati casi, il server può inviare messaggi al client in modo asincrono, sotto forma di notifiche, senza una richiesta esplicita da parte del client.

Parallelamente, il server gestisce anche tutte le richieste HTTP provenienti dal client tramite un set di **API REST**, utilizzate per operazioni sincrone legate alla gestione dell'applicazione, attraverso Socket.IO.

Infine, il server mantiene una comunicazione costante con il **database**, attraverso il quale esegue tutte le operazioni necessarie per la memorizzazione, l'aggiornamento e la consultazione dei dati legati al funzionamento dell'applicazione.

0.3.2 Structure

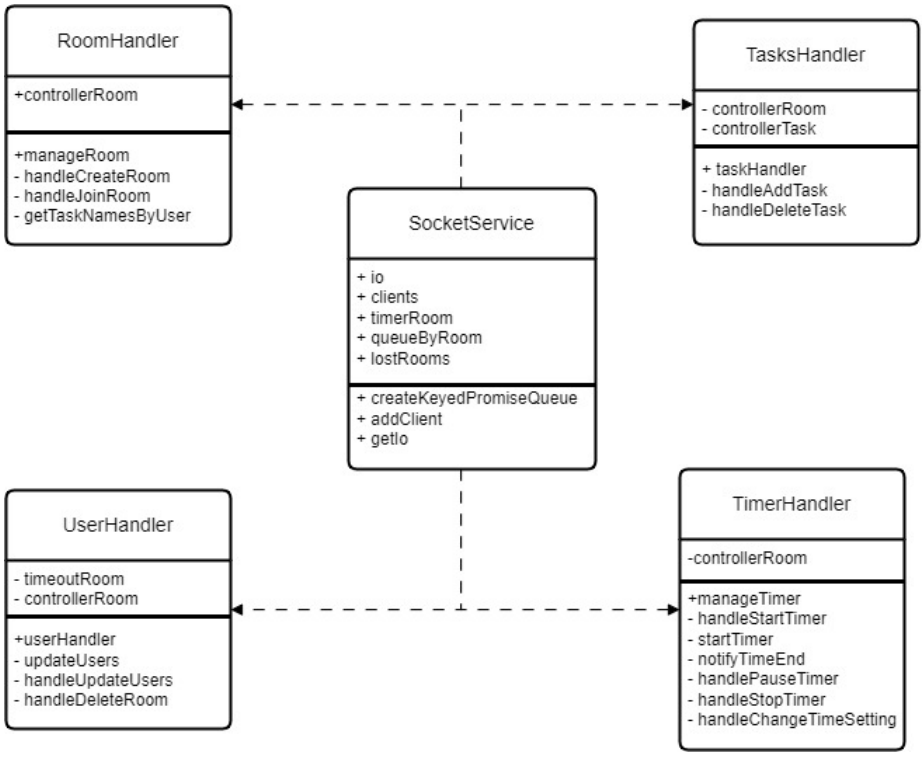

Figura 5: Modello di dominio

SocketService

È il cuore della comunicazione in tempo reale dell'applicazione. Gestisce le connessioni WebSocket (tramite l'attributo `io`, un'istanza di Socket.IO) e mantiene lo stato globale delle stanze attraverso le seguenti strutture dati:

- `timerRoom`: mappa che associa a ogni stanza un oggetto contenente il countdown, il riferimento al timer attivo e lo stato (Pomodoro, pausa lunga o corta).
- `clients`: mappa che per ogni `socketId` identifica l'utente e la stanza di appartenenza.
- `queueByRoom`: struttura per la gestione di code asincrone per ogni stanza.
- `lostRooms`: elenco delle stanze inattive, utile per la loro eventuale rimozione.

Metodi principali:

- `createKeyedPromiseQueue()`: crea code basate su Promise per gestire in ordine eventi concorrenti.
- `addClient()`: aggiunge un nuovo client alla mappa `clients`.
- `getIo()`: restituisce l'istanza dell'oggetto Socket.IO.

RoomHandler (Gestione Stanze)

Gestisce la creazione e l'accesso alle stanze virtuali. Si interfaccia con il `controllerRoom`, che contiene la logica di accesso ai dati.
Metodi principali:

- `manageRoom()`: punto d'ingresso generale per la gestione delle stanze.
- `handleCreateRoom()`: crea una nuova stanza e vi aggiunge l'utente, se la stanza non esiste.
- `handleJoinRoom()`: permette a un utente di unirsi a una stanza esistente.
- `getTaskNamesByUser()`: restituisce l'elenco dei task assegnati a un determinato utente.

Interazioni: comunica con `SocketService` per propagare gli aggiornamenti relativi alla gestione delle stanze.

TasksHandler (Gestione Attività)

Gestisce i task condivisi nelle stanze. Si interfaccia con `controllerRoom` e `controllerTask`.
Metodi principali:

- `taskHandler()`: punto d'ingresso generale per la gestione dei task.
- `handleAddTask()`: aggiunge un nuovo task per uno o più utenti nella stanza.
- `handleDeleteTask()`: elimina un task esistente.

Interazioni: usa `SocketService` per notificare in tempo reale le modifiche ai task.

UserHandler (Gestione Utenti)

Gestisce la presenza e lo stato degli utenti all'interno delle stanze.
Metodi principali:

- `userHandler()`: punto d'ingresso per la gestione degli eventi utente.
- `handleUpdateUsers()`: aggiorna la lista degli utenti nella stanza.

- `handleDeleteRoom()`: rimuove la stanza dopo un timeout, se non ci sono più utenti.

Interazioni: utilizza `SocketService` per sincronizzare lo stato degli utenti tra tutti i client connessi.

TimerHandler (Gestione Timer)

Gestisce i timer condivisi tra gli utenti di una stanza, fondamentali per l'applicazione della tecnica del Pomodoro. Utilizza `controllerRoom` per recuperare e modificare la configurazione.
Metodi principali:

- `manageTimer()`: punto d'ingresso generale per la gestione dei timer.
- `handleStartTime()`: avvia o riprende un timer nella stanza.
- `notifyTimeEnd()`: segnala la fine del tempo e la transizione alla fase successiva.
- `handlePauseTimer()` / `handleStopTimer()`: mettono in pausa o fermano il timer attivo.
- `handleChangeTimeSetting()`: aggiorna le durate configurate per lavoro e pause, propagandole ai client.

Interazioni: sfrutta `SocketService` per inviare aggiornamenti in tempo reale sullo stato del timer a tutti gli utenti nella stanza.

0.3.3 Behaviour

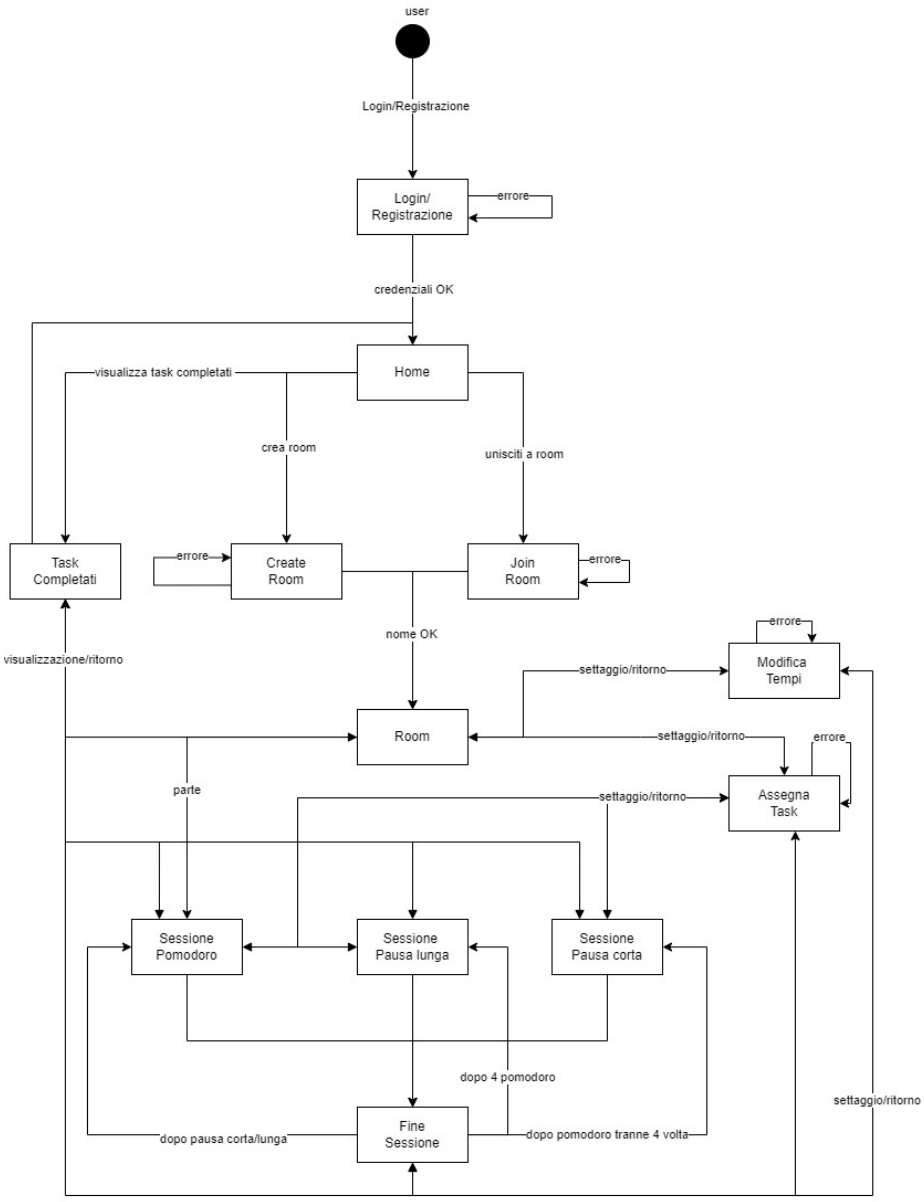

Figura 6: Diagramma degli stati

L'esperienza utente inizia con una fase di **Login/Registrazione**, indispensabile per accedere al sistema. In caso di successo, l'utente viene indirizzato alla scher-

mata **Home**, che funge da snodo principale. Da qui, è possibile intraprendere diverse azioni:

- **Visualizzare i Task Completati**: un'area dedicata al resoconto delle attività portate a termine.
- **Creare una nuova sessione di lavoro condivisa** tramite la funzione *Create Room*.
- **Unirsi a una sessione esistente** attraverso *Join Room*.

Eventuali errori durante l'autenticazione o la gestione delle stanze riportano l'utente al passaggio precedente per una nuova immissione dei dati.
Una volta creata o raggiunta una **Room**, l'utente si trova nell'ambiente di lavoro effettivo. Da questo stato centrale, è possibile:

- **Avviare la sessione di lavoro**, che conduce alla **Sessione Pomodoro**.
- Accedere alla sezione di configurazione per **Modificare i Tempi**, cosi da personalizzare la durata degli intervalli di lavoro e pausa.
- Utilizzare la funzione **Assegna Task**, per definire o gestire i compiti specifici per gli utenti.

Queste sezioni di configurazione permettono poi di ritornare alla *Room*, cosi da continuare la navigazione.
Il cuore dell'applicazione risiede nel ciclo **Pomodoro**. Una *Sessione Pomodoro* rappresenta un intervallo di lavoro concentrato. La logica del ciclo prevede le seguenti transizioni:

- Al termine di una *Sessione Pomodoro* (se non è la quarta di un ciclo), l'applicazione transita automaticamente a una **Sessione Pausa corta**.
- Dopo quattro *Pomodori*, invece, è prevista una **Sessione Pausa lunga** per un recupero più esteso.
- Sia le pause corte che quelle lunghe riconducono l'utente a una nuova *Sessione Pomodoro*.

Al termine di ogni sessione di lavoro o pausa, si può giungere alla **Fine Sessione**. Da questo stato, è possibile, ad esempio, accedere nuovamente alla visualizzazione dei *Task Completati* oppure modificare i tempi.

Nota: non è possibile modificare i tempi durante una sessione attiva.

0.3.4 Interaction

Approccio REST

L'autenticazione degli utenti, inclusa la registrazione e il login, così come la visualizzazione dei task completati, sono gestite tramite un'architettura RESTful. In questo sistema, il client interagisce con il server inviando richieste HTTP, le quali sono elaborate utilizzando il framework Express.js.

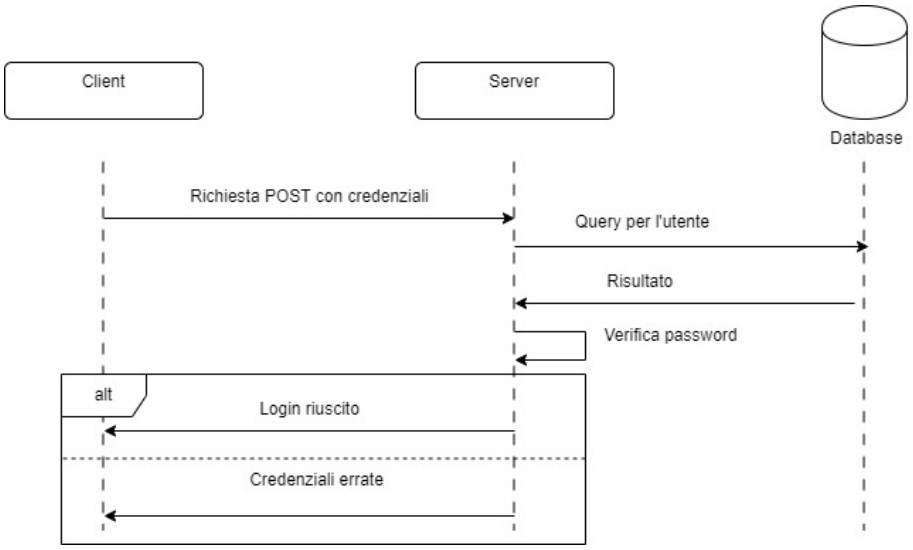

Figura 7: Login utente

Questo diagramma descrive il processo di login di un utente. Il Client invia una "Richiesta POST con credenziali" al Server. Il Server interroga il Database per autenticare l'utente e, una volta ottenuto il risultato, verifica la password. Se il login ha successo, il Client riceve una risposta positiva (200) e le sue informazioni(nome e cognome). Altrimenti, viene inviato il messaggio "Credenziali errate" al Client.

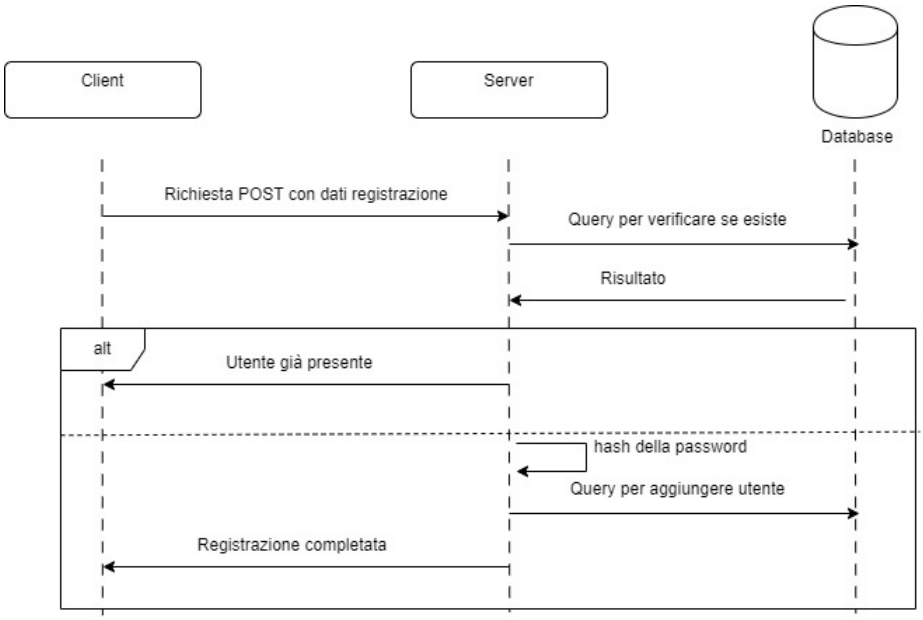

Figura 8: Registrazione utente

Questo diagramma, invece, illustra il processo di registrazione di un nuovo utente. Il Client invia una richiesta POST con i dati di registrazione al Server. Il Server verifica nel Database se l'utente è già presente. Se sì, il Client riceve un messaggio di "Utente già presente". Altrimenti, il Server esegue l'hashing della password e aggiunge il nuovo utente al Database, inviando una risposta positiva(201) con il messaggio "Registrazione completata" al Client.

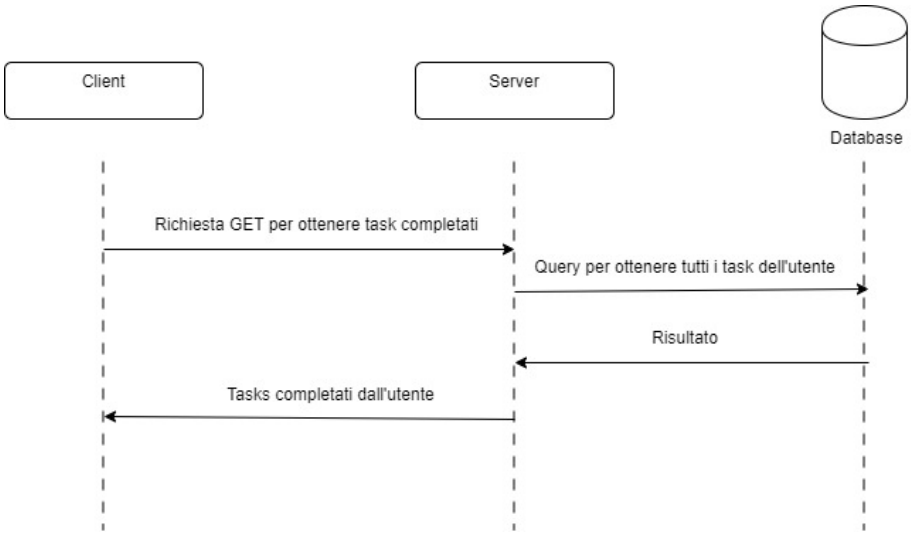

Figura 9: Visualizzazione task completati

Questo diagramma mostra come un client visualizza i task completati durante le sessioni nelle stanze. Il Client invia una richiesta GET per ottenere i task completati al Server. Il Server esegue una query sul Database per recuperare tutti i task completati dall'utente, e una volta ottenuto il risultato, invia la lista con tutti i task completati dal Client.

Approccio con Socket

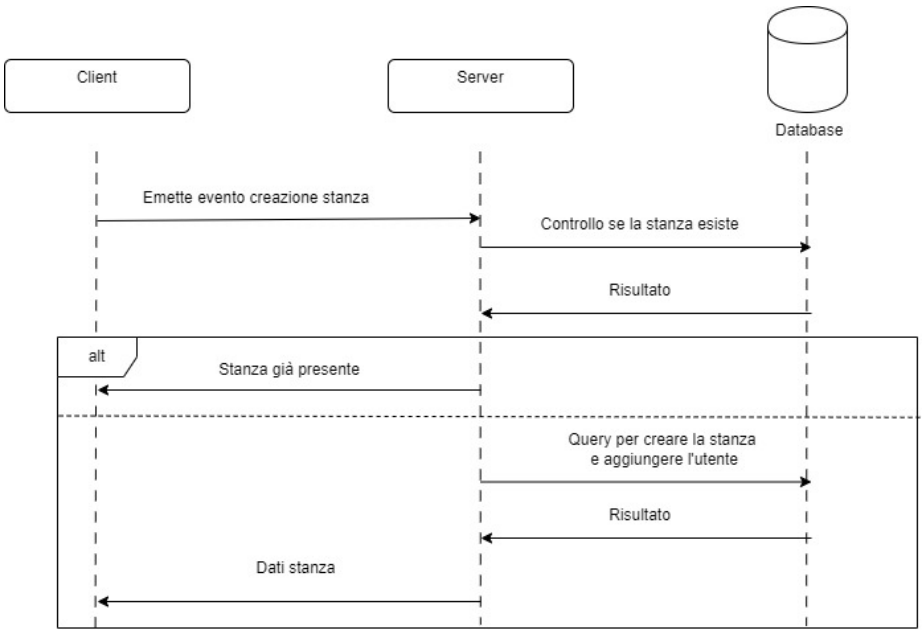

Figura 10: Gestione creazione stanza

Questo diagramma illustra il processo di creazione di una stanza. Il Client emette un evento di "creazione stanza" al Server. Il Server controlla nel Database se la stanza esiste già. Se presente, il Client riceve una callback con risposta negativa(stanza già presente). Altrimenti, il Server crea la stanza e aggiunge l'utente, restituendo i dati(lista utenti, tempistichee altre configurazioni iniziali) della stanza al Client.

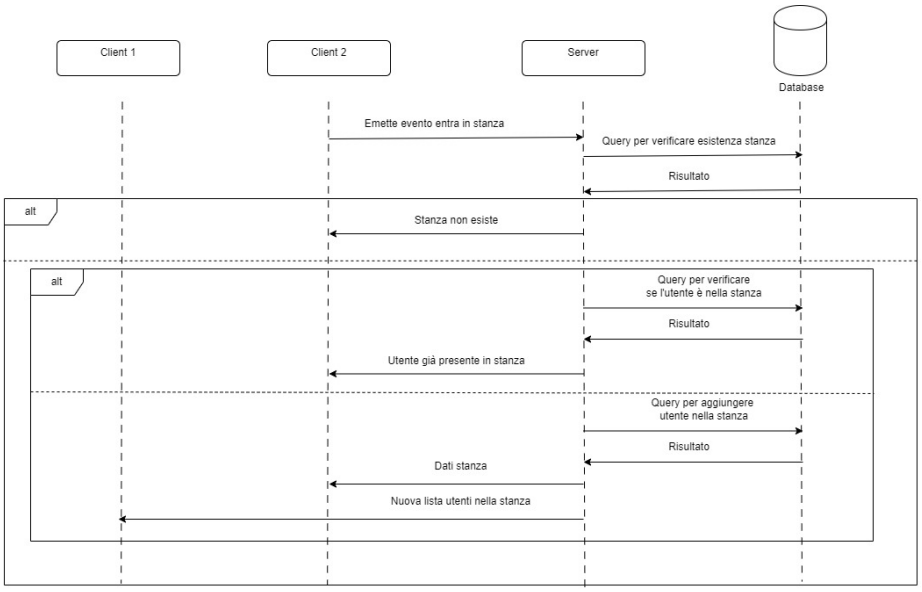

Figura 11: Gestione unione a stanza

il diagramma sopra descrive il processo di unione di un client a una stanza. Il Client emette un evento "entra in stanza". Il Server verifica l'esistenza della stanza nel Database. Se la stanza non esiste, il Client viene notificato attraverso una callback. Se la stanza esiste, il Server verifica se l'utente è già presente; in tal caso, il Client viene avvisato del problema sempre attraverso una callback. Altrimenti, l'utente viene aggiunto alla stanza, e il Server invia i dati della stanza al nuovo client e la nuova lista utenti a tutti i client della stanza.

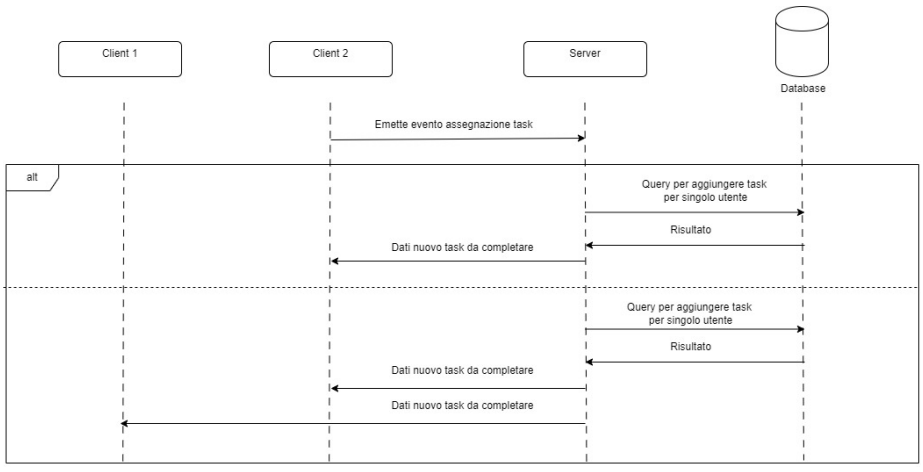

Figura 12: Gestione assegnazione task

Il diagramma illustra il processo di assegnazione di un task. Un client (Client 2) invia al server un evento di "assegnazione task". Il server salva il task nel database e, a seconda del destinatario, lo assegna al client specifico (Client 1, Client 2) oppure a tutti gli utenti presenti nella stanza. Successivamente, il task viene inviato al singolo utente o distribuito a tutti i client dentro la stanza.

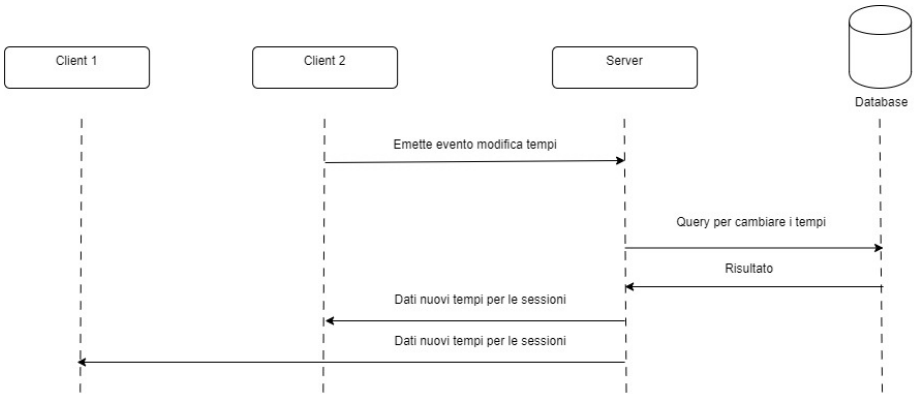

Figura 13: Gestione modifica tempistiche

Questo diagramma descrive la modifica dei tempi associati a una stanza. Un Client emette un evento di "modifica tempi" al Server. Il Server esegue una query sul Database per aggiornare i tempi e, una volta ottenuto il risultato, invia le nuove tempistiche per ogni sessione a tutti i Client connessi alla stanza.

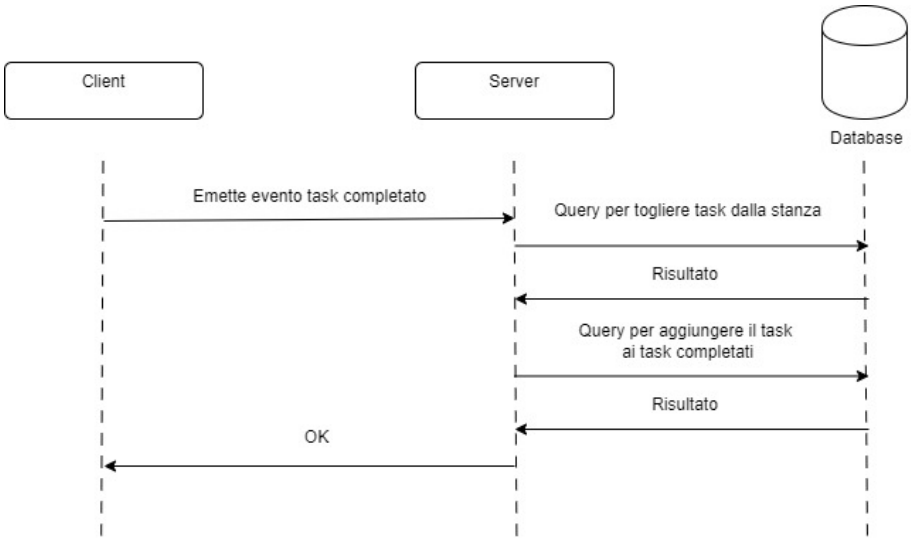

Figura 14: Gestione completamento task

Questo diagramma illustra il flusso per la gestione di un task completato. Il Client emette un evento di "task completato" al Server. Il Server esegue una query per rimuovere il task dalla stanza e un'altra query per aggiungerlo all'elenco dei task completati nel Database, rispondendo con un "OK" al Client una volta finita con successo l'operazione.

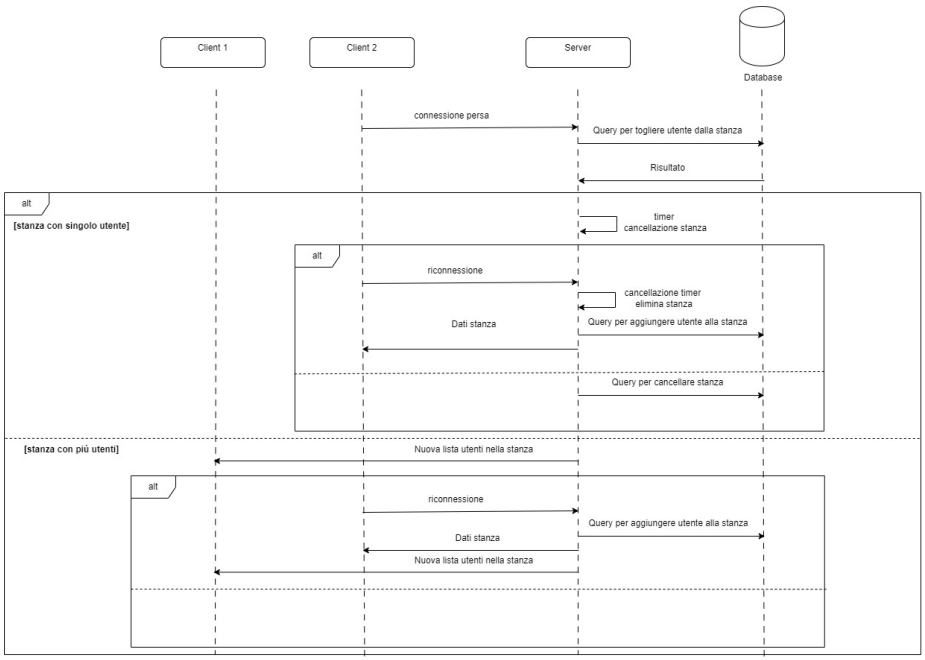

Figura 15: Gestione disconnessione client

Questo diagramma di sequenza mostra come viene gestita la disconnessione involontaria o meno di un client. In particolare vengono mostrate due casistiche:

- **Stanza con singolo utente**: In caso di perdita di connessione, il Server rimuove l'utente dalla stanza e avvia un timer per la cancellazione della stanza. Se il client si riconnette, il timer viene annullato e l'utente viene aggiunto alla stanza e gli vengono notificati nuovamente i dati della stanza. Altrimenti, Se la stanza non è più necessaria (risulta essere ancora vuota), viene cancellata. Se l'utente dovesse ritornare online dopo la cancellazzione della stanza, sarebbe reindirizzato alla homepage, subito dopo il messaggio di riconessione stabilita con il server, nel caso abbia avuto un problema di rete momentaneo.
- **Stanza con più utenti**: come nel caso precedente, il Server rimuove l'utente dalla stanza. Successivamente notifica ai client restanti la nuova lista di utenti online. Se poi il client si riconnette, viene aggiunto nuovamente alla stanza e riceve i dati della stanza, e tutti i client nella stanza

ricevono la nuova lista di utenti connessi. Nel caso in cui invece l'utente non si riconnette non succede nulla.

Gestione interruzione server

In caso di arresto del server, tutti gli utenti riceveranno una notifica dell'interruzione. Al momento del riavvio, verranno avvisati e reindirizzati automaticamente alla homepage. Tuttavia, i dati relativi alle stanze andranno persi.

0.3.5 Software architecture

L'architettura adottata combina due paradigmi fondamentali: il **pattern MVC (Model–View–Controller)** e il **modello event-driven**. Questa scelta consente di organizzare il sistema in modo modulare e scalabile, facilitando sia lo sviluppo di API RESTful sia la gestione della comunicazione in tempo reale.

Il pattern **MVC** (Model–View–Controller) è impiegato principalmente per strutturare l'interfaccia REST del sistema. In questo contesto:

- Il **Model** rappresenta la struttura dei dati e gestisce l'accesso al database.
- Il **Controller** contiene la logica applicativa, elaborando le richieste e interagendo con i dati.
- Le **Route** fungono da punto di ingresso, indirizzando le richieste HTTP ai controller appropriati.

Questo approccio consente di mantenere una chiara separazione delle responsabilità tra la gestione dei dati, la logica di business e la definizione degli endpoint, migliorando la leggibilità e la manutenibilità del codice.

Parallelamente, per la gestione delle comunicazioni asincrone e in tempo reale, è stato adottato un modello **event-driven**. In questo paradigma, le interazioni tra client e server si basano su eventi: il server ascolta determinati eventi emessi dal client e reagisce eseguendo determinate azioni, spesso rispondendo con ulteriori eventi.

La logica che gestisce questi eventi è integrata con i moduli del sistema MVC, riutilizzando sia i modelli per l'accesso ai dati, sia eventuali controller per la logica. Questo consente una coerenza architetturale tra l'API sincrona (REST) e quella asincrona (WebSocket).

0.4 Implementation Details

L'applicazione è sviluppata secondo un'architettura **client-server** con comunicazioni in tempo reale, basata sullo stack **MEVN** (MongoDB, Express.js, Vue.js, Node.js).

- **Database (MongoDB)**: gestisce in modo sicuro e strutturato i dati relativi agli account utente, alle stanze create e ai task completati, garantendo la persistenza delle informazioni tra diverse sessioni. L'interazione con il database avviene tramite **Mongoose**, che offre un'interfaccia *schema-based* per MongoDB.
- **Backend (Node.js + Express.js)**:
  - Espone **API REST** per operazioni sincrone meno frequenti ma fondamentali, come registrazione, login e visualizzazione dei task completati. La comunicazione con il frontend avviene attraverso **Axios**.
  - Utilizza **Socket.IO** per gestire in tempo reale le funzionalità interattive, tra cui la creazione e il controllo delle stanze, la sincronizzazione di un timer condiviso e l'assegnazione o l'aggiornamento dei task.
- **Frontend (Vue.js)**: organizzato in maniera modulare, impiega componenti sviluppati con la **Composition API** e utilizza **Vue Router** per la gestione della navigazione tra le varie viste.

0.4.1 Altre tecnologie

- **Docker** è una tecnologia ampiamente adottata per la containerizzazione e la distribuzione di sistemi software. All'interno del progetto è stato utilizzato per creare un'istanza del database MongoDB, frontend e backend in modo semplice ed efficiente, sfruttando il comando docker compose. Questo approccio ha facilitato la configurazione e il deploy dell'ambiente di sviluppo, garantendo portabilità e coerenza tra le diverse installazioni.
- **bcryptjs**: utilizzato per eseguire l'hashing sicuro delle password.
- **cors**: gestisce le richieste cross-origin, permettendo di configurare CORS in ambienti Express.
- **mongoose**: fornisce un'interfaccia ad oggetti per la gestione e modellazione dei dati su MongoDB.
- **Axios**: client HTTP basato su promise, utilizzato per effettuare chiamate verso il backend.

0.5 Self-assessment / Validation

0.5.1 Test automatici

Il testing è stato eseguito tramite test automatici utilizzando il framework Jest. È stato implementato un sistema di test completo e affidabile per l'applicazione, verificando che gli endpoint rispondessero correttamente alle richieste e restituissero i dati attesi, nonché la gestione adeguata degli errori. Tutti i componenti testati sono stati:

- **Autenticazione** Vengono testate le API di login e registrazione dell'applicazione. Verifica che la registrazione con dati validi avvenga correttamente e che venga restituito un errore se l'email è già registrata. Inoltre, controlla che il login fallisca con credenziali errate, assicurando la gestione appropriata degli errori di autenticazione.
- **Stanza**: Viene verificato il corretto funzionamento delle connessioni Socket.IO. Vengono testate la connessione e disconnessione di un singolo client, la gestione di un numero elevato di utenti, la creazione e gestione delle stanze virtuali, nonché il comportamento del server in caso di errori, come la creazione di stanze duplicate o l'accesso a stanze inesistenti.
- **Task**: Viene verificata la corretta gestione dei task all'interno delle stanze virtuali tramite Socket.IO. Si controlla che i task vengano assegnati correttamente a tutti gli utenti o a uno specifico, e che la rimozione dei task al completamento avvenga come previsto.
- **Timer**: Viene verificato il corretto funzionamento del sistema di timer condiviso tra utenti in una stanza virtuale. Vengono testate la creazione delle stanze con impostazioni di tempo predefinite, la modifica dei tempi, l'avvio, la pausa e la fine del timer, nonché il passaggio automatico tra le diverse fasi del ciclo di lavoro (es. pausa breve dopo il lavoro).

Per eseguire i test, seguire i seguenti passaggi:

1. Seguire il procedimento definito nel sezione deployment per inizializzare i componenti
2. Successivamente, spostarsi nelle cartella `server` del progetto;
3. Infine, eseguire `npm run test` per avviare i test.

0.5.2 Test manuali

Un campione ristretto di cinque utenti è stato coinvolto in una sessione di verifica dell'utilizzo del sistema. I partecipanti hanno fornito feedback iterativo sulle funzionalità della web app, permettendo così di distinguere gli aspetti consolidati da quelli da rivedere. Gli utenti sono stati assegnati a ruoli diversi e invitati a completare specifici task senza ricevere istruzioni preliminari, con l'obiettivo di

osservare le modalità d'interazione spontanea con l'interfaccia alla loro prima esperienza con il sistema. Gli utenti sono stati divisi in due categorie. Gli utenti con maggiore dimestichezza con la tecnica del pomodoro e quelli che non l'hanno mai utilizzata.

- `Utenti esperti`: Gli è stato richiesto di creare sessioni Pomodoro di gruppo, configurare i timer condivisi, assegnare task agli altri partecipanti. Queste azioni hanno permesso di valutare la semplicità d'uso, l'efficacia del sistema per la gestione centralizzata del lavoro e la componente personalizzazione.
- `Utenti base`: hanno agito come collaboratori, con il compito di partecipare attivamente alle sessioni Pomodoro, rispettando i cicli di lavoro e pausa proposti. Dovevano portare a termine i task loro assegnati e, in alcuni casi, proporne di nuovi da affidare ai compagni di lavoro. È stato chiesto loro di esplorare liberamente la dashboard per visualizzare i propri progressi.

L'attenzione è stata posta sull'affidabilità del sistema nel mantenere la sincronizzazione tra i timer dei vari partecipanti e sulla capacità del sistema di propagare correttamente gli aggiornamenti tra i nodi. È stata inoltre analizzata la gestione di eventuali conflitti, ad esempio nel caso in cui più utenti cercassero di modificare simultaneamente i task o lo stato della sessione. Infine, ma non per importanza, si è posta l'attenzione in eventuali bug.

Altri Test manuali

Disconnessione del Client per Problema di Rete

Per simulare una disconnessione a causa di un problema di rete, è stato utilizzato il browser in modalità Developer Tools. I passi seguiti sono stati i seguenti:

1. Aprire il browser in modalità DevTools (Ctrl + Shift + I su Chrome).
2. Ricaricare la pagina (Ctrl + R) per inizializzare una nuova sessione.
3. Accedere alla sezione Network di DevTools e, nella parte inferiore, modificare l'opzione `Network Throttling` da "No throttling" a "Offline".

Una volta applicata questa configurazione, il sistema simula una disconnessione di rete, e al termine del timeout del server, il frontend visualizza la schermata di disconnessione. Per ripristinare la connessione, sarà sufficiente tornare alla modalità "No throttling". Il frontend riprende la sincronizzazione con il server.

Arresto Anomalo del Server

Nel caso dell'arresto anomalo del server, il test è stato effettuato utilizzando l'interfaccia Docker per fermare il server. I passi seguiti sono stati:

1. Utilizzare l'interfaccia Docker per fermare il server.

2. Osservare il comportamento del frontend, che ha mostrato un messaggio di errore, segnalando l'impossibilità di comunicare con il server.

3. Dopo aver riavviato il server tramite Docker, il messaggio di errore nel frontend scomparirà.

Nel sezione Design sono stati dettagliati i comportamenti attesi in entrambi gli scenari.

0.6 Deployment Instructions

Esecuzione in locale con Docker

È possibile avviare l'applicazione **SharedPomodoro** in locale utilizzando `Docker`, seguendo i passaggi descritti di seguito:

1. Clonare il repository e spostarsi all'interno della cartella del progetto

2. Avviare il container tramite Docker Compose:

   ```
   docker-compose up --build
   ```

   Questo comando provvederà a creare e avviare i seguenti container:

   - **mongo**: database MongoDB, in ascolto sulla porta `27017`.
   - **server**: backend Node/Express, accessibile sulla porta `3000`.
   - **client**: frontend Vue, accessibile sulla porta `5173`.

3. Una volta avviati i container, è possibile accedere all'applicazione aprendo il browser all'indirizzo: `http://localhost:5173`

0.7 Usage Examples

Creazione/Unione a una stanza

Al primo accesso, l'utente viene accolto da una schermata di login, dove può inserire le proprie credenziali (email e password) per autenticarsi nel sistema, oppure puo scegliere di registrarsi.

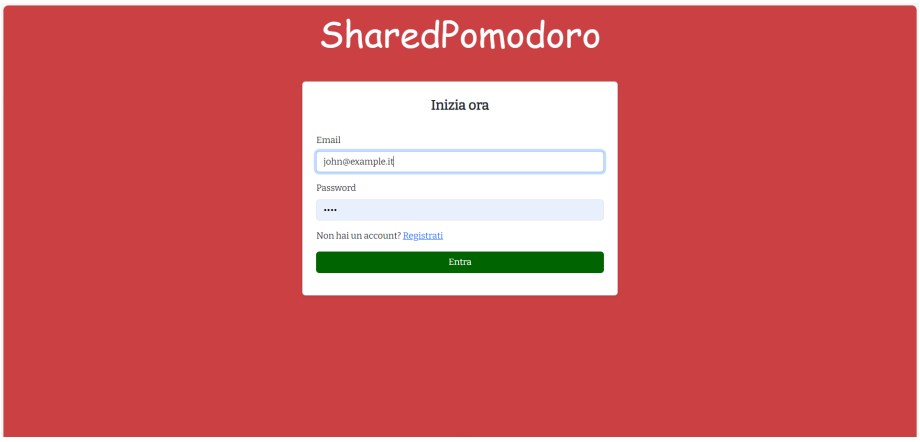

Figura 16: Rappresenta la schermata per fare il login.

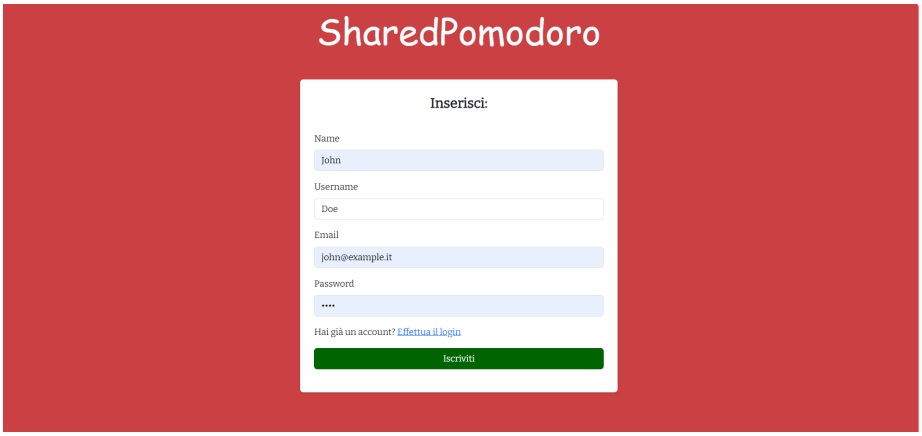

Figura 17: Rappresenta la schermata per effettuare la registrazione

Figura 18: Rappresenta la schermata per effettuare l'accesso alle stanze.

Nell'immagine in alto è raffigurata la schermata, che presenta un campo di input dedicato all'inserimento del nome di una stanza. Nella parte inferiore sono presenti due pulsanti, che consentono rispettivamente di creare una nuova stanza o di accedere a una già esistente. Sulla destra è presente una barra laterale con icone che rappresentano le principali sezioni dell'applicazione: profilo, log out, elenco dei task completati e homepage. Passando con il cursore sopra l'icona del profilo, viene mostrato il nome dell'utente.

Gestione Stanza

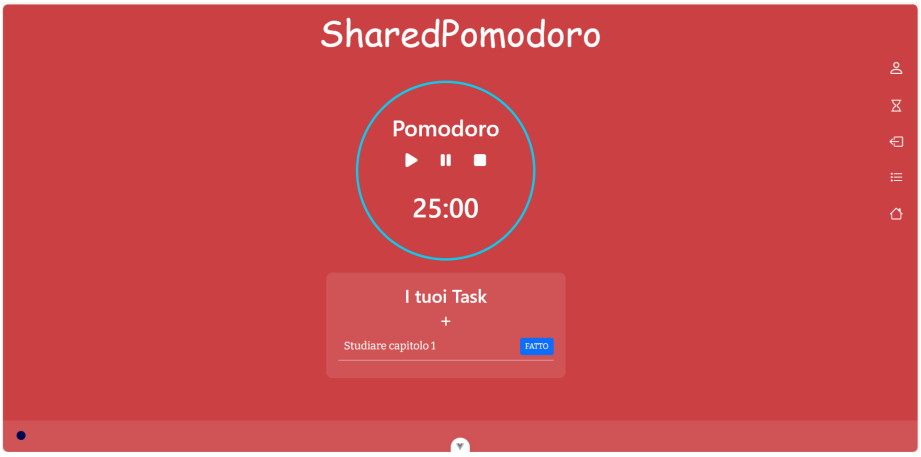

Figura 19: Rappresenta la stanza.

Una volta che l'utente crea o si unisce ad una stanza viene visualizzata questa schermata, al centro è ben visibile un cerchio che rappresenta il timer, affian-

cato dai controlli di avvio, pausa e arresto. Subito sotto, la scritta *"Tasks"* accompagnata da un'icona di aggiunta suggerisce la possibilità di aggiungere le attività. Subito sotto vengono mostrati i task da svolgere. Nella parte inferiore della schermata sono visualizzati gli utenti presenti nella stanza; passando con il cursore sopra ciascun utente, viene mostrato il relativo ID.

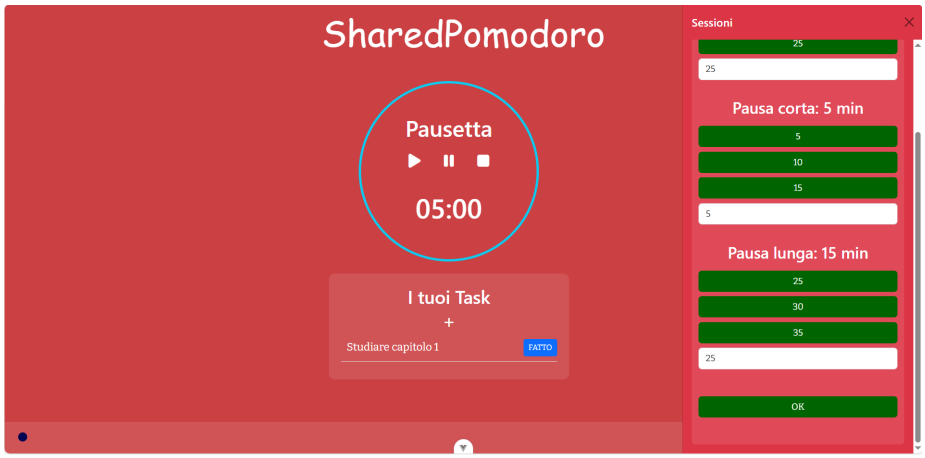

Figura 20: Rappresenta la schermata per modificare i tempi.

Cliccando sull'icona a forma di clessidra, si apre un riquadro laterale sulla destra intitolato *"Sessioni"*, che contiene i campi per impostare la durata del "Pomodoro", della "Pausa lunga" e della "Pausa breve", oltre a un pulsante "OK" per confermare le modifiche. Accanto a ciascuna etichetta di sessione è visualizzato il tempo attualmente valido, al fine di rendere più chiara la personalizzazione dei tempi.

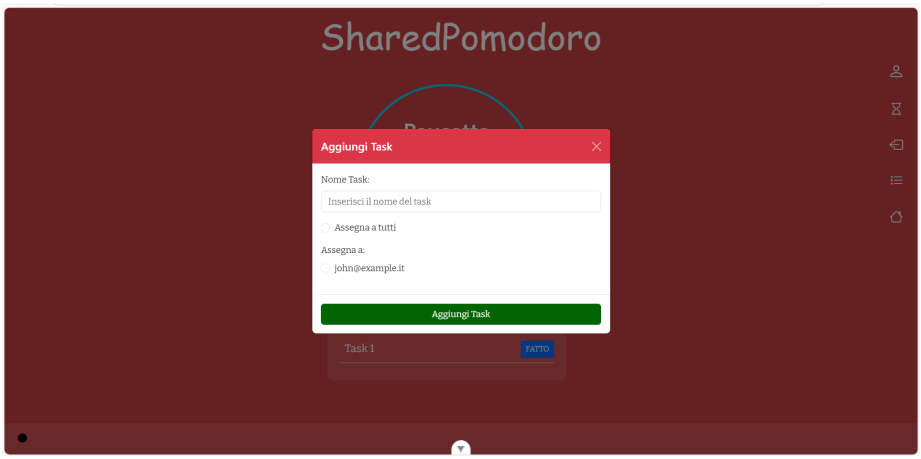

Figura 21: Rappresenta la schermata per aggiungere un task.

Per assegnare un nuovo task bisogna premere il pulsante con il *"+"*. Fatto ciò appare al centro un riquadro per inserire il *"Nome task"* e assegnarlo a uno specifico utente (user 1, user 2, user 3) o a tutti, tramite gli appositi pulsanti. In basso, il pulsante *"Aggiungi task"* consente di salvare l'attività.

Visualizzazione task completati

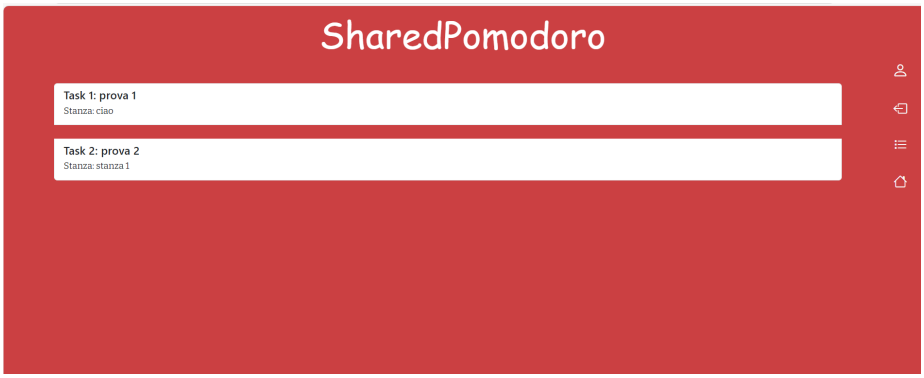

Figura 22: Rappresenta la schermata per visualizzare i task completati dall'utente.

Cliccando l'icona task (la terza, partendo dall'alto) nella barra laterale viene mostrata la sezione dedicata alle attività completate dall'utente. Al centro sono visualizzate delle cards, ciascuna contenente una descrizione con il nome del task e la stanza in cui è stato completato.

0.8 Conclusions

In conclusione, gli obiettivi inizialmente fissati per questo progetto sono stati raggiunti con successo. La realizzazione di un sistema con tanti utenti online ha richiesto l'integrazione di tecnologie moderne e affidabili, tra cui TypeScript, Vue.js, Node.js e WebSocket, che hanno consentito la costruzione di un'infrastruttura scalabile, reattiva e orientata alla comunicazione in tempo reale.

Il sistema sviluppato garantisce un'esperienza utente fluida e coinvolgente, grazie a un'interfaccia intuitiva e a una gestione efficiente delle interazioni tra i partecipanti. Le funzionalità real-time, elemento chiave in un contesto con tanti utenti, sono state implementate con successo, assicurando sincronizzazione, reattività e interattività continua tra i client.

0.8.1 Future Works

Nelle versioni future dell'applicazione è possibile introdurre le seguenti migliorie:

- Aggiunger una chat che permetta agli utenti di aiutarsi durante l'esecuzione dei task o per aggiornarsi semplicemente.
- Dare all'utente la possibilità di visualizzare le stanze in cui è presente.
- Migliorare l'interfaccia grafica in modo tale da garantire una migliore User Experience.

0.8.2 What did i learned

La realizzazione di questo progetto ha rappresentato un'importante occasione per consolidare e applicare in modo pratico i concetti fondamentali affrontati nel corso di Sistemi Distribuiti. In particolare, l'attenzione è stata rivolta all'implementazione di funzionalità real-time e alla gestione della comunicazione tra nodi in un contesto asincrono.

L'integrazione di tecnologie come Socket.io ha permesso di approfondire le dinamiche legate alla comunicazione event-driven e all'aggiornamento in tempo reale dello stato condiviso, aspetti centrali nei sistemi distribuiti moderni. Inoltre, la gestione delle problematiche legate alla sincronizzazione, alla disconnessione temporanea dei nodi e al ripristino delle connessioni ha evidenziato l'importanza di progettare sistemi resilienti e affidabili.