

SharedPomodoro
Applicazioni e Servizi Web

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

Maggio 2025

Indice

0.1	Introduzione	2
0.2	Requisiti	3
0.3	Design	5
0.3.1	Design architutturale	5
0.3.2	Design di dettaglio	6
0.3.3	Mockup	8
0.3.4	Target user analysis	13
0.3.5	Story boards	14
0.4	Tecnologie	18
0.4.1	Altre tecnologie	19
0.5	Codice	20
0.5.1	Backend	20
0.5.2	Frontend	21
0.6	Test	23
0.6.1	Test manuali	23
0.6.2	Test usabilità	23
0.6.3	Test automatici	24
0.7	Deployment	26
0.8	Conclusioni	26

0.1 Introduzione

Introduzione **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.

0.2 Requisiti

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.

0.3 Design

Il progetto è stato sviluppato adottando un approccio **User-Centered Design (UCD)**, centrato sull'utente finale. Sono state create delle **Personas** per comprendere meglio le esigenze degli utenti e definire le funzionalità del sistema. Inoltre, sono stati coinvolti utenti reali per raccogliere feedback diretti, permettendo un continuo affinamento del design in base alle loro preferenze e difficoltà.

Lo sviluppo è stato gestito tramite un approccio **AGILE**, suddiviso in sprint che hanno permesso di implementare e testare progressivamente le funzionalità, seguendo un processo incrementale e mirato.

Durante lo sviluppo, sono stati applicati i principi **KISS (Keep It Simple, Stupid)** per garantire un sistema intuitivo e facile da usare, e **DRY (Don't Repeat Yourself)** per ottenere un codice pulito, riutilizzabile e facilmente manutenibile.

Inoltre, l'interfaccia è stata progettata secondo i principi del **Responsive Design**, per assicurare un'esperienza utente ottimale su desktop e dispositivi mobili, migliorando l'accessibilità dell'applicazione.

0.3.1 Design architetturale

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).

0.3.2 Design di dettaglio

Backend

Il modulo Server gestisce in modo centralizzato tutte le operazioni relative ai client, alle stanze (room), ai task e al timer. Il server viene avviato attraverso il file `index.js`, che svolge le seguenti operazioni fondamentali:

- Creazione di un'istanza di **Express**.
- Connessione al database **MongoDB**, per garantire una gestione persistente dei dati.
- Definizione delle route dell'applicazione.

Le componenti principali del backend includono:

- **index**: gestisce l'avvio del server e la connessione al database.
- **handlers**: contiene la logica per la gestione degli eventi **WebSocket**, che consente una comunicazione asincrona e in tempo reale tra client e server.
- **services**: gestisce lo stato condiviso tra i componenti del server.
- **test**: raccoglie i file di test per garantire il corretto funzionamento e la validità del codice.
- **routes**: definisce le rotte che saranno utilizzate per gestire le richieste **HTTP** provenienti dai client.
- **Model e Controller**: rappresentano le strutture che consentono l'interazione con il database, gestendo la logica e l'accesso ai dati.

Il server è sviluppato in **TypeScript** ed eseguito in ambiente **Node.js**, sfruttando la robustezza di entrambi per ottenere un'applicazione scalabile e facilmente manutenibile. Per garantire una comunicazione in tempo reale tra client e server, è stato adottato il protocollo **WebSocket**, implementato tramite **Socket.io**. Questa scelta consente uno scambio bidirezionale di dati in tempo reale, eliminando la necessità di richieste esplicite da parte dei client, come accade invece nel tradizionale protocollo **HTTP** che è unidirezionale.

Il server gestisce tutte le interazioni, dalla creazione e connessione alle stanze fino alla gestione del timer e dei task. I due principali metodi di comunicazione utilizzati per interagire con i client sono:

- **emit to room**: invia messaggi a tutti i client connessi a una determinata stanza, per comunicare informazioni o eventi.
- **emit to socketId**: utilizzato per inviare messaggi al singolo client, garantendo così la sincronizzazione con il server.

Il server è configurato per ascoltare sulla porta 3000, e implementa rigidi controlli di sicurezza per garantire l'integrità delle operazioni eseguite e prevenire eventuali accessi non autorizzati o manipolazioni indesiderate. La netta separazione tra **Client** e **Server** ha reso il progetto modulare, scalabile e manutenibile. L'utilizzo delle WebSocket ha assicurato comunicazioni in tempo reale tra gli utenti, mentre i test automatici hanno garantito l'affidabilità del sistema.

Autenticazione e Gestione Task

Per la gestione dell'autenticazione e la visualizzazione dei task, l'applicativo prevede due moduli principali:

- **Autenticazione (Auth)**: consente agli utenti di registrarsi, eseguire il login. Per la registrazione, è previsto un **POST** a una specifica rotta che crea un nuovo utente nel sistema. Durante il login, l'utente invia le proprie credenziali tramite una richiesta **POST**.
- **Gestione dei Task (Tasks)**: i task rappresentano le attività che gli utenti possono creare, assegnare e completare. La visualizzazione dei task effettuati è resa possibile tramite una richiesta **POST**.

Frontend

Il modulo Client è responsabile della gestione dell'interfaccia utente e delle interazioni visibili all'utente. Sviluppato utilizzando TypeScript e Vue.js, il client comunica in tempo reale con il backend tramite WebSocket, grazie all'utilizzo di Socket.io. Inoltre, per le operazioni di login, registrazione e gestione dei task effettuati, è stato impiegato Axios per gestire le richieste HTTP in modo semplice ed efficace.

Le principali directory del frontend sono organizzate come segue:

- **components**: contiene i componenti Vue.js da renderizzare nell'interfaccia utente.
- **routes**: definizione delle rotte dell'applicazione.
- **style**: contiene i fogli di stile CSS per gestire l'aspetto grafico dell'applicazione.

Per quanto riguarda la gestione delle comunicazioni tramite WebSocket, il client utilizza i seguenti metodi fondamentali:

- **socket.emit(eventName, data)**: invia un evento con i dati associati al server.
- **socket.on(eventName, callback)**: riceve eventi dal server e attiva una funzione di callback associata all'evento.

Percorsi dell'applicazione

L'interfaccia utente è strutturata in diverse schermate, ognuna accessibile tramite le seguenti rotte:

- `/`: pagina iniziale di login.
- `/register`: pagina per la registrazione dell'utente.
- `/enterRoom`: pagina principale che offre le opzioni per creare una nuova stanza o entrare in una già esistente.
- `/room`: stanza in cui l'utente è attualmente presente, per interagire con altri partecipanti.
- `/tasksDone`: schermata che mostra i task completati dall'utente.

0.3.3 Mockup

Per la progettazione dell'interfaccia, si è puntato a uno stile semplice e intuitivo. Inizialmente sono stati creati mockup con Draw.io per visualizzare il design finale. L'app è ottimizzata per l'uso su desktop, dove offre la migliore esperienza utente e una maggiore produttività. Sono stati comunque fortemente considerati anche tablet e dispositivi mobili per garantire una fruizione soddisfacente su tutte le piattaforme. L'interfaccia finale è il risultato di primi sketch e feedback degli utenti.

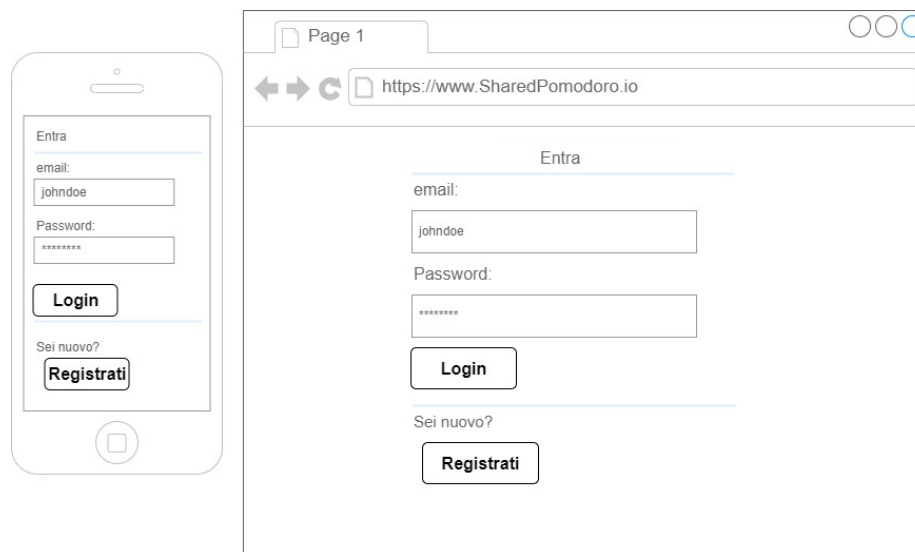


Figura 1: Rappresenta il login.

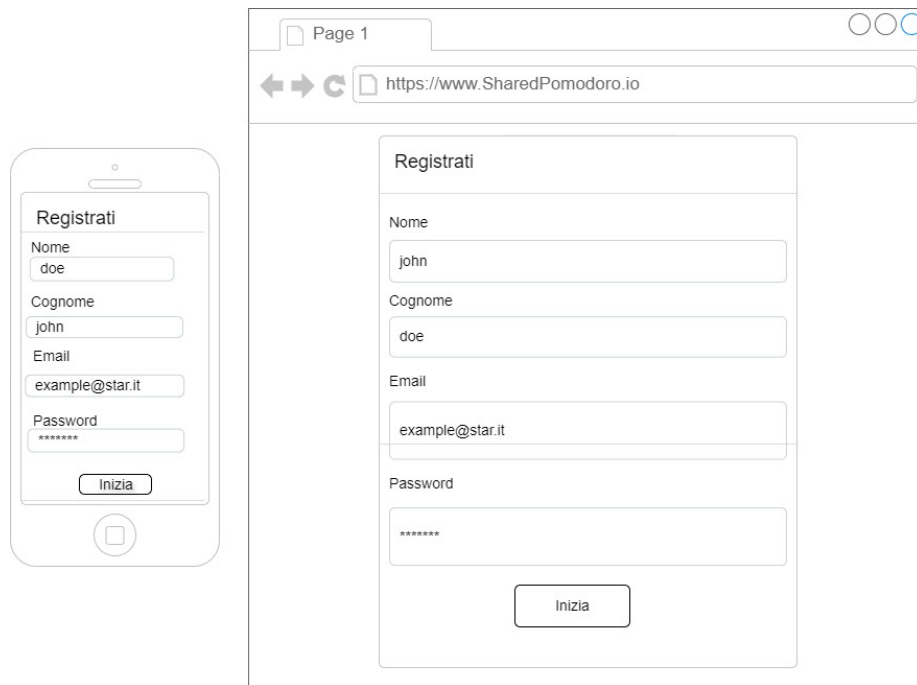


Figura 2: Rappresenta la registrazione.

Il sito si apre con un'interfaccia minimalista e funzionalità essenziali, invitando l'utente a fare il login o la registrazione. Dopo questa operazione, l'utente viene reindirizzato alla dashboard principale, che si presenta nel modo seguente:

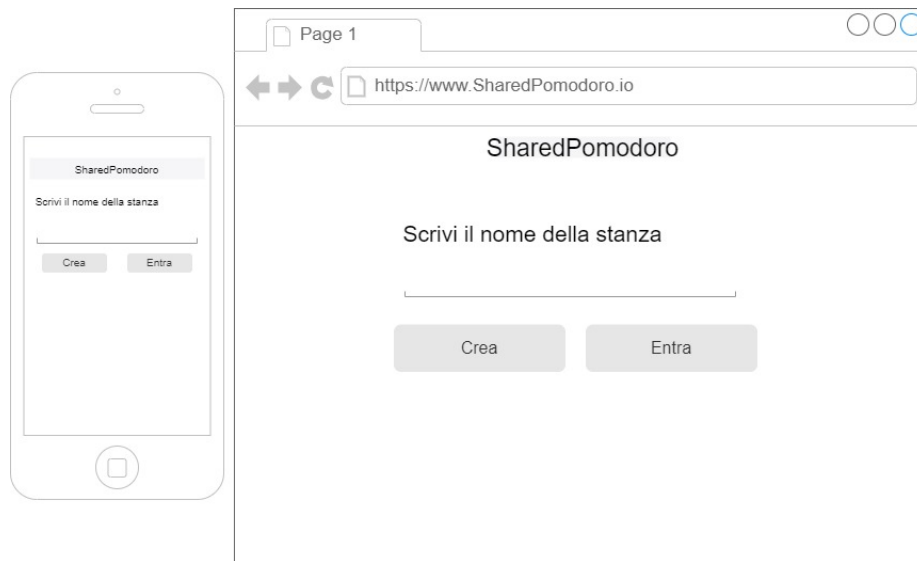


Figura 3: Rappresenta l'accesso alle stanze.

Il *mockup* della home page presenta un design pulito e minimalista, caratterizzato da un'interfaccia intuitiva progettata per guidare l'utente verso le funzionalità principali. Nella parte superiore della schermata è presente un campo di input in cui l'utente può inserire il nome di una stanza. Nella sezione inferiore, due pulsanti ben visibili e disposti orizzontalmente offrono le seguenti opzioni:

- **Entra:** consente di accedere a una stanza già esistente.
- **Crea:** permette di generare una nuova stanza.

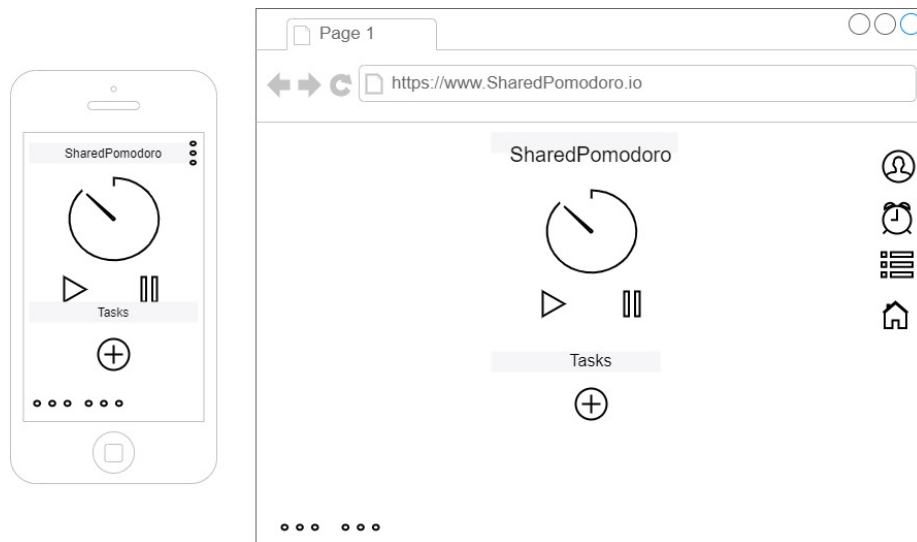


Figura 4: Rappresenta la stanza.

Al centro spicca un *timer* circolare, affiancato da controlli di riproduzione e pausa, suggerendo la funzionalità principale dell'applicazione. Sotto, la parola "Tasks" è presente un bottone per l'aggiunta. Una barra laterale a destra presenta icone per il profilo, timer, lista e homepage, delineando le sezioni dell'applicazione. Nella parte inferiore sono visualizzati gli utenti all'interno della stanza.

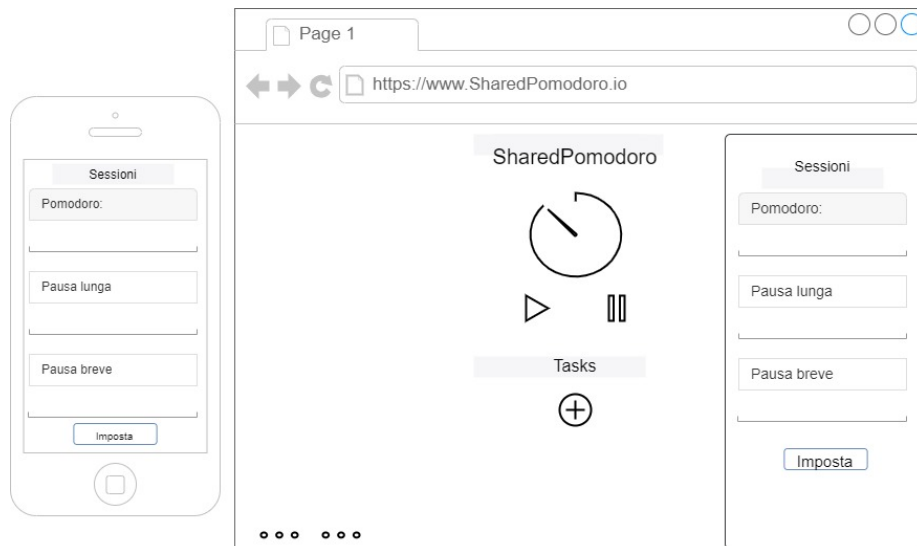


Figura 5: Rappresenta la schermata per modificare i tempi.

A destra compare un riquadro *"Sessioni"* con campi per il settaggio della durata del "Pomodoro", della "Pausa lunga" e della "Pausa breve", insieme a un pulsante "Imposta" per personalizzare i tempi.

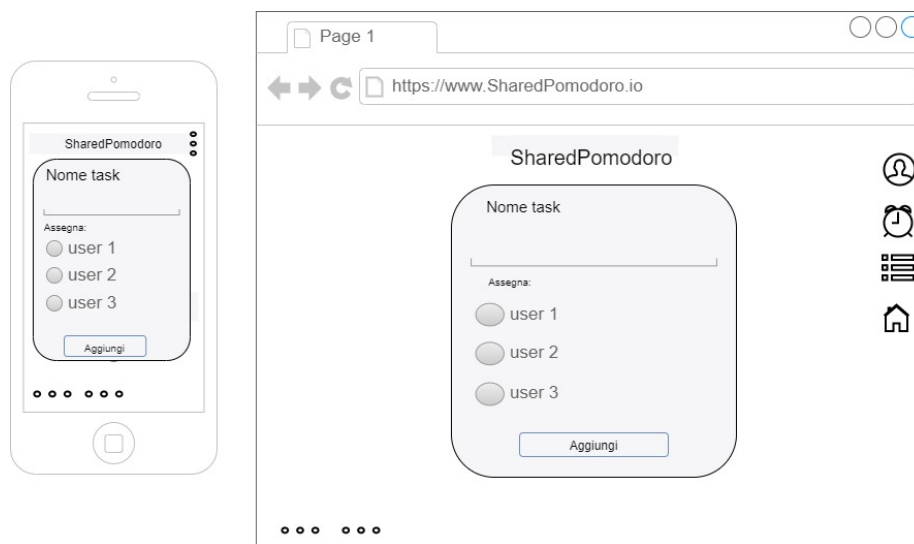


Figura 6: Rappresenta la schermata per aggiungere un task.

Al centro appare un riquadro per inserire il *"Nome task"* e per assegnare l'attività a uno tra "user 1", "user 2" e "user 3", con i relativi pulsanti. Il pulsante *"Aggiungi"* in basso indica l'azione di salvare la nuova attività.

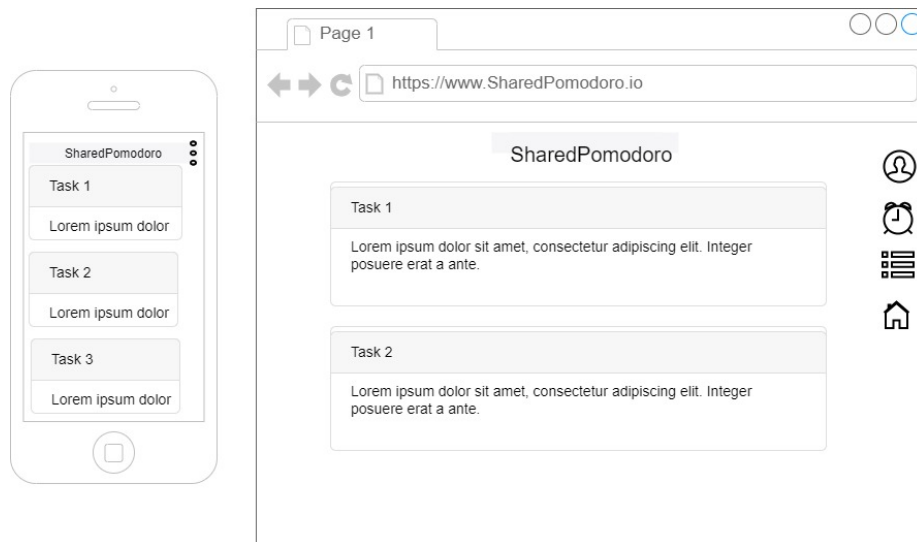


Figura 7: Rappresenta la schermata per visualizzare i task completati dall'utente.

Infine viene mostrata la sezione dedicata alle attività completate dall'utente. Al centro sono elencati i tasks, ognuno accompagnato da un testo descrittivo.

0.3.4 Target user analysis

L'applicativo si rivolge a studenti, lavoratori e gruppi di persone che desiderano aumentare la propria produttività tramite la tecnica del Pomodoro. Il focus principale è sulla collaborazione sincrona, ovvero l'utilizzo di un timer condiviso per coordinare sessioni di lavoro o studio con altri utenti, anche a distanza.

Laura (Mamma Lavoratrice)

Laura lavora da casa e ha un'agenda molto impegnata. Utilizza il sito per sfruttare al meglio brevi momenti liberi tra lavoro e famiglia. La tecnica del Pomodoro le consente di restare concentrata anche in sessioni brevi. Apprezza un'interfaccia semplice e la possibilità di personalizzare i tempi. Il suo obiettivo è migliorare la produttività senza trascurare la vita familiare.

Giovanni (Programmatore)

Giovanni è uno sviluppatore che necessita di sessioni intense e focalizzate. Per mantenere la concentrazione ed evitare il burnout, utilizza il Pomodoro con pause regolari. Cerca un'app configurabile che non lo distraiga. Il suo obiettivo è restare produttivo e concentrato su un compito alla volta.

Elisa (Studentessa Universitaria)

Elisa studia spesso in gruppo da remoto. Il timer condiviso le permette di

coordinarsi con le amiche e mantenere alta la motivazione. Le pause comuni favoriscono la comunicazione e il confronto. Il suo obiettivo è aumentare la disciplina nello studio e ridurre la procrastinazione.

Luca (Sviluppatore in remoto)

Luca lavora da remoto con un team distribuito. Utilizza il Pomodoro condiviso per sincronizzare i momenti di lavoro e mantenere la concentrazione. Le pause sono utili per aggiornarsi rapidamente. Il suo obiettivo è coordinare il team e migliorare il focus collettivo.

0.3.5 Story boards

Il prodotto finale rispecchia in larga parte i mockup iniziali, pur presentando alcune revisioni apportate in seguito ai feedback raccolti dagli utenti che hanno avuto accesso anticipato a un primo prototipo dell'applicazione. 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 può scegliere di registrarsi.

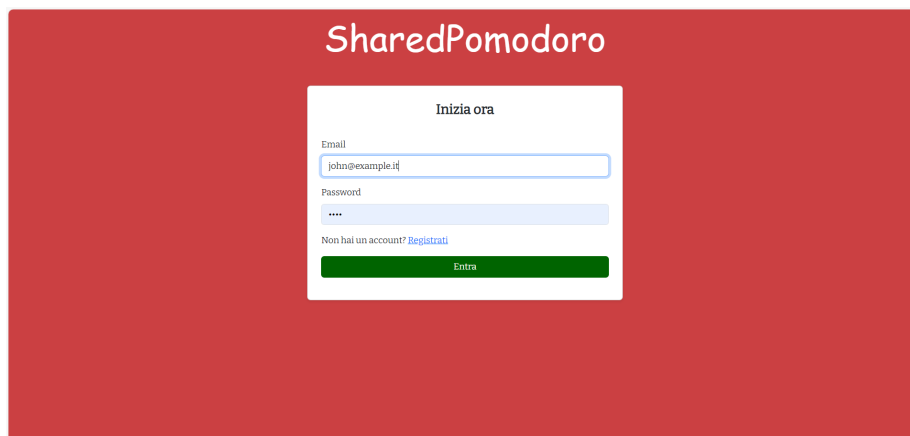


Figura 8: Rappresenta la schermata per fare il login.

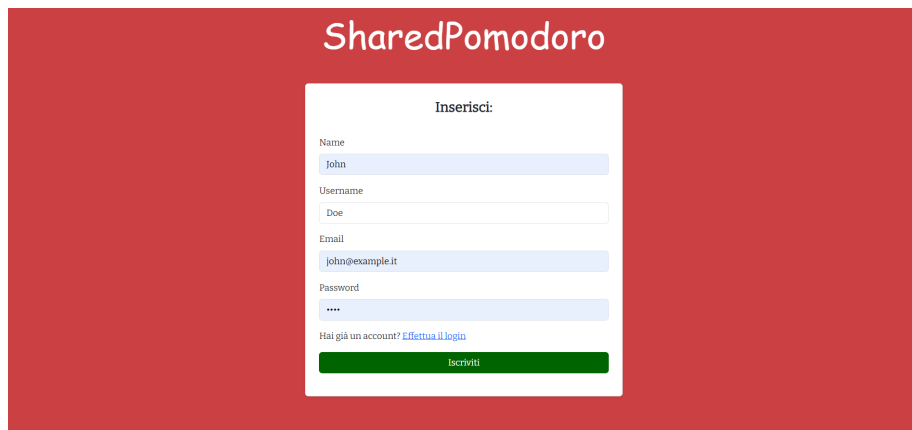


Figura 9: Rappresenta la schermata per effettuare la registrazione



Figura 10: Rappresenta la schermata per effettuare l'accesso alle stanze.

Nell'immagine in alto è raffigurata la schermata iniziale, che presenta un campo di input dedicato all'inserimento del nome di una stanza. Nella parte inferiore sono visibili 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, timer, elenco dei task completati e homepage. Passando con il cursore sopra l'icona del profilo, o cliccando nella versione mobile, viene mostrato il nome dell'utente.

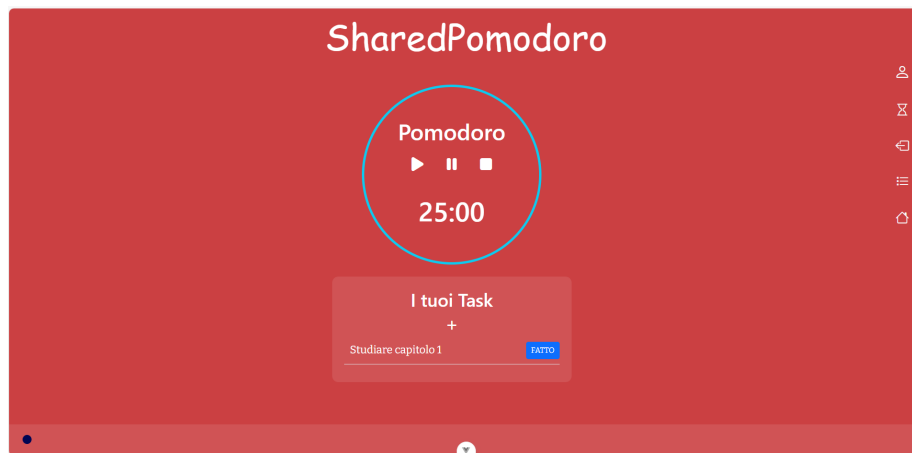


Figura 11: Rappresenta la stanza.

In questa schermata, al centro è ben visibile un cerchio che rappresenta il timer, affiancato dai controlli di avvio, pausa e arresto, evidenziando la funzionalità principale dell'applicazione. 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, o cliccando nella versione mobile, viene mostrato il relativo nome.

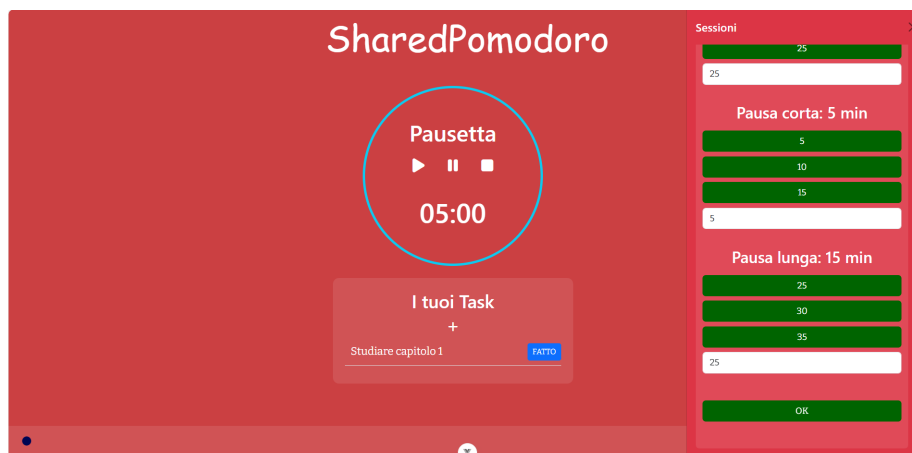


Figura 12: Rappresenta la schermata per modificare i tempi.

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. Durante la fase di testing, grazie al feedback degli utenti, è emersa l’utilità di mostrare accanto a ciascuna etichetta di sessione il valore aggiornato, al fine di rendere più chiara la personalizzazione dei tempi.

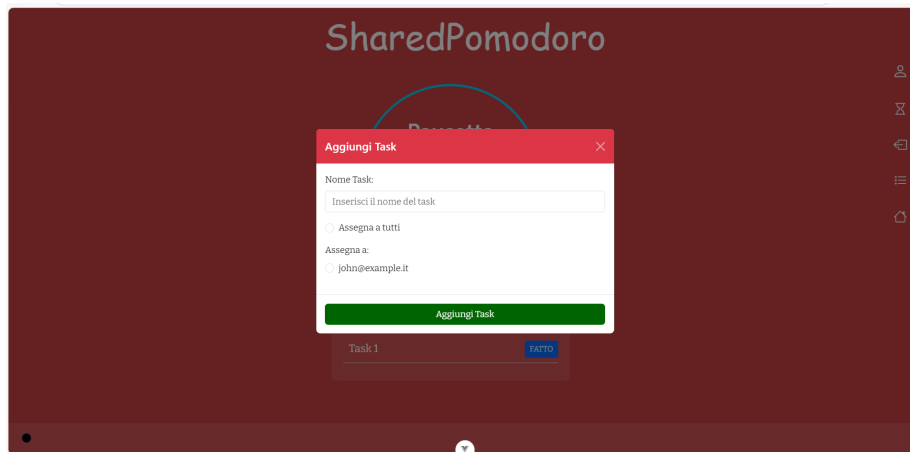


Figura 13: Rappresenta la schermata per aggiungere un task.

Una volta premuto il pulsante con il ”+” 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.



Figura 14: Rappresenta la schermata per visualizzare i task completati dall’utente.

Infine, 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.

0.4 Tecnologie

L'applicativo è stato sviluppato utilizzando lo **stack MEVN**, composto da:

- **MongoDB** per la gestione del database documentale,
- **Express.js** per la costruzione delle API backend,
- **Vue.js** per lo sviluppo dell'interfaccia utente,
- **Node.js** come ambiente di esecuzione lato server.

Questo stack ha permesso la realizzazione di un'architettura a tre livelli (frontend, backend, database), completamente basata su *JavaScript* e sul formato *JSON*, semplificando lo scambio di dati tra le varie componenti del sistema.

Lato frontend sono stati utilizzati **Vue.js** e strumenti correlati per la gestione dello stato e della reattività, mentre nel backend sono stati impiegati **Express** e **Mongoose** per la definizione dei modelli e l'interazione con **MongoDB**.

Per la progettazione delle API e la gestione delle funzionalità principali, come *login*, *registrazione* e *visualizzazione dei task completati*, è stato adottato il modello **REST (Representational State Transfer)**. Altre librerie utilizzate sono state:

- **Socket.io**: è una libreria JavaScript progettata per abilitare la comunicazione bidirezionale in tempo reale tra client e server nelle applicazioni web. Si compone di due componenti con la stessa API: una lato client e una lato server, quest'ultima utilizzata come libreria per Node.js. In questo progetto, Socket.io è stato impiegato per gestire il timer, i task e le stanze. Il suo funzionamento si basa su un meccanismo event-driven: al verificarsi di determinati eventi, il server invia direttamente i dati ai client interessati.
- **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.
- **Jest** Jest è un framework per il testing automatizzato in ambiente JavaScript, progettato per semplificare la scrittura e l'esecuzione dei test. Fornisce un ambiente completo che include strumenti per le asserzioni, il mocking e lo snapshot testing. La sua architettura consente una gestione efficace dei test asincroni e garantisce l'esecuzione isolata di ogni test, migliorando l'affidabilità delle verifiche.

0.4.1 Altre tecnologie

Backend

- **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.

Frontend

- **Axios**: client HTTP basato su promise, utilizzato per effettuare chiamate verso il backend.
- **Bootstrap**: framework CSS che semplifica lo sviluppo di interfacce responsive.

0.5 Codice

Di seguito sono presentate le porzioni di codice considerate più significative del progetto.

0.5.1 Backend

La funzione asincrona `handleAddTask` gestisce l'aggiunta di un nuovo task all'interno di una stanza in un'applicazione che sfrutta **Socket.IO** per la comunicazione in tempo reale tra il server e i client. La funzione riceve quattro parametri principali: `req`, che contiene i dati della richiesta (come il nome della stanza e il task da aggiungere); `socket`, il socket del client mittente; `clients`, un oggetto che mappa gli ID dei socket agli utenti connessi e alle loro stanze di appartenenza; e `io`, l'istanza di **Socket.IO** utilizzata per inviare eventi ai client.

La logica della funzione si suddivide in due scenari, in base alla destinazione del task. Se il task è destinato a tutti gli utenti della stanza, la funzione recupera i nomi degli utenti connessi, crea un task personalizzato per ciascuno e lo invia tramite i rispettivi socket ID. Per individuare i socket ID, viene utilizzato l'oggetto `clients`, che funge da mappa associando gli ID dei socket ai dati degli utenti, come il nome e la stanza di appartenenza. Per identificare gli utenti nella stanza, la funzione usa `Object.keys(clients)`, che restituisce un array con le chiavi (i socket ID). Successivamente, vengono applicati i metodi `filter()` e `map()` per estrarre solo gli utenti pertinenti, generando così dinamicamente un elenco di utenti a cui inviare i task.

Se il task è invece destinato a un singolo utente, il processo è semplificato: il task viene aggiunto e inviato esclusivamente al client specificato.

```
1  async function handleAddTask(req, socket, clients, io) {
2    const { room, newTask } = req;
3    const usersToNotify = newTask.userRef === "allUsers"
4      ? Object.keys(clients).filter(key => clients[key].
5        room === room).map(key => clients[key].name)
6        : [newTask.userRef];
7
7    for (const username of usersToNotify) {
8      const taskData = {
9        name: newTask.name,
10       userRef: username
11     };
12
12     const res = await controllerRoom.addTask(room,
13       taskData);
14
14     const targetSocketId = Object.keys(clients).find(
15       key => clients[key].name === username && clients
16         [key].room === room
17     );
```

```

18         if (targetSocketId) {
19             io.to(targetSocketId).emit('tasksToDo', res);
20         }
21     }
22 }
23 }

```

Listing 1: Codice gestione tasks

0.5.2 Frontend

Questo componente Vue è stato realizzato utilizzando la **Composition API** per organizzare il codice in modo modulare e leggibile. Ecco i punti principali:

1. **Gestione dello stato con ref:** Il componente utilizza **ref** per creare una variabile reattiva chiamata **tasks**. Questa variabile memorizza l'elenco dei task che l'utente ha completato, che verranno recuperati dal server tramite una chiamata API.
2. **Chiamata API con Axios:** All'interno del ciclo di vita del componente, viene usato il hook **onMounted** per eseguire una chiamata asincrona tramite **Axios**. Viene inviata una richiesta **POST** al server per recuperare i task completati dell'utente. L'ID dell'utente, che viene prelevato dal **sessionStorage**, viene passato al server come parte della richiesta. La risposta dell'API, contenente i task, viene assegnata alla variabile **tasks**, che aggiorna automaticamente la vista grazie alla reattività di Vue.
3. **Rendering dinamico con v-for:** Nel **template**, viene utilizzata la direttiva **v-for** per iterare attraverso l'array **tasks** e generare un elenco dinamico. Ogni elemento dell'array rappresenta un task, e per ciascuno viene creato un elemento della lista che mostra il nome del task e la stanza a cui è associato. L'indice dell'array viene usato per numerare i task visivamente, rendendo l'interfaccia utente più chiara.
4. **Integrazione del componente SideBar:** Alla fine del template, il componente include un altro componente figlio, **SideBar**. Questo componente viene utilizzato per visualizzare la barra laterale, le opzioni di navigazione.

```

1 <script setup>
2 import {onMounted, ref} from 'vue';
3 import SideBar from "@/components/gateway/SideBar.vue";
4 import axios from "axios";
5 const tasks = ref([]);
6
7 onMounted(async () => {
8     try {
9         const userId = JSON.parse(sessionStorage.getItem('user'))
          ?.email;

```

```

10     const res = await axios.post('http://localhost:3000/api/
      tasksDone', {userId});
11     tasks.value = res.data;
12   } catch (err) {
13     console.log("error", err.response.data.message);
14   }
15 });
16 </script>
17 <template>
18   <div class="d-flex vh-100">
19
20     <div class="container d-flex justify-content-center mt
      -5">
21       <div class="list-group w-100">
22         <a
23           v-for="(task, index) in tasks"
24           :key="index"
25           class="list-group-item list-group-item-action
      flex-column align-items-start mb-4"
26         >
27           <div class="d-flex justify-content-between">
28             <h5 class="mb-1">Task {{index+1}}: {{ task.name
      }}</h5>
29           </div>
30           <p class="mb-1">Stanza: {{ task.room }}</p>
31         </a>
32       </div>
33     </div>
34     <SideBar/>
35   </div>
36 </template>

```

Listing 2: Codice gestione tasks completati

0.6 Test

0.6.1 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, proporre di nuovi da affidare ai compagni di lavoro. È stato chiesto loro di esplorare liberamente la dashboard per visualizzare i propri progressi.

Un'attenzione particolare è stata riservata all'osservazione dell'intuitività dell'interfaccia, in particolare nella gestione in tempo reale del timer condiviso e nel passaggio tra le fasi di focus e di pausa. Sono stati valutati anche la facilità con cui si potevano assegnare o ricevere task, e il livello di coinvolgimento generato dalla collaborazione all'interno del sistema.

0.6.2 Test usabilità

Per garantire un'esperienza utente ottimale, sono stati applicati i principi delle euristiche di Nielsen, linee guida fondamentali per progettare interfacce intuitive e facili da usare. I seguenti punti descrivono come tali principi sono stati implementati nel design dell'interfaccia del sistema.

- **Visibilità dello stato del sistema:** sono stati gestiti con attenzione eventuali ritardi di risposta, in modo che l'utente riceva sempre un riscontro visivo immediato e coerente rispetto alle azioni compiute, senza percepire interruzioni o incertezze nel funzionamento del sistema.
- **Corrispondenza tra sistema e mondo reale:** il linguaggio adottato è stato pensato per risultare accessibile anche a utenti non esperti, evitando termini tecnici e privilegiando una comunicazione più vicina al

linguaggio quotidiano, così da favorire una comprensione immediata delle funzionalità.

- **Controllo e libertà dell'utente:** le operazioni necessarie per svolgere un compito sono state ridotte al minimo essenziale. Questo approccio ha permesso di semplificare l'interazione, diminuire la complessità percepita e offrire un'esperienza più fluida e naturale.
- **Coerenza e standard:** l'interfaccia mantiene una forte coerenza visiva e funzionale. I pulsanti condividono caratteristiche grafiche comuni e l'intera piattaforma fa uso di una palette cromatica ben definita e applicata in modo uniforme, così da guidare l'attenzione dell'utente verso gli elementi più rilevanti.
- **Prevenzione degli errori:** la struttura dell'applicazione è stata progettata per evitare situazioni che potrebbero indurre in errore l'utente. L'architettura dei percorsi non presenta vicoli ciechi o ambiguità che possano compromettere l'esperienza d'uso.
- **Riconoscimento anziché memorizzazione:** i layout sono stati progettati in modo semplice e razionale, facilitando l'individuazione immediata delle funzioni principali. L'utente può muoversi all'interno dell'interfaccia senza dover memorizzare informazioni da una schermata all'altra.
- **Flessibilità ed efficienza d'uso:** considerata la semplicità delle funzionalità offerte, non è stata ritenuta necessaria l'introduzione di scorciatoie o modalità avanzate. Il sistema si presta a un utilizzo immediato e intuitivo da parte di qualsiasi tipologia di utente.
- **Design estetico e minimalista:** il principio KISS (*Keep It Simple, Stupid*) è stato adottato come linea guida per la progettazione. Ogni pagina presenta un numero limitato di funzionalità, chiaramente distinguibili tra loro, così da garantire chiarezza e semplicità visiva.
- **Gestione degli errori:** eventuali errori di interazione vengono segnalati con messaggi chiari, comprensibili e non tecnici. L'utente viene guidato nella comprensione dell'errore e, quando possibile, gli viene offerta una soluzione concreta per risolverlo in modo semplice.
- **Aiuto e documentazione:** grazie alla struttura minimalista e all'interfaccia intuitiva, si è ritenuto che l'utente possa utilizzare l'applicazione senza bisogno di documentazione o guide specifiche. La facilità d'uso è stata garantita fin dalla progettazione, riducendo al minimo la necessità di supporto esterno.

0.6.3 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'applica-

zione, verificando che gli endpoint rispondessero correttamente alle richieste e restituissero i dati attesi, nonché la gestione adeguata degli errori.

Per eseguire i test, seguire i seguenti passaggi:

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

0.7 Deployment

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>

0.8 Conclusioni

La realizzazione di questo progetto ha contribuito significativamente all'arricchimento del mio bagaglio culturale, permettendomi di applicare le tecniche acquisite durante il corso in un contesto pratico, con particolare attenzione alla realizzazione di funzionalità real-time e all'uso di tecnologie diversificate all'interno della stessa soluzione. Nel corso dello sviluppo della piattaforma, uno degli strumenti chiave è stato Vue.js, una tecnologia che non utilizzavo da molto tempo. Grazie agli strumenti e alle risorse fornite dal corso, è stato possibile riprendere rapidamente l'utilizzo di Vue e sfruttarne le potenzialità per migliorare l'organizzazione del codice. Oltre a Vue.js, ho avuto l'opportunità di esplorare anche altre tecnologie come Socket.io, che erano precedentemente poco conosciute o mai utilizzate. Nel complesso, la combinazione di queste diverse tecnologie e l'approccio integrato hanno contribuito a sviluppare una soluzione innovativa e ben strutturata, favorendo la crescita professionale.