# DockerApp

Applicazioni e Servizi Web

Gloria Semprini - 0001033215 {gloria.semprini@unibo.it}

February 8, 2025

# Chapter 1

# Introduzione

Il progetto proposto consiste nella realizzazione di un'applicazione web che consente agli utenti di amministrare in modo semplice e interattivo i container Docker all'interno di gruppi di ricerca. L'applicazione facilita la gestione di operazioni primarie dei container, come l'avvio, l'arresto e il monitoraggio delle risorse, rendendo accessibile l'orchestrazione anche per chi non ha conoscenze approfondite della tecnologia.
Attraverso un sistema di notifiche in tempo reale, gli utenti possono ricevere aggiornamenti sugli stati del container, agevolando la comunicazione e il lavoro tra i partecipanti dei gruppi.

# Chapter 2

# Requisiti

Requisiti funzionali:

- Il sistema deve supportare due tipologie di utenti:
  - **Amministratori dei gruppi:** semplici utenti con il permesso di creare, gestire ed eliminare gruppi, ma anche gestire ruoli. Possono anche creare, eliminare, gestire e monitorare i container Docker associati al gruppo di cui fanno parte.
  - **Membri del gruppo:** utenti appartenenti ad un gruppo, possono interagire con i container assegnati al gruppo e monitorarli.
- Il servizio consiste nella gestione dei container Docker all'interno dei gruppi:
  - Gli utenti sono in grado di visualizzare i dettagli dei container (nome, stato, immagine), avviarli e fermarli.
  - Gli amministratori possono creare e eliminare nuovi container all'interno dei gruppi.
- Gli utenti ricevono notifiche in tempo reale:
  - Quando un container viene avviato o fermato, gli utenti ricevono una notifica che include il nome del container e lo stato dell'operazione (avviato o fermato).
  - Quando un container viene creato o eliminato, gli utenti ricevono una notifica che include il nome dell'immagine del container con esito (successo o insuccesso).
- Ai container sono associati gli identificativi dei gruppi per gestire al meglio la loro associazione.
- Ogni utente avrà un'area personale dove poter modificare i propri dati personali o eliminare l'account.

Requisiti non funzionali:

- L'interfaccia utente deve essere accessibile, equilibrato tra tecnologia e naturalezza, chiara e leggibile.

# Chapter 3

# Design

Il progetto è stato realizzato adottando un approccio **User-Centered Design (UCD)**, mettendo gli utenti al centro del processo decisionale al fine di garantire un'applicazione semplice e funzionale.Per comprendere meglio le esigenze degli utenti, sono state delineate personas che hanno guidato lo sviluppo delle funzionalità principali. Il modello di sviluppo adottato è stata una metodologia simil-agile con un'analisi dettagliata sulle attività da svolgere, definizione delle priorità e feedback esterno.
E' stato adottato il principio di **Minimum Viable Product (MVP)** che ha consentito di concentrarsi sulle funzionalità essenziali, rendendo possibile un rapido deployment e feedback continuo da parte del committente (anche fittizio).
Le interfacce utente sono state progettate seguendo i principi del **Kiss (Keep It Simple, Stupid)** per garantire un'esperienza chiara e intuitiva, riducendo al minimo la possibilità di errori.
Per garantire la fruibilità anche su dispositivi mobile, è stato adottato l'approccio **Mobile First**, assicurando compatibilità e usabilità con un design responsivo e accessibile.

## 3.1 Mockup

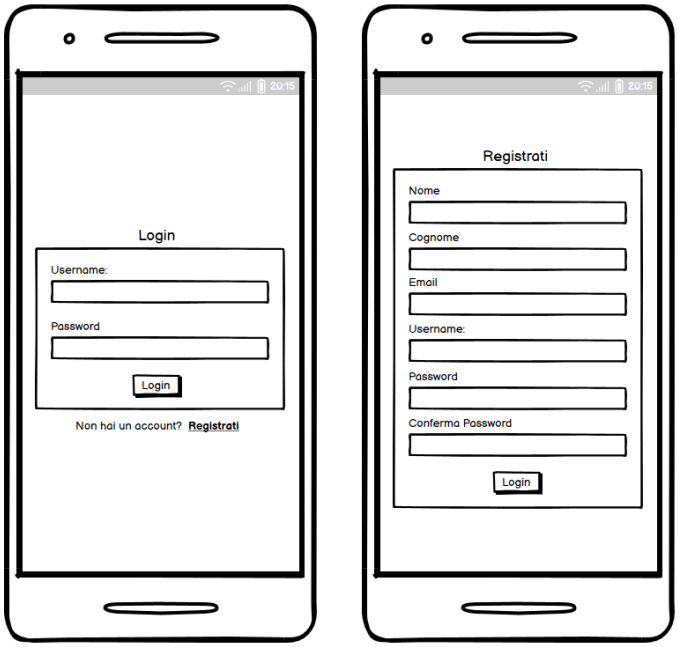


Figure 3.1: Mockup Registrazione e Login in versione mobile

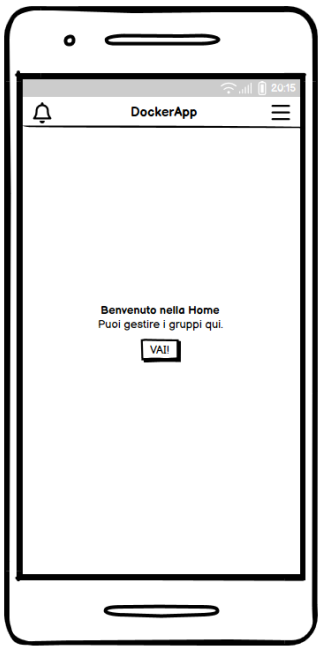


Figure 3.2: Mockup Home

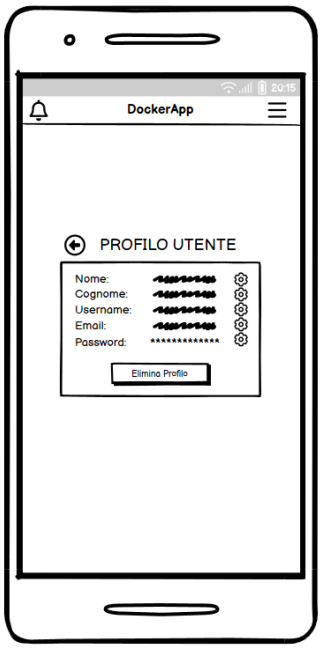


Figure 3.3: Mockup profilo utente

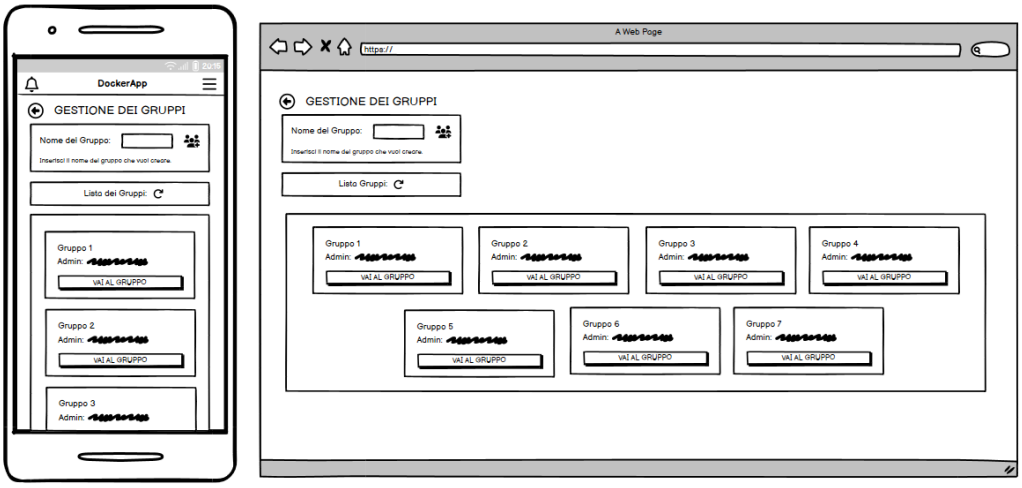


Figure 3.4: Mockup gestione dei gruppi

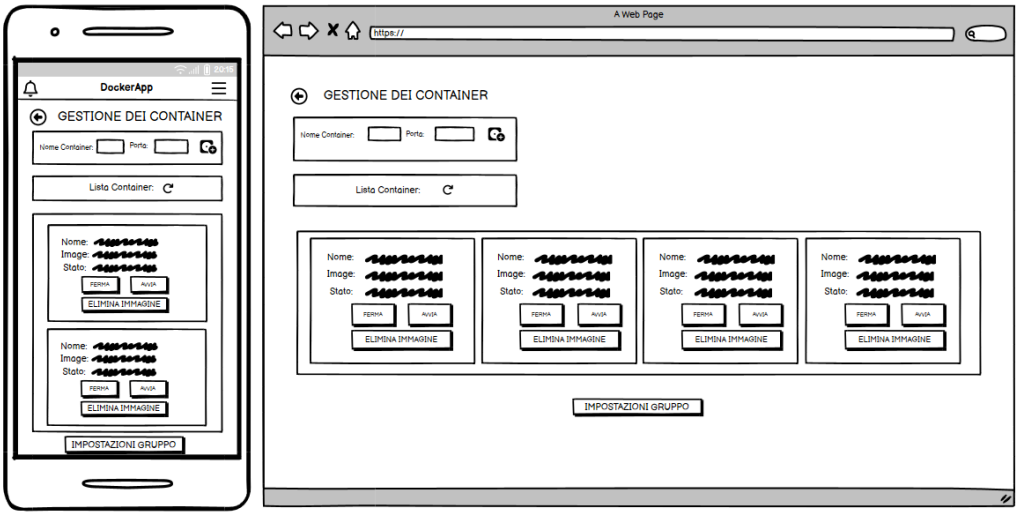


Figure 3.5: Mockup gestione container

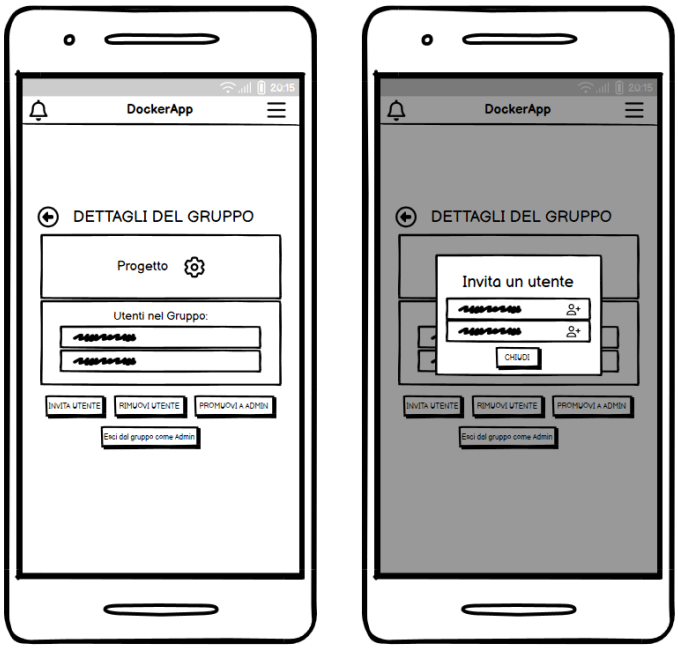


Figure 3.6: Mockup informazioni del gruppo e promozione ad admin, rimozione e invito di un utente

# Chapter 4

# Personas

Sono state costruite diverse Personas su due profili: *amministratore del gruppo* e *membro del gruppo*.

## Personas: Lorenzo

Lorenzo è uno studente del corso di laurea magistrale in Informatica all'Università di Bologna. Sta lavorando alla sua tesi, che prevede l'analisi di grandi dataset tramite algoritmi distribuiti su container Docker.

- Lorenzo ha bisogno di un ambiente flessibile per eseguire i suoi test senza dover configurare ogni volta il sistema da zero.
- Consulta spesso le risorse del progetto per capire come ottimizzare l'uso dei container per la sua ricerca.
- Condivide i risultati intermedi con il suo relatore, che gli fornisce indicazioni per migliorare il setup dei test.

## Personas: Elena

Elena è docente e responsabile di un laboratorio di ricerca presso l'Università di Bologna. Coordina un team di studenti che lavora su progetti legati all'intelligenza artificiale.

- Ha bisogno di una piattaforma semplice per monitorare l'uso delle risorse del laboratorio, compresi i container Docker avviati dagli studenti.
- Elena, grazie alle notifiche tempestive su eventuali problemi di sistema o aggiornamenti sui container rimane informata sui cambiamenti dei progetti.

- Apprezza la possibilità di visualizzare informazioni dettagliate sullo stato e sulle configurazioni dei container, permettendo di monitorare e dare feedback sui setup sperimentali dei suoi studenti.

## Personas: Luca

Luca è un ricercatore senior del dipartimento di Informatica all'Università di Bologna. Gestisce un gruppo di ricerca che si occupa di simulazioni scientifiche distribuite utilizzando container Docker per gestire gli ambienti di sviluppo e test.

- Luca ha la responsabilità di configurare e creare nuovi container per ogni esperimento, gestendo l'allocazione delle risorse e monitorando lo stato di ciascun container.
- Luca consulta regolarmente i progressi degli altri membri e fornisce supporto tecnico riguardo all'uso dei container per le attività sperimentali.

## Personas: Giovanni

Giovanni è un docente di ingegneria informatica all'Università di Bologna. Coordina progetti accademici finanziati, che richiedono una gestione attenta delle risorse, dei tempi e degli obiettivi, e supervisiona i suoi studenti.

- Quando un suo tesista si laurea, lo rimuove dal gruppo per fare spazio ai nuovi tesisti, favorendo così il continuo sviluppo dell'argomento di ricerca.
- Se il progetto ha aperto nuove possibilità, Giovanni decide di aprire un nuovo gruppo di ricerca e aggiunge nuovi utenti.

## 4.1 Storyboard

Il prodotto finale si avvicina molto ai mockup iniziali, con alcune revisioni basate sul feedback degli utenti che hanno testato una prima versione dell'applicazione. La schermata di login è la prima che l'utente vede all'accesso alla piattaforma. Qui può inserire le sue credenziali (username e password) per entrare nel sistema (Figura 4.2). Se l'utente non ha ancora un account, può registrarsi utilizzando il pulsante "Registrati" (Figura 4.1).

Una volta effettuato l'accesso, l'utente viene indirizzato alla schermata principale di benvenuto che può indirizzarlo verso la sezione Gruppi.

Se l'utente non è ancora membro di alcun gruppo, viene mostrata una sezione vuota con il pulsante "Crea Gruppo" (Figura 4.4).

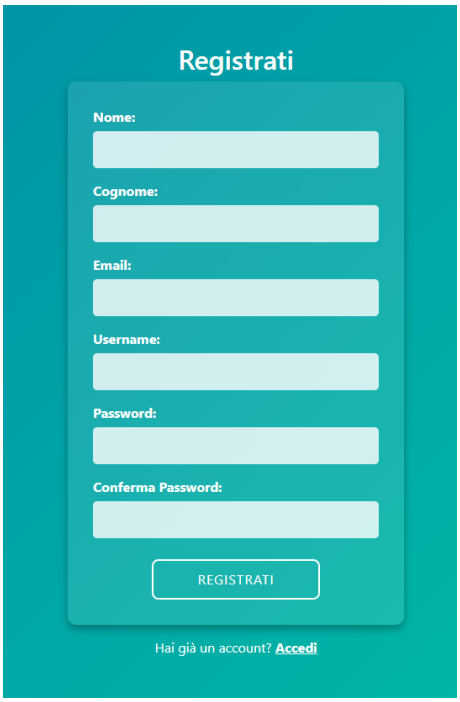


Figure 4.1: Registrazione

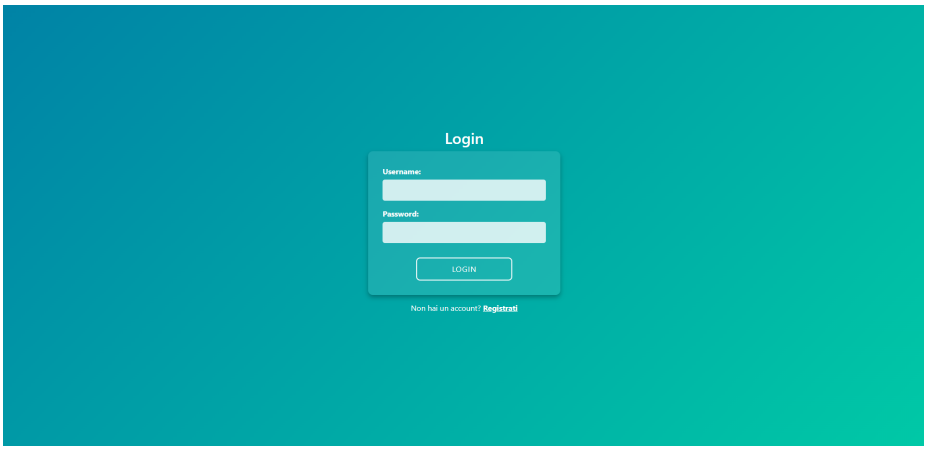


Figure 4.2: Login

Cliccando sul pulsante, l'utente può creare un nuovo gruppo, definendo nome, descrizione e altre informazioni.
Durante il processo di creazione del gruppo, l'utente visualizza un modulo in

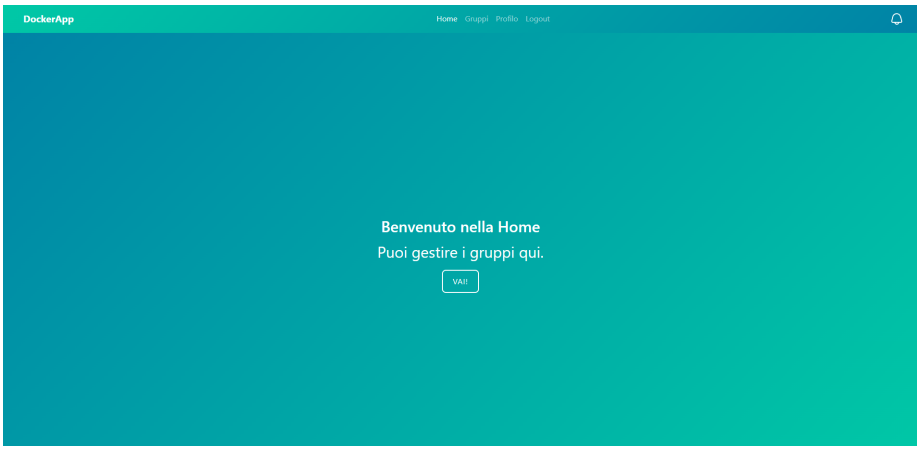


Figure 4.3: Home

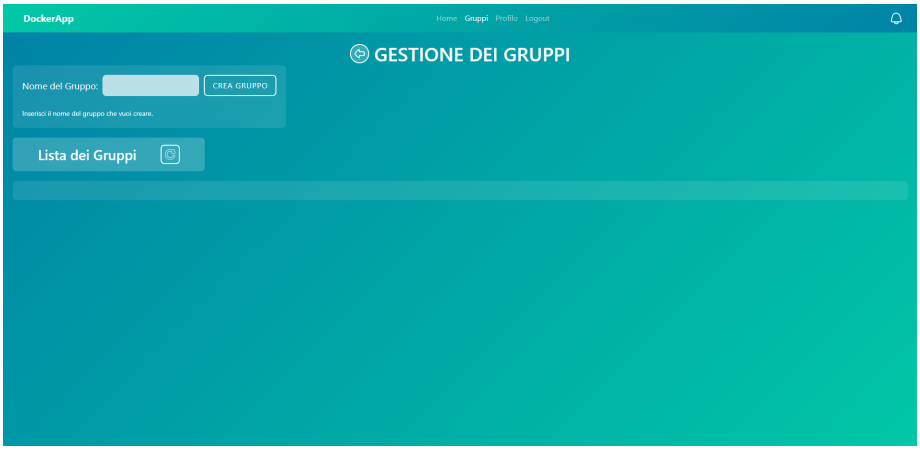


Figure 4.4: Elenco dei gruppi (vuoto)

cui inserire i dettagli del gruppo. Viene evidenziata la possibilità di scegliere i membri da aggiungere e di definire il gruppo come pubblico o privato. Una volta creato, il gruppo appare nella lista dei gruppi disponibili.

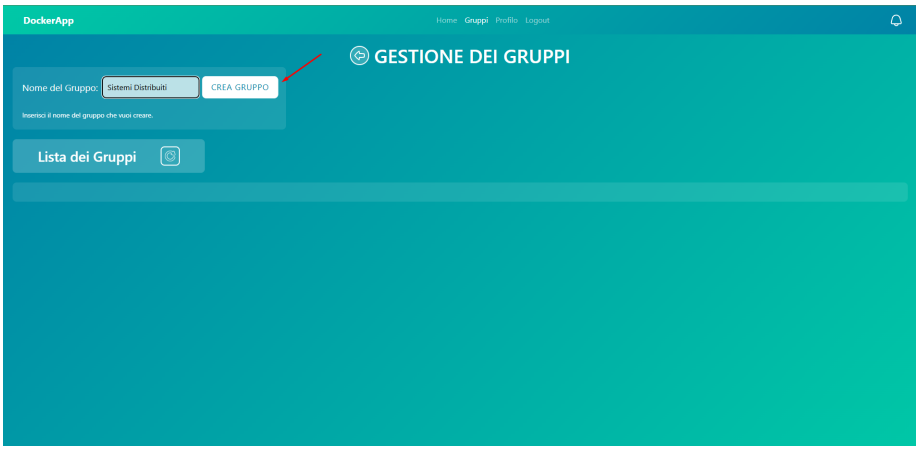


Figure 4.5: Creazione del gruppo

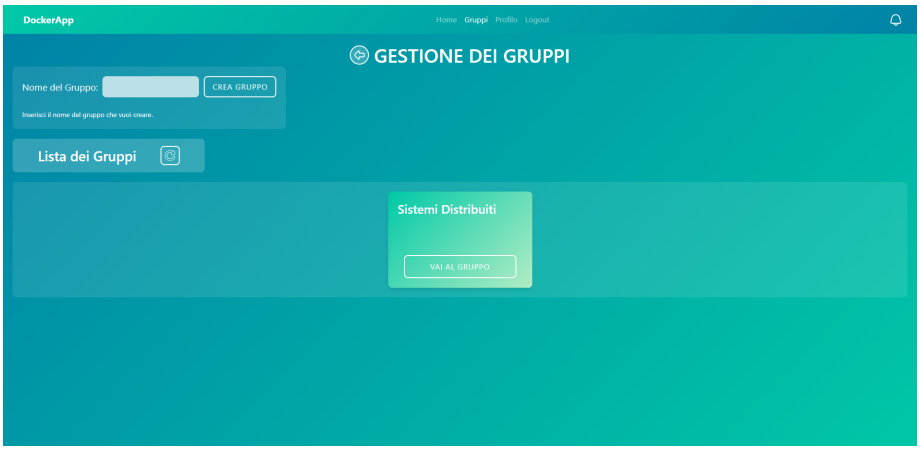


Figure 4.6: Lista dei gruppi

Cliccando sul gruppo nella schermata principale, l'utente entra nella sezione interna del gruppo. Qui l'interfaccia si divide in diverse aree, tra cui la sezione "Container", che mostra i container attivi e quelli in attesa di avvio.

Nella schermata di creazione del container, l'utente deve inserire le informazioni necessarie, come il tipo di container, le risorse richieste e le modalità di esecuzione. Una volta completato il modulo, l'utente può cliccare sul pulsante "Crea", il che genera il container per il gruppo. Un messaggio di conferma viene visualizzato per notificare l'utente che il container è stato creato con successo.

Una volta che il container è stato creato, nella lista dei container appare il nuovo container. Gli utenti con ruolo di amministratore vedranno i pulsanti "Ferma", "Avvia" e "Elimina" accanto al container. Questi pulsanti permet-

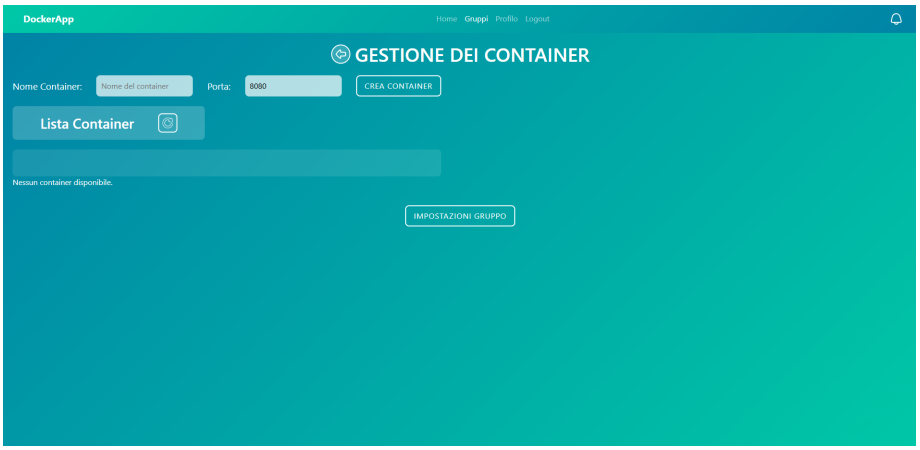


Figure 4.7: Lista dei container (vuota)

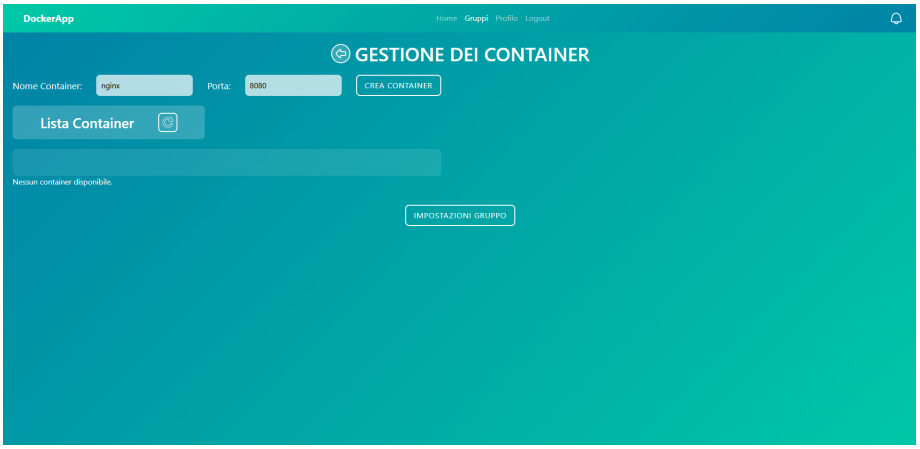


Figure 4.8: Creazione container (container nginx)

tono all'amministratore di gestire il ciclo di vita del container. I membri semplici vedranno solo i pulsanti "Ferma" e "Avvia", in quanto non hanno i permessi per eliminare il container. La schermata mostra anche un'icona di stato che indica se il container è attivo o inattivo.

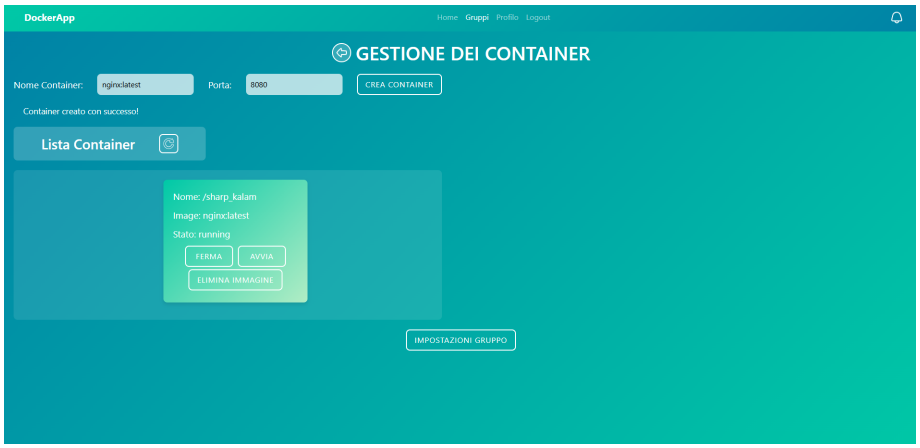


Figure 4.9: Lista dei container

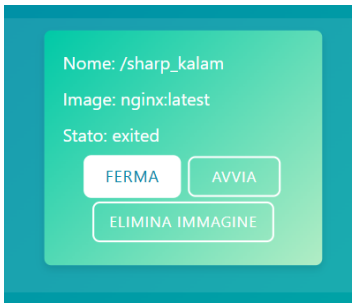


Figure 4.10: Container start/stop


Figure 4.11: Container start/stop e eliminazione dell'immagine

Ogni volta che un container viene creato, avviato o feramto, i membri del gruppo vengono notificati tramite un sistema di notifiche interne.

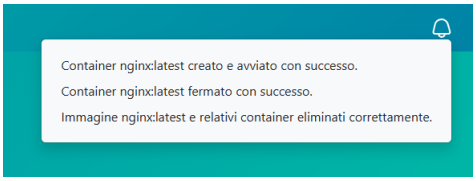


Figure 4.12: Notifiche del gruppo

Nella schermata delle impostazioni del gruppo, l'amministratore ha accesso a un pannello di controllo avanzato dove può aggiungere o rimuovere membri dal gruppo. I membri semplici, invece, hanno solo accesso a una vista limitata che permette di visualizzare il gruppo e interagire con i container, ma non di modificarne le impostazioni.

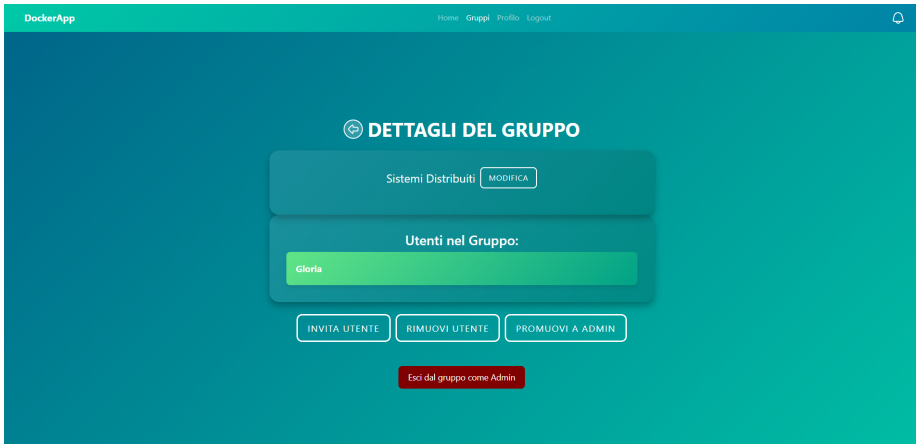


Figure 4.13: Informazioni del gruppo


Figure 4.14: Aggiunta di un utente


Figure 4.15: Promozione ad admin

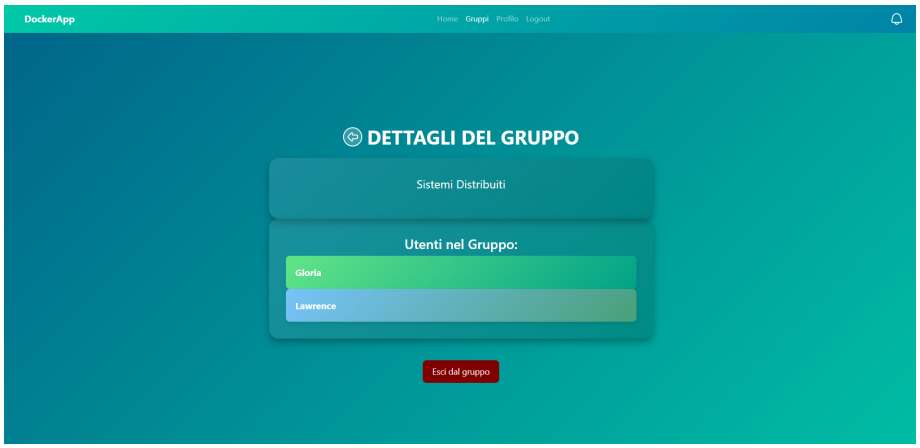


Figure 4.16: Utente non admin

## 4.2 Tecnologie

Il sistema è stato realizzato utilizzando lo Stack MEAN.
Si compone quindi di un backend basato su un server in **Node.js** con **Express** come framework di gestione delle richieste e database non relazionale con **MongoDB**. Il sistema viene poi reso disponibile agli utenti tramite un'interfaccia sviluppata con **Angular CLI**

## 4.3 Back-end

IL server viene avviato lanciando il file index.js, attraverso il quale viene creata un'istanza Express, viene stabilita una connessione col database MongoDB e vengono definite le diverse route utilizzate dall'applicazione. Le route sono suddivise nei seguenti moduli, ciascuno dei quali fa riferimento a uno schema Mongoose utilizzato nel database:

- `auth`
  - `register` (post): carica nel database un nuovo utente utilizzando le informazioni del corpo dell'oggetto richiesta.
  - `login` (post): verifica le credenziali dell'utente nel database e, in caso di successo, restituisce un token di autenticazione valido per le richieste future insieme all'ID dell'utente.
  - `logout` (post): termina la sessione dell'utente autenticato e invalida il suo token, disconnettendo l'utente dal sistema.
  - `validate-token` (post): verifica la validità del token di autenticazione. Se il token è valido, restituisce un messaggio di conferma, altrimenti un errore.

- `home`
  - `/` (get): restituisce un messaggio di benvenuto alla home page dell'applicazione.
- `profile`
  - `/profile/update-name-surname` (put): consente all'utente di aggiornare il nome e il cognome nel proprio profilo.
  - `/profile/update-username` (put): consente all'utente di aggiornare il proprio username nel profilo.
  - `/profile/update-email` (put): consente all'utente di aggiornare la propria email nel profilo.
  - `/profile/change-password` (put): consente all'utente di cambiare la propria password nel profilo.
  - `/profile/deleteAccount` (delete): permette all'utente di eliminare il proprio account dal sistema.
  - `/profile` (get): restituisce le informazioni del profilo dell'utente autenticato.
- `group`
  - `/create` (post): consente all'utente autenticato di creare un nuovo gruppo.
  - `/group-users` (get): restituisce gli utenti di un gruppo specificato tramite l'ID del gruppo.
  - `/user-groups` (get): restituisce i gruppi a cui l'utente autenticato è iscritto.
  - `/list-container` (get): restituisce la lista dei container associati a un gruppo specificato.
  - `/stop-container` (post): consente di fermare un container all'interno di un gruppo.
  - `/start-container` (post): consente di avviare un container all'interno di un gruppo.
  - `/leave` (delete): consente all'utente autenticato di uscire da un gruppo specificato.
  - `/info` (get): restituisce le informazioni di un gruppo specificato tramite il suo ID.
- `admin`
  - `/available-users` (get): restituisce gli utenti che non fanno parte di un gruppo e che possono essere aggiunti a un gruppo specifico.
  - `/inviteUser` (post): consente all'amministratore di un gruppo di invitare un utente a farne parte.

- `/change-admin` (put): consente all'amministratore di un gruppo di promuovere un utente a nuovo amministratore.
- `/removeUser` (delete): consente all'amministratore di rimuovere un utente da un gruppo.
- `/create-container` (post): consente all'amministratore di creare un nuovo container all'interno di un gruppo.
- `/delete-image` (delete): consente all'amministratore di eliminare un'immagine di un container all'interno di un gruppo.
- `/delete-group` (delete): consente all'amministratore di eliminare un gruppo.
- `/is-admin` (get): consente di verificare se l'utente autenticato è un amministratore di un gruppo.
- `/leave-group` (delete): consente all'amministratore di uscire da un gruppo.

Al momento del login, il server genera un **JWT (Json Web Token)** che viene inviato al client per confermare l'identità dell'utente. Questo token viene poi inviato dal client in ogni richiesta successiva, consentendo all'utente di accedere alle risorse protette dell'applicazione fino alla scadenza del token stesso. Il token viene salvato nel browser del client (ad esempio in localStorage o sessionStorage) per essere utilizzato nelle richieste successive. In questo modo, l'utente rimane autenticato finché il token è valido, senza dover effettuare un nuovo login.

## 4.4 Fronte-end

L'UI è stata realizzata con Angular, suddividendola in più componenti di cui i principali sono i seguenti:

- **Registration**: form per la registrazione di un nuovo utente;
- **Profile**: visualizza le informazioni relative all'utente loggato;
- **Notification**: gestisce e visualizza le notifiche per l'utente;
- **Navbar**: navbar visibile su tutte le pagine, con link differenti in base allo stato di login dell'utente;
- **Login**: form per l'accesso dell'utente al sistema;
- **Home**: pagina principale del sito, accessibile a tutti gli utenti;
- **Groups**: visualizza la lista dei gruppi a cui l'utente appartiene;
- **GroupContainers**: visualizza i container appartenenti a un gruppo specifico;
- **GroupList**: mostra la lista dei gruppi dell'utente;

- **GroupDetail**: visualizza i dettagli di un singolo gruppo, comprese le informazioni relative ai membri e alle attività;

È stato fatto uso del componente `Route` per visualizzare componenti diversi in base al percorso URL specificato.

- `profile`
  - **getProfile**: Effettua una richiesta GET per ottenere il profilo dell'utente e restituisce il risultato. In caso di errore, restituisce un valore predefinito o rilancia l'errore.
  - **updateNameSurname**: Effettua una richiesta PUT per aggiornare il nome e cognome dell'utente. Gestisce eventuali errori durante l'aggiornamento.
  - **updateUsername**: Effettua una richiesta PUT per aggiornare lo username dell'utente. Gestisce eventuali errori durante l'aggiornamento.
  - **updateEmail**: Effettua una richiesta PUT per aggiornare l'email dell'utente. Gestisce eventuali errori durante l'aggiornamento.
  - **changePassword**: Effettua una richiesta PUT per cambiare la password dell'utente. Gestisce eventuali errori durante il cambio della password.
  - **deleteProfile**: Effettua una richiesta DELETE per eliminare il profilo dell'utente. Gestisce eventuali errori durante l'eliminazione del profilo.
- `group-container`
  - **createContainer**: Esegue una richiesta POST per creare un nuovo container all'interno di un gruppo specificato, utilizzando il nome dell'immagine e il mapping delle porte. Gestisce errori di gruppo non trovato o errore interno del server.
  - **listContainers**: Esegue una richiesta GET per ottenere la lista dei container associati a un gruppo. Se non vengono trovati container, ritorna un array vuoto.
  - **stopContainer**: Esegue una richiesta POST per fermare un container all'interno di un gruppo. Gestisce errori di container non trovato o errore interno del server.
  - **startContainer**: Esegue una richiesta POST per avviare un container all'interno di un gruppo. Gestisce errori di container non trovato o errore interno del server.
  - **deleteImage**: Esegue una richiesta DELETE per eliminare un'immagine di container all'interno di un gruppo. Gestisce errori di immagine non trovata o errore interno del server.

  - **checkAdmin**: Esegue una richiesta GET per verificare se l'utente è amministratore di un gruppo. Se non trovati container, ritorna che l'utente non è amministratore.

- `group`
  - **listGroups**: Esegue una richiesta GET per ottenere la lista dei gruppi dell'utente. Se non vengono trovati gruppi, ritorna un array vuoto.
  - **createGroup**: Esegue una richiesta POST per creare un nuovo gruppo con il nome specificato. Gestisce errori relativi a problemi nella creazione del gruppo.
  - **getGroupDetails**: Esegue una richiesta GET per ottenere i dettagli di un gruppo specifico utilizzando l'ID del gruppo. Gestisce errori di gruppo non trovato.
  - **updateGroupName**: Esegue una richiesta PUT per aggiornare il nome di un gruppo specifico utilizzando l'ID del gruppo. Gestisce errori di gruppo non trovato.
  - **getGroupUsers**: Esegue una richiesta GET per ottenere la lista degli utenti appartenenti a un gruppo. Gestisce errori relativi al recupero degli utenti.
  - **addUser**: Esegue una richiesta POST per aggiungere un utente a un gruppo come amministratore. Gestisce errori relativi a gruppo o utente non trovato o altre problematiche.
  - **removeUser**: Esegue una richiesta DELETE per rimuovere un utente da un gruppo. Gestisce errori relativi a utente non trovato o errore del server.
  - **promoteToAdmin**: Esegue una richiesta PUT per promuovere un utente a amministratore di un gruppo. Gestisce errori nella promozione dell'utente.
  - **getAvailableUsers**: Esegue una richiesta GET per ottenere la lista degli utenti disponibili da aggiungere a un gruppo. Gestisce errori e ritorna un array vuoto se nessun utente è disponibile.
  - **leaveGroup**: Esegue una richiesta DELETE per fare uscire un utente dal gruppo. Gestisce errori durante il processo di uscita.
  - **adminLeaveGroup**: Esegue una richiesta DELETE per fare uscire un amministratore dal gruppo. Gestisce errori durante il processo di uscita.

- `auth`
  - **isAuthenticated$**: Restituisce un `Observable` che emette il valore di autenticazione dell'utente (vero o falso).

- **checkAuthentication**: Verifica se l'utente è autenticato controllando la presenza di un token e di un `userId` in `localStorage`.
- **register**: Esegue una richiesta POST per registrare un nuovo utente. Se la registrazione è completata con successo, reindirizza l'utente alla pagina di login.
- **login**: Esegue una richiesta POST per eseguire il login dell'utente. Se il login è riuscito, memorizza il token e l'ID utente nel `localStorage` e aggiorna lo stato di autenticazione.
- **logout**: Esegue una richiesta POST per eseguire il logout dell'utente. Dopo il logout, rimuove il token e l'ID utente da `localStorage` e aggiorna lo stato di autenticazione.
- **clearLocalStorage**: Rimuove il token e l'ID utente da `localStorage`.
- **validateToken**: Esegue una richiesta POST per validare il token di autenticazione. Restituisce `true` se il token è valido, altrimenti `false`.
- **handleError**: Gestisce gli errori generati dalle richieste HTTP. In caso di errore 401 (non autorizzato), avvisa l'utente che le credenziali non sono valide.

## 4.5 Codice

L'oggetto preso in riferimento è quello del token-interceptor, poichè è un servizio che intercetta tutte le richieste HTTP dell'applicazione, aggiungendo un token di autorizzazione (se presente) nelle intestazioni delle richieste per accedere a risorse protette. Se il token è assente o scaduto, la richiesta viene inviata senza di esso. L'interceptor gestisce anche l'eventuale errore `401 Unauthorized`, reindirizzando l'utente alla pagina di login. Inoltre, permette di bypassare il token su richieste che non richiedono autenticazione tramite l'intestazione `skipTokenInterceptor`. Il codice dell'interceptor per il token è il seguente:

```
intercept(req: HttpRequest<any>, next: HttpHandler):
    Observable<HttpEvent<any>> {
    if (req.headers.has('skipTokenInterceptor')) {
      const clonedRequest = req.clone({
        headers:
            req.headers.delete('skipTokenInterceptor')
      });
      return next.handle(clonedRequest);
    }

    const token = localStorage.getItem('token');
    if (token) {
      const cloned = req.clone({
```

```
            headers: req.headers.set('Authorization',
                `Bearer ${token}`)
        });
        return next.handle(cloned);
    }

    return next.handle(req);
}
```

Con il suo relativo `authenticate` in backend. ha lo scopo di garantire che solo gli utenti autenticati possano accedere alle risorse protette. Inizialmente, estrae il token JWT dalla richiesta HTTP, cercando nell'intestazione `Authorization`, che dovrebbe contenere il prefisso `Bearer` seguito dal token stesso. Se il token non è presente, la funzione restituisce un errore `401` con un messaggio che indica che il token è mancante o non valido. Se il token è presente, la funzione tenta di verificarne la validità usando la funzione `verifyJWT`. In caso di successo, il token viene decodificato per ottenere l'ID dell'utente. Successivamente, viene eseguita una ricerca nel database per trovare l'utente associato all'ID estratto dal token. Se l'utente non viene trovato, viene restituito un errore `401` con il messaggio `Utente non autorizzato`. Se tutte le operazioni hanno esito positivo, la funzione aggiunge i dati dell'utente (ID e username) all'oggetto `req.user` e permette alla richiesta di proseguire. Se si verifica un errore in qualsiasi fase (come un token non valido o problemi con il database), viene restituito un errore `401` con un messaggio di errore specifico. Questa soluzione di autenticazione permette di proteggere le risorse dell'applicazione e garantire che solo gli utenti legittimi possano accedervi.

```
const authenticate = async (req, res, next) => {
    const token = req.headers.authorization?.split('
        ')[1];

    if (!token) {
        return res.status(401).json({ message: 'Token
            mancante o non valido.' });
    }

    try {
        const decoded = verifyJWT(token);
        const user = await User.findById(decoded.id);

        if (!user) {
            return res.status(401).json({ message:
                'Utente non autorizzato.' });
        }

        req.user = { id: user._id, username:
            user.username };
```

```
        next();
    } catch (error) {
        res.status(401).json({ message: 'Token non
            valido.' });
    }
};
```

## 4.6 Test

Le funzionalità del sistema sono state testate su diverse piattaforme, inclusi browser come Chrome e Firefox, per garantire una corretta portabilità. Inoltre, l'applicazione è stata progettata seguendo un approccio mobile-first, per cui è stata testata anche su dispositivi mobili. L'obiettivo dei test è stato verificare che tutte le funzionalità principali e i task fondamentali funzionassero in modo consistente su tutti i dispositivi e browser, con particolare attenzione alla corretta visualizzazione su piattaforme diverse. Per quanto riguarda le API sviluppate lato server, sono state testate utilizzando strumenti come Postman, per garantire che rispondessero come previsto prima dell'integrazione con il front-end. Solo dopo aver validato il corretto funzionamento delle API, è stato eseguito il test finale per verificare l'integrazione completa e la funzionalità dell'applicazione.

- **Corrispondenza tra sistema e mondo reale:** Vengono utilizzati termini e icone che sono familiari agli utenti, richiamando quelli più comuni nei social network. Questo facilita la comprensione e l'orientamento dell'utente nell'interfaccia.
- **Controllo e libertà:** Viene ridotto al minimo il numero di operazioni necessarie per completare ogni task, assicurando che l'utente possa navigare nel sistema in modo rapido e senza frustrazioni.
- **Consistenza e standard:** Viene definito un set di colori coerente e uniforme in tutto il sistema, indirizzando l'attenzione dell'utente sugli elementi principali, creando un'esperienza visiva armoniosa e comprensibile.
- **Flessibilità ed efficienza d'uso:** Viene assicurato che le funzionalità del sistema siano semplici e intuitive, eliminando la necessità di scorciatoie complesse o modalità avanzate, e rendendo l'utilizzo del sistema facilmente accessibile a tutti.
- **Design ed estetica minimalista:** Viene adottato il principio delle regole KISS (Keep It Simple and Stupid) per garantire che l'interfaccia sia chiara, semplice e facile da usare, favorendo l'interazione senza distrazioni inutili.
- **Prevenzione dell'errore:** Viene garantito che l'interfaccia eviti situazioni ambigue per l'utente, minimizzando la possibilità di errori e aumentando l'affidabilità del sistema.

# Chapter 5

# Deployment

Per installare l'applicativo è necessario seguire le seguenti istruzioni:

1. **Clonazione repository:** https://github.com/gloriasemprini/dockerApp.git
2. Tramite una Shell Bash spostarsi all'interno della cartella dell'elaborato.
3. Eseguire i seguenti comandi:
   (a) cd ./webapp-frontend/
   (b) npm install
   (c) cd ..
   (d) cd ./webapp-backend/
   (e) npm install
   (f) npm run server
   (g) cd ..
   (h) cd ./webapp-frontend/
   (i) npm start

Ora è possibile collegarsi alla piattaforma tramite un browser: http://localhost:3000/
.

# Chapter 6

# Conclusioni

Conclusioni Il progetto ha portato alla realizzazione di un sistema completo e funzionale che integra diverse tecnologie moderne per offrire un'esperienza utente fluida e intuitiva. L'adozione di tecnologie come il MEAN stack per il backend, e strumenti come Postman per i test delle API, ha permesso di costruire una base solida per l'applicazione, con un sistema di autenticazione sicuro tramite token JWT e gestione delle risorse tramite servizi dedicati. Il sistema backend è stato sviluppato con un'architettura solida, mentre il frontend è stato ottimizzato per garantire un'interfaccia reattiva e facilmente navigabile, anche su dispositivi mobili. Il processo di testing ha garantito che tutte le funzionalità principali del sistema fossero operative su diverse piattaforme, mentre la gestione delle API e dell'autenticazione è stata verificata per assicurare la sicurezza e la robustezza dell'applicazione.

### 6.0.1 Commenti

A causa di limitazioni temporali il progetto è stato ridimensionato notevolmente rispetto a quanto inizialmente previsto, dando maggiore priorità alla produzione di codice di qualità anzichè ad un numero maggiore di funzionalità.