Mapsage

Applicazioni e Servizi Web

Federico Orazi - 0000928050
{federico.orazi@studio.unibo.it}

Matteo Andruccioli - 0000906869
{matteo.andruccioli2@studio.unibo.it}

14 Luglio 2020

Indice

Capitolo 1

Introduzione

Il progetto nasce dall'esigenza di realizzare una piattaforma che faciliti l'individuazione di massaggiatori da parte di potenziali clienti. Attualmente, non è presente sul mercato una piattaforma dedicata a tale scopo.

L'obiettivo del sistema è quello di consentire ad un utente di individuare i massaggiatori che si trovano in una città specificata ed eventualmente di potersi mettere in contatto con essi e accordarsi per una prestazione.

La Web Application si rivolge a due tipologie di utenti:

- **Massaggiatore**: professionista che esercita in libera professione presso un proprio studio del quale vuole condividere la posizione;
- **Utente**: qualsiasi potenziale cliente alla ricerca di un massaggiatore in una specifica area.

L'applicazione si presenta come un social network orientato alle specifiche categorie di utenti sopra indicate. Come tale, dispone di:

- una sistema di messaggistica istantanea tra un massaggiatore e un proprio cliente;
- un sistema di notifiche che permette ai massaggiatori di comunicare annunci;
- un sistema di geo-localizzazione che permette di associare un massaggiatore alla posizione del proprio studio.

Capitolo 2

Requisiti

Per poter ottenere un elenco di requisiti utile e che rispecchi gli interessi di un potenziale utente, è stata eseguita un'intervista con un massaggiatore. Riportiamo di seguito un estratto dell'intervista.

- *"Vorrei potermi registrare e indicare il nome, la posizione e il numero di telefono del mio studio"*,
- *"Vorrei che un potenziale cliente interessato ad un massaggio possa contattarmi in privato attraverso una chat per ricevere informazioni o concordare un appuntamento"*,
- *"Se mi capita di voler pubblicare un avviso o un offerta, vorrei poter informare i miei clienti attraverso il sito"*.

A seguito dell'intervista sono stati individuati i seguenti requisiti.

2.1 Requisiti funzionali

Massaggiatore

- Deve potersi registrare al sito indicando:
  1. le informazioni relative allo studio, ovvero: nome, numero di telefono e posizione dello studio in termini di coordinate geografiche;
  2. email e password;
  3. eventuale immagine del profilo;
  4. una breve presentazione delle proprie esperienze o capacità.

- Le informazioni relative allo studio indicate nel punto precedente devono poter essere modificabili;
- Ha a disposizione una chat per poter comunicare con un utente che lo ha precedentemente contattato;
- Deve poter pubblicare annunci sul proprio profilo.

Cliente

- Deve potersi registrare al sito indicando: nome e cognome, email e password, eventuale immagine del profilo;
- Sia che sia registrato o meno, deve poter effettuare una ricerca dei massaggiatori che si trovano in una certa città;
- Deve poter raggiungere il profilo dei massaggiatori risultanti dall'operazione di ricerca sulla città;
- Una volta raggiunto il profilo di un massaggiatore, se l'utente è loggato, deve poterlo:
  1. contattare tramite chat;
  2. seguire per tenersi aggiornato su eventuali annunci.
- Se l'utente non è loggato e tenta di effettuare una delle operazioni indicate nel punto precedente, deve venire reindirizzato sulla schermata di login.

Sistema

- Produce una notifica verso tutti i clienti che seguono un massaggiatore quando quest'ultimo pubblica un annuncio;
- Produce una notifica lato destinatario quando un messaggio viene inviato tramite chat;
- Permette la visualizzazione e la specifica delle posizioni geografiche attraverso una mappa;
- Visualizza su una mappa i risultati delle ricerche di massaggiatori in una certa città.

2.2 Requisiti tecnologici

Il requisito tecnologico su cui ci si è concentrati maggiormente è la **reattività**. Quest'ultima è un aspetto fondamentale nella gestione delle notifiche relative agli annunci e dei messaggi inviati sulla chat.

Capitolo 3

Design

Il sistema è stato realizzato seguendo un modello iterativo basato su **UCD (User Centered Design)** con utenti "virtualizzati". Per prima cosa, sono stati individuati i target, a partire dai quali sono state create una serie di **Personas** che sono poi state utilizzate per capire come le funzionalità dell'applicazione dovessero rispondere alle caratteristiche degli utenti.

Il modello classico UCD è stato integrato con alcune caratterische tipiche del modello **AGILE**. In particolare, sono stati svolti briefing giornalieri per la suddivisione del lavoro sulla base dello stato di avanzamento del progetto. In questo modo, i membri del team hanno potuto partecipare attivamente allo sviluppo delle varie funzionalità. Nonostante non siano state eseguite concretamente delle release periodiche verso un cliente, durante lo sviluppo del progetto ci si è attenuti il più possibile al principio **MVP (Minimum Viable Product)**: inizialmente è stata realizzata una versione minimale del sistema alla quale sono state via via integrate funzionalità aggiuntive in modo da avere ad ogni step un prodotto funzionante e testabile.

Dal punto di vista del design delle interfacce si è cercato di allinearsi il più possibile ai principi **KISS** e **"Less Is More"** con l'obiettivo di realizzare un'interfaccia semplice nell'utilizzo da parte dell'utente e allo stesso tempo funzionale. Inoltre, per rendere le interfacce utilizzabili da qualsiasi dispositivo e migliorare la User Experience, sono stati applicati i principi alla base del responsive design.

3.1 Target User Analysis

Sono stati individuati i seguenti target:

- **Massaggiatore**
- **Cliente**

Personas: Giorgio

Giorgio è un massaggiatore fisioterapista che esercita in libera professione in uno studio privato.

- **Contesto A**: Giorgio si è messo in proprio da poco tempo e deve trovare un modo per farsi conoscere al pubblico.

Scenario d'uso A	
1	Giorgio inserisce i propri dati personali
2	Giorgio inserisce le proprie esperienze
3	Giorgio inserisce la posizione del suo studio
4	Giorgio si registra al sito
5	Giorgio mantiene il suo profilo aggiornato in modo che chiunque possa visualizzarlo e conoscere tutto ciò che professionalmente c'è da sapere su di lui
6	Giorgio pubblica quotidianamente news sul suo profilo in modo che tutti i suoi follower vengano avvisati

- **Contesto B**: Giorgio si è già registrato al sito Mapsage.

Scenario d'uso B	
1	Giorgio viene contattato tramite il sito Mapsage da un cliente tramite la chat
2	Giorgio e il cliente si conoscono e si mettono d'accordo per un appuntamento

Personas: Giulia

Giulia è una ragazza nata nell'era digitale che pratica sport agonistico.

- **Contesto A**: Giulia ha necessità di trovare un fisioterapista in zona che la aiuti a recuperare da un infortunio

Scenario d'uso A	
1	Giulia cerca sul sito Mapsage il massaggiatore più vicino a lei nella sua città
2	Giulia scopre che lo studio di Giorgio si trova poco distante da casa sua
3	Giulia visualizza il profilo di Giorgio e verifica che quest'ultimo si occupi di fisioterapia riabilitativa
4	Giulia si registra al sito inserendo i propri dati personali
5	Giulia contatta Giorgio e chiede un appuntamento per un massaggio

- **Contesto B**: Giulia si è già registrata al sito Mapsage e lo utilizza regolarmente. Giulia è interessata agli annunci pubblicati da un certo massaggiatore.

Scenario d'uso B	
1	Giulia va sul profilo del massaggiatore
2	Giulia inizia a seguire il massaggiatore
3	Giulia riceve una notifica ogni volta che quel massaggiatore pubblica un annuncio

3.2 Story board

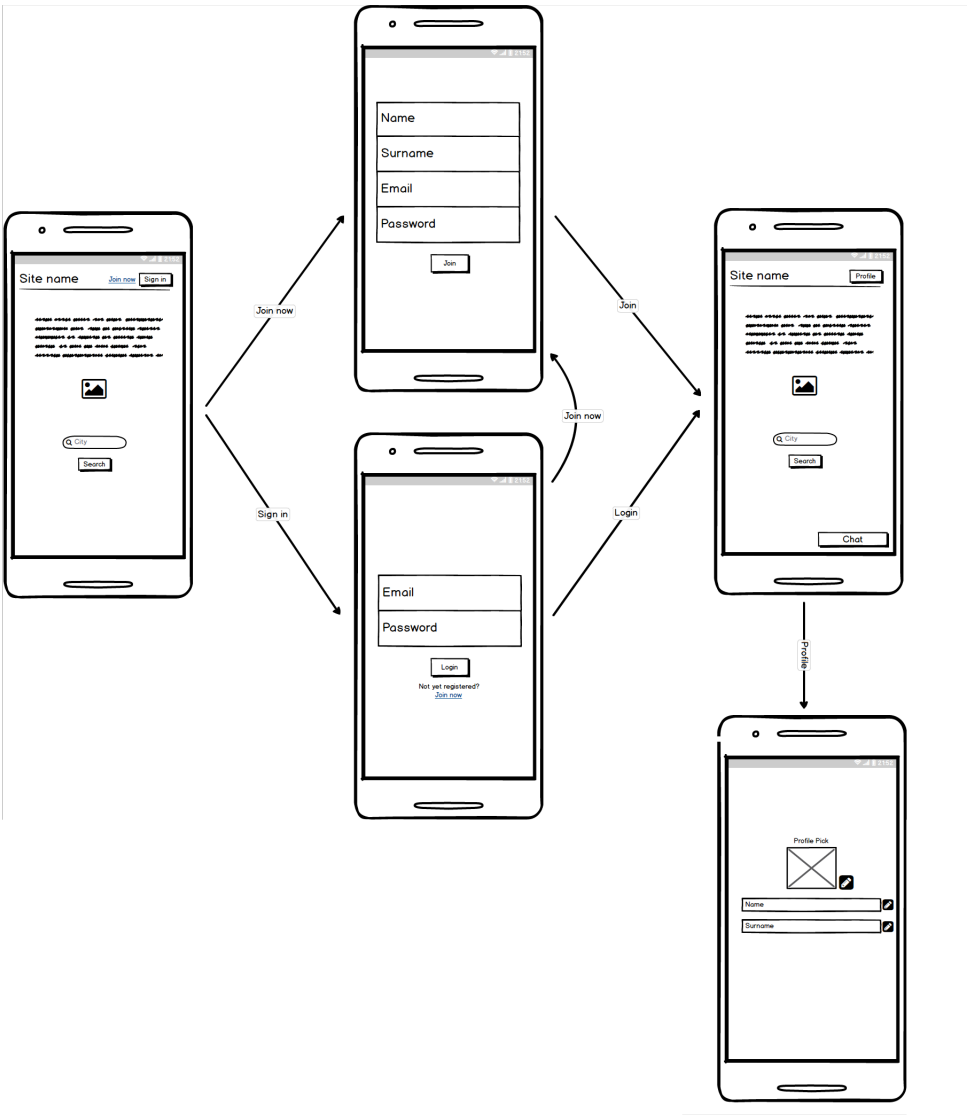

Figura 3.1: Login e registrazione utente

Come mostrato in figura 3.1, l'utente che desidera eseguire il login o registrarsi al sito, può raggiungere le schermate dedicate a partire dalla Home. A login/registrazione effettuata l'utente viene rediretto sulla schermata di Home dalla quale può visualizzare il proprio profilo.

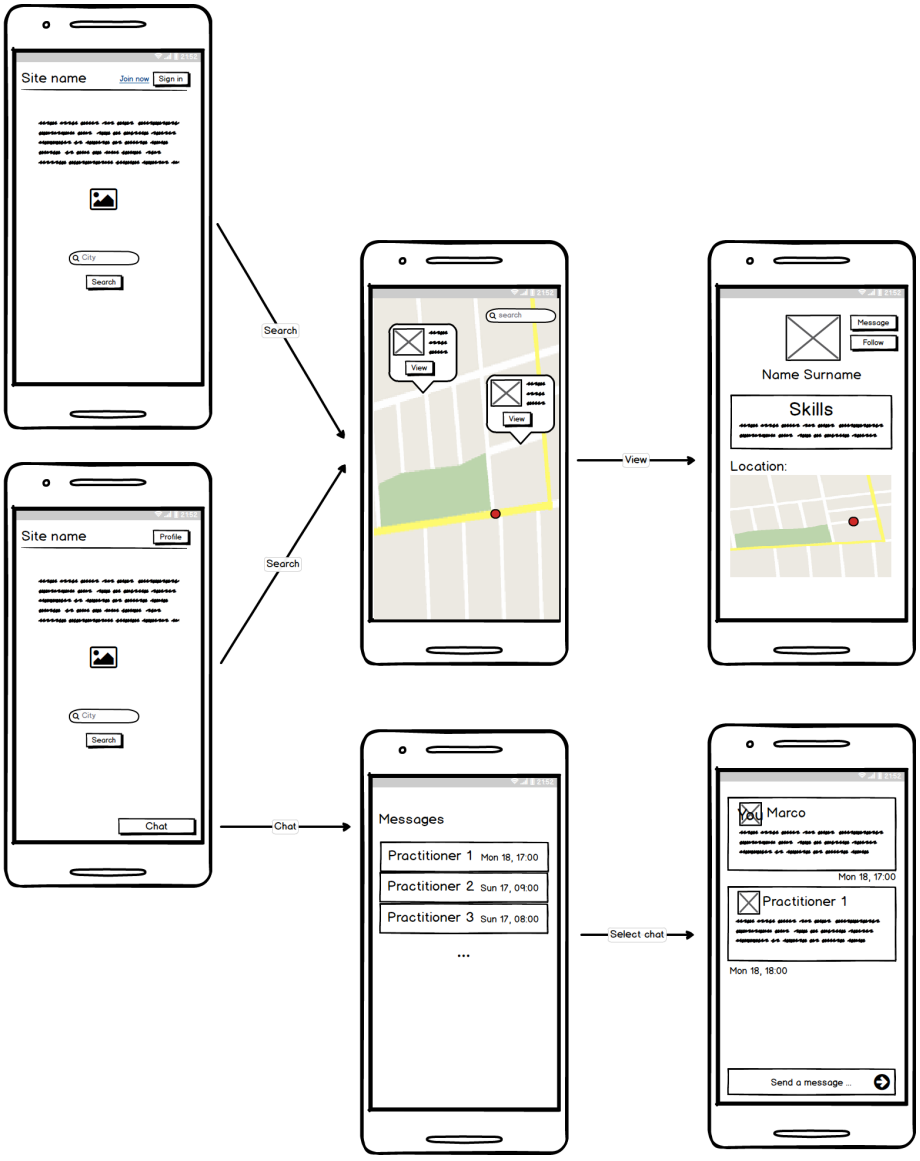

Figura 3.2: Ricerca massaggiatori in città

Come mostrato in figura 3.2, dalla schermata di Home l'utente (loggato o meno) può effettuare una ricerca dei massaggiatori presenti in una specifica città. Le posizioni degli studi dei massaggiatori che praticano in quella città vengono visualizzati su una mappa. Cliccando su uno dei risultati proposti, è possibile raggiungere al profilo del massaggiatore. Se l'utente che ha eseguito la ricerca risulta essere loggato, può aprire una chat con il massaggiatore e scambiare messaggi. In generale, un utente loggato deve sempre avere facile accesso alla lista delle proprie chat.

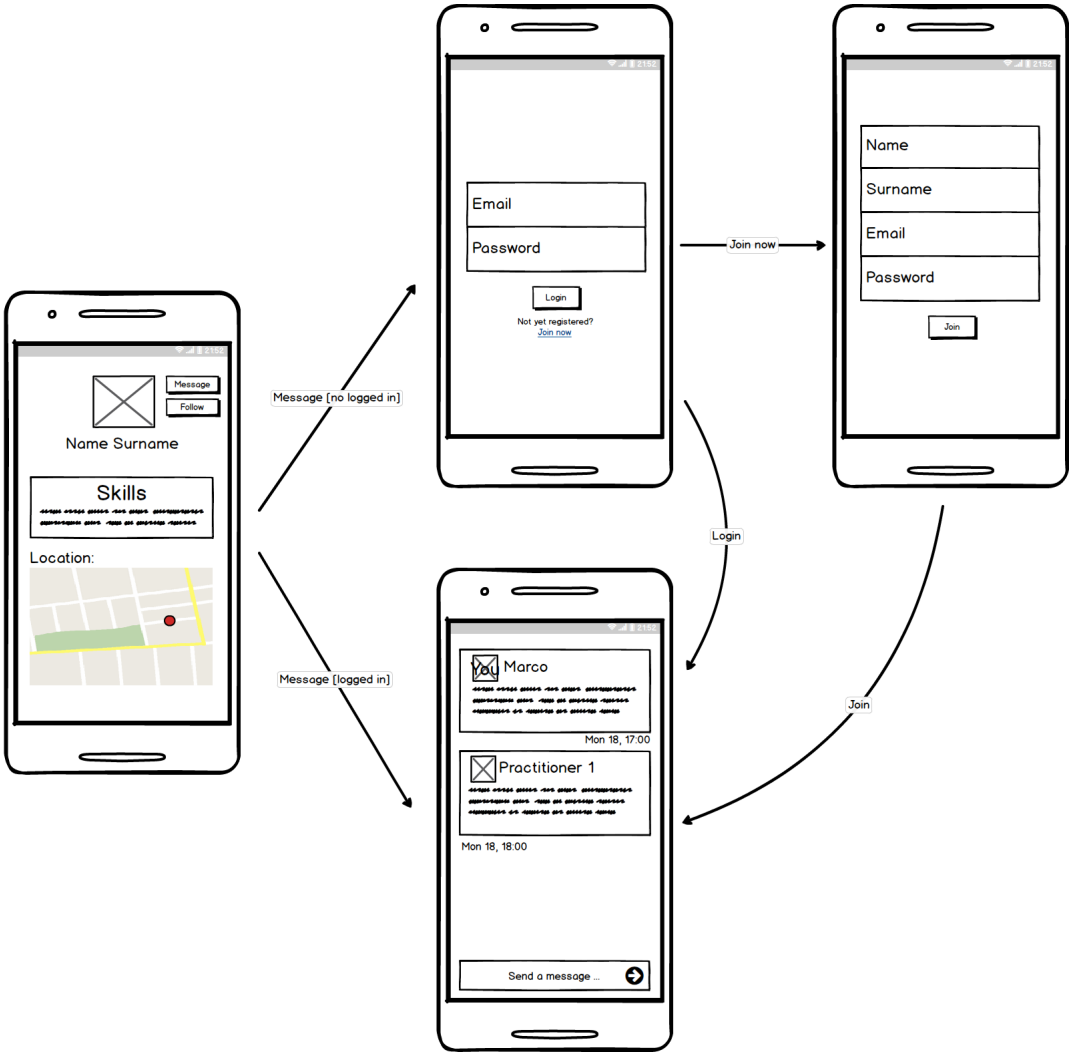

Figura 3.3: Avvio chat da profilo massaggiatore

Come mostrato in figura 3.3, un utente sul profilo del massaggiatore ha a disposizione un pulsante per avviare una chat con quest'ultimo. Nel caso in cui non sia loggato, viene rediretto verso la schermata di login, altrimenti la chat viene aperta.

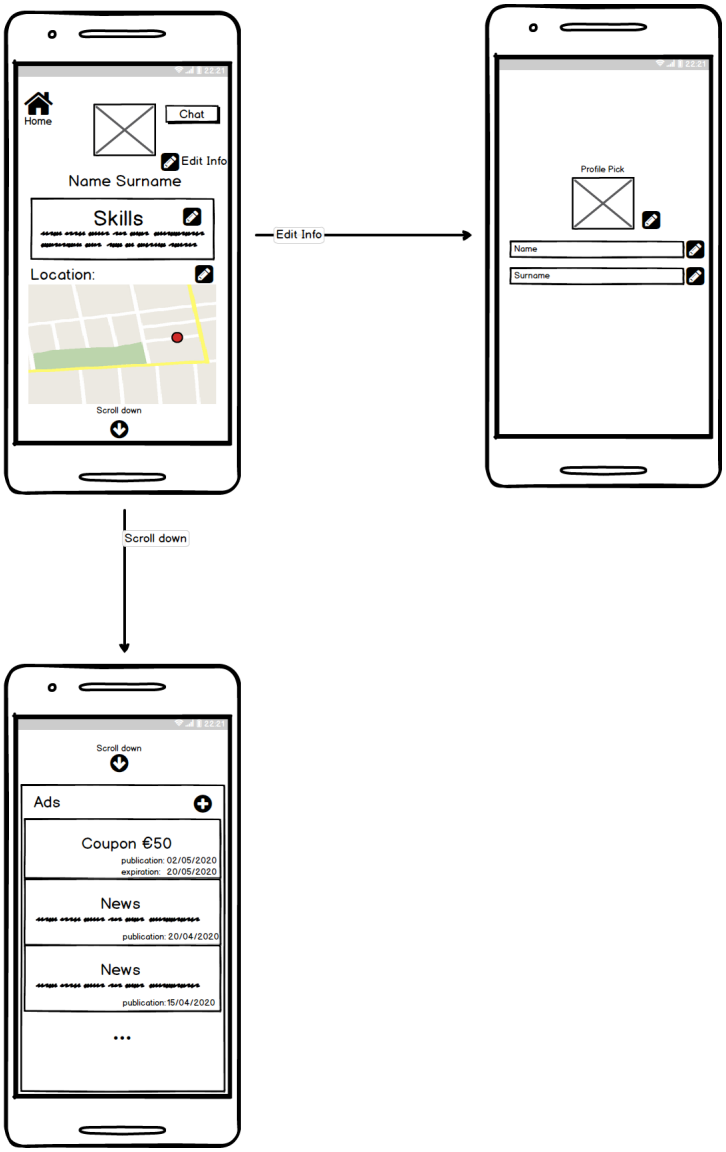

Figura 3.4: Profilo massaggiatore

Come mostrato in figura 3.4, il massaggiatore dopo essersi loggato può accedere al proprio profilo, pubblicare annunci ed eventualmente modificare le proprie informazioni personali.

Da notare che un massaggiatore è a sua volta un utente del sistema e come tale può eseguire ricerche per città a partire dalla home, visitare profili di altri massaggiatori, contattare questi ultimi, seguirli e ricevere notifiche quando pubblicano annunci.

Capitolo 4

Tecnologie

Il sistema è stato realizzato utilizzando il solution stack **MEVN**. Esso è stato scelto poiché da un lato risulta adeguato alla gestione delle funzionalità definite in precedenza e dall'altro è lo stack sul quale il team ha avuto modo di fare maggiore esperienza.

4.1 Supporto alla programmazione

Vue CLI

Poiché la complessità del sistema risulta essere elevata, si è deciso di ricorrere all'utilizzo della Vue CLI (1) che ha consentito una migliore organizzazione e strutturazione del codice lato client. Ulteriori vantaggi apportati sono stati:

- la possibilità di utilizzare a piena potenza le funzionalità Javascript introdotte con ES6;
- la possibilità di concentrare lo sviluppo sui singoli componenti, sia in termini di struttura HTML e Javascript che di stile.

Stile

Nel caso dello stile, ci si è appoggiati al toolkit **Bootstrap 4** (2) che utilizza **Flexbox** per realizzare il proprio Grid Stystem e permettere la realizzazione di siti caratterizzati da un layout fluido. Inoltre, in alcune parti del codice si è fatto direttamente uso del supporto nativo offerto da Flexbox che ha consentito di ottenere un comportamento più predicibile e stabile degli elementi della pagina quando questa doveva adattarsi a display di diversi device con diverse dimensioni.

Persistenza

Per quanto riguarda la persistenza delle informazioni lato server, si è fatto uso della liberia **Mongoose** (3) per interfacciarsi al database MongoDB. Si è scelta questa libreria poiché risulta essere una delle più utilizzate nel suo campo nonché l'unica presentata a lezione.

4.2 Autenticazione e Autorizzazione

Volendo allinearsi alle prassi di sicurezza previste per la memorizzazione delle password, si è deciso di fare uso del modulo **bcrypt** (4) che ha consentito di salvare le password su database in maniera sicura (sotto forma di hash).

Nell'elaborare una richiesta HTTP proveniente da un client, il server fa uso di un **JWT (Jason Web Token)** (5) per garantire allo stesso tempo autenticazione e autorizzazione. Il JWT viene generato dal server al momento della registrazione o del login dell'utente e restituito al client. Quest'ultimo provvederà a presentarlo al server allegandolo ad ogni richiesta futura.

4.3 Comunicazione Client-Server

Per la gestione delle richieste AJAX si fa uso della libreria **axios** (6). Per l'implementazione delle funzionalità real-time quali l'invio dei messaggi in chat e l'invio di notifiche relative alla pubblicazione di annunci da parte di massaggiatori, si è deciso di fare uso della libreria **socket.io** (7) presentata a lezione.

4.4 Geo-localizzazione

In tutti i casi in cui è richiesta la visualizzazione di coordinate geografiche, si ricorre all'utilizzo della libreria **leaflet** (8) la quale consente di visualizzare la mappa e fornisce il supporto di base per la visualizzazione su di essa di layers quali ad esempio: markers, boundaries, ecc.

Per la geo-localizzazione si è fatto uso del plugin open-source **esri-leaflet** (9) che fornisce una serie di componenti, tra cui **esri-leaflet-geocoder** (10), che fanno da intermediario verso le funzionalità messe a loro volta a disposizione dal servizio **ArcGIS** (11). Quest'ultimo è il servizio che realmente si occupa di implementare le funzioni di geo-localizzazione.

All'atto della ricerca di massaggiatori in una certa città, in aggiunta alle modalità di ricerca classiche, ovvero quelle che prevedono l'inserimento di un indirizzo o del nome della città, si è deciso di consentire anche l'utilizzo del **GPS** per identificare la posizione dell'utente. Nel caso di un massaggiatore, il GPS può essere inoltre utilizzato per indicare o modificare la posizione del proprio studio al momento della registrazione o in caso di aggiornamento del profilo.

Capitolo 5

Codice

5.1 Login e Registrazione

La Web Application riserva alcune funzionalità per i soli utenti loggati. In particolare, tali funzionalità sono:

- l'accesso alla chat per lo scambio di messaggi;
- la possibilità di seguire un massaggiatore e ricevere notifiche.

Queste esigenze si traducono nelle seguenti necessità:

- lato server, deve esistere un meccanismo di identificazione e autorizzazione che permetta di riconoscere il mittente di una richiesta e, di conseguenza, valutare il suo diritto di accedere alla funzione desiderata;
- lato client, deve essere possibile capire se l'utente è loggato o meno e a quale tipologia appartiene (cliente o massaggiatore).

Inizialmente, si era pensato di utilizzare un **cookie di sessione**. Tale opzione non consentiva di soddisfare entrambe le esigenze, in particolare non permetteva di accedere lato client alle informazioni contenute nel cookie. Per risolvere il problema, si è pensato di introdurre un ulteriore cookie creato dal client (dopo l'autenticazione) che contenesse le informazioni essenziali sull'utente e che fosse accessibile dal frontend. Tale cookie viene creato e successivamente acceduto utilizzando il modulo **vue-cookies** (12).

La soluzione sopra indicata è efficace, tuttavia non tiene in considerazione la possibilità da parte di un eventuale attaccante di creare un cookie di sessione ad hoc per impersonare un utente del sistema. Per risolvere il problema serebbero necessari:

- una connessione sicura (utilizzando il protocolo HTTPS);
- un token di autenticazione.

In quest'ottica, si è deciso di ricorrere ad un **Jason Web Token** che viene creato lato server al momento del login o della registrazione dell'utente e restituito al client. Per evitare attacchi di Cross Site Scripting (XSS) il JWT viene restituito al client all'interno di un **cookie HTTP-Only** che va a sostituire il cookie di sessione citato in precedenza. Tale cookie verrà allegato automaticamente alle successive richieste eseguite dal client e il server potrà recupare il JWT in esso contenuto e procedere alla verifica della firma.

```
1  if (bcrypt.compareSync(req.body.password, user.password)) {
2      const payload = { _id: user._id }
3      // JWT generation
4      const token = jwt.sign(payload, process.env.SECRET_KEY, {
5          expiresIn: 30 * 86400
6      })
7      const cookieConfig = {
8          httpOnly: true,
9          maxAge: 30 * 86400 * 1000, // 30 days cookie
10         signed: true
11     }
12     res.cookie('jwt', token, cookieConfig)
13     res.json({ profile_type: user_type, _id: user._id })
14 } else {
15     console.log("wrong password")
16     res.json({ error: 'Wrong password' })
17 }
```

Listing 5.1: Esempio di utilizzo JWT e bcrypt server-side

Poiché il sito non utilizza il protocollo HTTPS, rimane comunque vulnerabile ad attacchi di Man-In-The-Middle e Eaves-Dropping.

Volendo allinearsi alle prassi di sicurezza previste per la memorizzazione delle password, si è deciso di criptarle. In fase di registrazione, la password viene criptata lato client prima di inviarla attraverso una richiesta HTTP, poi nuovamente criptata lato server prima di memorizzarla sul database. In fase di login, la password viene criptata lato client prima dell'invio, poi nuovamente criptata lato server e confrontata con quella memorizzata su database. Le operazioni di cifratura sono state eseguite utilizzando il modulo **bcrypt** e SHA512 come algoritmo di hashing.

Nel momento in cui un utente decide di effettuare il logout, il client elimina il cookie da esso precedentemente creato, dopodiché esegue una apposita richiesta al server per distruggere il cookie HTTP-Only contenente il JWT.

```
try {
    jwt.verify(token, process.env.SECRET_KEY);
    res.clearCookie("jwt");
    res.json({ description: "Logout succeded" })
} catch (error) {
    res.sendStatus(401); // The JWT is not valid
}
```

Listing 5.2: Distruzione cookie HTTP-only server-side

```
axios.get('http://localhost:3000/users/logout', {
   withCredentials: true })
.then(res => {
    if (!res.data.error) {
        this.$cookies.remove('currentUser') // client cookie
   removing
    }).catch(err => {
        // errors management
    })
}
```

Listing 5.3: Distruzione cookie client-side

5.2 Gestione delle immagini

Per quanto riguarda la memorizzazione delle immagini del profilo degli utenti, si è deciso di procedere salvando le immagini sottomesse dai client all'interno di una apposita directory *"uploads"* presente lato server all'interno della cartella *"public"* referenziata staticamente. Se un utente, all'atto della registrazione, specifica un immagine, questa viene memorizzata all'interno di una sub-directory di *"uploads"* creata appositamente per quell'utente e denominata con l'email di quell'utente per garantire univocità (le email all'interno del sistema sono univoche). All'interno del database viene poi memorizzato l'URL HTTP per raggiungere staticamente l'immagine.

Il client codifica le immagini in **base64** e le trasferisce al server all'interno di un oggetto JSON insieme alle altre informazioni dell'utente. Alla ricezione, il server le decodifica e procede alla memorizzazione.

Per una migliore resa visiva delle immagini del profilo, si è deciso di uniformare l'aspect-ratio delle immagini memorizzate sul server. Questo è reso possibile effettuando un ritaglio dell'immagine al momento della selezione da parte dell'utente. Tale ritaglio non avviene in automatico ma l'utente ha il pieno controllo su di esso, infatti ha la possibilità di selezionare la porzione di

immagine da ritagliare. Per fare ciò si è fatto uso della libreria **Cropper.js** (13).

Figura 5.1: Esempio di ritaglio

5.3 Gestione geo-localizzazione

Per la geo-localizzazione si è fatto uso del plugin open-source **esri-leaflet** (9) ed in particolare del componente **esri-leaflet-geocoder** (10) da esso messo a disposizione. Le principali funzionalità supportate da esri-leaflet-geocoder che sono state utilizzate nel sistema, sono:

- **geocoding**, usato per recuperare le coordinate spaziali dato in input il nome di una città;
- **reverse-geocoding**, usato per recuperare il nome della città corrispondente alle coordinate fornite in input;
- **suggestions**, usato per recuperare e fornire suggerimenti all'utente mentre questo sta inserendo il nome di una città, di una via, ecc.

In particolare, la funzione di reverse-geocoding è stata utilizzata per rendere possibile l'utilizzo del GPS per individuare la posizione dell'utente che vuole eseguire la ricerca di massaggiatori nella città in cui si trova. Nel caso invece del massaggiatore che desidera modificare la posizione del proprio studio al momento registrata nel sistema con la posizione attualmente ricoperta, vengono utilizzate direttamente le coordinate restituite dal GPS. Per usufruire del GPS si è fatto direttamente uso del supporto nativo offerto dalla HTML Geolocation API ed in particolare dell'oggetto **Navigator** e della proprietà *"geolocation"* di quest'ultimo.

```
if(!("geolocation" in navigator)) {
    alert('Error: geolocation is not available.');
    return;
}
navigator.geolocation.getCurrentPosition(pos => {
    geocoding.reverseGeocode()
        .latlng(L.latLng(pos.coords.latitude, pos.coords.
    longitude))
        .run(function (error, result) {
            // result handling - result contains city info
        });
}, err => {
    // errors handling
})
```

Listing 5.4: Esempio d'uso del GPS abbinato a reverse-geocoding

Ogni volta che nell'applicazione è richiesto l'utilizzo della mappa, si fa uso dello stesso componente *"MapPanel"*. Esso contiene la mappa Leaflet e ogni volta che il componente viene usato, la mappa viene inizializzata sempre allo stesso modo. È poi possibile personalizzare la configurazione di base della mappa a seconda delle esigenze specifiche. Le personalizzazioni vengono innescate dal componente padre attraverso l'uso di props.

GeoJSON e ricerca per città

Nell'eseguire la memorizzazione su database delle coordinate spaziali relative alle posizioni degli studi dei massaggiatori, si è deciso di allinearsi al formato GeoJSON. Questo ha portato ad una serie di vantaggi notevoli poiché sia Leaflet che MongoDB offrono un supporto diretto a questo tipo di formato. Nel caso di Leaflet il supporto rende molto semplice la creazione di layers anche molto complessi senza il bisogno di particolari elaborazioni mentre nel caso di MongoDB esiste la possibilità di utilizzare **query geospaziali** per restituire risultati sulla base delle coordinate memorizzate.

```
const geoJson = L.geoJSON({
    type: "FeatureCollection",
    features: coordinates // collection of GeoJSON objects
}, {
    onEachFeature: function(feature, layer) {
        // GeoJSON objects management
    }
});
geoJsonLayer.addLayer(geoJson) // geoJsonLayer is a map layer
```

Listing 5.5: Esempio di uso del supporto GeoJSON di Leaflet

Quando un massaggiatore, dopo aver specificato la posizione del propio studio, si registra al sistema, il client invia direttamente le coordinate geo-spaziali al server insieme alle altre informazioni relative al massaggiatore che si vuole registrare. Sarà quindi il server, prima di memorizzare il nuovo utente sul database, a utilizzare tali coordinate per creare l'oggetto GeoJSON da salvare su DB.

```
const location = { // GeoJSON
    type: 'Feature',
    geometry: {
      type: 'Point',
      coordinates: req.body.coordinates
    },
    properties: {
      brand_name: req.body.brand_name,
      profile_picture: profileImagePath
    }
}
```

Listing 5.6: Creazione del GeoJSON server-side

Il sistema dà la possibilità ad un utente di specificare una città e di eseguire una ricerca dei massaggiatori il cui studio si trova all'interno dei confini di quest'ultima. Attualmente, il sistema supporta la maggior parte dei Comuni italiani. I confini dei Comuni supportati sono memorizzati in formato GeoJSON all'interno di una collection presente nel database MongoDB che è stata reperita dal seguente repository: `https://github.com/openpolis/geojson-italy`.

Le coordinate geo-spaziali della posizione specificata dall'utente al momento della ricerca insieme alle coordinate relative ai confini dei Comuni e a quelle relative le posizioni degli studi dei massaggiatori, vengono utilizzate per eseguire consecutivamente due geospatial query MongoDB:

- la prima, recupera dalla collection dei Comuni i confini del Comune a cui fanno riferimento le coordinate della posizione cercata dall'utente (che non necessariamente deve essere il nome specifico del Comune, ma può essere ad esempio anche il nome di uno dei suoi viali);
- la seconda, recupera dalla collection dei Massaggiatori tutti i massaggiatori che hanno uno studio posizionato all'interno di tali confini.

Un esempio di tali query viene mostrato di seguito. Nel primo caso (5.7) ne viene mostrato un uso in MongoDB, nel secondo caso (5.8) viene mostrato l'utilizzo concreto che è stato adottato lato server utilizzando Mongoose.

```
var municipality = db.municipalities.findOne({
    geometry: {
        $geoIntersects: {
            $geometry: {
                type: "Point",
                coordinates: [ 12.647171, 43.998850 ]
            }
        }
    }
})
db.masseurs.find({
    "location.geometry": {
        $geoWithin: {
            $geometry: municipality.geometry
        }
    }
})
```

Listing 5.7: Esempio di geospatial query MongoDB

```
Municipality.findOne({
    geometry: {
        $geoIntersects: {
            $geometry: {
                type: type,
                coordinates: coordinates
            }
        }
    }
}).then(municipality => {
    if (municipality !== null) {
        Masseur.find({
            "location.geometry": {
                $geoWithin: {
                $geometry: municipality.geometry
                }
            }
        }).then(masseursList => {
        // results handling and response to the client
        }).catch(err => /* errors handling */ })
    } else {
        // no municipalities found
    }
}).catch(err => /* errors handling */ })
```

Listing 5.8: Uso delle geospatial query con Mongoose

5.4 Gestione aspetti real-time

Chat

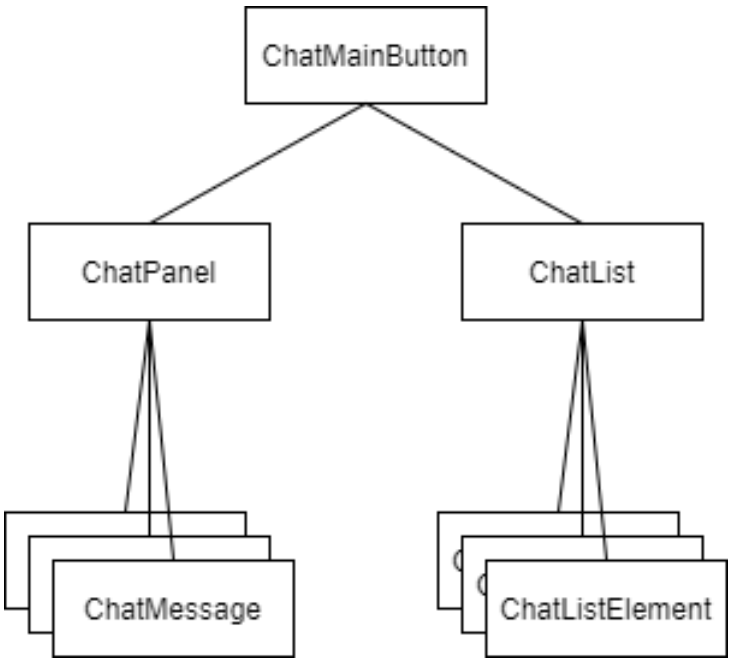

Figura 5.2: Gerarchia componenti chat

Il componente chat deve mettere a disposizione due funzionalità principali:

- visualizzazione dell'elenco delle chat attive per l'utente loggato;
- visualizzazione della chat specifica e invio di messaggi verso un altro utente.

Poiché la chat risulta essere una componente complessa, si è deciso di suddividerla gerarchicamente in sotto-componenti come mostrato in Figura 5.2. In particolare, *"ChatList"* è il sotto-componente che permette la visualizzazione dell'elenco delle chat attive (*"ChatListElement"*) mentre *"ChatPanel"* è quello che consente di visualizzare la chat specifica e fornisce la possibilità di inviare messaggi (*"ChatMessage"*).

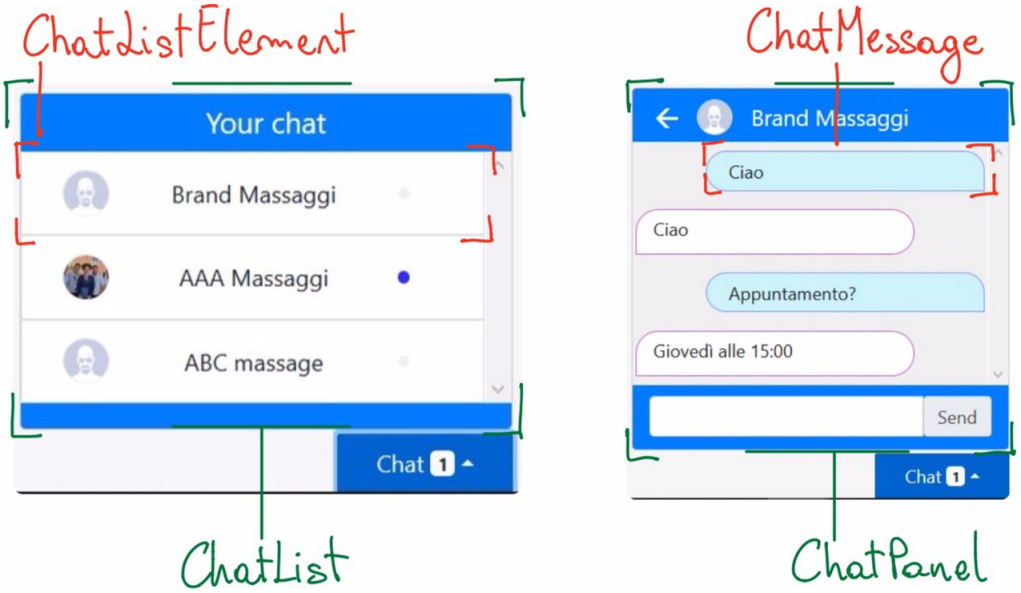

Figura 5.3: Vista componenti chat

I due sotto-componenti principali sono gestiti dal componente *"ChatMainButton"* che si occupa anche di tutte le interazioni (via socket.io o axios) che avvengono con il server. Le risposte provenienti dal server vengono propagate ai sotto-componenti attraverso l'uso di props, mentre le interazioni dell'utente con questi ultimi vengono propagate al componente padre attraverso eventi. Di seguito vengono proposti una serie di diagrammi che mostrano nel dettaglio come avvengono le comunicazioni.

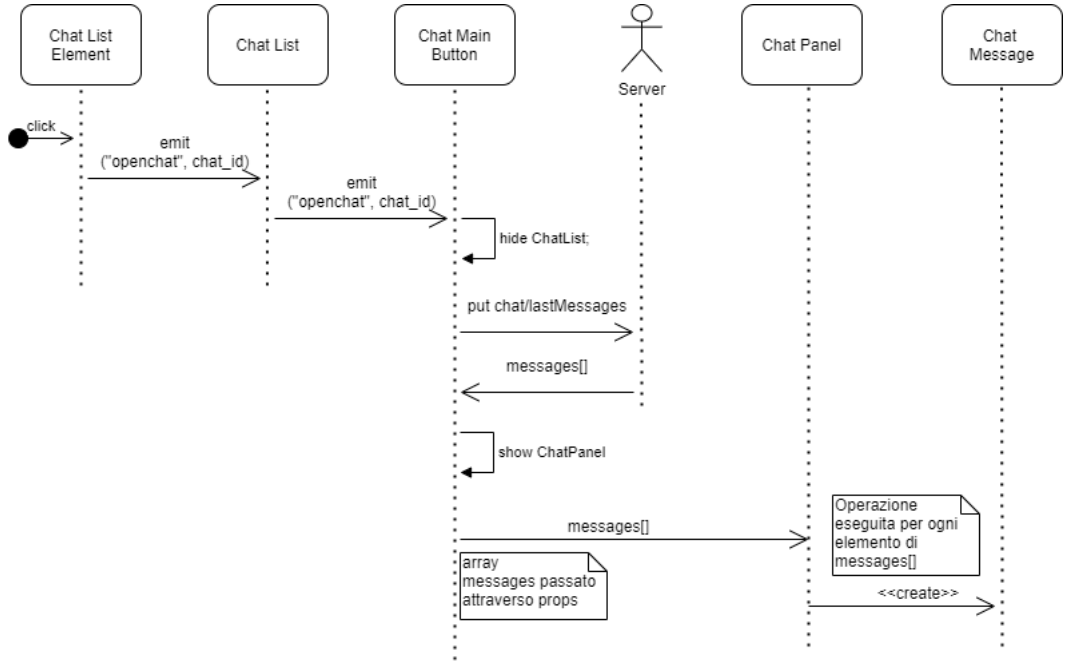

Figura 5.4: Download messaggi di una chat selezionata

Quando l'utente seleziona la chat che vuole aprire dall'elenco delle chat attive, il componente padre *"ChatMainButton"*:

- nasconde la *"ChatList"*;
- esegue il download dei messaggi correlati alla chat selezionata,
- visualizza il *"ChatPanel"* inviando ad esso i messaggi appena scaricati in modo che li possa visualizzare.

Notare che i messaggi non vengono scaricati nello loro totalità ma ci si limita solo ai più recenti. L'utente ha comunque la possibilità di scaricare anche i messaggi precedenti attraverso successive chiamate axios.

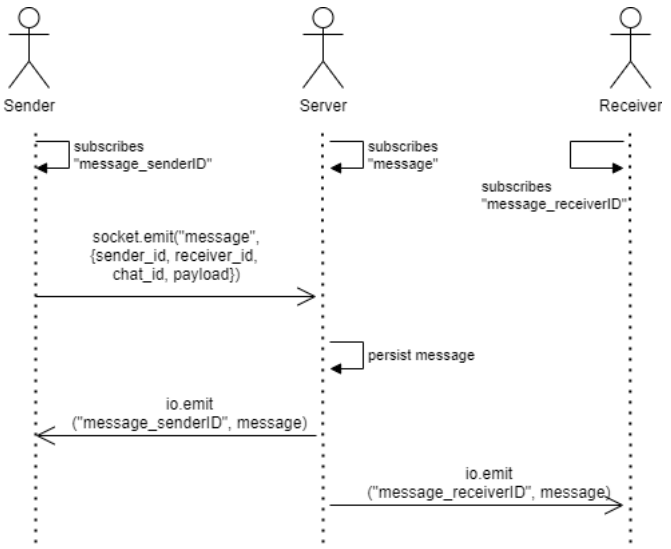

Figura 5.5: Messaggistica istantanea

Per quanto riguarda lo scambio istantaneo dei messaggi l'interazione con il server avviene attraverso **socket.io**. Ogni utente si sottoscrive ad un canale denominato *"message_userID"* (ad esempio: l'utente con *"userID=1"* si sottoscrive al canale *"message_1"*) e su questo canale attende la ricezione di eventuali messaggi rediretti dal server.

Il server si sottoscrive ad un canale denominato *"message"* sul quale ogni mittente invia i messaggi. Ogni volta che viene ricevuto un messaggio su questo canale, esso viene memorizzato sul database e poi rediretto sia verso il destinatario che verso il mittente per la visualizzazione.

News

Il componente delle notifiche deve consentire:

- la visualizzazione di una notifica ogni volta che un massaggiatore pubblica un nuovo annuncio;
- la possibilità di raggiungere il profilo del massaggiatore che ha pubblicato un certo annuncio cliccando sulla notifica relativa a quest'ultimo.

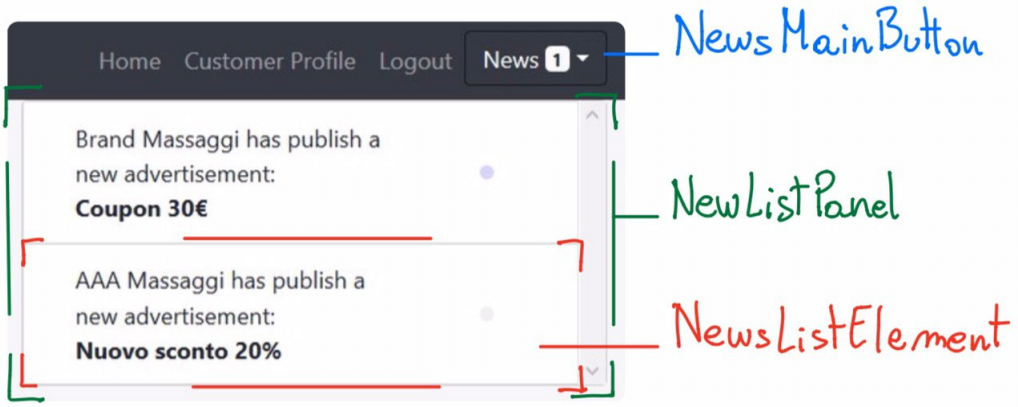

Figura 5.6: Vista componenti news

Come nel caso delle chat, anche per le notifiche si decide di strutturare i componenti in maniera gerarchica mantenendo nel componente padre *"NewsMainButton"* la logica di interazione con il server. Di seguito vengono proposti una serie di diagrammi che mostrano nel dettaglio come avvengono le comunicazioni.

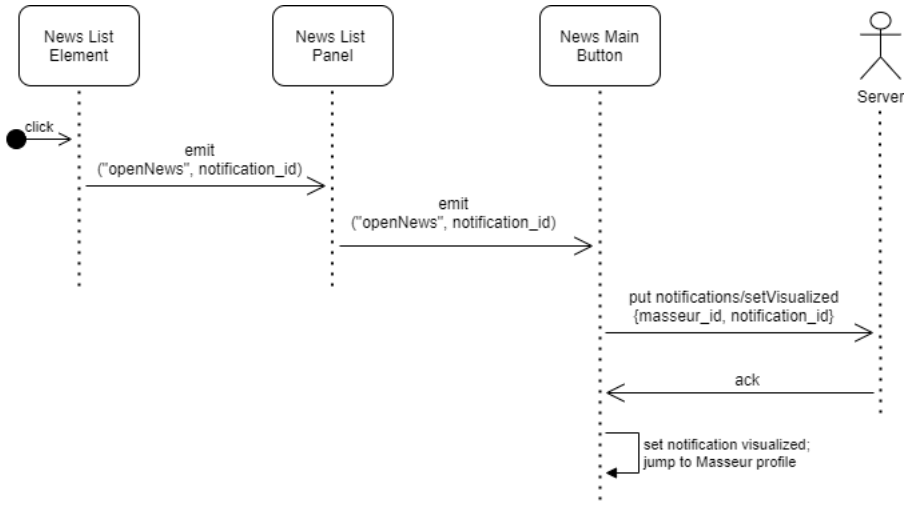

Figura 5.7: Click su una notifica

Quando un utente clicca su una notifica, essa deve venire etichettata sul database come visualizzata e l'utente deve essere rediretto sulla pagina della massaggiatore che ha pubblicato quella notifica.

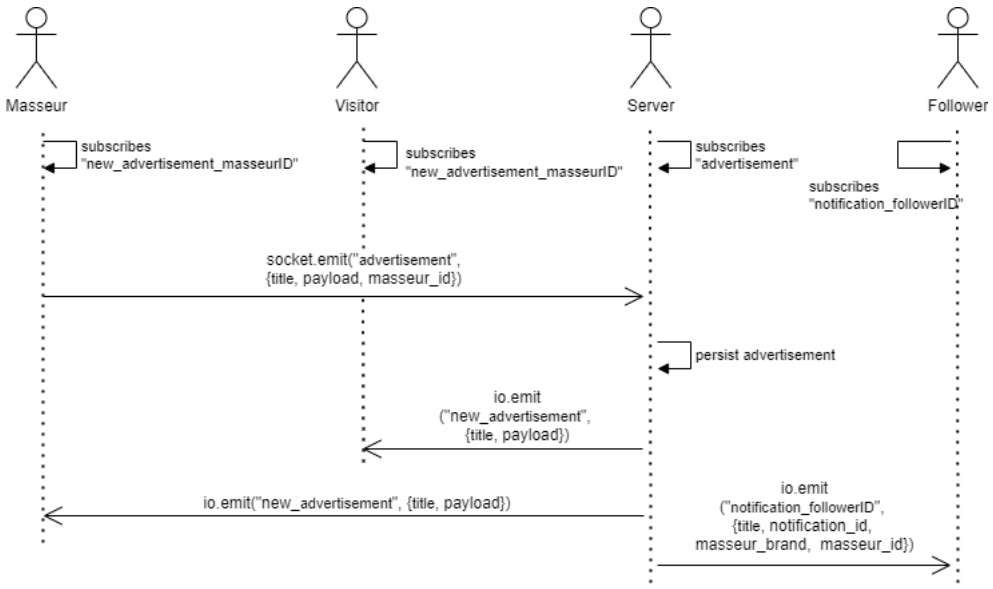

Figura 5.8: Gestione notifiche

Per quanto riguarda la gestione degli annunci, l'interazione con il server avviene attraverso **socket.io**. Ogni volta che un utente visita il profilo di un massaggiatore, si sottoscrive ad un canale denominato *"new_advertisement_masseurID"* (ad esempio: se il massaggiatore ha *"masseurID=1"*, l'utente si sottoscrive al canale *"new_advertisement_1"*) e su questo canale attende la ricezione di eventuali annunci rediretti dal server. Tale canale viene distrutto quando l'utente esce dal profilo. Notare che anche il massaggiatore stesso, quando accede al proprio profilo, si registra allo stesso canale. Questo permette al massaggiatore che pubblica un nuovo annuncio e a tutte le persone sul suo profilo (followers e non) di visualizzare l'annuncio.

I follower di un massaggiatore si sottoscrivono ad un canale dedicato per ricevere le notifiche di pubblicazione annunci. Tale canale è denominato *"notification_followerID"*.

Il server si sottoscrive ad un canale denominato *"advertisement"* sul quale ogni massaggiatore invia i nuovi annunci. Ogni volta che viene ricevuto un annuncio su questo canale, esso viene memorizzato sul database e poi rediretto sul canale *"new_advertisement_masseurID"* per la visualizzazione. Inoltre, per ogni follower del massaggiatore, viene generata una notifica relativa all'annuncio che viene memorizzata sul database e poi inviata sul canale *"notification_followerID"*.

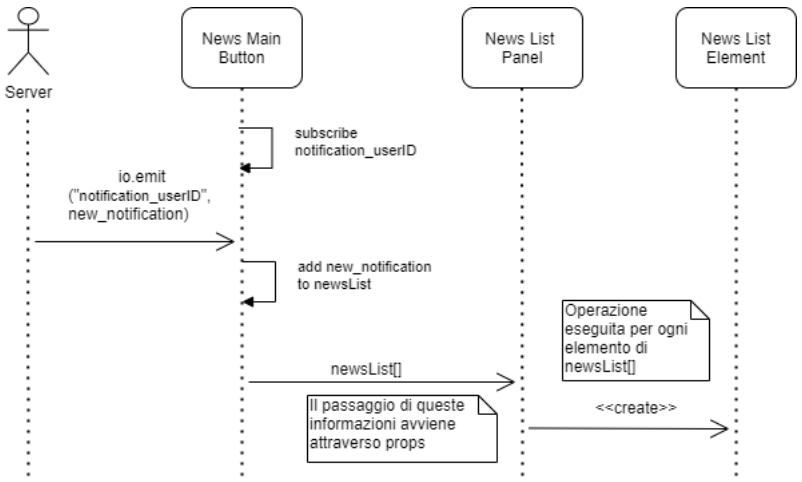

Figura 5.9: Ricezione notifiche

Questo diagramma, che rappresenta la continuazione del precedente, mostra la sequenza di eventi che portano alla visualizzazione di una notifica relativa ad un nuovo annuncio da parte dei followers.

Notare che, come avviene per i messaggi delle chat, anche le notifiche non vengono scaricate nella loro totalità ma ci si limita alle le più recenti.

5.5 Struttura rotte lato Server

Le API esposte dal server sono organizzate sulla base del dominio su cui agiscono. Per questo motivo sono state suddivise in differenti file.

Di seguito la specifica tecnica delle API.

Chat Routes				
Metodo	**Rotta**	**Parametri**	**Risposta**	**Descrizione**
PUT	/chat/lastMessages	chat_id: String firstElement: Number	messages: [{ sender: ObjectId, _id: ObjectId, body: String }]	Restituisce N messaggi della chat a partire da firstElement

PUT	/chat/chatInfo	receiver: String	{
			visualized: Boolean,
			chat_id: ObjectId,
			receiver_fullName: String,
			receiver_imgPath: String,
			receiver_id: ObjectId
			}
				Restituisce info sulla chat indicata da (user_id, receiver_id)

GET	/chat/chatInfo/allOfUser		chats: [{ visualized: Boolean, chat_id: ObjectId, receiver_id: ObjectId, receiver_fullName: String, receiver_imgPath: String }]	Restituisce le informazioni di base di tutte le chat che appartengono a un certo utente

Customer Routes				
Metodo	**Rotta**	**Parametri**	**Risposta**	**Descrizione**

POST	/customers/register	first_name: String,
		last_name: String,
		email: String,
		password: String,
		profile_picture: String,
			{
			_id: ObjectId
			}
				Permette di registrarsi, setta il token jwt

GET	/customers/profile		{
			first_name: String,
			last_name: String,
			email: String,
			profile_picture: String,
			notifications: []
			}
				Ritorna le principali informazioni relative al customer identificato dall'id contenuto nel JWT allegato alla richiesta

Follower Routes				
Metodo	**Rotta**	**Parametri**	**Risposta**	**Descrizione**
PUT	/follow/add	masseur_id: String		Aggiunge l'utente loggato (che ha effettuato la richiesta) alla lista dei follower del masseur indicato dall'id passato come parametro
PUT	/follow/remove	masseur_id: String		Rimuove l'utente loggato (che ha effettuato la richiesta) dalla lista dei follower del masseur indicato dall'id passato come parametro

Masseurs Routes				
Metodo	**Rotta**	**Parametri**	**Risposta**	**Descrizione**
POST	/masseurs/register	brand_name: String, email: String, password: String, phone_number: String, expertise: String, profile_picture: String, coordinates: Array[],	_id: ObjectId	Permette di registrarsi, setta il token jwt
GET	/ masseurs/profile		*oggetto contenente informazioni utente*	Ritorna le principali informazioni relative al customer identificato dall'id contenuto nel JWT allegato alla richiesta
GET	/ masseurs/:id	Id: String	*oggetto contenente informazioni utente*	Ritorna le principali informazioni relative al customer identificato dall'id
POST	/masseurs/masseursByLocation	type = String coordinates = Array[]	cityBoundaries: Array[GeoJSON], masseursLocations: Array[GeoJSON]	Restituisce un oggetto contenente le coordinate dei confini del Comune e le coordinate dei massaggiatori presenti in quel comune
POST	/masseurs/edit	edit_brand_name: String, edit_phone_number: String, edit_expertise: String	*oggetto contenente informazioni utente*	Aggiorna le informazizoni relative al masseur che effettua la richiesta, individuato dall'user id in JWT
PUT	/masseurs/editLocation	edit_brand_name: String, edit_phone_number: String, edit_expertise: String	*oggetto contenente informazioni utente*	Aggiorna le informazizoni relative alla posizione del masseur che effettua la richiesta, individuato dall'user id in JWT

Notifications Routes				
Metodo	**Rotta**	**Parametri**	**Risposta**	**Descrizione**
PUT	/notifications/setVisualized	notification_id: String		Imposta la notifica identificata dal notification_id passato come parametro come visualizzata
GET	/notifications/getSet		notification_id: ObjectId, masseur_id: ObjectId, masseur_brand: String, advertisement_title: String, visualized: Boolean	Ritorna un set di N notifiche a partire da quella con l'id specificato, relative al client identificato dall'id passato come parametro nel JWT

Users Routes				
Metodo	**Rotta**	**Parametri**	**Risposta**	**Descrizione**
POST	/users/login	email: String, password: String	profile_type: String _id: ObjectId	Permette all'utente (Customer o Masseur) di effettuare il log in nel sistema; setta il token jwt
GET	/users/logout			Rimuove il cookie http-only contenente il JWT permettendo il logout dell'utente identificato dall'id passato come parametro nel JWT

5.6 Struttura database

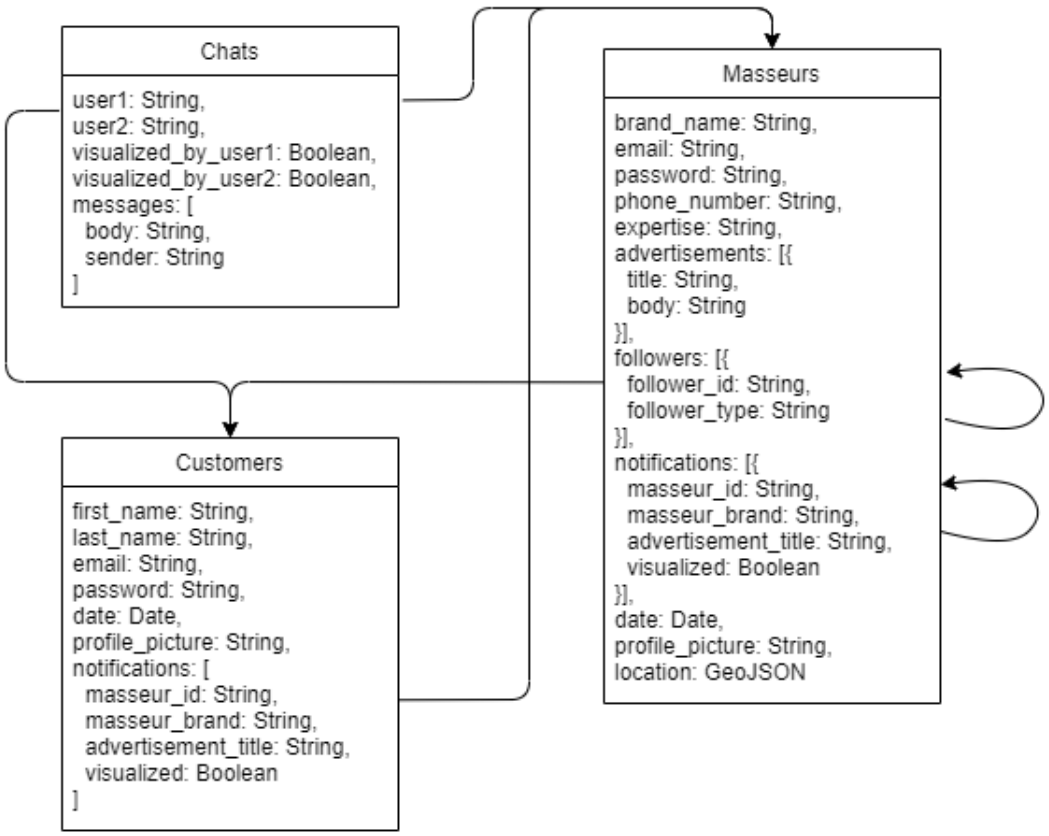

Figura 5.10: Schema MongoDB

La scelta di memorizzare le informazioni relative ad "advertisements", "followers", "notifications" e "location" all'interno degli oggetti a cui fanno

riferimento è giustificata da motivi di performance. Infatti, nella maggior parte dei casi in cui si ha la necessità di accedere a tali informazioni, in realtà è necessario accedere anche alle altre informazioni dell'utente.

Per quanto riguarda le chat, si è deciso di memorizzarle in una collezione a sé stante per vari motivi, tra cui:

- le chat contengono informazioni che interessano due utenti, i quali potenzialmente vi accederanno con la stessa frequenza. Per questo motivo non ha senso privilegiare l'accesso da parte di uno dei due utenti memorizzando le chat nell'oggetto relativo a quest'ultimo;

- ogni volta che si ha la necessità di scaricare i messaggi relativi ad una certa chat, è sufficiente individuare lo specifico oggetto "Chat" attraverso gli "UserID" dei due utenti coinvolti;

- si tratta di una collezione i cui oggetti tenderanno a crescere molto rapidamente e quindi mantenendola separata dalle altre collezioni si ottiene sicuramente un miglioramento dell'organizzazione del database.

Capitolo 6

Test

Le funzionalità del sistema sono state testate dal team sia su browser **Chrome** che su browser **Firefox** con l'obiettivo di garantire portabilità. In particolare, i test hanno voluto verificare che funzionalità complesse come quelle real-time funzionassero allo stesso modo in entrambi i browser.

Le API sviluppate lato server sono state opportunamente testate utilizzando lo strumento **Postman** che ha permesso di verificare che tali API avessero il comportamento desiderato prima di procedere con l'integrazione con il front-end e quindi con il test della funzionalità completa.

6.1 Euristica di Nielsen

Dal punto di vista dell'usabilità dell'interfaccia, il sistema è stato sottoposto preventivamente ad **euristica di Nielsen**. Di seguito si riportano alcune considerazioni emerse.

- **Visibilità dello stato del sistema**: le eventuali latenze presenti danno un feedback visivo all'utente che lo rendono consapevole del fatto che il sistema sta elaborando.
- **Corrispondenza tra sistema e mondo reale**: termini e icone utilizzate nel sistema sono in linea con quelli che caratterizzano i più famosi social network.
- **Controllo e libertà**: si è cercato di ridurre il più possibile il numero di operazioni necessarie all'utente per portare a termine un task. Non esistono diversi modi per svolgere lo stesso task.
- **Consistenza e standard**: i pulsanti che abilitano l'avvio di un task presentano le stesse caratteristiche visive. Definita inizialmente la

gamma dei colori da utilizzare, si è fatto uso uniforme di questi all'interno del sistema in modo da rivolgere l'attenzione dell'utente verso gli elementi principali delle varie schermate.

- **Prevenzione dell'errore**: si è cercato il più possibile di evitare situazioni ambigue per l'utente che lo potessero portare a commettere errori. L'utente è guidato nelle procedure che abilitano l'esecuzione di un task attraverso label, suggerimenti e messaggi di errore.
- **Riconoscimento anziché ricordo**: le immagini utilizzate sono autoesplicative e lo stile di ogni pagina è uniforme per tutto il sistema.
- **Flessibilità ed efficienza d'uso**: le funzionalità offerte dal sistema sono già molto semplici per cui non si è ritenuto necessario introdurre scorciatoie o modalità di utilizzo alternative.
- **Design e estetica minimalista**: KISS è stato il principio chiave che si è deciso di seguire. Lo si nota già dalla schermata di "Home": essa rivolge l'attenzione dell'utente verso la funzionalità principale del sistema ovvero quella della ricerca dei massaggiatori.
- **Aiuto all'utente**: i messaggi di errore che vengono visualizzati sono espressi in un linguaggio comprensibile all'utente.
- **Documentazione**: data la semplicità di utilizzo del sistema non si è ritenuto necessario produrre una guida per l'utente.

6.2 Usability test

Il sistema, una volta realizzato, è stato sottoposto all'attenzione di entrambe le tipologie di utente coinvolte in modo da valutare le loro reazioni e comprendere cosa potrebbe essere migliorato nelle evoluzioni future.

Ad entrambe le categorie di utenti sono stati assegnati un elenco di task da portare a termine. Il team in questa fase ha ricoperto un ruolo di osservatore cercando di intervenire il meno possibile.

Di seguito viene riportato un sunto dei risultati ottenuti.

Massaggiatore

Il massaggiatore intervistato in fase di stesura dei requisiti è stato poi coinvolto nuovamente durante la fase di testing. I task che sono stati individuati per quest'ultimo sono:

- **Task 1**: il massaggiatore deve eseguire una registrazione completa;
- **Task 2**: il massaggiatore, una volta registrato, deve raggiungere il proprio profilo e modificare le informazioni in esso contenute;
- **Task 3**: il massaggiatore deve creare e pubblicare un annuncio;
- **Task 4**: il massaggiatore deve interagire con i clienti che lo hanno contattato attraverso la chat. In questo caso, un membro del team ha assunto il ruolo di cliente.

Il massaggiatore è riuscito a compiere quasi tutti i task senza la necessità di intervento da parte del team. L'unica operazione nella quale il massaggiatore ha incontrato delle difficoltà, poi superate grazie ad un suggerimento, è stata la modifica della posizione dello studio dalla schermata del profilo. I pulsanti presenti sulla mappa che abilitano tale operazione non sono risultati sufficientemente immediati. Per tale motivo, il massaggiatore ha consigliato di rivedere quella parte e renderla più accessibile.

Altre proposte mosse dal massaggiatore riguardano il miglioramento generale della grafica del profilo e l'aggiunta di funzionalità relative agli annunci. In particolare, sono state evidenziate le seguenti necessità:

- poter aggiungere una scadenza per limitare la validità degli annunci;
- poter eliminare annunci precedentemente inseriti;
- poter accedere ad uno storico degli annunci per poter riproporre annunci passati che si sono dimostrati efficaci.

Cliente

Si è voluto valutare la reazione al sistema da parte di utenti appartenenti a diverse fasce di età. Sono stati coinvolti:

- **Utente A**: un utente giovane (25 anni) pratico di applicazioni Web di vario genere;
- **Utente B**: un utente adulto (56 anni) che fa un uso limitato del Web, tendenzialmente applicazioni di messaggistica e Facebook, raramente le mappe (sporadici utilizzi del navigatore).

A entrambi sono stati sottoposti i seguenti task:

- **Task 1**: l'utente deve eseguire una ricerca di massaggiatori in una città a sua scelta;

- **Task 2**: l'utente deve aprire una chat con il massaggiatore scelto (per fare questo è necessario registrarsi al sito);

- **Task 3**: l'utente deve seguire il massaggiatore scelto e gestire le notifiche quando questo pubblica annunci.

L'utente A è riuscito a portare a termine tutti i task con successo, senza la necessità di intervento da parte del team. Ha trovato scomodo il fatto di dover passare sempre per la mappa per raggiungere il profilo di un massaggiatore che magari aveva già cercato/contattato in passato. I suggerimenti evidenziati sono stati i seguenti:

- ha suggerito di introdurre la possibilità di risalire al profilo del massaggiatore cliccando sull'immagine del profilo presente nella chat;

- ha suggerito di consentire la ricerca massaggiatori specificando una distanza dalla propria posizione attuale entro un certo range "con un effetto visivo tipo radar".

L'utente B ha trovato difficoltà ad eseguire la ricerca tramite la mappa. Non avendo esperienze con questo tipo di sistemi di ricerca non gli è risultato intuitivo il fatto che, per dover risalire al profilo di un massaggiatore, fosse necessario interagire con i marker visualizzati come risultati della ricerca sulla mappa. Inoltre, una volta raggiunto il profilo del massaggiatore, l'utente è riuscito ad individuare il pulsante che consente l'apertura di una chat con quel massaggiatore, ma ha trovato inaspettato il fatto di essere rediretto sulla schermata di login dalla quale si sarebbe dovuto autenticare prima di poter procedere all'invio di messaggi. Nel caso specifico del Task 3, non ha avuto particolari difficoltà.

L'utente B non ha prodotto dei veri e propri suggerimenti ma durante l'esperienza ha commentato il fatto che trovandosi sul profilo del massaggiatore e avendo a quel punto disponibile il suo numero di telefono, lo avrebbe contattato telefonicamente piuttosto che registrarsi al sito e utilizzare la chat.

Capitolo 7

Deployment

7.1 Installazione

- Clonare il repository
```
git clone https://github.com/nem3s1s/mapsage-for-good.git
```
- Dalla cartella principale del progetto, inizializzare il front-end
```
cd frontend
npm install
```
- Dalla cartella principale del progetto, inizializzare il back-end
```
cd backend
npm install
```
- Nella cartella principale del progetto, è presente il file "municipalities.json". Esso contiene la collection dei Comuni e va importato in MongoDB nel database "mapsage".
```
mongoimport --collection municipalities
--db mapsage .\municipalities.json
```

7.2 Messa in funzione

- Dalla cartella principale del progetto, avviare il server
```
cd backend
node index.js
```
- Dalla cartella principale del progetto, avviare la vue-cli
```
cd frontend
npm run serve
```
- Collegarsi al sito da un browser
```
http://localhost:8080/
```

Capitolo 8

Conclusioni

I risultati ottenuti dai test effettuati e i consigli espressi dagli utenti coinvolti, ha portato il team alle seguenti considerazioni.

Per quanto riguarda le funzionalità che interessano i massaggiatori un'evoluzione futura del sistema, potrà sicuramente concentrarsi sul rendere più intuitiva l'interfaccia per la modifica della posizione dello studio dal profilo del massaggiatore. Inoltre, come indicato dal massaggiatore intervistato, si potrebbe introdurre un'estensione delle funzionalità relative agli annunci, che potrebbero comprendere l'aggiunta di una scadenza per la validità degli annunci e la possibilità di cancellare questi ultimi così come l'introduzione di uno storico. Oltre a queste funzionalità suggerite, si potrebbe integrare il sistema con:

- la possibilità di specificare filtri di ricerca per tipologia di massaggiatore che renderebbero più efficace la ricerca;
- un aumento del livello di personalizzazione dei profili e di aggiornamento delle informazioni (ad esempio l'update dell'immagine del profilo);
- l'introduzione di una sezione dedicata alle recensioni che potranno essere utilizzate dai massaggiatori per ricevere un feedback da parte dei propri clienti e costituiranno la base per lo sviluppo di ulteriori funzionalità;
- un supporto statistico a disposizione dei massaggiatori per monitorare il traffico e le interazioni da parte dei clienti sul proprio profilo.

Per risolvere i problemi evidenziati dai clienti coinvolti in fase di testing riguardo la difficoltà nel raggiungere i profili dei massaggiatori dalla mappa, potrebbe essere utile inserire nella "Home" uno storico delle ricerche effettuate in passato dall'utente in modo che quest'ultimo possa riutilizzarle per

saltare immediatamente sul profilo del massaggiatore desiderato. Inoltre, può essere sicuramente d'aiuto implementare la funzionalità che permette di raggiungere il profilo di un massaggiatore, cliccando sull'immagine in chat.

Per aumentare ulteriormente la flessibilità nella ricerca di massaggiatori potrebbe essere utile aggiungere la possibilità di eseguire una ricerca all'interno di una circonferenza centrata nella posizione dell'utente, individuata dal GPS, con raggio a piacere.

Commenti finali

Siamo consapevoli del fatto che durante la fase di testing sarebbe stato più efficace coinvolgere gruppi di persone più numerosi per ogni tipologia di utente in modo da ottenere feedback più rappresentativi. Tuttavia, si è scelto di coinvolgere un numero limitato di persone: sia perché il progetto non racchiude un numero molto elevato di funzionalità, sia per i limiti di tempo imposti. Nonostante ciò, si è cercato di selezionare le persone coinvolte in modo che rappresentassero al meglio i target.

Lo sviluppo di questo progetto ha contribuito ad arricchire il bagaglio di conoscenze tecniche acquisite fino ad ora, dandoci la possibilità di metterci alla prova con le funzionalità real-time e coinvolgendo tante tecnologie diverse in un'unica soluzione. Nella realizzazione del sistema ha giocato un ruolo fondamentale l'utilizzo della Vue CLI, che ha contribuito a migliorare l'organizzazione del codice e ci ha permesso di apprendere l'utilizzo di un tool di supporto alla programmazione lato client.

Questo progetto è stato utile anche a sviluppare skill di programmazione Javascript pura; cosa che non è stata possibile nei progetti della triennale a causa dell'utilizzo di framework come JQuery. Tra l'altro, Javascript è l'unico linguaggio interpretato visto fin'ora nella carriera universitaria e quindi è stato utile poterlo approfondire ai fini di confronto con i linguaggi compilati.

L'idea di questo progetto è partita da una proposta mossa dal massaggiatore che poi è stato coinvolto durante la stesura dei requisiti e la fase testing. Questo ci ha dato modo di sperimentare alcune aspetti del rapporto con un vero committente. Primo fra tutti, la difficoltà nello stendere un elenco dei requisiti che rappresentino concretamente l'idea reale del committente. Questo poi ha portato, in fase di testing, ad evidenziare degli aspetti che il committente pensava sarebbero stati implementati, ma che di fatto non erano espressi da nessun requisito.

Bibliografia

[1] "*Vue CLI*", https://cli.vuejs.org/

[2] "*Bootstrap 4*", https://getbootstrap.com/

[3] "*Mongoose*", https://mongoosejs.com/

[4] "*bcrypt*", https://github.com/kelektiv/node.bcrypt.js

[5] "*Json Web Token*", https://github.com/auth0/node-jsonwebtoken

[6] "*Axios*", https://github.com/axios/axios

[7] "*Socket.IO*", https://socket.io/

[8] "*Leaflet*", https://leafletjs.com/

[9] "*Esri Leaflet*", https://esri.github.io/esri-leaflet/

[10] "*Esri-Leaflet-Geocoder*",https://github.com/Esri/esri-leaflet-geocoder

[11] "*ArcGIS*", https://developers.arcgis.com/javascript/

[12] "*vue-cookies*", https://github.com/cmp-cc/vue-cookies

[13] "*Cropper.js*", https://github.com/fengyuanchen/cropperjs