UNIVERSITÀ POLITECNICA DELLE MARCHE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea triennale in
Ingegneria Informatica e dell'Automazione

Sviluppo di un'app di messaggistica attraverso l'utilizzo dello stack MERN

Development of a messaging app through the use of the MERN stack

Relatore: Chiar.mo
Prof. Aldo Franco Dragoni

Tesi di Laurea di:
Nicholas Urbanelli

Anno Accademico 2021/2022

Sommario

Lo scopo del seguente documento è quello di illustrare le attività svolte sulla base del tiricinio avuto luogo presso l'azienda Infoservice S.r.l. al termine del percorso di studi della laurea triennale in Ingegneria Informatica e dell'Automazione, avente una durata di centocinquanta ore.
Uno degli obiettivi principali del tirocinio era quello di sviluppare una applicazione full stack di messaggistica istantanea, che l'azienda avrebbe poi successivamente integrato su alcuni software di sua proprietà.

Questo documento descrive tutti i passi e le scelte che hanno portato alla realizzazione del progetto, a partire dagli strumenti di sviluppo utilizzati fino alla progettazione pratica.
Nel primo capitolo della tesi si analizza il concetto di Messaggistica istantanea, il suo funzionamento e come si è sviluppata nel corso degli anni, divenendo parte della nostra quotidianità.
Nel capitolo successivo vengono trattate le varie tecnologie presenti nell'ambito dello sviluppo software e i loro pregi e difetti. Inoltre vengono descritti gli strumenti informatici e i linguaggi utilizzati successivamente in fase di progettazione ed implementazione della piattaforma.
Il terzo capitolo ha come oggetto la progettazione dell'applicazione attraverso l'analisi dei requisiti e la descrizione dei casi d'uso, oltre alla realizzazione del database.
Il quarto capitolo risulta essere quello di maggiore importanza poiché rappresenta il nucleo dell'elaborato. Qui verrà di fatto analizzata la codifica delle funzionalità dell'applicazione. Verranno inoltre mostrate al lettore le istantanee delle schermate dell'applicazione effettuate su browser, riportando porzioni di codice del front e back-end.
Infine verranno esposte le conclusioni che possono essere tratte dall'esperienza di stage in azienda, le possibili implementazioni e migliorie future dell'applicazione e le conoscenze acquisite mediante la fase di sviluppo.

Indice

Elenco delle figure

Elenco delle tabelle

Codici

Capitolo 1

Messaggistica

1.1 Introduzione

Le telecomunicazioni si evolvono con passi da gigante e così anche il nostro modo di comunicare. L'avvento dei servizi di messaggistica istantanea ha rivoluzionato completamente il nostro modo di comunicare e di rapportarci con le persone. In un passato non troppo remoto, l'unico mezzo per poter comunicare a distanza era quello della lettera scritta. Metodo che, per quanto possa essere considerato 'romantico', pecca di inefficienza in quanto risulta essere costosa, lenta e scomoda.
I moderni sistemi di comunicazione invece aggirano con facilità questi ostacoli. Basta disporre di un'applicazione, di una connessione ad internet e del contatto della persona con cui si vuole comunicare per poter scambiare messaggi, file ed effettuare chiamate video e audio in tempo reale, anche con persone che si trovano in zone geografiche molto distanti tra loro.

La Messaggistica Istantanea consente la conversazione e la comunicazione in tempo reale, anche con scambio di *file* e *documenti*, tra due o più interlocutori.
Attraverso la messaggistica è possibile rimanere in contatto con persone distanti o con altre che non si ha tempo di visitare di persona, dare e ricevere informazioni, rendere la comunicazione *facile*, *veloce* e *comoda*. Le esigenze possono essere molteplici, come il superamento della timidezza e dell'imbarazzo di un'interazione faccia a faccia.
A questo si aggiunge la *personalizzazione*, con cui ogni utente può rendere davvero unico il suo spazio digitale, mostrando una propria identità agli altri contatti ed amici. Avere il proprio colore, le proprie emoticon, il proprio status attuale.

1.1.1 Storia della Messaggistica istantanea

Anche se il termine risale agli anni novanta, la messaggistica istantanea precede Internet, apparendo per la prima volta su sistemi operativi multi-utente come Compatible Time-

Sharing System (CTSS) e Multiplexed Information and Computing Service (Multics) a metà degli anni sessanta. Inizialmente, questi sistemi sono stati utilizzati esclusivamente come sistemi di notifica per servizi come la stampa, ma rapidamente la loro mansione è diventata quella di facilitare la comunicazione con altri utenti connessi alla stessa macchina.
Parallelamente alla messaggistica istantanea nascono i primi servizi di chat online, il primo dei quali era Talkomatic creato nel 1973 sul sistema PLATO, presso l'università dell'Illinois che permetteva a 5 persone di chattare contemporaneamente.
Con il passare del tempo sempre più sistemi hanno incorporato funzionalità di chat simili alla messaggistica istantanea, ma il primo servizio commerciale di chat online a disponibilità generale fu il CompuServe CB Simulator nel 1980, creato dal dirigente di CompuServe Alexander "Sandy" Trevor a Columbus, Ohio. A quel tempo, i concetti di chat multiutente e messaggistica istantanea erano estranei, per questo il dirigente Alexander Trevor scelse appositamente l'acronimo CB facendo riferimento alla CB radio, ovvero, una tipologia di radio destinata all'uso privato collettivo e utilizzata da autotrasportatori, forze locali o semplici appassionati. Attraverso l'etichetta *CB*, CB simulator ha contribuito a introdurre i concetti di chat multiutente e messaggistica istantanea al pubblico.

Figura 1.1: CB radio

Sempre intorno al 1980 il MIT Project Athena inventò lo Zephyr Notification Service (ancora in uso presso alcune istituzioni), che consentiva ai fornitori di servizi di individuare e inviare messaggi ai propri clienti.
I primi programmi di messaggistica istantanea erano principalmente a scrittura in tempo reale, dove i caratteri apparivano mentre venivano digitati. Questo include il programma a riga di comando Unix "talk" e "WinPopUp" su MsWindows. Questi due sistemi però, si basavano sulla coppia utente-macchina e non utente-utente.
Nella seconda metà degli anni '80 e nei primi anni '90, il servizio online Quantum Link

per i computer Commodore 64 offriva la possibilità di scambiare messaggi tra utenti connessi contemporaneamente che chiamavano questo tipo di messaggi "Messaggi on-line" e successivamente "FlashMail". Quantum Link in seguito divenne **AOL Instant Messenger**. Mentre il software client Quantum Link girava su un Commodore 64 lo schermo era visivamente diviso in sezioni e i messaggi ricevuti apparivano come una barra gialla che citava "Message From:" e il nome del mittente, il relativo messaggio e un serie di opzioni per rispondere. Questo può essere considerato come un inizio di interfaccia utente grafica (GUI), anche se molto primitiva rispetto al successivo software GUI IM basato su Unix, Windows e Macintosh.

Figura 1.2: AOL Instant Messenger

Nel frattempo, altre aziende come Yahoo o Ubique hanno sviluppato il loro software di messaggistica personale; ciascuno con il proprio client e i propri protocolli. Gli utenti dovevano quindi eseguire più applicazioni client, una per ogni piattaforma.
Il problema venne risolto nel 2000, quando è stata lanciata un'applicazione open-source e un protocollo basato su standard aperti chiamato Jabber. Il protocollo è stato standardizzato sotto il nome **Extensible Messaging and Presence Protocol (XMPP)** e i server XMPP potevano fungere da gateway per altri protocolli di messaggistica, riducendo la necessità di eseguire più client.
Nel 2010, la messaggistica istantanea sul Web era in forte declino a favore delle funzionalità di messaggistica sui social network. Grandi compagnie come facebook crearono i

primi social network che offrivano un'enorme varietà di servizi tra cui anche quello dedicato alla messaggistica istantanea, infatti, gli utenti registrati al social network potevano scambiarsi messaggi all'interno di chat private. Inoltre, non si aveva più il bisogno di scaricare un client, bastava digitare su un qualsiasi browser il nome del social network, iscriversi o autenticarsi attraverso il login e si era pronti a chattare con i propri amici o conoscenti.
La popolarità della messaggistica istantanea è stata rilanciata con nuovi servizi sotto forma di applicazioni mobili. Le prime app mobile che si occupavano di messagistica erano BlackBerry Messenger, rilasciato per la prima volta nel 2005, e WhatsApp rilasciato per la prima volta nel 2009. A differenza delle precedenti applicazioni di messaggistica istantanea, le nuove app giravano solo su dispositivi mobile e coincidevano perfettamente con la crescente popolarità degli smartphone abilitati a Internet. Ad oggi WhatsApp è l'app di messaggistica più utilizzata in tutto il mondo.

1.1.2 Applicazioni nel mondo del business

La messaggistica istantanea ha dimostrato di essere simile a personal computer, e-mail e World Wide Web, in quanto la sua adozione per l'uso come mezzo di comunicazione aziendale è stata guidata principalmente da lavoratori che preferiscono utilizzare i servizi di messaggistica istantanea, piuttosto che per utilizzare metodi di comunicazione formali e impegnativi. Decine di milioni di account di messaggistica istantanea vengono di fatti utilizzati per scopi commerciali da parte dei dipendenti di aziende e organizzazioni.
In risposta alla richiesta di servizi di messaggistica istantanea a livello aziendale e alla necessità di garantire la sicurezza e la conformità legale, nel 1998 l'azienda Lotus Software creò un nuovo tipo di servizio dedicato appositamente alle imprese, chiamato "Enterprise Instant Messaging" ("EIM"). Microsoft ha seguito l'esempio con *Microsoft Exchange Instant Messaging*, in seguito ha creato una nuova piattaforma chiamata *Microsoft Office Live Communications Server* e ha rilasciato *Office Communications Server 2007* nell'ottobre 2007. Anche Oracle Corporation è entrata nel mercato con il suo software *Oracle Beehive*. A partire dal 2010, le principali piattaforme per l'Entreprise Instant Messaging (EIM) includono *IBM Lotus Sametime*, *Microsoft Office Communications Server*, *Jabber XCP* e *Cisco Unified Presence*.
L'adozione di servizi di IM attraverso le reti aziendali, al di fuori del controllo delle organizzazioni IT crea rischi e responsabilità per le aziende che non gestiscono e supportano efficacemente l'uso dell'IM. Le aziende, per tutelarsi, implementano prodotti e servizi di archiviazione e sicurezza necessari per mitigare questi rischi e fornire ai propri dipendenti servizi sicuri e produttivi. L'IM sta diventando sempre più una caratteristica del software aziendale piuttosto che un'applicazione stand-alone.
I prodotti IM di solito possono essere classificati in due tipi: **Enterprise Instant Messaging (EIM)** e **Consumer Instant Messaging (CIM)**. Le soluzioni aziendali spesso utilizzano un server IM interno ma tuttavia ciò non è sempre fattibile, in particolare per

le piccole imprese con budget limitati. La seconda opzione invece fornisce il vantaggio di essere poco costosa e facile da implementare.

1.1.3 Differenza con gli SMS

SMS è l'acronimo di "short message service" e consente agli utenti di telefonia mobile di inviare messaggi di testo senza una connessione Internet. Prima dell'arrivo degli smartphone, l'SMS era la forma di comunicazione più dominante. Mentre gli SMS si basano sui tradizionali servizi telefonici a pagamento, le app di messaggistica istantanea sono disponibili gratuitamente all'interno di negozi digitali chiamati "store" e funzionano attraverso la connessione ad internet. Grazie a una semplice connessione a internet l'utente può scrivere messaggi in quantità illimitata, Al contrario degli sms che solitamente vengono forniti dal gestore telefonico in numero limitato.
Un altro vantaggio apportato dalle app di messaggistica per smartphone è la possibilità di creare chat di gruppo, novità determinante che ha contribuito ulteriormente all'adozione delle app a sfavore degli SMS.
Prima dell'introduzione delle app di messaggistica, gli utenti di smartphone potevano interagire con una sola persona alla volta tramite chiamate vocali mobili o SMS. Con l'introduzione delle applicazioni di messaggistica, la funzionalità di chat di gruppo consente a tutti i membri di vedere un intero thread di risposte di tutti. I membri possono anche rispondere direttamente tra loro, piuttosto che dover passare attraverso il membro che ha creato il gruppo, per trasmettere le informazioni. Grazie alle chat di gruppo un utente può entrare a far parte di un "gruppo digitale" e può relazionarsi con gli altri membri del gruppo per discutere di argomenti più o meno importanti e scambiarsi opinioni a riguardo.
Nonostante l'introduzione delle chat app mobile, l'SMS rimane ancora molto popolare in alcune parti del mondo come negli Stati Uniti perché incluso gratuitamente nelle promozioni delle varie compagnie telefoniche. Infatti, dal 2013 a oggi l'uso degli SMS negli stati Uniti è diminuito di circa un quarto rispetto ad altri paesi come la Danimarca o la Spagna dove l'uso degli SMS è sceso fino ai due terzi.

1.1.4 Sicurezza e archiviazione

I cracker (hacker malintenzionati o black hat) hanno costantemente utilizzato le reti IM come vettori per attuare tentativi di phishing, "URL avvelenati" e allegati di file carichi di virus.
Gli hacker utilizzano principalmente due metodi di consegna di codice dannoso attraverso la messaggistica istantanea: la consegna di virus, cavalli di Troia, o spyware all'interno di un file infetto, e l'uso di un indirizzo web che invoglia il destinatario a fare clic su di esso e che lo collega a un sito web che scarica codice dannoso.
Una volta che il virus entra nel computer della vittima esso si propaga inviandosi rapida-

mente attraverso l'elenco dei contatti dell'utente infetto. Un attacco efficace utilizzando un URL avvelenato può raggiungere decine di migliaia di utenti in un breve periodo in quanto l'elenco dei contatti di ogni utente riceve messaggi che sembrano provenire da un amico fidato. I destinatari fanno clic sull'indirizzo web e l'intero ciclo ricomincia.
Nei primi anni 2000, nasce una nuova classe di provider di sicurezza informatica per fornire protezioni e rimedi contro i rischi che le società affrontano utilizzando l'IM per le comunicazioni aziendali. I provider di sicurezza dell'IM hanno creato nuovi prodotti da installare nelle reti aziendali allo scopo di archiviare, scansionare i contenuti e scansionare il traffico di tutti i messaggi in entrata e in uscita dalla società. Analogamente ai fornitori di filtraggio delle e-mail, i provider di sicurezza dell'IM si concentrano sui rischi e sulle responsabilità descritti in precedenza.
A metà degli anni 2000 il continuo aumento delle applicazioni di messaggistica istantanea sul posto di lavoro ha portato a un conseguente aumento della domanda di prodotti di sicurezza. Nel 2007, la piattaforma preferita per l'acquisto di software di sicurezza era diventato il "computer appliance", ovvero, un dispositivo hardware provvisto di un software integrato utilizzato per eseguire particolari funzioni applicative.
IDC ha stimato che nel 2008, l'80% dei prodotti di sicurezza di rete è stato consegnato tramite il "computer appliance".
Tuttavia, nel 2014, il livello di sicurezza offerto dalle app di IM era ancora estremamente basso. Secondo una scorecard realizzata dalla Electronic Frontier Foundation, solo 7 delle 39 app messagistica istantanea avevano ricevuto un punteggio perfetto, mentre i più famosi software di IM avevano raggiunto il punteggio di 2 su 7. Inoltre, diversi studi hanno dimostrato come i servizi di messaggistica istantanea non siano in grado di garantire la tutela della privacy del consumatore.

Crittografia

La crittografia è il metodo principale che le app di messaggistica utilizzano per proteggere la privacy e la sicurezza dei dati degli utenti. I messaggi SMS non sono crittografati e ciò li rende insicuri, in quanto il contenuto di ciascun messaggio SMS è visibile alle compagnie telefoniche, ai governi e può essere intercettato da terzi.
I messaggi SMS, inoltre, portano con se informazioni -come i numeri di telefono del mittente e del destinatario- che non rappresentano il contenuto del messaggio stesso e che permettono di identificare le persone coinvolte nella conversazione.
Un'altra considerazione da fare è la falsificazione del messaggio e del mittente del messaggio, i quali possono essere modificati per impersonare un'altra persona.
Le applicazioni di messaggistica sul mercato che utilizzano la crittografia end-to-end includono Signal, WhatsApp, Wire e iMessage. Grazie a questo tipo di crittografia nessuno può leggere i messaggi durante il trasfermiento tra i vari dispositivi.
Di fatti, utilizzando la crittografia end-to-end, tutte le chat, inclusi il testo o gli eventuali file o contenuti multimediali, vengono criptate in fase di trasferimento, convertendo i dati

in testo illeggibile che può essere decifrato solo da una chiave segreta.
La chiave segreta è di fatti una stringa che possiede diverse caratteristiche:

- Viene creata solo sul dispositivo del mittente e del destinatario del messaggio. Esiste quindi fisicamente solo su questi due dispositivi.Viene creata solo sul dispositivo del mittente e del destinatario del messaggio. Esiste quindi fisicamente solo su questi due dispositivi.
- Viene generata di nuovo per ogni messaggio.
- Viene eliminata automaticamente dal dispositivo del mittente quando il messaggio criptato viene creato e dal dispositivo del destinatario quando il messaggio viene decriptato.

I server di recapito dei messaggi e qualsiasi terza persona che dovesse ottenere l'accesso ai dati scambiati non potrà quindi decifrarli, non essendo in possesso della chiave segreta.

Rischi di conformità

Oltre alla minaccia dovuta alla potenziale presenza di codice dannoso, l'uso della messaggistica istantanea sul posto di lavoro crea anche un rischio di non conformità alle leggi e regolamenti che disciplinano l'uso delle comunicazioni elettroniche nelle imprese.
Solo negli Stati Uniti ci sono oltre 10.000 leggi e regolamenti relativi alla messaggistica digitale e alla conservazione dei dati.
L'autorità di regolamentazione del settore finanziario statunitense (FINRA) ha fornito chiarimenti alle imprese associate nel settore dei servizi finanziari nel dicembre 2007, rilevando che "comunicazioni elettroniche", "e-mail" e "corrispondenza elettronica" possono essere utilizzate in modo intercambiabile e possono includere forme di messaggistica elettronica come la messaggistica istantanea e la messaggistica di testo.
La maggior parte delle nazioni regolano anche l'uso della messaggistica elettronica e la conservazione dei dati in modo simile agli Stati Uniti. I regolamenti più comuni relativi alla messaggistica istantanea sul lavoro comportano la necessità di archiviare le comunicazioni commerciali per soddisfare le richieste governative o giudiziarie ai sensi di legge.[1]

Capitolo 2

Sviluppo mobile

2.1 Il mondo delle app

2.1.1 Introduzione

Un'applicazione mobile meglio conosciuta con il nome di app, è un programma o meglio un'applicazione software progettata per funzionare su un dispositivo mobile come un telefono, un tablet o un orologio. Le applicazioni mobile differiscono dalle applicazioni desktop che vengono create allo scopo di funzionare sul desktop dei computer e con le applicazioni web che funzionano, invece, sui browsers del dispositivo ma non sul dispositivo stesso.
L'obiettivo iniziale delle app era quello di fornire supporto alla produttività dell'utente, infatti, le applicaioni servivano alla gestione delle email, il calendario e i contatti; a seguito della richiesta di nuove app da parte del pubblico il mondo app ebbe una rapida espansione in molti altri campi come quello dei giochi mobile, dei servizi basati sulla nostra posizione, gestione degli ordini e molto altro.

Le app possono essere scaricte tramite piattaform digitali di distribuzione, che a loro volta sono delle app.
Possiamo classificare le app in tre tipologie differenti:

- Web app
- App native
- App ibride

Le web app sono scritte in HTML e/o CSS e tipicamente funzionano attraverso l'utilizzo di un browser. Le applicazioni native, invece, sono progettate specificatamente per un sistema operativo mobile, normalmente iOS o Android.

Infine le app ibride sono costruite usando tecnologie web come JavaScript, CSS e HTML, e funzionano come dell app web ma nascoste in un contenitore nativo.

2.1.2 Web App

Una web application non è altro che un'applicazione client-server in cui le funzionalità applicative risiedono su un application server e il client vi accede in un web browser. Quando si parla di web site si utilizza il termine "web application" se le funzionalità proposte sono simili a quelle di un software desktop. Mentre un software desktop è sviluppato per una piattaforma e installato su essa, le web app sono utilizate dal client tramite l'accesso a internet.
I vantaggi delle web app sono i seguenti:

- Il client può usufruire della web app senza dover rispettare dei vincoli di compatibilità.
- L'utente risparmia spazio sul proprio computer perché non ha bisogno di scaricare in locale l'appicazione.
- Diminuiscono i problemi legati a software pirati.

Le prime applicazioni client-server iniziano a diffondersi a partire dalla metà degli anni ottanta dove l'utente necessitava di installare il lato client dell'applicazione sul proprio computer. Ad ogni nuovo aggiornamento lato server seguiva anche la rispettiva modifica dell'applicazione lato client, ciò aveva diverse conseguenze in particolare per i costi del produttore. Inoltre, sia la parte client che quella server erano legate a uno specifico sistema operativo, era necessario effettuare il porting verso altri sistemi e questo aumentava ancora di più i costi.
Le moderne applicazioni web, invece, utilizzano documenti web scritti in formato standard come HTML o Javascript. Tutti i web browser sono in grado di interpretare in modo opportuno questi documenti. Nelle web application il software del client viene scaricato non appena l'utente visita la pagina web associata all'applicazione tramite il protocollo HTTP. Gli aggiornamenti avvengono ogni volta che l'utente visita la pagina web. Possiamo quindi considerare il web browser come un clinet universale per tutte le web app perché riceve le pagine dal server, le interpreta e ne mostra i risultati su schermo.
Nelle prime web app il client naviga in una serie di pagine ricevute dal server sotto forma di documenti statici e ogni volta che la pagina richiesta veniva modificata il server veniva chiamato per effettuare il refresh della pagina.
Nel 1995 Netscape introdusse Javascript, un linguaggio di scripting client-side. Questo linguaggio forniva la possibilità di sviluppare pagine web dinamiche, attraverso l'esecuzione di un codice direttamente sul client senza dover ogni volta chiamare il server.

Nel 2005 Avvenne un'altra svolta importante, ovvero, l'introduzione di Ajax. Ajax è un insieme di tecniche per lo sviluppo web che permette di creare web app asincrone e con parti client più interattive. Questo tramite l'uso di script client-side che contattano il server per effettuare il download di dati senza dover scaricare nuovamente lintera pagina web.
Infine nel 2011 venne completato HTML5, tramite il quale si possono realizzare interfacce utente più ricche di animazioni attraverso l'introduzione di nuovi elementi che gestiscono in modo nativo i contenuti multimediali e grafici all'interno della pagina web.
Al giorno d'oggi, HTML5 si è affermato nell'uso tradizionale e fornisce agli sviluppatori web una serie di funzionalità che permettono di spostare nel client diversi lavori che prima erano a carico del server.

Progressive Web App

Negli ultimi anni si è diffuso il concetto di Progressive Web Application (PWA), con questo termine si fa riferimento a un metodo che si basa sul concetto di fornire all'utente una navigazione sul sito che sembra quella di un'app nativa, per poi dare la scelta di installare l'app dal sito in modo semplice e veloce.

ASP.NET Core e Blazor

ASP.NET è un framework per lo sviluppo di web application messo a disposizione da Microsoft, uno dei suoi componenti principali è Blazor a sua volta definito come framework per creare web app usando C e HTML.
Blazor è in continuo sviluppo e tramite esso lo sviluppatore può creare app single page, un metodo molto usato nello sviluppo di web app.
Blazor deriva dall'accostamento di Browser e Razor, il secondo è un sistema di scripting lato server utilizzato per generare e gestire pagine web dinamiche, tramite razor si può scrivere in C e Visual Basic incorporando il codice scritto nelle pagine web.
Inoltre, ASP.NET mette a disposizione altre funzione come, per esempio, EntityFramework (EF) per la gestiore e l'interazione con database SQL.

Esistono due tipologie di app Blazor che utilizzano metodo diversi che dipendono dall'esecuzione e dalla comunicazione:

- **Blazor Server**: L'applicazione viene eseguita nel server di un'app ASP.NET Core tramite l'utilizzo di alcuni segnali chiamati SignalR avviene la comunicazione tra browser e server. I signalR hanno il compito di aggiornare gli elementi della UI ma necessitano di una costante connessione a internet.

- **Blazor Webassembly**: Il browser esegue il modello principale per Blazor e vengono scaricate le varie dipendenze e il runtime *.NET* nel browser. La gestione

degli eventi e l'aggiornamento dell'interfaccia utente avviene in un solo processo. Le risorse dell'app sono distribuite sotto forma di file statici attraverso dei server web.
Grazie a questa metodologia si è in grado di fornire supporto offline, non necessitando di una comunicazione continua con il server per la modifica della pagina.

A seguito un'immagine delle due metodologie e la loro modalità di comunicazione:

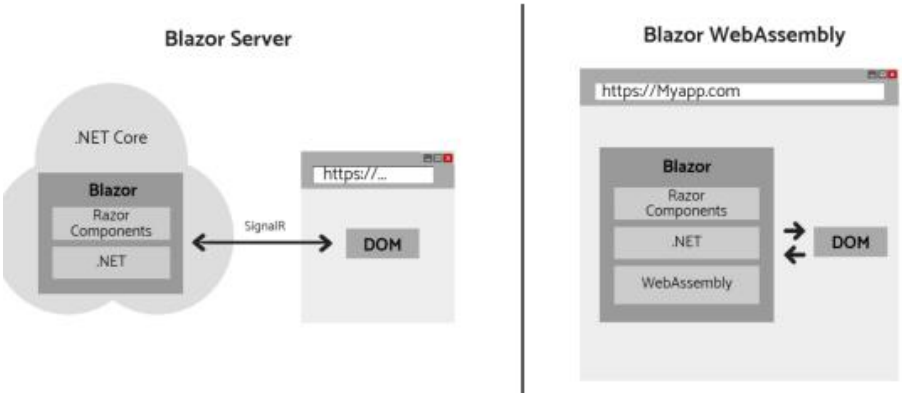

Figura 2.1: Funzionamento di Blazor Server e Webassembly

Se si sceglie l'approccio Webassembly si avrà un download più pesante all'inizio perché è necessario avere tutto su lato client per garantirne il corretto funzionamento, al contrario di Blazor Server. [2]

Angular con Express e Node, nasce lo stack MEAN

AngularJS è un framework open source creato da Google che offre la possibilità di sviluppare sia app web che single page application. Questo framework si basa su JavaScript ma spesso si utlizza la sua evoluzione, ovvero, TypeScript. Con AngularJS si scrive la parte front-end e, grazie a delle chiamate, si può comunicare con il server.
Angular non è un framework full-stack e per poterlo usare come tale è necessario un qualcosa che si interfacci a lato server; una delle scelte più comuni è Node.js, utilizzato per installare dipendenze e per eseguire il web server locale preconfigurato. Nel caso si scelga di utilizzare Node come piattaforma lato server, un valido framework da utilizzare per la gestione del back-end è Express.
Node, Angular ed Express sono una buona combinazione di framework per lo sviluppo web, utilizzando Javascript su entrambi i lati. La comunicazione avviene in modo asincrono grazie a Node.js e le chiamate avvengono utilizzando AJAX poichè viene usato il modello *Single Page Application*.
Con la nascita dei database non relazionali è nata l'idea di unire questi tre framework

con MongoDB, dando vita allo stack *MEAN*.

Nell'immagine seguente è possibile vedere la rappresentazione dello stack MEAN:

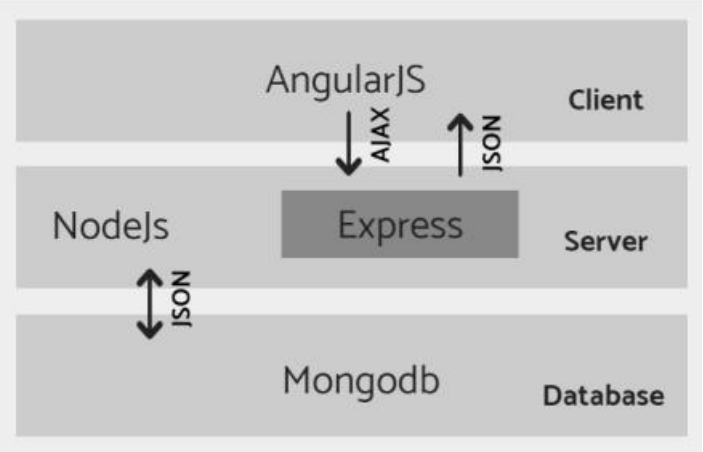

Figura 2.2: Stack Mean

React con Express e Node, nascita dello stack MERN

Iniziamo col definire cosa si intende con il termine stack.
Uno stack tecnologico non è altro che un insieme di framework e strumenti utilizzati per sviluppare un software. Questo insieme di framework e strumenti è scelto in modo specifico per lavorare all'unisono nella creazione di un software ben funzionante.
Lo stack **MERN** è un framework di sviluppo web. Esso è formato da 4 componenti: MongoDB, ExpressJS, React e NodeJS.

- **MongoDB**: un databse noSQL, orientato ai documenti, usato per conservare i dati della nostra applicazione.
- **NodeJS**:l'ambiente di runtime di JavaScript. È utilizzato per eseguire il codice JavaScript su macchina piuttosto che nel browser.
- **ExpressJS**: un framework che si colloca logicamente "sopra" NodeJS, utilzzato per sviluppare la parte relativa al backend di un sito mediante l'utilizzo di funzioni e strutture di quest'ultimo. Dato che NodeJS non era stato sviluppato per la progettazione di siti web ma bensì per eseguire JavaScript su calcolatore, ExpressJS è stato sviluppato per sopperire a questa mancanza.
- **ReactJS**: libreria creata da Facebook. È utilizzata per lo sviluppo di componenti UI, che costituiscono l'interfaccia utente della nostra single page web application.

Figura 2.3: Stack Mern

Come mostrato nell'illustrazione qui situata, l'utente interagisce con le componenti UI di ReactJS. Il frontend è servito dal backend dell'applicazione che risiede in un server, tramite ExpressJS in esecuzione su NodeJS.
Ogni interazione che causa una richiesta di cambiamento di dati è inoltrata al server Express, che prende i dati dal database MongoDB se richiesto, e restituisce i dati al frontend dell'app, che vengono poi mostrati all'utente.[3]

Quando utilizziamo lo stack MERN, lavoriamo con React per quanto riguarda la presentazione, con Express e Node per lo strato che riguarda l'applicazione e MongoDB per il database.
Di seguito verrano illustrati in dettaglio i componenti che formano i 3 strati dello stack MERN, ovvero il Front-end, il server e il database.

React.js Front-end

Lo strato superiore dello stack MERN è React.js.
React è una libreria JavaScript per la creazione di interfacce utente. Questa consente di sviluppare applicazioni dinamiche che non necessitano di ricaricare la pagina per visualizzare i dati modificati. Inoltre nelle applicazioni React le modifiche effettuate sul codice si possono visualizzare in tempo reale, permettendo uno sviluppo rapido, efficiente e flessibile delle applicazioni web. [4] React sfrutta l'approccio dichiarativo (un linguaggio di markup che estende l'HTML), per comporre l'interfaccia delle single page applications.
React è probabilmente la prima libreria JavaScript che nasce con una vocazione specifica: diventare la soluzione definitiva per sviluppatori frontend e app mobile basate su HTML5, il tool che permetta di costruire UI dinamiche e sempre più complesse rimanendo comunque semplice e intuitivo da utilizzare.
La creazione di applicazioni Web, indipendentemente dal framework scelto per lo sviluppo, coinvolge necessariamente i tre linguaggi fondamentali della piattaforma: HTML per la struttura, CSS per lo stile e JavaScript per la logica.
Per molte delle librerie esistenti, e React non fa eccezione, il linguaggio HTML nello specifico viene utilizzato quasi esclusivamente per creare "componenti Web" riutilizzabili, a volte estendendo il linguaggio HTML stesso.

Le pagine complete invece sono ridotte al minimo, anzi molto spesso a una sola, tant'è vero che queste applicazioni prendono il nome di Single Page Applications (SPA) e che servono da "contenitore" in cui creare e gestire l'interfaccia utente.
La forza di React rispetto ad altre librerie è quella di consentire l'uso di un **approccio dichiarativo** simile all'HTML per definire i componenti che rappresentano parti significative e logiche dell'interfaccia utente.
Benché dichiarativa, la rappresentazione del componente in realtà si traduce in chiamate all'API di React che intervengono in modo molto rapido sul DOM della pagina per creare gli elementi necessari.
Se dovessimo inquadrare React all'interno di un paradigma, potremmo dire che esso è la "V" di MVC: nasce per gestire la parte relativa alle view, ossia alla presentazione, e all'intercettazione degli eventi di input dell'utente, senza forzare l'adozione di specifiche funzioni né limitare l'interfacciamento ad altre librerie per quanto riguarda l'eventuale comunicazione con un server di backend, o specifiche architetture di binding ai dati, frangenti in cui lo sviluppatore ha libera scelta sugli strumenti con cui integrare React. [5]

Express.js e Node.js, Server Tier

Il livello successivo è il framework lato server Express.js, in esecuzione all'interno di un server Node.js. Express.js si autodefinisce un "framework web veloce, senza pretese e minimalista per Node.js", ed è esattamente quello che è. Express.js dispone di potenti modelli per l'instradamento degli URL (corrispondenza di un URL in entrata con una funzione server) e per la gestione di richieste e risposte HTTP.

Effettuando richieste HTTP XML (XHR) o GET o POST dal front-end React.js, puoi connetterti alle funzioni di Express.js che sono implementate nella tua applicazione. Queste funzioni a loro volta usano i driver di Node.js di MongoDB, tramite callback per l'utilizzo di Promises, per accedere e aggiornare i dati nel database MongoDB. Qui di seguito verrano analizzati singolarmente i due componenti appena descritti.

- **Node.js**
  Node.js è un framework per realizzare applicazioni Web in JavaScript, permettendoci di utilizzare questo linguaggio, tipicamente utilizzato nella 'client-side', anche per la scrittura di applicazioni 'server-side'.
  La piattaforma è basata sul JavaScript Engine V8, che è il runtime di Google utilizzato anche da Chrome e disponibile sulle principali piattaforme, anche se maggiormente performante su sistemi operativi UNIX-like.
  La caratteristica principale di Node.js risiede nella possibilità che offre di accedere alle risorse del sistema operativo in modalità *event-driven* e non sfruttando il classico modello basato su processi o thread concorrenti, utilizzato dai classici server

web.
Il modello event-driven, o "programmazione ad eventi", si basa su un concetto piuttosto semplice: si lancia una azione solo quando accade qualcosa. Ogni azione quindi risulta **asincrona** a differenza dei pattern di programmazione più comune in cui una azione succede ad un'altra solo dopo che essa è stata completata.
Ciò dovrebbe garantire una certa efficienza delle applicazioni grazie ad un sistema di callback gestito a basso livello dal runtime.
L'efficienza dipenderebbe dal considerare che le azioni tipicamente effettuate riguardano il networking, ambito nel quale capita spesso di lanciare richieste e di rimanere in attesa di risposte che arrivano con tempi che, paragonati ai tempi del sistema operativo, sembrano ere geologiche.
Grazie al comportamento asincrono, durante le attese di una certa azione il runtime può gestire qualcos'altro che ha a che fare con la logica applicativa, ad esempio. [6]

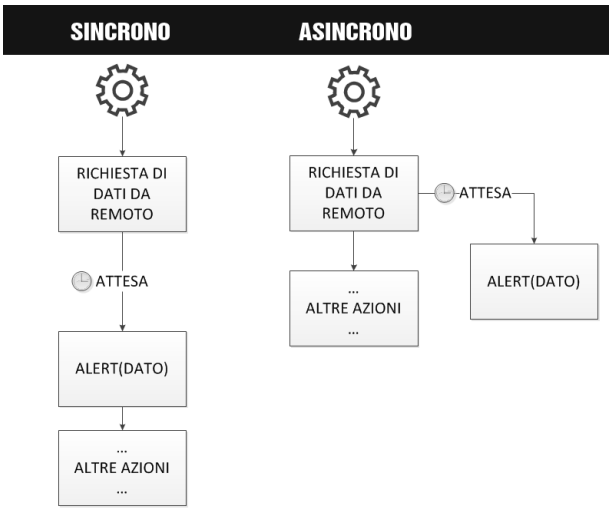

Figura 2.4: Approccio sincrono e asincrono a confronto

- **Express**
  Express.js è un framework open-source Node.js per la programmazione di applicazioni web e mobile. Minimalista, leggero e veloce, Express potenzia Node.js senza comprometterne le funzionalità.
  Il framework Express consente di creare potenti API di routing e di impostare middleware per rispondere alle richieste HTTP, fornendo semplici meccanismi di debugging e una rapida integrazione con vari motori di templating.
  Si utilizza Express in combinazione con Node.js poiché questo presenta una serie di vantaggi come la sua efficienza, la velocità elevata, l'approccio asincrono e la facilità di utilizzo. [7]

MongoDB Database tier

Se la tua applicazione memorizza dati (profili utente, contenuti, commenti, caricamenti, eventi, ecc.), allora vorrai un database con cui sia facile lavorare in sinergia con React, Express e Node.

È qui che entra in gioco MongoDB: i documenti JSON creati nel front-end React.js possono essere inviati al server Express.js, dove possono essere elaborati e (supponendo che siano validi) archiviati direttamente in MongoDB per un successivo recupero.

Altri framework per lo sviluppo web

Come si è visto in precedenza è possibile scegliere diversi framework e integrarli tra loro per creare app full stack dato che molti di essi forniscono supporto solo lato client o lato server.

I framework visti in precedenza operano con Javascript, tuttavia esistono altri framework o librerie molto diffusi che usano linguaggi differenti:

- **Laravel**:
  Un framework che segue il modello MVC ed è il più popolare in PHP. Presenta un'interfaccia a linea di comando chiamata *Artisan* che è in continuo aggiornamento. Viene usato per la programmazione lato server tramite l'uso di un sistema di gestione di pacchetti modulare e un gestore delle dipendenze dedicato. [8]

- **Ruby on Rails**:
  Detto anche RoR è un framework open source scritto in linguaggio *Ruby*. Anch'esso utilizza un modello MVC e viene usato lato server.[9]

- **Django**:
  un framework in Pyton lato server open source per lo sviluppo di applicazioni web utilizzando il modello MVC.[10]

- **Codelgniter**:
  Un altro framework in PHP per la creazione di app web seguendo un modello MVC. È un prodotto open source e di veloce funzionamento.[11]

- **jQuery**:
  Una delle librerie JavaScript più diffuse, che si occupa di manipolare e gestire gli eventi e animare gli elementi del DOM in pagine HTML.[12]

- **Bootstrap**:
  Framework composto da un insieme di modelli di progettazione per la creazione di siti e applicazioni.

Famoso principalmente per la sua utilità in ambito responsive, è una delle librerie più diffuse attualmente nello sviluppo web lato client. I modelli di bootstrap interpretabili dal browser sono scritti in HTML, CSS e JavaScript. [13]

Quadro generale dello sviluppo web e Progressive Web Apps

Quando si decide di utilizzare un framework web o più framework è necessario conoscere la tipologia di sviluppo che offre (front-end o back-end) e i linguaggi utilizzati.

Un altro aspetto fondamentale da considerare è la presenza di esigenze specifiche da parte dei clienti, che necessitano essere soddisfatte. Per questo motivo c'è bisogno di effettuare un'analisi dei requisiti dell'applicazione e una ricerca per scegliere il framework che più soddisfa le esigenze.

Quale metodologia scegliere

Dopo aver analizzato le caratteristiche delle varie metodologie di sviluppo è possibile avere un quadro più accurato e dettagliato.
Quando si ha bisogno di sviluppare un'app mobile ma con tempi e costi ridotti la soluzione ibrida risulta essere quella che va più incontro a queste esigenze, offrendo buone performance e dando la possibilità di creare applicazioni quasi native.
Se, invece, si vuole sviluppare un'applicazione mobile ma che possieda anche un comparto desktop allora lo sviluppo web è la scelta migliore per via delle applicazioni create, comodamente accessibili da tutti i dispositivi dotati di un browser.
Un altro fattore fondamentale che riguarda la scelta è rappresentato dalla rete poichè in alcuni luoghi la connessione può non essere costante e in questo caso scegliere uno sviluppo web risulta la scelta peggiore.
In altri casi si può avere bisogno di un'applicazione che si possa interfacciare con dispositivi particolari o alcuni terminali che sfruttano teconlogie e sistemi operativi particolari. In questo caso è necessario verificare che il framework offra la possibilità di sviluppare con tali restrizioni.
Se non ci sono esigenze particolari come quelle sopra elencate sta al team di sviluppo scegliere il framework migliore una volta scelta la tipologia.

2.1.3 App Native

Le applicazioni native ricoprono la maggior parte delle mobile applications, esse vengono create per essere utilizzate su una determinata piattaforma e vengono sviluppate con tecnologie e linguaggi ad hoc.
Le due principali tipologie di app native che vengono create in questo periodo sono Android e iOS, le prime per essere utilizzate dai dispositivi Android mentre le seconde per i dispositiv Apple.

Le app native vengono create mediante l'utilizzo di ambienti di sviluppo che hanno il nome di IDE (integrated Development Environment) che vengono eseguiti sulla stessa piattaforma per cui l'applicazione viene costruita. Gli IDE mettono a disposizione diversi strumenti ed interfacce che possono essere sviluppati per lo sviluppo e la documentazione del software, identificati con la parola SDK (software Development kit).[14]
Le applicazioni native offrono i seguenti *vantaggi*:

- Utilizzo delle funzionalità del dispositivo per cui sono create in maniera rapida ed efficiente.
- Maggiore familiarità da parte dell'utente tramite componenti sviluppati per la piattaforma stessa garantendo consistenza con le altre applicazioni.
- Le app vengono installate e lanciate localmente sul dispositivo garantendo potenza di calcolo ed velocità.
- Possibilità di ottenere un corretto funzionamento anche offline, ovvero, senza connessione internet.
- Maggiore semplicita ed efficacia per la distribuzione dell'app, grazie agli store la distribuzione e l'istallazione è molto più facile ed intuitiva.
- Grande velocità d'interazione con l'utente e tempi d'attesa minimi durante il cambio di schermate.
- Maggiore possibilità di sfruttare le varie interazioni con il dispositivo come i comandi vocali e le gestures.
- Sicurezza e protezione garantita dallo store dato che l'app deve essere approvata prima di poter essere pubblicata.
- I dati vengono salvati sul dispositivo per poterne beneficiare a un nuovo rilancio dell'app.

Sfortunatamente lo sviluppo di app native non comporta solo vantaggi, ma anche diversi svantaggi. Di seguito verranno elencati alcuni degli *svantaggi* più importanti:

- La stessa applicazione deve essere sviluppata per ogni piattaforma differente questo implica che bisogna conoscere i vari linguaggi di programmazione per ogni piattaforma per cui si vuole creare l'applicazione.
- Maggiori costi e tempi di sviluppo data la necessitò di utilizzo di diversi linguaggi.
- Investimento di tempo e denaro per la manutenzione dell'app data un'assenza di retrocompstibilità nel caso il dispositivo venga aggiornato.

- Necessità di essere approvati dallo store nel quale si vuole pubblicare l'applicazione.

Come abbiamo visto, le app native sono caratterizzati sia da aspetti positivi che aspetti negativi. [15]

2.1.4 Android

Android è uno dei sistemi operativi più utilizzati per lo sviluppo di applicazioni native. Al momento più del 70% dei dispositivi nel mondo utilizza Android, il resto adotta iOS.[16]
Per sviluppare in android si utilizza come IDE Android Studio o Eclipse mentre il codice scritto si basa su Java o su un suo derivato, ovvero, Kotlin. Android è creato da Google ed è quindi possibile utilizzare diversi strumenti di sviluppo forniti dall'azienda come:[17]

- **Andorid Jetpack**:
  Una serie di librerie, strumenti e informazioni che guidano lo sviluppatore a creare app di qualità fornendo linee guida e semplificando l'implementazione di funzioni più complesse. [18]
- **Firebase**:
  Una piattaforma utilizzata per lo sviluppo di applicazioni web e mobili.
- **Android SDK**:
  L'insieme di strumenti di sviluppo fornito per Android.

Sviluppare in Andorid porta a diversi vantaggi, il più importante è la sua natura open source, infatti si ha accesso a più funzionalità rispetto ad iOS. Inoltre, in Andorid il processo di pubblicazione dell'applicazione sullo store non è così lungo ma, anzi, può essere terminato in poche ore.
Scegliere di sviluppare in questo modo un'app nativa porta anche ad alcuni svantaggi, il più noto è la quantità ingente di tempo necessaria per testare le applicazioni; questo è dovuto alla presenza di diverse versioni e un grande numero di dispositivi a differenza di iOS.

2.1.5 iOS

Il sistema operativo iOS è l'altro ambiente di sviluppo più comunemente usato, rispetto ad Android occupa un mercato inferiore ma è comunque in continua espansione. La maggior parte delle app Android vengono sviluppate anche per iOS.
Per sviluppare in iOS occorre utilizzare come IDE *Xcode* e come linguaggio di programmazione *Swift* e *Objective-C*.
Lo sviluppo in iOS permette di beneficiare dei seguenti strumenti:

- **Swift Playgrounds**: un'applicazione sviluppata da Apple per poter programmare in Swift, interfacciandosi con un ambiente semplice e intuitivo.
- **TestFlight**: app utilizzata per testare le applicazioni sviluppate in iOS.
- **iOS SDK**: un insieme di strumenti di sviluppo fornito dalla Apple utilizzate tramite un'integrazione con Xcode.

Se si decide di sviluppare in iOS si ottengono dei vantaggi che Android non possiede. Innanzitutto vi è un maggiore profitto economico perchè i possessori di Apple spendono di più rispetto agli utenti Android.
Il secondo punto a favore è che il numero di dispositivi Apple è inferiore rispetto a quelli Android, quindi il problema di creare app compatibili con molti dispositivi si riduce. Inoltre, con Apple possiamo beneficiare di una serie di interfacce grafiche create dagli sviluppatori per rendere più semplice ed intuitiva l'user experience.
Ovviamente non mancano gli aspetti negativi utilizzando questa metodologia; in primo luogo vi è una limitata flessibilità data dal fatto che il sistema è chiuso e di conseguenza non open source.
Il secondo aspetto riguarda l'app store, che richiede una serie di regole più particolari per quanto riguarda la pubblicazione della propria app e per questo motivo è richiesta una tempistica maggiore per la pubblicazione all'interno dello store online.

2.1.6 Confronto tra architetture e caratteristiche

Sviluppare un'app nativa può portare a molti vantaggi e, come visto in precedenza, questi cambiano a seconda dell'ambiente in cui si vuole sviluppare. Oltre alle differenze sopra elencate ce ne sono altre dal punto di vista tecnico come l'architettura e il kernel. Di seguito verranno riportate le principali differenze.

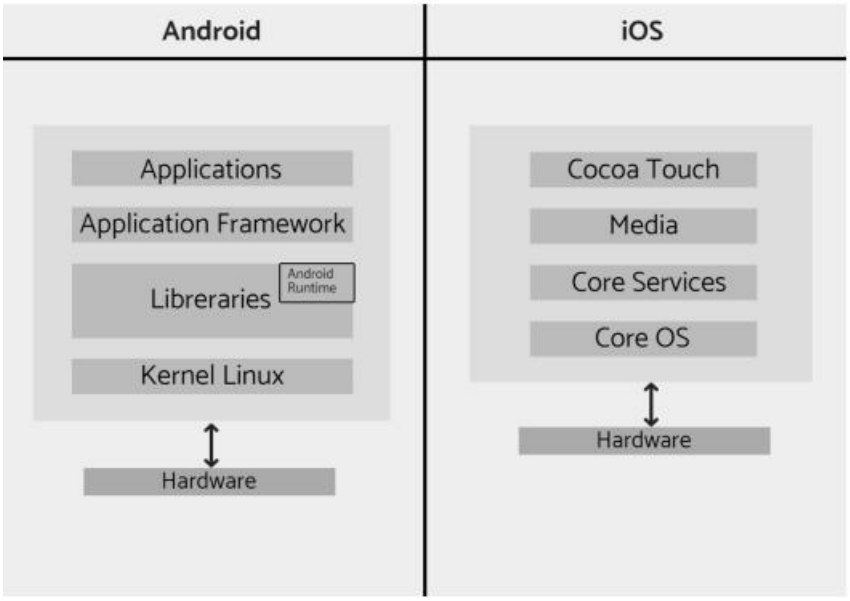

Figura 2.5: Architetture Android e iOS a confronto

Nella parte sinistra dell'immagine possiamo notare come Android utilizzi un kernel Linux (open source). Questo contiene tutti i driver essenziali come audio, camera ecc.
Sopra sono presenti le librerie open source tra le quali abbiamo SSL, SQLite e molte altre.
Possiamo notare come nello stesso livello sia presente la scritta Android Runtime; essa contiene una Java Virtual Machine specificatamente fatta per Android. Grazie a questa JVM si a la possibilità di scrivere app in Java.
Il livello successivo, ovvero *Application framework* fornisce vari servizi di alto livello che possono essere utilizzati dagli sviluppatori, mentre l'ultimo livello rappresenta l'applicazione installata sul dispositivo.[19]

Android, inoltre, introduce il concetto di indipendenza tra ogni differente applicazione. Identificata formalmente come "utente", ogni app viene eseguita con un processo indipendente sulla JVM. Ogni "utente" è installato in una sua directory sul file system, delimitando quindi dei confini virtuali e facendo in modo che un'app non influenzi il comportamento di un'altra, evitando la nascita di conflitti.

Sulla destra, invece, è presente l'architettura iOS. Iniziando dal fondo si ha il *core OS* che contiene le funzionalità di basso livello necessarie alle applicazioni. Il livello *Core Services* offre la possibiltà di sfruttare alcuni servizi attraverso dei framework e fornisce varie interfacce. Successivamente si ha il livello *Media* che supporta l'uso di audio-video con semplicità e il livello denominato *Cocoa Touch* che contiene una collezione di fra-

mework utilizzati per lo sviluppo di applicazioni iOS.

2.1.7 App Ibride

Le applicazioni ibride hanno la principale caratteristica di poter essere create per diverse piattaforme in una sola volta e scrivendo un codice universale.
Un'app ibrida racchiude tutti i vantaggi delle applicazioni native e web in un'unica soluzione di sviluppo.
Principalmente le app ibride vengono sviluppate con due approcci differenti: il primo è chiamato *Web Wrapper* e consiste nella crazione di un nucleo in HTML, CSS e JavaScript, successivamente incapsulato all'interno di applicazioni native. Per poter avere accesso a tutte le funzionalità del dispositivo vengono utilizzati dei plugin specifici, questo metodo viene spesso utilizzato da Ionic. In questo caso il punto di forza è l'incapsulamento di un'applicazione web dentro un'applicazione nativa che dipende dalla piattaforma in cui viene installata. Tramite questo procedimento il nucleo dell'app può essere eseguito in un browser incorporato nell'app nativa che rimane invisibile all'utente finale così da avere una grafica simile a quando si naviga su internet.
Per visualizzare l'applicazione nel seguente modo è necessario crare un *WebView* per Android e una *WKWebView* per iOS.[20]

Il secondo metodo riguarda sviluppare tutto tramite un framework o un SDK che restituirà come risultato le applicazioni native, andando a tradurre il codice generato nel codice nativo di quella piattaforma come avviene con Xamarin, React Native e Flutter.
I maggiori punti di forza che riguardano lo sviluppo di app ibride sono i costi di sviluppo inferiori, la rapidità con cui si può creare l'applicazione simultanemente per diversi dispositivi e il fatto che essendo un unico codice è più facile l'aggiornamento o la modifica.

Occorre ricordare che le app ibride non danno la steesa User Experience e performance delle app native.
Di seguito verranno approfonditi alcuni framework o SDK che possono essere utilizati per la creazione di applicazioni ibride evidenziandone le principali caratteristiche. Principalmente la scelta di questi framework viene fatta in base al linguaggio utilizzato, degli IDE e dei vari strumenti di cui lo sviluppatore ha bisogno.

Xamarin

Xamarin è una piattaforma open source utilizzabile in combinazione con Visual Studio.
Il suo compito è la comunicazione tra il codice condiviso e la piattaforma sottostante.
Le funzionalità native dei vari sistemi operativi mobile sono esposte come classi .NET

le quali consentono un accesso completo alle API per essere più vicine possibili alle app native.[21]

I linguaggi attualmente supportati per uno sviluppo con Xamarin sono C# e F#. Attraverso l'integrazione con Visual Studio è possible utilizzare diversi strumenti utili allo sviluppo come, ad esempio, il designer per creare interfacce grafiche o *IntelliSense* per la documentazione delle API native.
Xamarin, inoltre, offre uno strumento denominato *Xamarin.Forms* che permette di creare interfacce grafiche condivisibili su tutte le piattaforme; questo dato dal fatto che, compilando le soluzioni create in questo modo, viene generata in un singolo colpo un'app per ogni piattaforma. Ovviamente si può anche scegliere di condividere tutto il codice che non sia un'interfaccia grafica e sviluppare diverse UI per ogni piattaforma, andandole ad integrare con il codice condiviso.

Un altro strumento utile è *Xamarin.Essentials*, una libreria che fornisce tutte le API necessarie per poter utilizzare al meglio alcune funzionalità del dispositivo.
Le applicazioni Xamarin possono essere compilate sia in file .apk (per le app Android) sia in file .ipa (per le app iOS).

React Native

React Native è un framework open source creato da Facebook per dare la possibilità di utilizzare ReactJS nella creazione di applicazioni native. Il codice viene scritto in JavaScript e successivamente convertito in codice nativo.

Un fatto molto importante di React Native è la user interface in quanto possiede delle interfacce responsive chiare, efficienti e intuitive.
La metodologia di REACT si basa sull'avere componenti che racchiudano il codice nativo e comunichino con le API native attraverso JavaScript e i paradigmi dell'interfaccia utente dichiarativa di REACT.[22]
Ciò comporta che quando l'app viene creata vengono utilizzati gli elementi nativi grafici del sistema operativo mobile.
La chiave del funzionamento di React Native è proprio il mappare i componenti grafici JavaScript con i componenti nativi del sistema operativo. I componenti dello sviluppo nativo si chiamano Native Component a differenza di quelli usati in ReactJS chiamati Web Component.

Per programmare con React occorre utilizzare Node.js, e quindi anche dei gestori di pacchetti come Npm o Yarn.
Inoltre, si ha la possibilità di scrivere codice usando Objective-C/Swift o Java per creare

dei componenti nativi e integrarli alla nostra app.
Di seguito è riportata una foto dell'architettura di React Native:

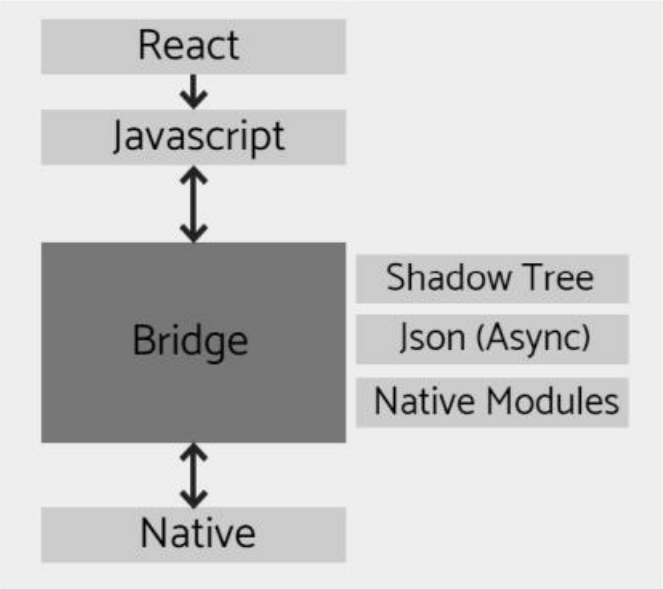

Figura 2.6: Architettura React Native

La struttura è formata da 4 livelli; i livelli React, Javascript e Native sono facilmente comprensibili e rappresentano il codice scritto in React con i vari componenti.
L'aspetto principale si trova nel blocco *Bridge*, che attua il collegamento tra la parte nativa e quella in JavaScript.
Tramite questo componente avviene la comunicazione asincrona e bidirezionale tra le due parti grazie all'uso di messaggi JSON ed è creato attraverso linguaggi C o C++, quindi multipiattaforma. Lo *Shadow Tree*, invece, ha il compito di creare l'albero di elementi che compongono l'interfaccia grafica attraverso i messaggi ricevuti.

In sintesi, il funzionamento di React Native si basa nell'integrazione di questi componenti e nella loro connessione.

Flutter

Flutter è un nuovissimo ambiente di sviluppo basato su Dart. Flutter non è solo un framework ma anche un Software Development Kit (SDK) permettendo di creare applicazioni utilizzando un qualsiasi editor di testo, purché collegato con il tool di linea di comando offerto da Google per lo sviluppo di app ibride.
Per quanto riguarda il suo funzionamento, le applicazioni scritte in Dart vengono pas-

sate al motore per la compilazione anticipata che si interfaccia in modo diretto con la piattaforma di destinazione come mostrato nell'immagine riportata al di sotto.

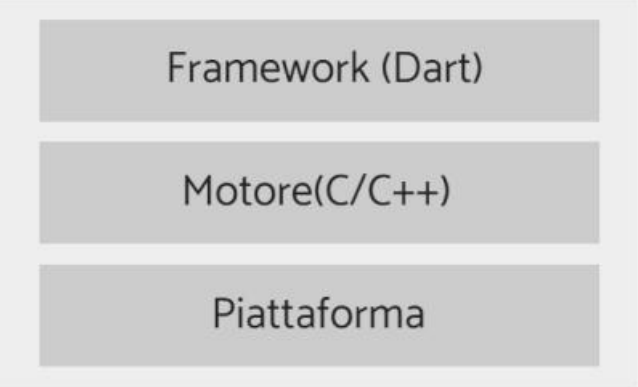

Figura 2.7: Struttura di Flutter

Il *motore* fornisce un supporto per il rendering a basso livello, utilizzando delle librerie grafiche di Google chiamate Skia.
L'organizzazione di Flutter è basata sul concetto di widget con cui si identificano i blocchi base delle UI create con questo SDK. I widget possono essere di due tipi, stateless e stateful, che si differenziano per la presenza dello stato. [23]

Ionic

Ionic è un SDK open source utilizzato per sviluppare app ibride e si basa sulla creazione di un nucleo simile a una web app che viene incapsulato in un wrapper, in modo tale da essere utilizzato come se fosse un'app nativa.
I componenti di Ionic vengono scritti con i classici linguaggi dei browser come HTML, CSS e JavaScript.[24]

Inizialmente le applicazioni Ionic venivano create in Cordova, il cui compito è quello di impacchettare il codice scritto e fornire dei plugin per l'accesso alle funzionalità native nel caso necessario. Nella maggior parte dei casi però, viene utilizzato Capacitor che crea il Wrapper tra l'app scritta in tecnologie web e il dispositivo nativo.

Nell'immagine sottostante possiamo visualizzare l'architettura utilizzata da Ionic per creare app mobile attraverso l'uso di Cordova.

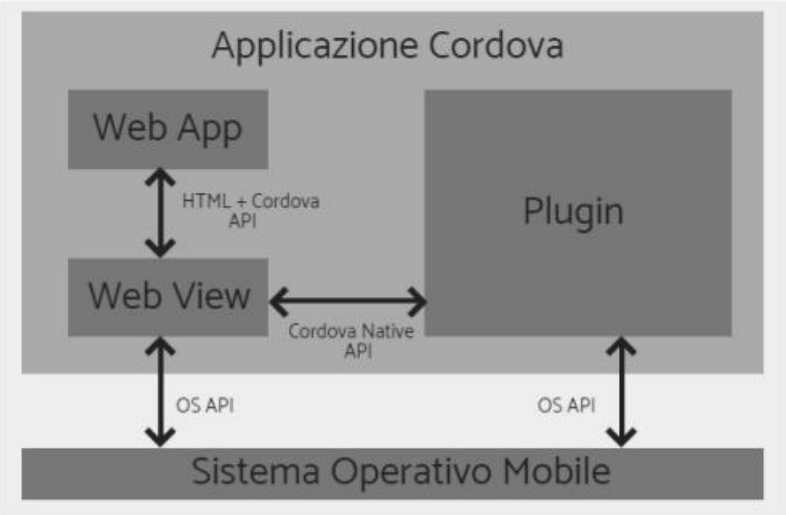

Figura 2.8: Architettura Ionic

Come si può vedere è necessario utilizzare un plugin e una Web View per permettere la scrittura di app ibride attraverso l'uso di linguaggi di programmazione web. La comunicazione con il sistema operativo, invece, avviene con l'uso di API gestite da Cordova o Capacitor.[25]

Capitolo 3

Progettazione

3.1 Obiettivo

Il software utilizzato dall'azienda Infoservice S.R.L. fornisce supporto per la gestione dei dispositivi antincendio nelle aziende. L'azienda aveva la necessità di ampliare l'applicazione sviluppando un modulo chat. L'obiettivo principale è quello di rendere più efficiente l'applicazione che attualmente utilizzano i tecnici in modo da migliorarne l'efficienza e l'operato. Attraverso questo nuovo modulo, i tecnici possono parlare tra loro all'interno della piattaforma senza dover utilizzare applicazioni terze.

3.2 Analisi dei requisiti

In questa sezione verranno discussi i requisiti e le funzionalità che l'applicazione si propone di soddisfare, tenendo conto delle richieste dell'azienda e degli utilizzatori della piattaforma.

3.2.1 Requisiti funzionali

L'applicazione dovrà garantire le seguenti funzionalità:

- Possibilità di registrarsi all'applicazione.
- Possibilità di autenticarsi all'applicazione.
- Possibilità di effettuare il logout dalla piattaforma.
- Possibilità di inviare e ricevere messaggi di testo all'interno della chat con ogni utente registrato all'applicazione.
- Possibilità di inviare e ricevere file da altri utenti.

- Possibilità di inviare e ricevere foto da altri utenti.
- Possibilità di consultare la lista dei media scambiati con l'utente della singola chat.
- Possibiltà di attivare e disattivare la modalità nottura all'interno dell'applicazione
- Possibilità di cercare un utente attraverso l'utilizzo dell'apposita barra di ricerca.
- Possibilità di inviare e ricevere emoticon.
- Possibilità di visualizzare lo stato attuale degli utenti (online/offline).
- Possibilità di visualizzare se il messaggio inviato è stato letto.

3.2.2 Requisiti non funzionali

- Si necessita di un database remoto per contenere tutti i dati necessari all'applicazione.
- Si necessita di una connessione a internet per poter utilizzare l'applicazione.
- La registrazione dell'utente va a buon fine soltanto se tutti i campi sono compilati nella maniera corretta (il formato dell'email deve essere corretto, la password deve essere lunga almeno 6 caratteri, le due password devono coincidere)

3.2.3 Casi d'uso

In questa sottosezione andremo ad identificare gli attori e i casi d'uso, ovvero coloro che interagiscono con l'applicazione e le possibili azioni che possono compiere.

Attori del sistema

Nella tabella seguente sono riportati gli attori che compongono il sistema o che interagiscono con esso:

Attori	Descrizione
Utente non registrato	L'utente non registrato ha accesso alla pagina di Login da cui potrà successivamente passare a quella di Registrazione per registrarsi.
Utente non autenticato	L'utente non autenticato ha accesso alla pagina di Login dell'app per potersi autenticare.
Utente autenticato	L'utente autenticato ha accesso completo all'applicazione e può utilizzare tutte le funzioni messe a disposizione.

Tabella 3.1: Attori dell'applicazione

Diagramma dei casi d'uso

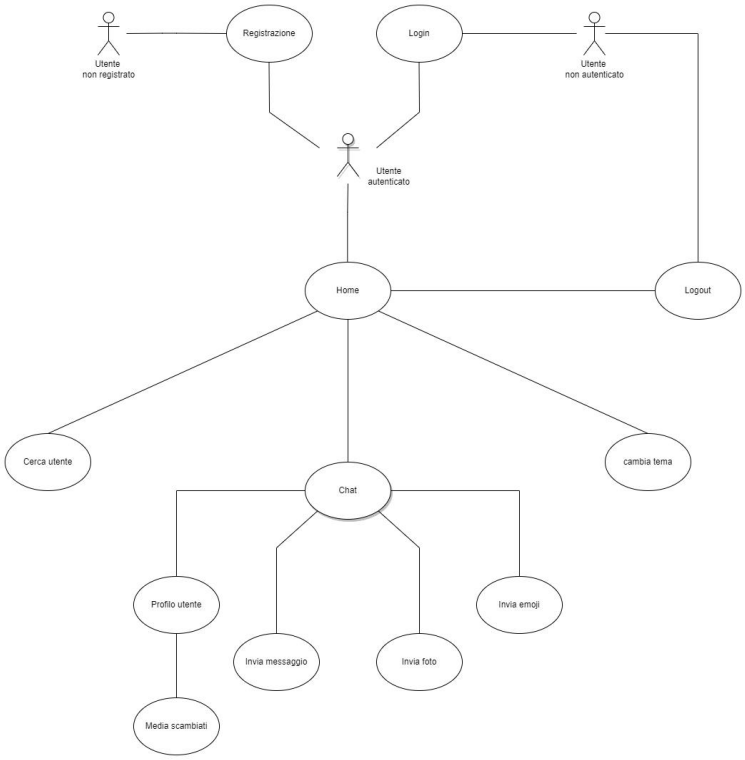

Figura 3.1: Diagramma dei casi d'uso

Elenco dei casi d'uso

I casi d'uso vengono riportati ognuno sotto un titolo dopo averli individuati tramite l'Analisi dei Requisiti, seguito da un'individuazione degli attori coinvolti, una descrizione testuale, il flusso delle azioni descritte dal caso d'uso, condizioni precedenti ed eventualmente condizioni a posteriori.

Registrazione

- **Attori**: Utente non registrato.
- **Scopo e descrizione**: Questo caso d'uso permette all'utente di richiedere l'autorizzazione di accesso al sistema inserendo i propri dati nella form di registrazione. Dopo avere inserito le informazioni richieste, l'utente invia i dati e il sistema conclude la procedura di registrazione, non prima di aver effettuato un controllo sulla validità dei dati inseriti nei campi.
- **Precondizione**: Il sistema mostra la form di input necessaria per l'inserimento dei dati da parte dell'utente.
- **Flusso principale degli eventi**: L'utente inserisce i dati all'interno della form.
- **Postcondizione**: Il sistema inserisce i dati immessi all'interno del database remoto.
- **Scenari alternativi**: L'utente visualizza un alert di errore relativo ai valori errati che sono stati inseriti.

Login

- **Attori**: Utente non autenticato.
- **Scopo e descrizione**: Questo caso d'uso consente l'accesso al sistema ed è regolamentato da una form in cui vanno inseriti l'email e la password dell'utente che deve aver precedentemente effettuato la registrazione. Il sistema, una volta eseguita l'autenticazione, consente l'accesso all'applicazione e all'utilizzo delle sue funzionalità.
- **Precondizione**: Il sistema mostra la form di input necessaria per l'inserimento dei dati da parte dell'utente.
- **Flusso principale degli eventi**: L'utente inserisce i dati all'interno della form.
- **Postcondizione**: Il sistema ha eseguito l'autenticazione dell'utente.

- **Scenari alternativi**: L'utente visualizza un alert di errore in quanto le credenziali inserite non risultano essere corrette.

Logout

- **Attori**: Utente autenticato.
- **Scopo e descrizione**: Questo caso d'uso consente all'utente di effettuare il logout dall'applicazione.
- **Precondizione**: Il sistema ha una sessione attiva dell'utente e consente a quest'ultimo la possibilità di effettuare il logout.
- **Flusso principale degli eventi**: L'utente effettua il logout.
- **Postcondizione**: Il sistema ha eseguito il logout con successo.

Home

- **Attori**: Utente autenticato
- **Scopo e descrizione**: Questo caso d'uso consente all'utente di essere all'interno della Homepage.
- **Precondizione**: L'utente deve aver effettuato l'accesso.
- **Flusso principale degli eventi**: L'utente ha a disposizione diverse funzionalità all'interno della Homepage.
- **Postcondizione**: Nessuna.

Ricerca utente

- **Attori**: Utente autenticato.
- **Scopo e descrizione**: Questo caso d'uso consente all'utente di visualizzare la lista degli user registrati all'applicazione, appositamente filtrata per nome attraverso quanto inserito nell'apposita casella di ricerca sovrastante.
- **Precondizione**: Il sistema permette di filtrare la lista degli utenti attraverso una casella di testo.
- **Flusso principale degli eventi**: L'utente ha la possibilità di inserire l'username da ricercare e la lista sottostante verrà filtrata in modo dinamico.
- **Postcondizione**: Il sistema mostra la lista degli utenti il cui username corrisponde con quello desiderato.

Visualizza chat con l'utente

- **Attori**: Utente autenticato.
- **Scopo e descrizione**: All'interno di questa schermata l'utente visualizza la lista dei messaggi scambiati con l'user scelto.
- **Precondizione**: Il sistema fornisce delle funzionalità che l'attore può eseguire.
- **Flusso principale degli eventi**:
  - L'attore deve poter visualizzare la lista dei messaggi.
  - L'attore deve poter inviare un messaggio.
- **Postcondizione**: Il sistema aggiorna la lista dei messaggi in seguito alla selezione di una funzione.

Invia messaggio

- **Attori**: Utente autenticato.
- **Scopo e descrizione**: Questo caso d'uso consente all'utente di inviare un nuovo messaggio ad un altro utente registrato alla piattaforma.
- **Precondizione**: Il sistema restituisce la lista dei messaggi nella chat.
- **Flusso principale degli eventi**: L'utente immette il messaggio attraverso il box di input e premendo il pulsante provvede ad inviarlo.
- **Postcondizione**: Il sistema inserisce il messaggio all'interno del database e aggiorna automaticamente la schermata della chat.

Invia immagini

- **Attori**: Utente autenticato.
- **Scopo e descrizione**: Questo caso d'uso consente all'utente di caricare un'immagine contenuta all'interno del proprio dispositivo.
- **Precondizione**: L'utente ha selezionato l'immagine da caricare all'interno delle immagine contenute nel proprio dispositivo.
- **Flusso principale degli eventi**: L'utente seleziona l'immagine da caricare e premendo il pulsante provvede ad inviarla;
- **Postcondizione**: Il sistema inserisce il file all'interno del database e aggiorna la schermata contenente la lista dei media scambiati con l'altro utente.

Visualizza immagini

- **Attori**: Utente autenticato.
- **Scopo e descrizione**: Questo caso d'uso consente all'utente di visualizzare all'interno di questa schermata la lista delle immagini scambiate con l'altro utente.
- **Precondizione**: Il sistema fornisce delle funzionalità che l'attore può eseguire.
- **Flusso principale degli eventi**:
  - L'attore deve poter visualizzare la lista delle foto.
  - L'attore deve poter inviare una foto.
- **Postcondizione**: Il sistema aggiorna la lista delle foto scambiate in seguito alla selezione di una funzione.

Attivare la modalità notturna

- **Attori**: Utente autenticato.
- **Scopo e descrizione**: Questo caso d'uso consente all'utente di attivare la modalità notturna per la propria applicazione.
- **Precondizione**: Il sistema fornisce delle funzionalità che l'attore può eseguire.
- **Flusso principale degli eventi**: L'utente preme il pulsante attivando o disattivando la modalità notturna.
- **Postcondizione**: Il sistema cambia il tono dei colori una volta attivata la modalità notturna.

Inviare emoticon

- **Attori**: Utente autenticato
- **Scopo e descrizione**: Questo caso d'uso consente all'utente di inviare e ricevere emoticon all'interno della chat con un altro user.
- **Precondizione**: Il sistema fornisce delle funzionalità che l'attore può eseguire.
- **Flusso principale degli eventi**: L'utente preme il pulsante delle emoticon e sceglie l'emoticon da inviare
- **Postcondizione**: Il sistema inserisce il messaggio all'interno del database e aggiorna automaticamente la schermata della chat.

Visualizzare lo stato degli utenti

- **Attori**: Utente autenticato.
- **Scopo e descrizione**: Questo caso d'uso consente all'utente di vedere lo stato degli altri utenti (se sono online o offline).
- **Precondizione**: Il sistema fornisce delle funzionalità che l'attore può eseguire.
- **Flusso principale degli eventi**: L'utente può vedere lo stato degli altri user attraverso la lista utenti.
- **Postcondizione**: Non avviene nulla.

Visualizzare lo stato del messaggio

- **Attori**: Utente autenticato.
- **Scopo e descrizione**: Questo caso d'uso consente all'utente di vedere lo stato del messaggio inviato a un altro user, ovvero se il messaggio è stato già letto o meno.
- **Precondizione**: Il sistema fornisce delle funzionalità che l'attore può eseguire.
- **Flusso principale degli eventi**: L'utente può vedere lo stato del messaggio aprendo la chat con lo user interessato.

3.3 Progettazione del database di supporto

Il database rappresenta l'insieme di tutte le informazioni da cui il sistema può attingere. La corretta progettazione di un database consente di accedere e utilizzare tutte le informazioni raccolte con lo scopo di raggiungere gli obiettivi prefissati.
Esistono due tipi di database, relazionali e non relazionali. Per la creazione di questa applicazione è stato utilizzato un database non relazionale ovvero **MongoDB**.
MongoDB è un database non relazionale, scalabile e ad alte prestazioni progettato per gestire l'archiviazione orientata ai documenti. MongoDB è un DBMS è classificato come database NoSQL e si allontana dalla tradizionale struttura basata su tabelle dei classici database relazionali in favore di documenti in stile JSON con schema dinamico rendendo l'integrazione di dati di alcune applicazioni più semplice ed efficiente.
Le caratteristiche principali di MongoDB sono le seguenti:[26]

- **Query ad hoc**:
  MongoDB supporta ricerche per campi, intervalli e regular expression. Le query possono restituire campi specifici del documento e includere funzioni definite dall'utente in JavaScript.

- **Indicizzazione**:
  Qualunque campo in MongoDB può essere indicizzato (gli indici sono concettualmente similari a quelli tradizionali RDBMS). Inoltre, sono disponibili anche indici secondari, indici unici, indici sparsi, indici geospaziali e indici full text.

- **Alta affidabilità**:
  MongoDB fornisce alta disponibilità e aumento del carico gestito attraverso i replica set. Un replica set consiste in due o più copie dei dati. Ogni replica può avere il ruolo di copia primaria o secondaria in qualunque momento. La replica primaria effettua tutte le scritture e le letture. Le repliche secondarie mantengono una copia dei dati della replica primaria attraverso un meccanismo di replicazione incluso nel prodotto. Quando una replica primaria fallisce, il replica set inizia automaticamente un processo di elezione per determinare quale replica secondaria deve diventare primaria. Le copie secondarie possono anche effettuare letture, con dati eventualmente consistenti di default.

- **Sharding e bilanciamento dei dati**:
  MongoDB scala orizzontalmente usando lo sharding. L'utente deve scegliere una chiave di sharding, che determina come i dati di una collection saranno distribuiti tra i vari nodi. I dati sono divisi in intervalli (basati sulla chiave di shard) e distribuiti su molteplici shard (uno shard è un replica set, quindi con una replica primaria e due o più repliche secondarie). MongoDB include anche un meccanismo di bilanciamento dei dati attraverso il quale sposta gli intervalli di dati da uno shard troppo carico a uno shard meno carico, in modo da bilanciare la distribuzione dei dati all'interno del cluster.

- **File storage**:
  MongoDB può essere usato anche come un file system, traendo vantaggio dalla caratteristiche di replicazione e di bilanciamento su più server per memorizzare file, anche di grandi dimensioni. Questa funzione, chiamata GridFS, è inclusa nei driver di MongoDB e disponibile facilmente per tantissimi linguaggi di sviluppo. MongoDB espone delle funzioni per la manipolazione dei file. GridFS è usato, ad esempio, nei plugin di NGINX e lighttpd. Invece di memorizzare il file in un singolo documento, GridFS divide il file in tante parti più piccole, chiamate chunks, e memorizza ognuno di questi chunk in un documento separato. Ai file possono essere associati dei metadati, su cui è possibile anche creare degli indici full-text.

- **Aggregazione**:
  MongoDB supporta due modalità di aggregazione dei dati: il MapReduce e l'Aggregration Framework. Quest'ultimo lavora come una pipeline e permette di ottenere risultati molto più rapidamente del MapReduce grazie all'implementazione in C++.

- **Capped collection**:
  MongoDB supporta collection a dimensioni fisse chiamate capped collection. Questo tipo di collection mantiene l'ordine di inserimento e una volta raggiunta la dimensione definita, si comporta come una coda circolare.

3.3.1 Vantaggi e svantaggi di MongoDB

MongoDB ha anche una serie di vantaggi e svantaggi. I **vantaggi** sono i seguenti:

- È altamente adattabile e flessibile per soddisfare i requisiti e le situazioni di business che cambiano.
- Facile da scalare sia verso l'alto che verso il basso.
- Permette di eseguire query e restituire campi all'interno di un documento.
- Supporta la replicazione dei dati in modo da poter conservare copie dei vostri dati e non perderli mai.
- Permette l'archiviazione di diversi tipi di file di diverse dimensioni senza influenzare il vostro stack tecnologico.
- Permette la creazione di indici per migliorare le prestazioni di ricerca.
- Funziona su più server e offre la duplicazione dei dati e il load balancing, quindi funziona anche durante un guasto hardware.
- Segue il modello BASE per offrire una maggiore disponibilità dei dati.
- Facile da usare.

Gli **svantaggi** invece sono i seguenti:

- Il modello ACID non è forte rispetto ad altri sistemi di database.
- Non fornisce alcuna opzione per le Stored Procedures, il che significa che non potrete implementare la vostra logica di business a livello di database, a differenza dei database relazionali.
- Le transazioni a volte possono essere complesse o insicure.
- Curva di apprendimento un po' ripida.
- La documentazione è poco strutturata.
- Implica un maggiore consumo di memoria e manca di join o di analisi integrate.

3.3.2 Confronto tra MongoDB e MySQL

Prima di entrare nello specifico ed effettuare il confronto tra i due database occorre una breve introduzione sul database MySQL.

MySQL è un sistema di gestione di database relazionali (RDBMS) gratuito e open-source. Organizza e memorizza i dati in un formato tabellare con righe e colonne in cui i tipi di dati sono correlati. MySQL utilizza il linguaggio SQL, ovvero, un linguaggio di programmazione specifico del dominio che può gestire i dati in un RDBMS eseguendo funzioni sui dati, incluso creare, estrarre, cancellare e modificare. MySQL funziona con molti sistemi operativi, come Windows, macOS e Linux per implementare RDBMS nel sistema di archiviazione di un dispositivo, consentire l'accesso alla rete, gestire gli utenti, facilitare i test di integrità del database e creare backup.

Dopo questa breve descrizione sul database MySQL verranno di seguito elencate le principali differenze tra le due piattaforme di base di dati per dare maggiore consapevolezza su quale scegliere delle due. [27]

Architettura

L'architettura è alla base di ogni sistema e stabilisce il quadro in cui tutte le caratteristiche e le funzionalità possono essere introdotte. È quindi importante conoscere innanzitutto le differenze tra l'architettura di MongoDB e l'architettura di un database MySQL.

MongoDB è basato sull'architettura Nexus, un'architettura che combina le funzionalità dei database relazionali. Può soddisfare le esigenze delle applicazioni moderne offrendo alta scalabilità, disponibilità globale e uno schema flessibile. Pertanto, apportare modifiche al suo design è piuttosto facile.

MySQL d'altra parte, include un'architettura client-server con storage ottimizzato per offrire alte prestazioni e multithreading. La sua documentazione mostra alcune tecniche di ottimizzazione delle prestazioni che riguardano la configurazione invece di mettere a punto le misure SQL.

Schema

Per quanto riguarda lo schema, MongoDB fornisce uno schema flessibile per permettere agli utenti di cambiare il design in base ai requisiti. Permette di combinare e memorizzare facilmente diversi tipi di dati e di modificare lo schema dinamicamente senza tempi morti. Si possono memorizzare più documenti in una collezione anche senza alcuna relazione tra loro, dato che è un sistema di database non relazionale.

In MySQL, occorre definire chiaramente colonne e tabelle prima di memorizzare i dati insieme a righe e colonne. Qui, ogni campo comprende una riga e una colonna. Questo significa che la memorizzazione dei dati non dà la stessa flessibilità che offre MongoDB e questo significa anche un processo di distribuzione e sviluppo più lento. Se, al contrario, si ha uno schema fisso MySQL è meglio: offre una migliore coerenza dei dati senza dover cambiare lo schema più volte.

Linguaggio

Il linguaggio di interrogazione in MongoDB è il MongoDB Query Language (MQL), un linguaggio espressivo e ricco che supporta le funzioni CRUD. Queste funzioni consentono di creare, leggere, aggiornare e cancellare i dati. Inoltre, facilita anche l'aggregazione dei dati e la ricerca testuale.

MySQL vede l'impiego del classico linguaggio SQL che può portare dati da diverse tabelle supportando la funzionalità di join. Attraverso questa operazione è possibile collegare i dati da più tabelle in una query.

Velocità

Dal punto di vista della velocità MongoDB memorizza un grande volume di dati non strutturati e segue un approccio di memorizzazione basato su documenti. Questo comporta che MongoDB memorizza i dati in un singolo documento per un'entità e aiuta a leggere o scrivere i dati più velocemente. Le sue prestazioni sono anche migliori quando si tratta di oggetti a causa del suo storage di oggetti tipo Json.

MySQL, invece, può mostrare prestazioni lente quando ha a che fare con un enorme volume di dati. Questo perché memorizza le tabelle in modo normalizzato e se si ha bisogno di cambiare i dati o estrarli occorre passare attraverso molte tabelle per scrivere e/o leggere i dati e ciò aumenta il carico del server e influisce sulle sue prestazioni.

Sicurezza

Infine, dal punto di vista della sicurezza, mongoDB sfrutta i controlli di accesso basati sui ruoli con permessi flessibili per utenti e dispositivi. A ogni utente viene assegnato un ruolo in base al quale gli vengono dati permessi specifici per accedere ai dati ed eseguire operazioni.

D'altra parte, MySQL ha controlli di accesso basati sui privilegi. Supporta anche strutture di crittografia come MongoDB con un modello di autenticazione simile, che include autorizzazione, autenticazione e auditing.

3.3.3 Somiglianze tra MongoDb e MySQL

Questi due database non presentano solo delle differenze ma anche alcune somiglianze che vengono riportate di seguito:

- Sono database open source e gratuiti.
- Usano un potente linguaggio di interrogazione.
- Supportano la ricerca full-text utilizzando la ricerca per frase e per termine.
- Offrono la ricerca per indici con l'aiuto della ricerca per frase e per testo.
- Hanno un forte supporto comunitario con migliaia di professionisti.
- Offrono l'ottimizzazione dell'indice.
- Offrono la replicazione dei dati attraverso la configurazione principale-secondaria.

3.3.4 Installazione e connessione al database

Il primo passo da fare è installare MongoDB sul proprio pc e mettere il percorso della cartella a partire dal "bin" tra le variabili d'ambiente. Successivamente, aprendo l'applicazione appena scaricata si aprirà una schermata nel quale si può creare il proprio database. Per semplicità sono state lasciate tutte le impostazioni di default compreso l'url che sarà **mongodb://localhost:27017**. Infine, per completare l'operazione, basta premere la scritta **"connect"**

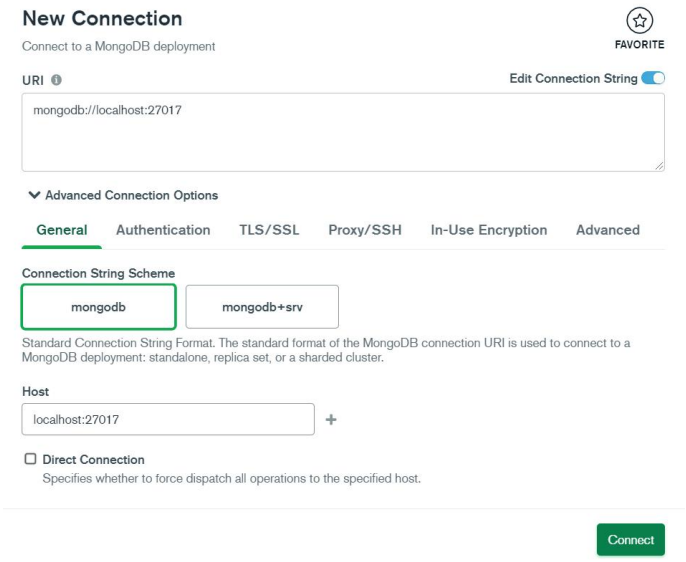

Figura 3.2: Creazione Database

Il passo successivo è quello di installare MongoDB all'interno del progetto e per farlo occorre aprire il terminale, posizionarsi nella cartella del progetto e scrivere **"npm i mongoose"**. Una volta fatto ciò occorre scrivere la connessione al database attraverso l'uso di di codice:

Listato 3.1: database.js

```
const mongoose = require('mongoose');

const databaseConnect = () => {
     mongoose.connect(process.env.DATABASE_URL,{
          useNewUrlParser : true,
          useUnifiedTopology : true
     }).then(()=>{
          console.log('Mongodb Database Connected')
     }).catch(error=>{
          console.log(error)
     })
}
module.exports = databaseConnect;
```

L'url del database si trova nel file *config.env* all'interno della variabile **DATABASE.URL**.

Capitolo 4

Implementazione

Questo capitolo descrive la fase di implementazione delle funzionalità della piattaforma sia lato front-end che back-end. In particolare verranno mostrate le schermate dell'applicazione e le porzioni di codice che hanno reso possibile la realizzazione della stessa e delle sue funzionalità.

Inoltre, occorre sottolineare come l'applicazione è stata progettata e sviluppata non solo da me, ma anche da un mio collega al quale è stato assegnato il medesimo progetto dall'azienda Inforservice S.R.L. . In questo capitolo verranno mostrate le parti di cui mi sono occupato per la maggior parte del tempo, questo però, non ha nulla a che vedere con la conoscenza globale dell'applicazione. Infatti, per ogni parte del progetto, c'è stato un grande lavoro collaborazione con il mio collega e posso affermare di conoscere l'intero progetto, ma di essermi concentrato più su alcuni argomenti rispetto ad altri per una semplice questione di divisione dei compiti e ottimizzazione del lavoro.

Prima di addentrarci nel vivo della parte relativa alla programmazione è necessario introdurre il concetto di Web Socket e Socket.IO poiché sono delle tecnologie di fondamentale importanza nello sviluppo del nostro software.

4.1 Socket.IO

In passato i siti web venivano ricaricati ogni volta che veniva richiesta una risorsa. Ciò ha introdotto ritardi inutili che hanno aumentato il tempo medio di attesa infatti spesso si è arrivati a dover attendere alcuni minuti per recuperare una pagina o un file particolare. Le applicazioni in tempo reale (messaggeria istantanea, giochi online, notifiche push ecc..), invece, sono quelle applicazioni che vengono eseguite in un determinato intervallo di tempo in modo tale che all'utente venga presentata una copia *immediata* e *aggiornata* della risorsa. La latenza di queste applicazioni deve essere mantenuta il più bassa possi-

bile per offrire un'esperienza utente fluida e coerente.
Socket.IO è una di queste librerie JavaScript che viene utilizzata dai programmatori nello sviluppo di "applicazioni Web" in tempo reale.

4.1.1 Perché scegliere Socket.IO

La maggior parte delle applicazioni su Internet oggi sono basate sull'architettura client-server. Un client è qualcuno che richiede qualcosa da un server. Un server, in base alla richiesta, risponde con risultati appropriati. Queste due entità sono completamente diverse l'una dall'altra a causa della natura dei compiti che svolgono.
Un browser è un perfetto esempio di applicazione client. I client (browser) generalmente comunicano con i server tramite richieste e risposte HTTP. Il problema con questa comunicazione è che è possibile inviare una richiesta o una risposta alla volta. Inoltre, le intestazioni HTTP contengono molte informazioni ridondanti, che sono inutili una volta stabilita una connessione tra client e server.
I socket funzionano in maniera differente, lavorando sul layer di trasporto del Network Stack.
Socket.IO si basa sullo stesso concetto e consente la comunicazione **bidirezionale** tra client Web e server. Per gestirli separatamente ed efficientemente, si compone di due parti;

- una libreria client JavaScript che viene eseguita sui browser.
- un server Node.js

Socket.IO si basa su *Engine.IO*, che è l'implementazione del livello di comunicazione bidirezionale cross-browser/cross-device basato sul trasporto.
Esso porta le seguenti funzionalità in Socket.IO:

- **Affidabiità**: può stabilire la connessione anche in presenza di proxy, load-balancer,firewall e software antivirus.
- **Supporto per la riconnessione automatica**: a meno che non sia indicato esplicitamente nel codice, la libreria client tenterà di riconnettersi per sempre, fino a quando il server non sarà nuovamente disponibile.
- **Rilevamento della disconnessione**: consente sia al server che al client di sapere quando l'altro non risponde più.
- **Supporto multiplexing**: permette di avere più canali di comunicazione sulla medesima connessione.
- **Supporto per lo streaming binario**: consente di emettere qualsiasi dato binario serializzabile come ArrayBuffer, Blob, ecc.

4.1.2 Cos'è un Web Socket

Un Web Socket è un protocollo che fornisce una comunicazione full-duplex, ovvero consente la comunicazione in entrambe le direzioni contemporaneamente.
È una moderna tecnologia web in cui esiste una connessione continua tra il browser (client) dell'utente e il server. In questo tipo di comunicazione (tra il server web e il browser web), entrambi possono scambiarsi messaggi in qualsiasi momento. Tradizionalmente sul Web, avevamo un formato di richiesta/risposta in cui un utente invia una richiesta HTTP e il server risponde a quella. Questo è ancora applicabile nella maggior parte dei casi, in particolare quelli che fanno uso di RESTful API. Ma si sentiva la necessità che il server comunicasse anche con il client, senza essere necessariamente interrogato da parte del client. Il server stesso dovrebbe essere in grado di inviare informazioni al client o al browser.

4.1.3 Differenze tra Web Socket e protocollo HTTP

Sia Web Socket che HTTP sono protocolli di comunicazione utilizzati nella comunicazione client-server.

Protocollo HTTP

Il Protocollo HTTP è unidirezionale, in cui il client invia la richiesta e il server invia la risposta. Facciamo un esempio: quando un utente invia una richiesta al server questa richiesta va sotto forma di HTTP o HTTPS. Dopo che il server riceve la richiesta, esso inoltra la risposta al client.
Ogni richiesta è associata alla sua corrispondente risposta, infatti dopo aver inviato la risposta la connessione viene chiusa. Ogni richiesta HTTP o HTTPS stabilisce ogni volta la nuova connessione al server e dopo aver ricevuto la risposta la connessione viene terminata da sola. L'HTTP è infatti un protocollo *stateless* che viene eseguito su TCP.
HTTP può essere eseguito su qualsiasi protocollo affidabile orientato alla connessione come TCP, SCTP. Quando un client invia una richiesta HTTP al server, viene aperta una connessione TCP tra client e server e dopo aver ricevuto la risposta, la connessione TCP viene terminata.

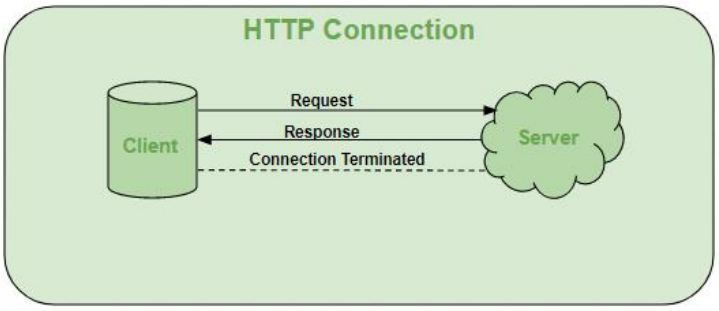

Figura 4.1: Connessione HTTP

Web Socket

WebSocket è bidirezionale, un protocollo full-duplex utilizzato nello stesso scenario di comunicazione client-server, a differenza di HTTP inizia da **ws:**// o **wss:**//.
È un protocollo *stateful*, il che significa che la connessione tra client e server rimarrà attiva fino a quando non verrà terminata da una delle parti (client o server). Dopo aver chiuso la connessione da parte del client e del server, la connessione viene terminata da entrambe le estremità.
Facciamo un esempio di comunicazione client-server, c'è il client che è un browser web e un server, ogni volta che avviamo la connessione tra client e server, il client-server effettua l'handshaking e decide di creare una nuova connessione e questa connessione rimarrà attiva fino a quando non sarà terminata da uno dei due componenti. Quando la connessione è stabilita e attiva, la comunicazione avviene utilizzando il medesimo canale di connessione, fino a che non viene terminata.
Una volta che il collegamento di comunicazione e la connessione sono state aperte, lo scambio di messaggi avverrà in modalità bidirezionale fino a quando la connessione tra client e server persiste. Se qualcuno di loro decide di chiudere la connessione, essa verrà chiusa da entrambe le parti. Il modo in cui funziona il socket è leggermente diverso da come funziona HTTP, il codice di stato 101 indica il protocollo di commutazione in WebSocket.

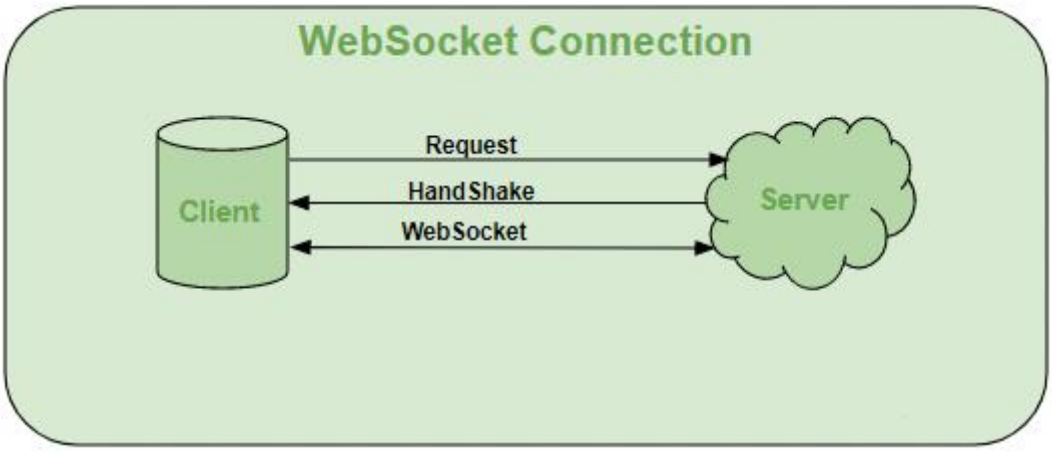

Figura 4.2: Connessione WebSocket

4.1.4 Quando è possibile utilizzare Web Socket

- **Applicazioni Web in tempo reale**: Le applicazioni Web in tempo reale utilizza un socket Web per mostrare i dati al client, che vengono continuamente inviati dal server back-end. In WebSocket, i dati vengono continuamente trasmessi nella stessa connessione che è già aperta, ecco perché WebSocket è più veloce e migliora le prestazioni dell'applicazione.

- **Applicazioni di gioco**: In un'applicazione di gioco i dati vengono continuamente ricevuti dal server e, senza aggiornare l'interfaccia utente, avranno effetto sullo schermo. L'interfaccia utente viene aggiornata automaticamente senza nemmeno stabilire la nuova connessione, quindi è molto utile in questa categoria di app.

- **Applicazioni di chat**: Le applicazioni di chat utilizzano WebSocket per stabilire la connessione una sola volta per lo scambio, la pubblicazione e la trasmissione del messaggio tra gli utenti. Riutilizza la stessa connessione WebSocket, per l'invio e la ricezione del messaggio e per il trasferimento dei messaggi uno a uno.

4.1.5 Quando non utilizzare Web Socket

WebSocket può essere utilizzato se desideriamo flussi di dati aggiornati o continui in tempo reale che vengono trasmessi sulla rete. Se vogliamo recuperare vecchi dati, o vogliamo ottenere i dati solo una volta per elaborarli con un'applicazione, dovremmo utilizzare il protocollo HTTP, i vecchi dati che non sono richiesti molto frequentemente o recuperati solo una volta possono essere interrogati dalla semplice richiesta HTTP, quindi in questo scenario, è meglio non utilizzare WebSocket.

4.2 Funzionalità

4.2.1 Accesso e login

Appena l'applicazione viene lanciata l'utente viene portato nella pagina di login dove può autenticarsi con le proprie credenziali di accesso o se non ha un account registrarsi.

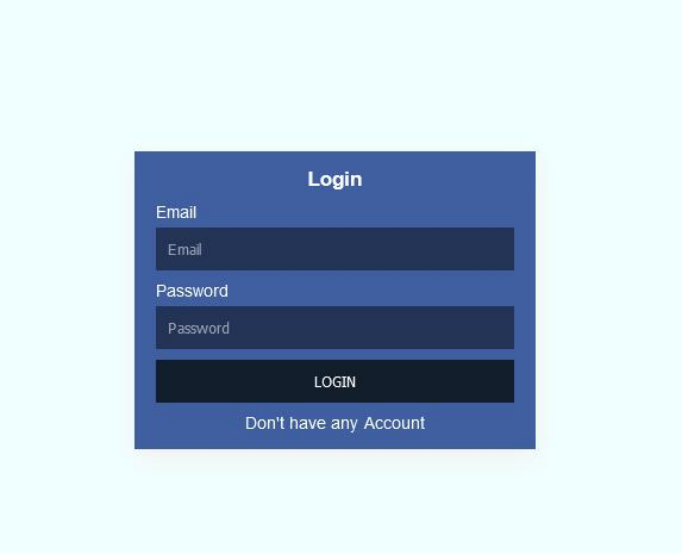

Figura 4.3: Pagina di Login

Il sistema effettua anche un controllo per verificare se l'utente si è già autenticato e se questa condizione è verificata verrà direttamente portato nella schermata principale.

Listato 4.1: Login.jsx

```
1 useEffect(()=>{
2           if(authenticate){
3                 navigate('/');
4           }
```

La funzione **if** presente nel listato 4.1 si trova anche nella pagina di registrazione così che entrambe le pagine non siano più accessibili una volta che l'utente si è autenticato. Se l'utente, invece, non ha effettuato il login deve inserire le proprie credenziali nella form di login della pagina *Login.jsx*.

Listato 4.2: Login.jsx

```
1
2  const Login = () => {
3  ...
4       const [state, setState] = useState({
5            email: '',
6            password : ''
7       });
8
9       const inputHendle = e => {
10           setState({
11                ...state,
12                [e.target.name] : e.target.value
13           })
14      }
15
16      const login = (e) => {
17           e.preventDefault();
18           dispatch(userLogin(state))
19      }
20
21      useEffect(()=>{
22           if(authenticate){
23                navigate('/');
24           }
25           if(successMessage){
26                alert.success(successMessage);
27                dispatch({type : SUCCESS_MESSAGE_CLEAR })
28           }
29           if(error){
30                error.map(err=>alert.error(err));
31                dispatch({type : ERROR_CLEAR })
32           }
33
34      },[successMessage,error])
35 ...
```

I dati inseriti nei campi vengono passati alla pagina *authAction* attraverso l'uso del metodo **dispatch**.

Listato 4.3: authAction.js

```
export const userLogin = (data) => {
    return async (dispath) => {

        const config = {
            headers: {
                'Content-Type': 'application/json'
            }
        }

        try {
            const response = await axios.post('/api/messenger/user-login', data, config);
            localStorage.setItem('authToken', response.data.token);
            dispath({
                type: USER_LOGIN_SUCCESS,
                payload: {
                    successMessage: response.data.successMessage,
                    token: response.data.token
                }
            })
        } catch (error) {
            dispath({
                type: USER_LOGIN_FAIL,
                payload: {
                    error: error.response.data.error.errorMessage
                }
            })
        }
    }
}
```

Questa pagina ha il compito di portare i dati nella parte backend dell'applicazione e per farlo utilizza la rotta **userLogin** presente nella pagina *authRoute.js* grazie alla libreria AXIOS. Questa libreria viene utilizzata per effettuare le chiamate HTTP e comunicare con le rotte del nostro progetto React.

Listato 4.4: authRoute.js

```
...
router.post('/user-login',userLogin);
...
```

Nel listato 4.4 si può leggere la rotta **'/user-login'** di tipo **POST** che invoca il metodo **userLogin** presente nella pagina *authController.js*.

Listato 4.5: authController.js

```
module.exports.userLogin = async (req, res) => {
  const error = [];
  const { email, password } = req.body;
  if (!email) {
    error.push("Per favore inserisci un'email");
  }
  if (!password) {
    error.push("Per favore inserisci una password");
  }
  if (email && !validator.isEmail(email)) {
    error.push("Per favore inserisci un'email valida");
  }
  if (error.length > 0) {
    res.status(400).json({
      error: {
        errorMessage: error,
      },
    });
  } else {
    try {
      const checkUser = await registerModel
        .findOne({
          email: email,
        })
        .select("+password");

      if (checkUser) {
        const matchPassword = await bcrypt.compare(
          password,
          checkUser.password
        );

        if (matchPassword) {
          const token = jwt.sign(
            {
              id: checkUser._id,
              email: checkUser.email,
              userName: checkUser.userName,
              image: checkUser.image,
```

```
40                registerTime: checkUser.createdAt,
41              },
42              process.env.SECRET,
43              {
44                expiresIn: process.env.TOKEN_EXP,
45              }
46            );
47            const options = {
48              expires: new Date(
49                Date.now() + process.env.COOKIE_EXP * 24 * 60 * 60 *
    1000
50              ),
51            };
52
53            res.status(200).cookie("authToken", token, options).json
    ({
54              successMessage: "Login avvenuto con successo",
55              token,
56            });
57          } else {
58            res.status(400).json({
59              error: {
60                errorMessage: ["La tua password non    valida"],
61              },
62            });
63          }
64        } else {
65          res.status(400).json({
66            error: {
67              errorMessage: ["La tua email non    stata trovata"],
68            },
69          });
70        }
71      } catch {
72        res.status(404).json({
73          error: {
74            errorMessage: ["Internal Server Error"],
75          },
76        });
77      }
78    }
79  };
```

In questa pagina vengono effettuati i controlli sui dati inseriti nei campi come, ad esempio, se la password combacia con quella presente nel database. Una volta verificate tutte le condizioni il sistema prenderà dal database il token insieme ai dati dell'utente all'interno della collection **Users** del database ed effettuerà l'accesso con successo.

4.2.2 Registrazione

Se l'utente non si è mai registrato nella piattaforma, può farlo passando dalla schermata di Login a quella di Registrazione premendo sulla scritta **"don't have any account"**, riportata sotto i campi per la form del login.

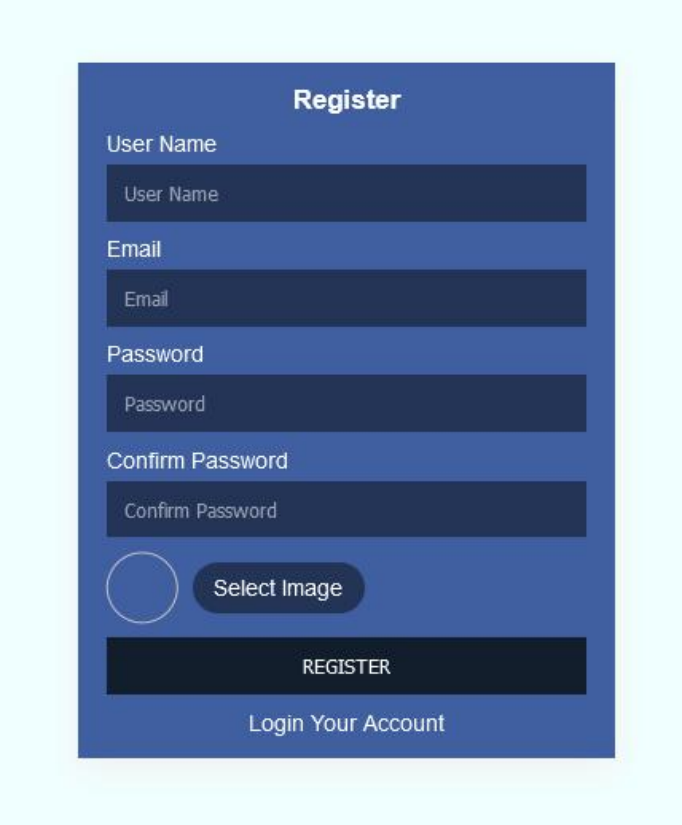

Figura 4.4: Pagina di Registrazione

Nella schermata di registrazione è presente una form con i campi relativi alle generalità dell'utente. Dopo che ogni campo è stato riempito con i propri dati si può premere il pulsante di registrazione.

I dati inseriti nei campi vengono presi dalla parte frontend della pagina di registrazione *Register.jsx* per essere successivamente passati alla parte backend dell'applicazione.

Listato 4.6: Register.jsx

```
const Register = () => {

     const navigate = useNavigate();
     const alert = useAlert();

     const {loading,authenticate,error,successMessage,myInfo} =
   useSelector(state=>state.auth);
     console.log(myInfo);

     const dispatch = useDispatch();

     const [state,setstate] = useState({
          userName : '',
          email:'',
          password:'',
          confirmPassword : '',
          image : ''
     })

     const [loadImage, setLoadImage] = useState('');

     const inputHendle = e => {
          setstate({
               ...state,
               [e.target.name] : e.target.value
          })
     }

     const fileHendle = e =>{
          if(e.target.files.length !==0){
               setstate({
                    ...state,
                    [e.target.name] : e.target.files[0]
               })
          }

          const reader = new FileReader();
```

```
37          reader.onload = () => {
38               setLoadImage(reader.result);
39          }
40          reader.readAsDataURL(e.target.files[0]);
41     }
42
43     const register = e =>{
44
45          const {userName,email,password,confirmPassword, image} =
    state;
46          e.preventDefault();
47
48          const formData = new FormData();
49
50          formData.append('userName',userName);
51          formData.append('email',email);
52          formData.append('password',password);
53          formData.append('confirmPassword',confirmPassword);
54          formData.append('image',image);
55
56          dispatch(userRegister(formData));
57     }
```

I dati vengono formattati sotto forma di coppia chiave-valore con il metodo .append e successivamente con il metodo **dispatch** vengono passati alla pagina *authAction.js*.

Listato 4.7: authAction.js

```
1  export const userRegister = (data) => {
2       return async (dispatch) => {
3
4            const config = {
5                 headers: {
6                      'Content-Type':'application/josn'
7                 }
8            }
9            try{
10                const response = await axios.post('/api/messenger/
    user-register',data,config);
11                localStorage.setItem('authToken',response.data.
    token);
12
13                dispatch({
14                     type : REGISTER_SUCCESS,
15                     payload:{
```

```
16                          successMessage: response.data.
    successMessage,
17                          token : response.data.token
18                      }
19                  })
20
21          } catch(error){
22                dispatch({
23                    type: REGISTER_FAIL,
24                    payload:{
25                          error : error.response.data.error.
    errorMessage
26                    }
27                })
28          }
29      } }
```

authAction.js ha il compito di portare i dati nella parte backend dell'applicazione e per farlo utilizza la rotta userRegister presente nella pagina authRoute.js grazie alla libreria AXIOS.

Listato 4.8: authRoute.jsx

```
1 ...
2 router.post('/user-register',userRegister);
3 ...
```

In questa pagina è presente la rotta '/**user-register**' di tipo **POST** che invoca il metodo **userRegister** presente nella pagina *authController.js*.

Listato 4.9: authcontroller.js

```
1  module.exports.userRegister = (req, res) => {
2    const form = formidable();
3    form.parse(req, async (err, fields, files) => {
4      const { userName, email, password, confirmPassword } = fields;
5
6      const { image } = files;
7      const error = [];
8
9      if (!userName) {
10       error.push("Inserire l'username");
11     }
12     if (!email) {
13       error.push("Inserire l'email");
14     }
```

```
15      if (email && !validator.isEmail(email)) {
16        error.push("Inserire un'email valida");
17      }
18      if (!password) {
19        error.push("Inserire la password");
20      }
21      if (!confirmPassword) {
22        error.push("Inserire la password ripetuta");
23      }
24      if (password && confirmPassword && password !==
    confirmPassword) {
25        error.push("Le due password non coincidono");
26      }
27      if (password && password.length < 6) {
28        error.push("La passoword deve essere lunga almeno 6
    caratteri.");
29      }
30      if (Object.keys(files).length === 0) {
31        error.push("Inserire l'immagine del profilo");
32      }
33      if (error.length > 0) {
34        res.status(400).json({
35          error: {
36            errorMessage: error,
37          },
38        });
39      } else {
40        const getImageName = files.image.originalFilename;
41        const randNumber = Math.floor(Math.random() * 99999);
42        const newImageName = randNumber + getImageName;
43        files.image.originalFilename = newImageName;
44
45        const newPath =
46          __dirname +
47          `../../../frontend/public/image/${files.image.
    originalFilename}`;
48
49        try {
50          const checkUser = await registerModel.findOne({
51            email: email,
52          });
53          if (checkUser) {
54            res.status(404).json({
```

```
55              error: {
56                errorMessage: ["L'email e' gia' in uso"],
57              },
58            });
59          } else {
60            fs.copyFile(files.image.filepath, newPath, async (error)
     => {
61              if (!error) {
62                const userCreate = await registerModel.create({
63                  userName,
64                  email,
65                  password: await bcrypt.hash(password, 10),
66                  image: files.image.originalFilename,
67                });
68
69                const token = jwt.sign(
70                  {
71                    id: userCreate._id,
72                    email: userCreate.email,
73                    userName: userCreate.userName,
74                    image: userCreate.image,
75                    registerTime: userCreate.createdAt,
76                  },
77                  process.env.SECRET,
78                  {
79                    expiresIn: process.env.TOKEN_EXP,
80                  }
81                );
82
83                const options = {
84                  expires: new Date(
85                    Date.now() + process.env.COOKIE_EXP * 24 * 60 *
    60 * 1000
86                  ),
87                };
88
89                res.status(201).cookie("authToken", token, options).
    json({
90                  successMessage: "registrazione avvenuta con
    successo",
91                  token,
92                });
93              } else {
```

```
94                res.status(500).json({
95                  error: {
96                    errorMessage: ["Internal Server Error"],
97                  },
98                });
99              }
100           });
101         }
102       } catch (error) {
103         res.status(500).json({
104           error: {
105             errorMessage: ["Internal Server Error"],
106           },
107         });
108       }
109     }
110   });
111 };
```

In questa pagina verranno effettuati una serie di controlli sui dati inseriti nella form. Una volta effettuati i controlli verrà effettuato un ulteriore controllo sulla mail, ovvero, se è già esistente nel database o no. Terminata questa fase preliminare verrà cambiato il nome dell'immagine del profilo e verrà utilizzato lo schema della pagina authModel.js per inserire i dati nella collection Users del database. Inoltre, verrà creato il token appartenente all'utente appena registrato.

Listato 4.10: authModel.js

```
1  const {model,Schema} = require('mongoose');
2
3  const registerSchema = new Schema({
4       userName : {
5            type : String,
6            required : true
7       },
8       email : {
9            type: String,
10           required : true
11      },
12      password : {
13           type: String,
14           required : true,
15           select : false
16      },
```

```
17      image : {
18           type: String,
19           required : true
20      }
21 },{timestamps : true});
22
23 module.exports = model('user',registerSchema);
```

Il model svolge la funzione di schema per caricare i dati in maniera corretta all'interno del database.

4.2.3 Home

Una volta che l'utente si è autenticato viene reindirizzato alla schermata principale, la quale rappresenta l'unica dell'app.

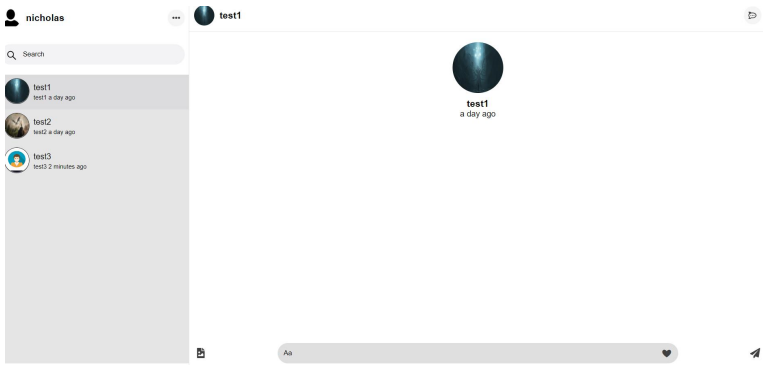

Figura 4.5: Homepage

Partendo da sinistra e dall'alto verso il basso si può vedere il profilo dell'utente, la barra di ricerca e gli altri utenti della chat. Se premiamo su un utente si aprirà al centro una chat con quest'ultimo e se abbiamo già chattato in precedenza verranno mostrati anche i messaggi vecchi.

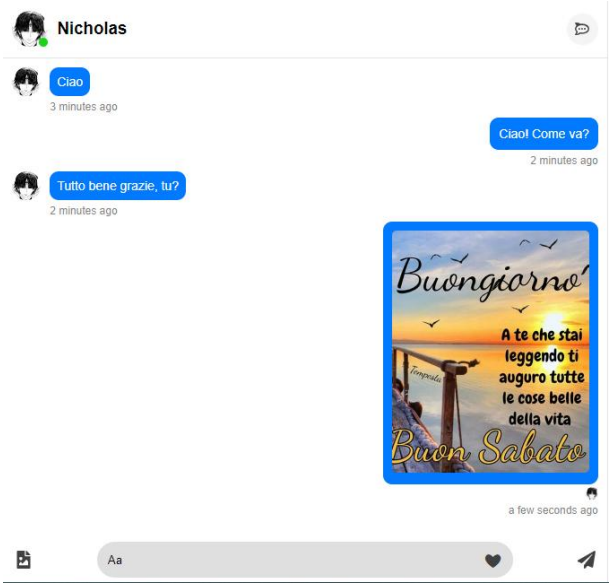

Figura 4.6: dettaglio chat con utente

All'interno della chat si possono scorrere i vecchi messaggi o scriverne di nuovi, inoltre, premendo il pulsante delle foto si possono inviare delle immagini e premendo il pulsante del cuore si possono inviare delle emoji. In alto a destra se viene premuto il pulsante della nuvoletta verrà aperta un'ulteriore schermata sulla destra che serve a mostrare i media scambiati tra i due utenti della chat.

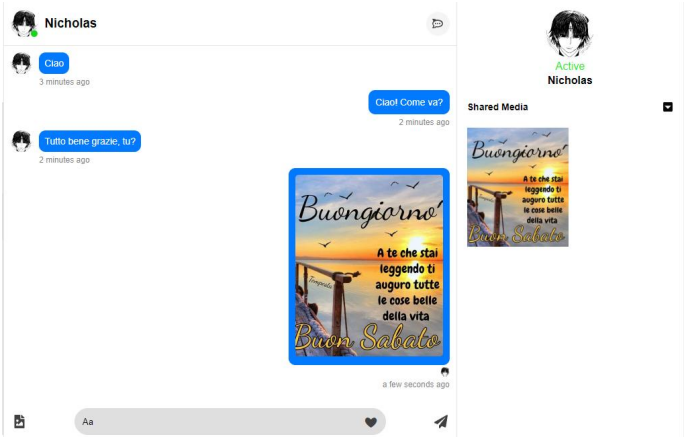

Figura 4.7: Media condivisi

La pagine principale è chiamata *Messenger.jsx* e quest'ultima funge da nucleo dell'applicazione in quanto tutte le funzionalità a disposizione dell'utente sono contenute in quest'unica pagina.

Listato 4.11: Messenger.jsx

```
...
return (
    <div className={themeMood === "dark" ? "messenger theme" : "
   messenger"}>
      <Toaster
        position={"top-right"}
        reverseOrder={false}
        toastOptions={{
          style: {
            fontSize: "18px",
          },
        }}
      />

      <div className="row">
        <div className="col-3">
          <div className="left-side">
            <div className="top">
              <div className="image-name">
                <div className="image">
                  <img src={`./image/${myInfo.image}`} alt="" />
                </div>
                <div className="name">
                  <h3>{myInfo.userName} </h3>
                </div>
              </div>

              <div className="icons">
                <div onClick={() => setHide(!hide)} className="
   icon">
                  <FaEllipsisH />
                </div>

                <div className={hide ? "theme_logout" : "
   theme_logout show"}>
                  <h3>Dark Mode </h3>
                  <div className="on">
                    <label htmlFor="dark">ON</label>
                    <input
                      onChange={(e) => dispatch(themeSet(e.target.
   value))}
                      type="radio"
```

```
                        value="dark"
                        name="theme"
                        id="dark"
                      />
                    </div>

                    <div className="of">
                      <label htmlFor="white">OFF</label>
                      <input
                        onChange={(e) => dispatch(themeSet(e.target.
    value))}
                        type="radio"
                        value="white"
                        name="theme"
                        id="white"
                      />
                    </div>

                    <div onClick={logout} className="logout">
                      <FaSignOutAlt /> Logout
                    </div>
                  </div>
                </div>
              </div>

              <div className="friend-search">
                <div className="search">
                  <button>
                    {" "}
                    <FaSistrix />{" "}
                  </button>
                  <input
                    onChange={search}
                    type="text"
                    placeholder="Search"
                    className="form-control"
                  />
                </div>
              </div>

              <div className="friends">
                {friends && friends.length > 0
                  ? friends.map((fd) => (
```

```
                    <div
                      onClick={() => setCurrentFriend(fd.fndInfo)}
                      className={
                        currentfriend._id === fd.fndInfo._id
                          ? "hover-friend active"
                          : "hover-friend"
                      }
                    >
                      <Friends
                        activeUser={activeUser}
                        myId={myInfo.id}
                        friend={fd}
                      />
                    </div>
                  ))
                : "No Friend"}
            </div>
          </div>
        </div>

        {currentfriend ? (
          <RightSide
            currentfriend={currentfriend}
            inputHendle={inputHendle}
            newMessage={newMessage}
            sendMessage={sendMessage}
            message={message}
            scrollRef={scrollRef}
            emojiSend={emojiSend}
            ImageSend={ImageSend}
            activeUser={activeUser}
            typingMessage={typingMessage}
          />
        ) : (
          "Per favore seleziona il tuo amico"
        )}
        </div>
        </div>   ); };
export default Messenger;
```

Lista utenti

Per ottenere la lista utenti avviene sempre un collegamento tra la parte frontend, ovvero, *Messenger.jsx* e la parte backend.

Listato 4.12: Messenger.jsx

```
...
const dispatch = useDispatch();
  useEffect(() => {
    dispatch(getFriends());
    dispatch({ type: "NEW_USER_ADD_CLEAR" });
  }, [new_user_add]);
 ...
```

Utilizzando il metodo **Dispatch** passiamo la richiesta a *messengerAction.js*

Listato 4.13: messengerAction.js

```
...
export const getFriends = () => async (dispatch) => {
  try {
    const response = await axios.get("/api/messenger/get-friends")
   ;
    dispatch({
      type: FRIEND_GET_SUCCESS,
      payload: {
        friends: response.data.friends,
      },
    });
  }
  catch (error) {

    console.log(error.response.data);
  }
};
...
```

Dal listato **4.13** vediamo come la funzione si colleghi al backend attraverso l'uso di una rotta che si trova nella pagina *messengerRoute.js* .

Listato 4.14: messengerRoute.js

```
...
router.get("/get-friends", authMiddleware, getFriends);
...
```

Questa è una rotta di tipo **GET** e chiama la funziona **getFriends** presente nella pagina *messengerController.js* .

Listato 4.15: messengerController.js

```
....
module.exports.getFriends = async (req, res) => {
  const myId = req.myId;
  let fnd_msg = [];
  try {
    const friendGet = await User.find({
      _id: {
        $ne: myId,
      },
    });
    for (let i = 0; i < friendGet.length; i++) {
      let lmsg = await getLastMessage(myId, friendGet[i].id);
      fnd_msg = [
        ...fnd_msg,
        {
          fndInfo: friendGet[i],
          msgInfo: lmsg,
        },
      ];
    }
....
```

La funzione **getFriends** prende tutti gli utenti nel database e li mostra nella parte sinistra dell'app. Per evitare che nella lista utenti venga mostrato anche l'utente loggato si è fatto uso di un middleware.

Listato 4.16: authMiddleware.js

```
....
module.exports.authMiddleware = async(req,res,next) => {
     const {authToken} = req.cookies;   //preso dai cookies
     if(authToken){
          const deCodeToken = await jwt.verify(authToken,process.
   env.SECRET);
          req.myId = deCodeToken.id;
          next();
     }else{   res.status(400).json({
               error:{ errorMessage: ['Please Loing First']
               }  })  } }
...
```

Questa funzione prende il token dell'utente loggato dai cookies dell'applicazione e in seguito lo passa a *messengercontroller.js*. In questo modo toglieremo dalla lista utenti, l'utente con il nostro stesso authtoken (ovvero noi stessi).

Listato 4.17: messengerController.js

```
...
const friendGet = await User.find({
      _id: {
        $ne: myId,  //con questo escludiamo il nostro id
      },
    });
...
```

Con questa funzione vengono presi gli id degli utenti tranne quello dello user che si è autenticato.

Ricerca Utenti

Subito sopra alla lista utenti è presente una barra di ricerca utilizzata per cercare il nome di un utente in particolare.

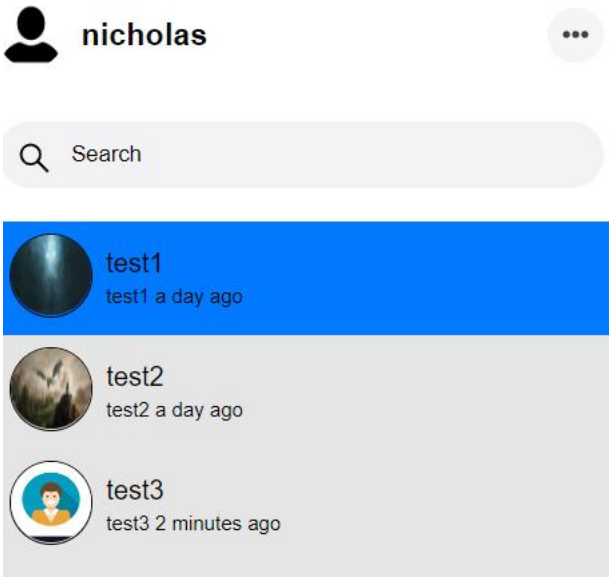

Figura 4.8: Barra di ricerca

I risultati della barra di ricerca cambiano ogni volta che si modifica il testo che si sta scrivendo. Questa funzione è stata implementata nella pagina principale *Messenger.jsx*.

Listato 4.18: Messenger.jsx

```
.....
const search = (e) => {
    const getFriendClass = document.getElementsByClassName("hover-
   friend");
    const frienNameClass = document.getElementsByClassName("
   Fd_name");
    for (var i = 0; i < getFriendClass.length, i < frienNameClass.
   length; i++) {
      let text = frienNameClass[i].innerText.toLowerCase();
      if (text.indexOf(e.target.value.toLowerCase()) > -1) {
        getFriendClass[i].style.display = "";
      } else {
        getFriendClass[i].style.display = "none";
      }
    }
  };
 ....
```

Attraverso questa breve funzione è possibile cercare gli utenti tramite il loro nome rendendo la ricerca molto più semplice e veloce.

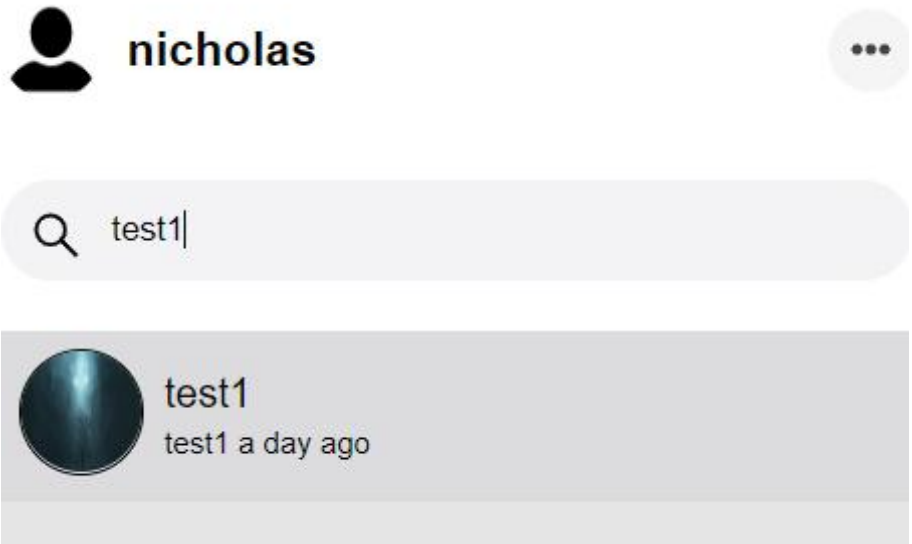

Figura 4.9: Barra di ricerca in funziona

Modalità Notturna

Un'altra funzionalità che l'app possiede è quella di cambiare tema, infatti, l'utente può scegliere se tenere il tema chiaro o cambiarlo in scuro. Per farlo basta premere sul pulsante dei tre puntini verticali.

Figura 4.10: Icona dei 3 puntini verticali

Premendo il pulsante comparirà un rettangolo di colore blu che contiene al suo interno i pulsanti per attivare o disattivare la "dark mode" e il pulsante per effettuare il logout.

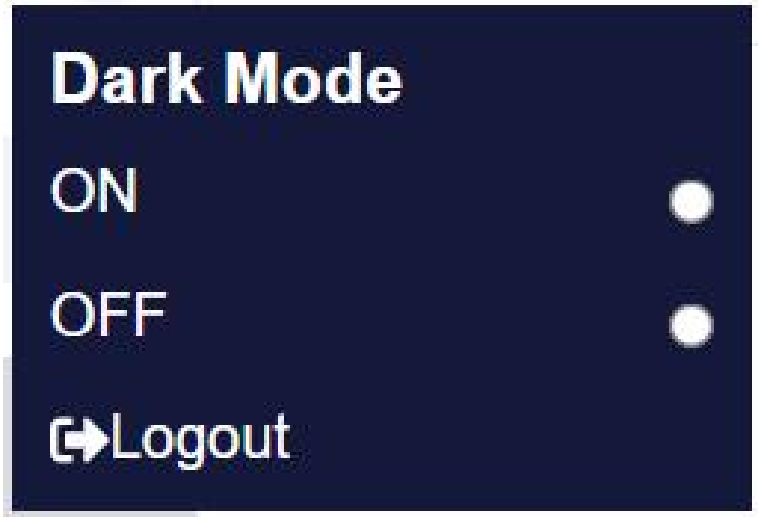

Figura 4.11: Pulsanti per Dark mode

All'interno della pagina *messenger.jsx* è presente una funzione che si attiva nel momento in cui premiamo il pulsante *on* o *off* per il tema scuro.

Listato 4.19: Messenger.jsx

```
...
useEffect(() => {
    dispatch(getTheme());
  }, []);
...
<h3>Dark Mode </h3>
```

```
                    <div className="on">
                      <label htmlFor="dark">ON</label>
                      <input
                        onChange={(e) => dispatch(themeSet(e.target.
    value))}
                        type="radio"
                        value="dark"
                        name="theme"
                        id="dark"
                      />
                    </div>

                    <div className="of">
                      <label htmlFor="white">OFF</label>
                      <input
                        onChange={(e) => dispatch(themeSet(e.target.
    value))}
                        type="radio"
                        value="white"
                        name="theme"
                        id="white"
                      />
                    </div>
...
```

Questa funzione passa la scelta "white" o "dark" alla pagina *messengerAction.js* all'interno del quale sono presenti due funzioni: **getTheme** e **themeSet**.

Listato 4.20: messengerAction.js

```
export const getTheme = () => async (dispatch) => {
  const theme = localStorage.getItem("theme");
  dispatch({
    type: "THEME_GET_SUCCESS",
    payload: {
      theme: theme ? theme : "white",
    },  }); };
export const themeSet = (theme) => async (dispatch) => {
  localStorage.setItem("theme", theme);
  dispatch({
    type: "THEME_SET_SUCCESS",
    payload: {
      theme: theme,
    }, }); };
```

la funzione **getTheme** utilizza il local storage per prendere la scelta fatta dall'utente, ovvero, quale pulsante ha premuto tra **darkmod on** o **darkmode off**. In seguito la scelta viene passata al metodo **themeSet** che va a settare il tema della pagina in base alla scelta.

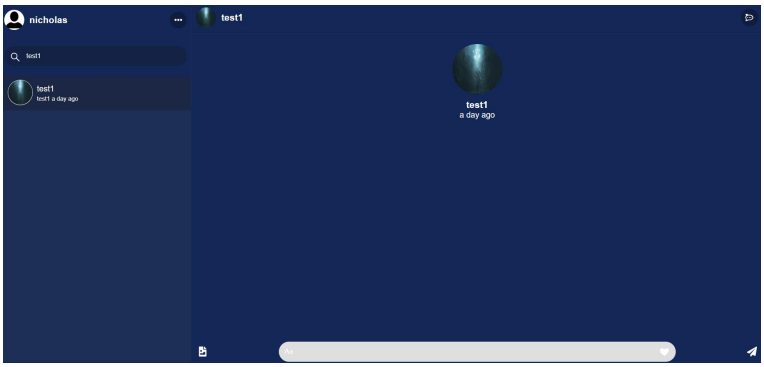

Figura 4.12: Home con il tema scuro attivo

4.2.4 Logout

Una volta terminate tutte le operazioni l'utente può effettuare il logout. Per effettuare il logout occorre, come detto in precedenza, premere l'icona dei tre puntini verticali e premere il pulsante di logout.

Figura 4.13: Pulsante di logout

Dal punto di vista del codice tutto inizia sempre dalla parte frontend ed è molto simile al login. La pagina del frontend è sempre *Messenger.jsx*.

Listato 4.21: Messenger.jsx

```
...
const logout = () => {
    dispatch(userLogout());
    socket.current.emit("logout", myInfo.id);
  };
...
```

Dal listato **4.21** si può notare come venga richiamato il metodo **userLogout** presente in *authAction.js*.

Listato 4.22: authAction.js

```
...
export const userLogout = () => async(dispatch) => {
     try{
         const response = await axios.post('/api/messenger/user-
   logout');
         if(response.data.success){
             localStorage.removeItem('authToken');
             dispatch({
                 type : 'LOGOUT_SUCCESS'
             })
         }

     }catch (error) {

     }
}
...
```

Si può osservare come la funzione va a chiamare la rotta **userLogout** presente in *authRoute.js*.

Listato 4.23: authroute.js

```
...
router.post('/user-logout',authMiddleware,userLogout);
...
```

La rotta, infine, chiama il metodo **userLogout** presente nella pagina *authcontroller.js*.

Listato 4.24: authController.js

```
...
module.exports.userLogout = (req, res) => {
  res.status(200).cookie("authToken", "").json({
    success: true,
  });
};
...
```

Questo metodo molto semplicemente svuota il campo dell'authtoken dell'utente corrente andando così ad effettuare il logout dall'applicazione.

Capitolo 5

Conclusioni

Grazie allo studio teorico effettuato nei primi capitoli è stato possibile apprendere e mostrare le diverse tecnologie all'avanguardia per lo sviluppo di applicazioni e le loro differenze. Questo è stato importantissimo per la realizzazione dell'applicazione, infatti, è stato possibile progettarla e svilupparla nel tempo previsto, nonostante richiedesse una serie di tecnologie e linguaggi non conosciuti. Tutto questo grazie a un duro lavoro di apprendimento che ha condotto agli obiettivi prefissati accomodando tutte le richieste avanzate dall'azienda. Inoltre, il lavoro presentato offre la possibilità di imparare ad implementare una web app full-stack con una serie di tecnologie differenti combinate tra loro.
Terminato il progetto di stage è stato possibile analizzare il lavoro svolto ed esaminare il rapporto con i tre anni accademici precedenti a quest'ultima esperienza.
L'università ha il compito di formare gli studenti che, terminato il percorso accademico, devono entrare nel mondo del lavoro con un buon livello di conoscenza ed esperienza sul campo. Compito diventato sempre più difficile negli anni a causa della continua evoluzione del settore e una grande competitività tra le aziende.
Il progetto è stato sviluppato a partire da conoscenze acquisite attraverso esami sostenuti nel triennio come ad esempio *Programmazione ad Oggetti, Sistemi informativi e Basi di dati e Tecnologie Web*; ma ciò non è bastato, infatti, c'è stato un periodo iniziale di ricerca e studio per quanto riguarda lo sviluppo di un'applicazione full-stack e il corretto utilizzo delle varie tecnologie richieste per sviluppare la stessa.
La difficoltà maggiore è stata proprio la mancanza di conoscenza iniziale sull'argomento la quale ha comportato il rischio di non rispettare i tempi di sviluppo richiesti o, peggio ancora, di non essere in grado di portare a termine il compito.
Nonostante tutto, le nozioni fornite dall'università sono state di vitale importanza per capire in modo efficacie ed efficiente i nuovi argomenti in quanto i concetti alla base dell'informatica risultano essere i medesimi.
Oltre agli aspetti tecnici questo progetto ha fornito la possibilità di integrarsi in un ambiente lavorativo vero acquisendo un'enorme esperienza. Grazie allo stage è stato

possibile comprendere le dinamiche aziendali e professionali relative al lavoro svolto. Infatti è stato possibile dialogare con i colleghi che si sono resi disponibili a condividere le proprie conoscenze ed esperienze. Inoltre, non sono mancati momenti in cui si ha avuto l'opportunità di mostrare le proprie competenze personali favorendo lo scambio di conoscenze e una discussione costruttiva.
Oltre al risultato didattico ottenuto occorre anche considerare gli obiettivi raggiunti e le competenze acquisite.

L'obiettivo richiesto dal tutor aziendale era quello di supportare l'azienda sviluppando interamente un'applicazione full-stack. La progettazione e l'implementazione dell'app rispettano i parametri e le funzionalità richieste in fase di analisi soddisfacendo in modo adeguato le aspettative dell'azienda. L'applicazione è completamente funzionante, tuttavia occorre che venga collegata in modo corretto ai server dell'azienda.
Dal punto di vista delle competenze acquisite, invece, si è riscontrata un'elevata difficoltà in fase iniziale, come già descritto nelle frasi precedenti, in quanto c'è stata una forte necessità di studiare e approfondire i nuovi argomenti alla base delle tecnologie da impiegare per lo sviluppo dell'applicazione. Inoltre, l'applicazione è stata costruita in intera autonomia senza alcun utilizzo di esempi o "frame" offerti dall'azienda ospitante. Spesso in ambito universitario ci si concentrava sulla realizzazione di un singolo aspetto di un'applicazione, in questo caso però, sono stati sviluppati sia il server (con il database) che il client. Anche la conoscenza dal punto di vista dei linguaggi di programmazione è stata ampliata attraverso un uso approfondito di Javascript.
Il risultato finale rispetta l'idea nata in fase di analisi e ha lasciato soddisfatta l'azienda. Questo non significa che l'app non possa essere migliorata in alcun modo. Lo sviluppo di base è completo ma ci sono ancora diverse implementazioni che possono essere fatte. In particolare, si potrebbe aggiungere la funzionalità delle chiamate vocali o videochiamate tra gli utenti, dando loro la possibilità di discutere in modo più accurato di temi che risultano complessi da esprimere tramite messaggio. La funzione delle chiamate potrebbe essere implementata sfruttando le API di Zoom ma resta il problema che per effettuare una riunione occorre comunque essere in possesso dell'applicazione e, ogni volta che si partecipa o si avvia una chiamata, ci si sposta su Zoom, risultando un po' scomodo.
Un altro spunto di riflessione riguarda l'implementazione delle chat di gruppo; quest'ultima potrebbe essere una funzione molto utile in quanto gli utenti avrebbero la possibilità di discutere su uno stesso argomento senza dover parlare privatamente e perdere tempo a scrivere a più persone gli stessi messaggi. Infine, occorre anche collegare l'applicazione al database aziendale in quanto per la realizzazione è stato utilizzato un database personale ovvero MongoDB.
Alla luce delle considerazioni effettuate l'applicazione risulta essere funzionante e rispetta i requisiti aziendali, tuttavia, maggiori competenze e conoscenze potrebbero rendere l'app molto più efficiente e competitiva per essere fruita al meglio dai clienti.

Sitografia

[1] Wikipedia. *Instant Messaging wiki*. URL: https://en.wikipedia.org/wiki/Instant_messaging.

[2] Microsoft. *Blazor*. URL: https://docs.microsoft.com/it-it/aspnet/core/blazor/hosting-models?view=aspnetcore-3.1.

[3] educative.it. *What is Mern stack*. URL: https://www.educative.io/edpresso/what-is-mern-stack.

[4] <geekandjob/>. *ReactJS*. URL: https://www.geekandjob.com/wiki/react.

[5] html.it. *oReact*. URL: https://www.html.it/pag/55052/react-introduzione/.

[6] html.it. *Node.js*. URL: https://www.html.it/pag/32814/introduzione-a-node-js/.

[7] <geekandjob/>. *Express.js*. URL: https://www.geekandjob.com/wiki/express.

[8] Wikipedia. *Laravel*. URL: https://it.wikipedia.org/wiki/Laravel.

[9] Wikipedia. *Ruby on Rails*. URL: https://it.wikipedia.org/wiki/Ruby_on_Rails.

[10] Wikipedia. *Django*. URL: https://it.wikipedia.org/wiki/Django_(informatica).

[11] Wikipedia. *CodeIgniter*. URL: https://it.wikipedia.org/wiki/CodeIgniter.

[12] Wikipedia. *jQuery*. URL: https://it.wikipedia.org/wiki/JQuery.

[13] Wikipedia. *Bootstrap*. URL: https://it.wikipedia.org/wiki/Bootstrap_(informatica).

[14] Wikipedia. *Software Development Kit*. URL: https://it.wikipedia.org/w/index.php?title=Software_development_kit&oldid=108041678.

[15] Apppartner. *Native apps vs Web apps*. URL: https://www%20.apppartner.com/native-apps-vs-web-apps-strengths-and-weaknessesyou-need-to-know/.

[16] StatCounter. *Mobile Operating System Market Share Worldwide*. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[17] The App Solutions Inc. *IOS VS Android*. URL: https://theappsolutions.com/blog/development/iosvs-android/#contents_0.

[18] Android. *Android Jetpack*. URL: https://developer.android.com/jetpack/.

[19] tutorialspoint. *Android Architecture*. URL: https://www.tutorialspoint.com/android/android_architecture.html.

[20] Chris Griffith. *What is Hybrid App Development?* URL: https://ionicframework.com/resources/articles/what-is-hybrid-appdevelopment.

[21] Microsoft. *What is Xamarin?* URL: https://docs.microsoft.com/it-it/xamarin/get-started/what-is-xamarin.

[22] Facebook. *React Native*. URL: https://reactnative.dev/.

[23] Flutter. *Technical overview*. URL: https://flutter.dev/docs/%20resources/technical-overview.

[24] Ionic. *Ionic Framework 5*. URL: https://ionicframework.com/.

[25] Nick Hyatt. *Capacitor*. URL: https://ionicframework.com/blog/introducing-capacitor-supportfor-ionic-appflow/.

[26] Wikipedia. *MongoDB wikipedia*. URL: https://it.wikipedia.org/wiki/MongoDB.

[27] Kinsta. *MongoDB vs MySQL*. URL: https://kinsta.com/it/blog/mongodb-vs-mysql/.