ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea in Ingegneria Elettronica, Informatica e Telecomunicazioni

SVILUPPO DI UN MODULO DI MESSAGGISTICA REAL-TIME A SUPPORTO DI SISTEMI DI GESTIONE DELL'EMERGENZA

Elaborato in
FONDAMENTI DI INFORMATICA A

Tesi di Laurea di
RICCARDO DELVECCHIO

Relatore
Prof. MIRKO VIROLI

Correlatori
Ing. SIMONE GROTTI

Anno Accademico 2015 – 2016
III[a] Sessione di Laurea

PAROLE CHIAVE

Real-Time Communication

Web Service REST

Web Application

JavaScript

HTML5

Ai miei genitori, ai miei amici e compagni di studio,
a tutti coloro che mi hanno accompagnato durante
questo lungo percorso.

Indice

Introduzione

Durante il corso della storia l'aspettativa di vita dell'uomo si è alzata considerevolmente, specialmente negli ultimi due secoli, a partire dalla rivoluzione industriale, la quale ha dato inizio ad un processo di produzione di tecnologie sempre nuove e al loro graduale inserimento nella vita quotidiana, non limitando la loro applicazione al solo ambito scientifico.
Il notevole incremento tecnologico ha portato a innumerevoli sviluppi in tutti i settori, ma uno in particolare, quello sanitario, ha beneficiato del progresso, frutto di una costante e oculata ricerca, che ha come oggetto di studio tre campi in particolare: prevenzione, cura e soccorso. Sono infatti degni di nota i traguardi raggiunti nell'evitare l'incorrere di problemi di salute, ad esempio ottenuti da studi sul genoma umano, sull'anatomia o tante altre branche della medicina; inoltre è altrettanto valido lo sviluppo delle strutture ospedaliere, le quali dispongono di macchinari all'avanguardia e in continua evoluzione per l'individuazione di malattie e la loro cura; infine anche per quanto riguarda il soccorso, c'è da considerare l'impatto che hanno avuto strumenti, mezzi e sistemi innovativi, che hanno consentito un abbondante miglioramento per ciò che concerne la tempistica e la qualità del soccorso.

È in questo contesto che si inserisce il caso di studio oggetto di questo elaborato, che consiste nella realizzazione di un modulo di messaggistica real-time, inserito in un sistema per la gestione delle emergenze. Esso deve offrire agli utenti la possibilità di scambiare messaggi in tempistiche brevi, a prescindere dal contesto in cui essi si trovino, in modo tale da offrire la possibilità di far viaggiare informazioni di una certa rilevanza in rete, mantenendo sempre aggiornati i dati. Questo modulo è stato pensato per riferire quel genere di dati che normalmente impiegherebbero troppo tempo per giungere a destinazione, ad esempio un tipico scenario può essere l'insorgere di un imprevisto, con la conseguente necessità di comunicare tempestivamente un cambio di programma, garantendo in questo modo il minimo spreco di risorse e tempo.

Capitolo 1

Background

Le cause di possibili emergenze sono tantissime al giorno d'oggi, si va da eventi naturali, ad esempio terremoti, inondazioni e eruzioni, a eventi ad opera dell'uomo, come disastri ambientali, incidenti sul lavoro, guerre e esplosioni nucleari, per questo la gestione di emergenze è una tematica che viene presa seriamente in considerazione.

L'obiettivo è quello di utilizzare la tecnologia per rendere il più performante possibile il lavoro degli operatori che gestiscono queste emergenze; che siano essi medici, infermieri, operatori sul campo e non, il fine è di salvare quante più vite possibile, in questo contesto i moderni sistemi a disposizione dei team di soccorso hanno senza dubbio perfezionato il loro compito. Questo è il motivo per cui la costante ricerca di sempre nuove tecnologie da applicare in ambito sanitario è di grande rilievo all'interno del contesto scientifico odierno.

1.1 Descrizione generale del sistema

Il sistema in cui verrà inserito il modulo sviluppato in questo elaborato consiste in componenti hardware e software che dovranno essere impiegati in situazioni di incidente maggiore, supportando il coordinamento delle operazioni da parte della catena di controllo dei sanitari, la quale ha il compito di regolare le manovre di soccorso e gli interventi del personale medico sui pazienti. Il sistema è composto da tre sottoinsiemi principali:

- **Back-end**, parte del sistema non visibile all'utente, che incorpora il server sia per i device mobili, che per i diversi tipi di utenti che interagiscono con la control-room. Questo componente si occupa di raccogliere e memorizzare i dati raccolti, gestire le connessioni e autenticazioni da parte

degli utenti, elaborare i dati reperiti sul campo, diversa a seconda dei privilegi legati al ruolo dell'utente, infine nel back-end sono implementate le politiche di sicurezza e controllo dell'accesso al sistema.

- **Applicazione Android**, parte destinata agli operatori sul campo, che raccoglieranno, salveranno e invieranno valutazioni della salute delle persone soccorse durante l'emergenza.

- **Applicazione web**, parte che, essendo responsive, può essere acceduta dai browser dei vari dispositivi, assumendo la funzionalità di una control-room per l'interazione con gli altri utenti e la visualizzazione dei dati raccolti durante l'emergenza, i quali sono diversi a seconda dei permessi dell'utente autenticato, è inoltre progettata per visualizzare i dati in maniera sintetica e riassuntiva, in base alla figura operatore che li sta guardando.

Dal punto di vista logico l'architettura del sistema è suddivisa in tre differenti macro parti, ognuna delle quali è suddivisa in diversi layer, come mostrato in 1.1: ogni livello logico rappresentato si appoggia su quelli sottostanti per l'utilizzo delle relative funzionalità.

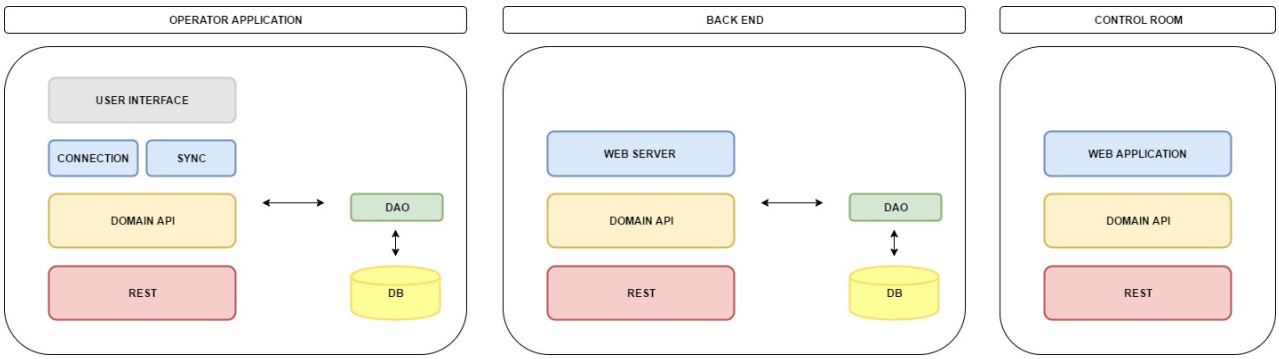

Figura 1.1: Architettura ad alto livello di astrazione del sistema.

1.1.1 Obiettivo

L'obiettivo del sistema in questione è di gestire l'emergenza nella maniera più opportuna, cercando di portare in salvo il maggior numero possibile di persone coinvolte, il contributo che apporta è supportare e organizzare i team di soccorso e le azioni sul campo, gestire in maniera più efficiente possibile i mezzi e e le strutture ospedaliere, oltre a ottimizzare l'operato del personale sanitario, rendendo più agevole la prima valutazione dello stato di salute.

Il sistema persegue l'obiettivo di incrementare l'efficienza nella gestione dell'evento di emergenza e dell'utilizzo delle risorse a disposizione migliorando in maniera considerevole la comunicazione, facendo viaggiare i dati in rete in maniera rapida, consistente e precisa, permettendo alla catena di comando di gestire l'intervento, coordinando al meglio tutte le risorse coinvolte.

1.1.2 Scenario di emergenza e operatori

Sfruttando l'esperienza ricavata dalla gestione operativa di varie maxi emergenze in ambiti differenti, è stato possibile, dopo un'accurata analisi, individuare caratteristiche comuni a tutti gli eventi di questo tipo: tra queste si possono individuare le differenti aree in cui la scena dell'emergenza viene suddivisa e le differenti figure di riferimento per il personale addetto alla gestione di queste aree, che insieme formano la catena di comando della gestione della maxi emergenza.

Lo scenario operativo può essere così suddiviso in diverse aree di lavoro:

- una zona adibita alla gestione dell'incidente maggiore, nella quale opera chi coordina gli operatori sul campo e dalla quale partono le richieste e le disposizioni.
- un'altra designata alla funzione di parcheggio per i mezzi di trasporto, coordinati da uno specifico operatore che ne gestisce la mobilità.
- un altro luogo è attribuito ad accogliere i pazienti che devono essere stabilizzati prima di essere trasportati verso un ospedale, gestito anch'esso da uno specifico operatore.
- i cantieri reali del disastro, in cui un componente in particolare del personale dirige le squadre di soccorso.
- l'insieme delle strutture ospedaliere, destinazione finale dei pazienti.

Un tipico scenario di maxi emergenza si articola dunque in differenti fasi che coinvolgono i concetti appena esposti:

- all'attivazione della situazione di emergenza le squadre vengono inviate sul campo d'azione per svolgere le attività di soccorso e gestione dell'evento;
- quando arrivano sul posto le squadre vengono coordinate dall'operatore addetto alla direzione dei soccorsi nella valutazione dei feriti e dell'entità dell'evento: in particolare le squadre svolgono una valutazione sanitaria dei feriti tramite protocolli di triage cartacei 1.2, riportando i risultati

alla figura preposta alla gestione del campo, che deve far giungere queste informazioni alle figure di controllo di tutta l'emergenza;

- tipicamente vi è quindi la necessità di gestire diversi trasferimenti di mezzi e operatori, in modo da coordinare al meglio i soccorsi e gestire le risorse a disposizione: ad esempio l'arrivo di un mezzo con personale di soccorso sul campo d'azione, oppure il trasferimento di feriti verso strutture ospedaliere temporanee e non.

In particolare quindi tra le varie figure preposte alla gestione delle emergenze spiccano quella relativa alla gestione del campo d'azione, quella che ha in carico la gestione dei mezzi, quella preposta alla gestione della struttura ospedaliera temporanea di soccorso per i feriti ed in particolare il coordinatore di tutta l'emergenza che deve avere in ogni momento una visione il più completa possibile dell'evento, a cui tutti fanno riferimento.

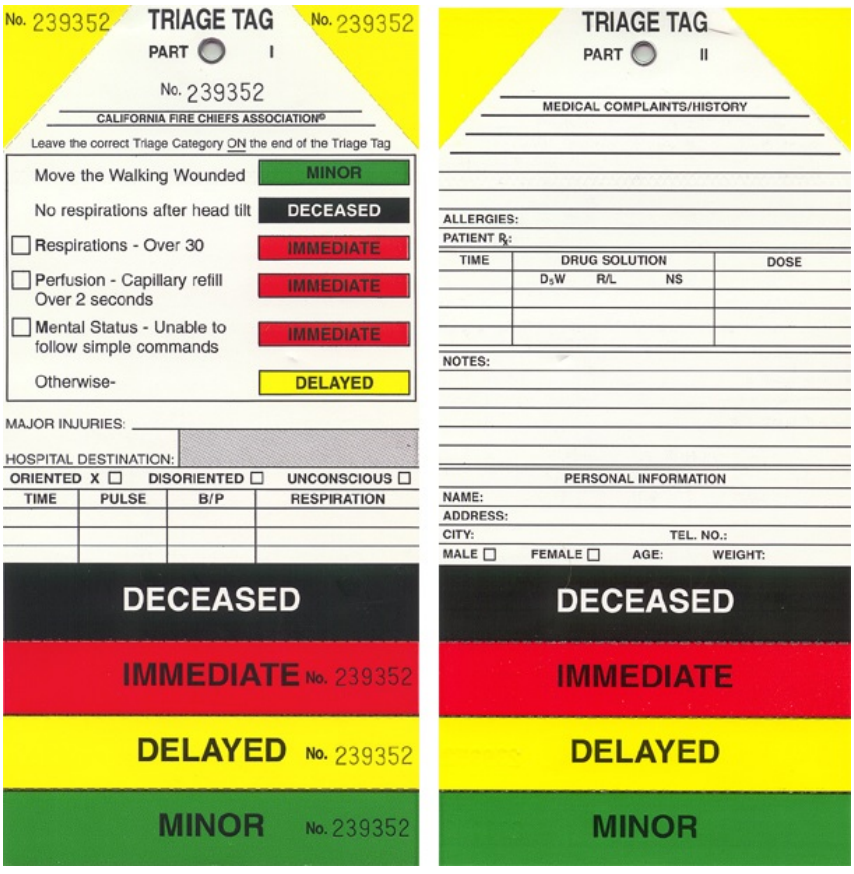

Figura 1.2: Un esempio di triage cartaceo.

1.2 Tecnologie utilizzate nello sviluppo del sistema

1.2.1 AngularJS

Figura 1.3: Logo di AngularJS.

"Angular è quello che HTML avrebbe dovuto essere se fosse stato progettato per sviluppare applicazioni."[1]

AngularJS[1] è un framework per lo sviluppo di applicazioni web dinamiche lato client, offrendo una duplice funzionalità: da un lato la possibilità di definire un'interfaccia grafica, che sfrutta l'approccio dichiarativo del di HTML, estendendone la sintassi, dall'altro offre gli strumenti per implementare la struttura e la logica applicativa di un'applicazione web.
AngularJS segue il pattern di presentazione MVC (*Model View Controller*), seguendo quindi la politica di separazione dei componenti in base ai compiti nel modo seguente: il Model si occupa della gestione dei dati, la View è designata alla visualizzazione dei dati e dell'interazione con l'utente, infine il Controller, che è il responsabile dell'interazione fra Model e View, riceve comandi in input dalla View e va a modificare il modello di conseguenza.
Un aspetto fondamentale di AngularJS consiste del data binding bidirezionale (Two-way data binding). Il data binding monodirezionale consiste nel combinazione dei dati del Model con un template HTML per produrre una View, ma una volta che questa è stata prodotta, da questo momento in poi eventuali modifiche nella View non avranno riscontro nel Model e viceversa, l'unico

[1] https://docs.angularjs.org/guide/introduction

modo è scrivere del codice per farlo. Nel data binding bidirezionale (Figura 1.4), al contrario, si ha un legame fra View e Model: il template HTML viene compilato dal browser producendo una View, modificando quest'ultima, i cambiamenti si rispecchiano automaticamente nel Model e viceversa, per cui non è necessario scrivere del codice per gestire queste modifiche, semplificando notevolmente la programmazione per lo sviluppatore, si potrebbe pensare alla View come un proiezione istantanea dei dati del Model.

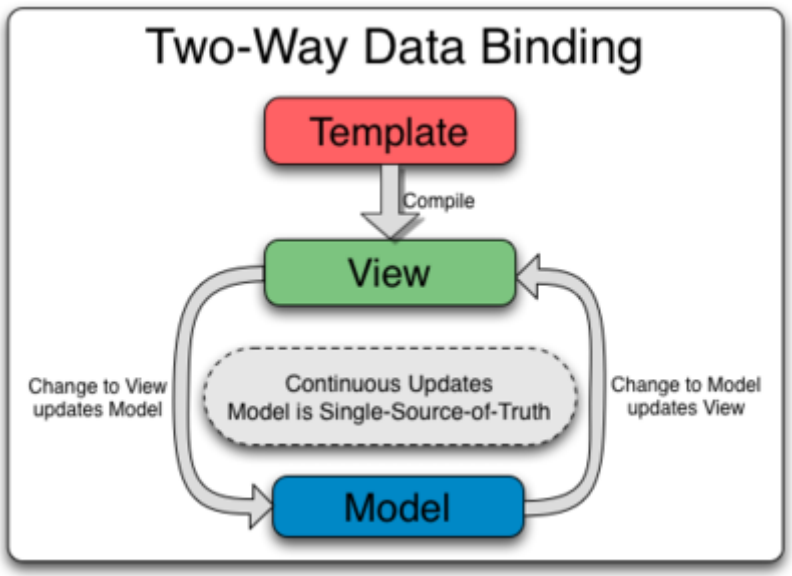

Figura 1.4: Data binding bidirezionale.

Un altro punto da considerare nello sviluppo di applicazioni web con AngularJS è l'architettura SPA (*Single Page Application*), eseguendo la SPA in una sola pagina HTML, le cui risorse vengono caricate dinamicamente, offrendo alla singola pagina HTML più viste, a seconda dell'interazione con l'utente. Il vantaggio nell'utilizzo di questo metodo di programmazione consente di creare applicazioni web responsive, dando comunque all'utente l'illusione di caricare pagine diverse, in quanto ognuna delle View è mappata in un URL.

Infine è degno di nota il design pattern Dependency Injection, che consente di delegare a terzi, in questo caso al framework, il compito di individuare e procurare una risorsa di cui l'oggetto in questione ha bisogno, se un componente ha bisogno di funzionalità offerte da un altro componente non deve far altro che specificarlo come parametro nella propria definizione.

1.2.2 Bootstrap

Figura 1.5: Logo di Bootstrap.

Bootstrap[2] è considerato uno dei framework front-end più rinomati per quanto riguarda la progettazione di siti web responsive e mobile first, è considerato un tool in grado di gestire la fase di avvio di un progetto, creando componenti riusabili e personalizzabili, ai quali però possa essere un'impronta stilistica personalizzata. Per progettare un sito web è necessario tener conto delle diverse dimensioni degli schermi dei dispositivi, oltre che delle inconsistenti dimensioni delle finestre; per far fronte a questi problemi bisogna progettare il sito seguendo un design responsive.

> *"Il controllo che posseggono i designer su un mezzo come la stampa, lo stesso che vorrebbero avere anche sul web, è semplicemente una funzione della limitatezza della pagina stampata. Dovremmo accettare il fatto che il web non possiede le stesse restrizioni e progettare un design che tenga presente questa flessibilità"*[2]

Ciò significa che il contenuto deve essere ideato per essere adattabile alle dimensioni, senza bisogno di dover creare un sito ad hoc per diversi dispositivi, per poi fare un redirect a seconda di quello in questione. Per ottenere questo risultato si sfrutta un layout che si ridimensiona dinamicamente, in base alle caratteristiche del dispositivo, rendendo le librerie di Bootstrap multidispositivo e multipiattaforma.

Nel caso di Bootstrap questo layout responsive è una griglia, divisa in righe, ognuna delle quali organizzata al suo interno in 12 colonne; esistono inoltre 4 dimensioni di schermo diverse, organizzate in base alla loro larghezza in pixel. Quando si definisce un componente si può personalizzare singolarmente il suo comportamento, definendo il numero di colonne che andrà a occupare, nella dimensione desiderata, perciò esso si adatterà dinamicamente alle dimensioni dello schermo. Infine seguire una linea di sviluppo mobile-first significa partire la progettazione del sito destinato a dispositivi mobili, con il fine di definire

[2]John Allsopp, https://alistapart.com/article/responsive-web-design

una prima struttura con i contenuti essenziali, per poi via via arricchirla nei dispositivi più capaci, man mano che aumentano le dimensioni dello schermo.

1.2.3 Web Service REST

Un web service è un sistema progettato per permettere la creazione di sistemi distribuiti, permettendo interoperabilità di calcolatori diversi, situati in nodi diversi della rete. Questa proprietà è attuata definendo un'interfaccia che manifesti all'esterno i servizi disponibili, in modo che altri sistemi possano usufruirne e attivare le operazioni definite nell'interfaccia tramite richieste. Esistono due linee guida differenti per la creazione di web service: il primo è l'approccio architetturale REST[**?**], che definisce un insieme di risorse con le quali il client può interagire con richieste HTTP (GET, PUT, POST, DELETE)(Figura 1.6), mentre il secondo, l'approccio SOAP definisce un insieme di metodi richiamabili da remoto.

In questo elaborato si tratta solo il primo, in quanto è quello adottato per lo sviluppo del sistema. I principi di progettazione su cui si basa REST sono:

- Le funzionalità sono definite in risorse web.
- Ogni risorsa deve essere identificabile unicamente (URI).
- Un insieme di operazioni ben definite.
- Un protocollo client-server,stateless, a livelli.
- Il formato per la rappresentazione della risorsa, è chiamato media type e in genere si utilizzano linguaggi come XML o JSON.

Perciò, ricapitolando, l'elemento fondamentale su cui si basano i sistemi REST è la risorsa, per risorsa si intende un qualsiasi elemento del sistema che può essere oggetto di elaborazione. I principi di REST stabiliscono che ognuna di esse deve essere identificata univocamente, siccome le linee guida REST sono utilizzate per progettare sistemi distribuiti, la soluzione più semplice e immediata consiste nell'assegnare un URI a ogni risorsa, rendendola accessibile da remoto, tramite richieste HTTP. Il sistema reagisce a una chiamata eseguendo le operazioni assegnatele dallo sviluppatore in fase di progettazione.

Restlet

Restlet è un framework open source per lo sviluppo di web service in Java, tramite le API fornite è possibile realizzare applicazioni web che soddisfino i

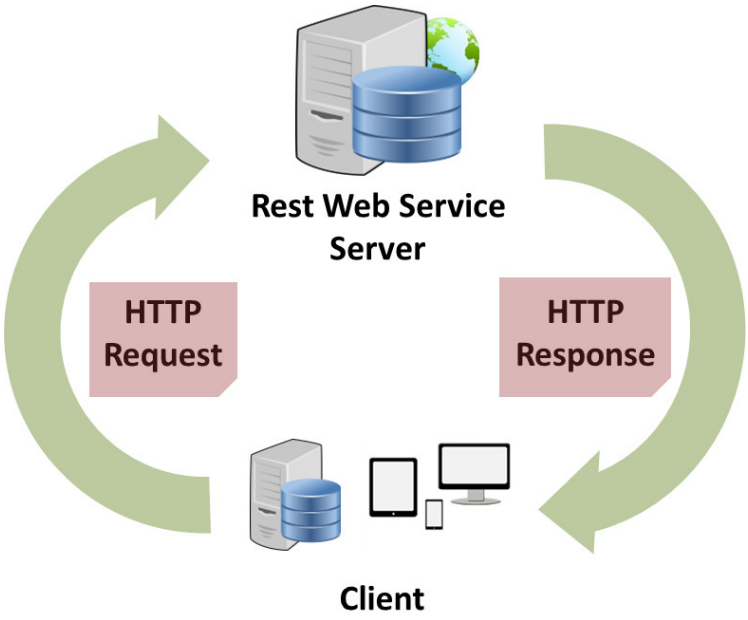

Figura 1.6: Modello di comunicazione REST.

requisiti REST di interscalarità e interoperabilità. Il framework è composto da due parti: le Restlet API e il Restlet engine; le prime facilitano l'implementazione di applicazioni, sia server side, che client side, mentre il secondo che supporta o implementa le Restlet API. Il vantaggio nell'utilizzo di Restlet è l'ampia gamma di funzionalità del modello REST che Restlet mette a disposizione per lo sviluppo in Java:

- risorse, ossia il concetto fondamentale di REST, per fornire accesso e la manipolazione dei dati da remoto;
- componenti, che sono contenitori logici per applicazioni Restlet;
- connettori, i quali abilitano le comunicazioni fra entità REST;
- rappresentazioni, per esporre lo stato delle risorse.

1.3 Modulo per la messaggistica

Allo stato attuale del sistema per la gestione di emergenze esistente, preso come caso di studio di riferimento, manca un modulo per la messaggistica real-time, il cui sviluppo come componente integrato è l'obiettivo di questo elaborato. Nonostante al sistema iniziale siano già stati integrati altri moduli[3],

Figura 1.7: Logo di Restlet.

non risulta ancora coperto per ciò che concerne l'invio di dati, al fine di consegnare informazioni i maniera rapida e intuitiva a uno specifico destinatario, scelto all'interno delle figure di gestione di emergenza. Il contributo apportato consiste nello sviluppo e nell'integrazione di un modulo che consenta di scambiarsi messaggi real-time, allo scopo migliorare ulteriormente il livello di comunicazione e coordinazione, assicurando quindi un minor speco di risorse e la tracciabilità di informazioni che altrimenti non sarebbero recuperabili.
Un tipico scenario di utilità del modulo in questione può essere l'insorgere di un imprevisto, con quindi un conseguente cambio di programma inaspettato. Nel sistema sviluppato finora potrebbe volerci del tempo prima che la modifica venga comunicata a tutto il personale interessato, in quanto potrebbe non disporre dei mezzi adatti a tal scopo, mentre con l'introduzione del nuovo modulo si potrebbe inviare tempestivamente un messaggio per avvisare e fare in modo che il resto degli operatori si organizzino di conseguenza, impedendo quindi che vadano sprecati mezzi, tempo e ogni altro genere di risorsa, garantendo inoltre che l'informazione trasmessa possa essere recuperata in qualsiasi momento perché se ne tiene traccia all'interno del sistema.

Capitolo 2

Analisi

I requisiti richiesti da un modulo di messaggistica real-time da utilizzare in questi contesti possono essere divisi in due tipi, i funzionali descrivono le funzionalità che si devono sviluppare, mentre quelli non funzionali descrivono i vincoli dei servizi offerti dal modulo e dallo stesso processo di sviluppo.

2.1 Requisiti funzionali

È richiesto un modulo che debba essere integrato in un sistema già esistente di gestione di emergenze, suddiviso in tre macro blocchi: un'applicazione web che permetta la corretta visualizzazione di dati consistenti, consentendo all'utente autenticato di interagire con tutti gli altri operatori e manipolare i dati in base ai permessi a lui riservati, una seconda parte di back end, che racchiude le funzionalità di web server, si occupa di gestire tutte le connessioni, autenticazioni e della persistenza dei dati in un database, mentre la terza parte comprende un'applicazione su su dispositivo mobile per eseguire tutte le operazioni di raccoglimento dati sul campo e invio di quest'ultimi al server.

L'obiettivo principale da raggiungere è l'integrazione completa con il sistema appena descritto di un modulo, che permetta agli utenti di scambiarsi messaggi real-time attraverso la rete, ossia di poter inviare informazioni in formato testuale che raggiungano tempestivamente il destinatario e vengano istantaneamente notificate all'utente. Risulta quindi necessaria la modifica dell'applicazione web, affinchè comprenda la parte dedita a contenere l'interfaccia per lo scambio di messaggi, oltre all'aggiunta, nel back end di un componente che garantisca il trasferimento dei dati attraverso la rete il più velocemente possibile. Il modulo deve prevedere tre diverse modalità di invio di messaggi, suddivise in base al destinatario: la prima prevede la consegna di un singolo messaggio a un singolo utente scelto fra quelli disponibili (recuperabili dal si-

stema), la seconda invia un messaggio broadcast a tutti gli utenti disponibili, mentre la terza invia un messaggio collettivo agli utenti filtrati in base al ruolo che ricoprono (anch'esso ottenibile dal sistema). È prevista inoltre la tracciabilità dei messaggi inviati, funzionalità di cui si deve occupare il back end, in modo tale che un utente offline, una volta fatta l'autenticazione e ottenuto l'accesso al sistema, abbia la possibilità di visualizzare tutti i messaggi a lui destinati, mentre non era connesso. Infine va sottolineato che l'applicazione web, nelle condizioni attuali, è predisposta per l'internazionalizzazione e la conseguente traduzione dei testi mostrati a video, per ora nelle lingue italiano e inglese; perciò è importante, per ottenere una piena integrazione, che anche il modulo di messaggistica segua questa linea di sviluppo. Le funzionalità richieste possono essere riassunte nel seguente diagramma dei casi d'uso:

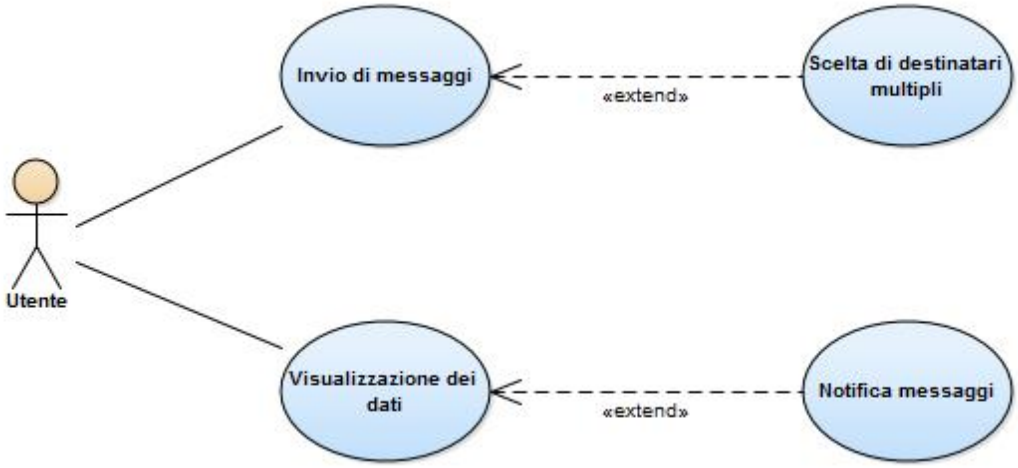

Figura 2.1: Casi d'uso che rappresentano le funzionalità richieste per il modulo di messaggistica real-time da implementare.

Come si evince dal diagramma l'utente del sistema può quindi usufruire della due funzionalità principali: invio di messaggi real-time verso un utente remoto e visualizzazione dei messaggi a lui destinati. Le due rimanenti invece estendono le precedenti, ossia utilizzano la visualizzazione e l'invio dei messaggi per arricchire ulteriormente il set di funzionalità a disposizione dell'utente. In altri termini, la notifica dei messaggi sfrutta la visualizzazione, per rendere noto all'utente che c'è un messaggio in arrivo, mentre la scelta di un destinatario multiplo utilizza l'invio di messaggio per mandarne uno a ogni a gruppi di utenti.

2.2 Requisiti non funzionali

Nello sviluppo del modulo bisogna tener conto anche di altri fattori che non riguardano funzionalità aggiuntive del sistema, ma bensì proprietà che dovranno essere soddisfatte, prendendo in considerazione il contesto di utilizzo e il tipo di utenza che ne farà uso. Oltre a dover migliorare la comunicazione e coordinazione del sistema, le qualità di cui deve essere dotato il modulo sono le seguenti:

- **Correttezza**: è definito corretto se, una volta ultimato, rispetta le specifiche che erano state decise in fase di progettazione.
- **Affidabilità**: è affidabile se si può dipendere da esso, concetto chiave in questo ambito, siccome il fine è di salvare vite umane.
- **Robustezza**: può essere considerato robusto se a fronte di un imprevisto non considerato in fase di progetto, continua a comportarsi in maniera ponderata.
- **Facilità d'uso**: affinchè sia di facile utilizzo è necessario che l'interfaccia utente sia semplice da comprendere e da utilizzare, in modo che egli possa interagire con essa in maniera naturale.
- **Portabilità**: per avere una piena integrazione con il sistema esistente è necessario che il modulo sia multipiattaforma e multi-dispositivo.

Infine è necessario fare una considerazione sul fatto che il sistema è stato creato per essere utilizzato da personale non informatico, perciò è fondamentale, oltre alla facilità d'uso, che sia il più possibile immediato e intuitivo.

2.3 Scenari

I possibili scenari di utilizzo del modulo di messaggistica sono:

- ogni volta che un utente deve comunicare con un altro utente remoto, il messaggio viene inviato, il sistema si occupa di farlo viaggiare in rete e si assicura che venga consegnato al destinatario giusto;
- ogni volta che un messaggio viene recapitato a un utente, quest'ultimo dovrà essere informato tramite una notifica visiva;
- quando un utente si autentica ed accede alla pagina della chat, il sistema deve mostrare tutti i messaggi a lui destinati, che non ha potuto visualizzare durante la sua assenza;

- il sistema deve tener traccia dei messaggi, ogni volta che ne viene inviato uno, dovrà essere salvato in una apposita struttura, in modo che siano reperibili alla visione in qualsiasi momento.

Scenario: Invio di un messaggio

ID (Nome)	Scambio di un messaggio.
Descrizione	Permette agli utenti della web application di spedire e ricevere un messaggio.
Attori principali	Utenti della web application.
Condizioni Preesistenti	Non vi siano malfunzionamenti nel sistema, connessione alla rete disponibile.
Corso principale	Un utente desidera inviare un messaggio a un altro utente, o un gruppo di essi.
Condizioni di successo	Il messaggio arriva integro a destinazione e viene visualizzato correttamente.
Condizioni di fallimento	Il messaggio non viene recapitato, arriva al destinatario sbagliato, non viene elaborato in maniera adeguata e non viene visualizzato correttamente.

Scenario: Notifica di un messaggio

ID (Nome)	Notifica di un messaggio.
Descrizione	Permette agli utenti della web apllication di ricevere una notifica quando un messaggio viene consegnato.
Attori principali	Utenti della web application.
Condizioni Preesistenti	Non vi siano malfunzionamenti nel sistema, connessione alla rete disponibile.
Corso principale	È stato consegnato un messaggio.
Condizioni di successo	Il messaggio consegnavo viene notificato all'utente che ha la possibilità di visualizzarlo.
Condizioni di fallimento	Il messaggio non viene notificato.

Scenario: Recupero dei messaggi

ID (Nome)	Recupero dei messaggi.
Descrizione	Permette alla web application di recuperare i messaggi destinati a un utente, mentre quest'ultimo era offline e di renderli disponibili alla visione.
Condizioni Preesistenti	Non vi siano malfunzionamenti nel sistema, connessione alla rete disponibile.
Corso principale	Un utente si autentica e accede alla pagina della chat.
Condizioni di successo	I messaggi vengono recuperati correttamente e resi disponibili alla visione.
Condizioni di fallimento	I messaggi non vengono recuperati, i messaggi vengono recuperati ma non resi disponibili alla visione.

Scenario: Tracciabilità dei messaggi

ID (Nome)	Tracciabilità dei messaggi.
Descrizione	Tiene traccia di tutti i messaggi trasmessi in rete.
Condizioni Preesistenti	Non vi siano malfunzionamenti nel sistema, connessione di rete disponibile.
Corso principale	Un nuovo messaggio da consegnare arriva al web server REST.
Condizioni di successo	Il messaggio è tracciato correttamente.
Condizioni di fallimento	Il messaggio non viene tracciato.

2.4 Architettura

Per portare a compimento gli obiettivi prefissati nei requisiti è necessario che il modulo da implementare sia distribuito nel sistema, ossia va suddiviso in componenti da aggregare rispettivamente ai sottosistemi interessati (la Web Application e il Web Server REST, vedi Figura 2.2), si rende inoltre necessario realizzare ex novo il componente che si occupa della trasmissione dei dati real-time. Si può quindi definire uno schema di funzionamento del modulo come quello nella figura 2.2:

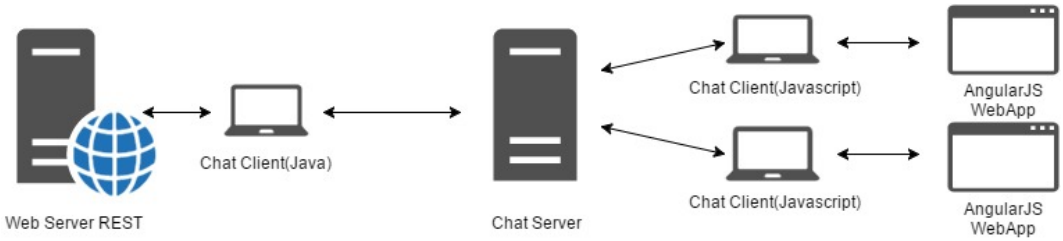

Figura 2.2: Architettura logica del modulo di messaggistica.

2.4.1 Architettura logica

A fronte dell'analisi effettuata si può ridefinire l'architettura descritta in Figura 2.3 nel modo seguente:

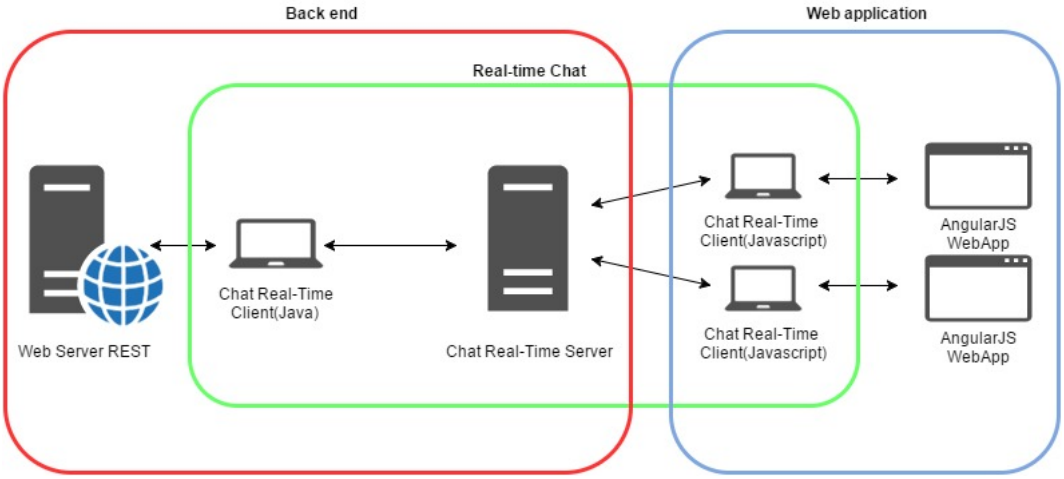

Figura 2.3: Architettura logica del modulo di messaggistica, integrato al sistema preesistente.

Come si può vedere dalla Figura 2.3, non solo la web application, ma anche il web server REST si interfaccia al server real-time, questo si può spiegare con la necessità del sistema di consegnare istantaneamente i messaggi al destinatario. Da tutto ciò si evince che il server real-time non può operare a polling, inviando richieste a intervalli regolari al web server REST, ma è necessario creare un componente in grado di inviare notifiche push, che, per definizione, consentono al messaggio di essere recapitato al destinatario senza che questi debba effettuare alcuna richiesta, diminuendo decisamente non solo il tempo di consegna, ma anche il traffico di rete.

2.5 Web Application

Il componente del modulo che va aggregato alla web application, oltre a inviare dati, ha il compito elaborare quelli in arrivo dal back end, per renderli fruibili alla visione dell'utente. Lo si può organizzare quindi, implementando il pattern MVC, in due sotto-componenti: il controller, che si occupa di gestire tutte le operazioni in ingresso e uscita, che siano esse per instaurare connessioni o per l'invio e la ricezione di dati, e la view, che gestisce la visualizzazione dei messaggi, specificando informazioni aggiuntive ad esempio mittente, ora e così via. Perciò dai requisiti si evince che la view, al suo interno, ha uno o più componenti per mostrare all'utente i messaggi ricevuti, popolati di volta in volta dal controller, che, man mano che analizza i messaggi in ingresso, riproduce i contenuti nella view. Inoltre si deduce che nella view dovranno essere presenti tre componenti, che specifichino il tipo di destinatario scelto e diano all'utente la possibilità di passare facilmente da uno all'altro.

In conclusione, dopo l'analisi dei requisiti, le funzionalità del componente web application del modulo si possono riassumere in:

- recupero dei messaggi destinati all'utente, ma non ancora ricevuti, perché in precedenza era offline.
- ricezione di nuovi messaggi e il loro collocamento all'interno dei componenti designati nella view, per essere sottoposti alla visione dell'utente.
- notifica all'utente di un nuovo messaggio.
- invio del messaggio con scelta del destinatario, fra le modalità singolo utente, scelta per ruolo e broadcast.

2.6 Back end

Il componente del modulo da inserire nel back end è a sua volta diviso in due sotto-componenti: la parte da integrare al web server REST e quella che gestisca lo scambio di informazioni real-time. Dal requisito sulla tracciabilità dei dati che viaggiano in rete si evince che il primo componente è designato al mantenimento dei dati e alla preservazione della loro consistenza, nonchè al salvataggio dinamico di nuovi messaggi, i quali devono essere disponibili alla web application, qualora gli utenti necessitassero di visualizzarli. Una volta tenuta traccia di un nuovo messaggio arrivato, il sotto-componente REST dovrebbe delegare alla parte real-time il compito di consegnarlo al giusto destinatario. Tutti questi aspetti possono essere incapsulati all'interno di una

risorsa REST, implementando dovutamente il comportamento che dovrà avere in risposta alle chiamate HTTP. Mentre il secondo sotto-componente andrebbe a svolgere la mansione di "dispatcher" dei messaggi che transitano nella rete, smistandoli real-time ai dovuti destinatari.

Dopo l'analisi dei requisiti, le funzionalità del componente back end del modulo si possono riassumere in:

- tracciabilità dei messaggi all'interno del sistema.
- gestione delle connessioni dei vari componenti al server per le comunicazioni real-time.
- gestione dei e smistamento dei messaggi.
- ritorno dei messaggi destinati a un utente che ne faccia richiesta.

In conclusione è necessario soffermarsi nuovamente sulla differenza sostanziale tra le due diverse modaità di ottenimento dei dati, pull e push: la prima esegue ripetutamente a intervalli di tempo prefissati delle richieste al server, richiedendo se ci sono nuovi dati, mentre la seconda ha bisogno di una sola richiesta per ottenere le informazioni desiderate, in quanto non è il client a fare richieste, ma bensì è il server a notificare che la e consegnare il dato al client, con una notifica. Questa tecnica di comunicazione ha un impatto enorme, in termini di performance, rendendola una delle più adatte per lo sviluppo di sistemi real-time.

Capitolo 3

Progettazione del modulo

Da qui in poi la parte del sistema che riguarda la mobile application verrà tralasciata, in quanto l'obiettivo di questo elaborato è mirato allo scambio di messaggi fra utenti della web application, mentre la parte in questione si concentra più sul soccorso dei pazienti, perciò i componenti inseriti saranno concentrati nella web application e nel back end. Nel capitolo verranno descritte ed evidenziate le scelte progettuali effettuate per giungere all'integrazione del modulo di messaggistica nel sistema esistente, sarà anche descritta la struttura delle classi e le operazioni svolte dai singoli componenti. Dovendo aggiungere componenti a un sistema già consolidato e funzionante, si è deciso di seguire la linea di sviluppo scelta per il sistema, adottando quando possibile i framework, le tecnologie e i paradigmi già utilizzati, al fine di agevolare l'integrazione.

3.1 Progettazione del back end

La fase progettuale per la realizzazione del componente lato back end prevede l'aggiunta di diversi componenti:

- per accedere ai dati del sistema, sia in lettura che in scrittura verrà aggiunta una risorsa;
- verrà definita una classe che sarà designata ad incapsulare la logica del messaggio;
- un client e un server, per l'invio dei messaggi tramite notifiche push.

3.1.1 Descrizione e struttura delle classi

In questo paragrafo verrà descritta la struttura delle classi che sono state aggiunte al sistema, andando a descrivere la loro semantica e le loro operazioni.

Messaggio

Innanzitutto è stata progettata una classe che rappresentasse il messaggio, ossia la struttura dati che viaggerà attraverso la rete fra i due endpoint. Nella seguente figura si può vedere come è stato ideato:

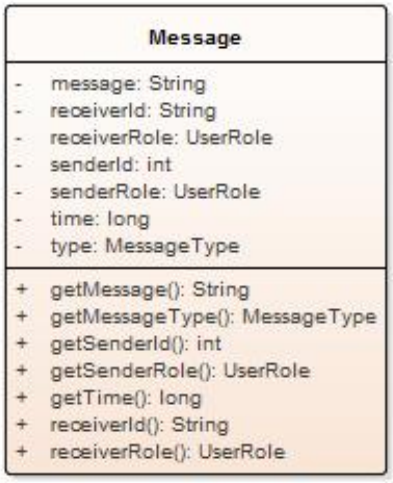

Figura 3.1: Classe che rapprenta un messaggio nelle rete.

All'interno questa classe incapsula tutti i dati utili al sistema per inviare, consegnare e visualizzare correttamente il messaggio. Come dati utili sono stati individuati:

- il tipo di messaggio, che fungerebbe da discriminante nel caso possa essere utile definire messaggi semanticamente diversi.
- l'identificatore e il ruolo del mittente del messaggio, che dovranno essere resi noti al destinatario.
- l'identificatore e il ruolo del destinatario, dati che saranno utilizzati prima nella web application per specificare il destinatario, poi nel back end per indirizzare il messaggio verso l'utente giusto.
- il momento in cui è stato inviato il messaggio, perchè può tornare utile nel caso si volesse dare un ordine ai messaggi ricevuti, oppure anche solo per pura visualizzazione.
- il messaggio vero e proprio.

Web Server REST

Per quanto riguarda il server REST, si è giunti alla conclusione di aggiunger una risorsa, in cui è definita l'interfaccia di accesso ai dati; è stato pensato anche un database che conservi tutti i messaggi, come si può vedere nell'immagine seguente:

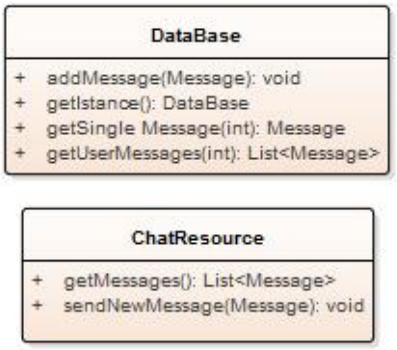

Figura 3.2: Le classi in figura rapprentano, rispettivamente dall'alto verso il basso, un database, per la tracciabilità dei dati e la risorsa che può essere acceduta attraverso le chiamate HTTP.

L'interfaccia definita nella risorsa indica le regole di accesso ai dati contenuti nel database, quindi bisognerà definire una prima operazione che prelevi tutti i messaggi destinati a un utente (tramite una richiesta HTTP di tipo GET), perchè saranno richiesti nel momento in cui un utente si autentica e accede al sistema, mentre la seconda operazione (invocabile con una richiesta HTTP di tipo POST) sarà utilizzata per salvare un messaggio nuovo nel database, per poi inviarlo al server real-time, che si occuperà di farlo arrivare a destinazione. Di conseguenza le operazioni da fare sul database saranno le stesse, ossia prelevare tutti i messaggi e inserirne uno nuovo, operazioni che saranno utilizzate dalla risorsa per eseguire le proprie elaborazioni. Come database al momento dell'implementazione si potrà scegliere se modificare quello già esistente per adattarlo a contenere i dati relativi ai messaggi, oppure crearne uno dedicato, però è importante considerare che, nel caso se ne crei uno ex novo, esso dovrà essere unico in tutto il sistema, per evitare di incorrere in problemi di inconsistenza dei dati.

Componenti real-time

I componenti che aggiungeranno al sistema la funzionalità di scambio di messaggi real-time sono due, un primo che sarà il client dedicato al server REST, avrà il compito di inviare i messaggi al secondo, il server, che a sua

volta penserà a smistarli ai destinatari corretti. Nell'immagine sottostante si può osservare come è stato pensato il modello per questi due componenti:

Figura 3.3: Le classi in figura rappresentauno il client real-time utilizzato dal server REST per comunicare con il server real-time, che è l'altro componente in figura.

Il funzionamento di queste due classi è molto intuitivo: il client farà pervenire il messaggio al server che si occuperà di reperire il destinatario dai dati, per poi inviarli, avendo come unica restrizione il tempo in cui queste operazioni devono essere eseguite.

3.1.2 Flusso di operazioni

Come si può vedere dalla Figura 3.5, ci sono due tipi di interazioni nel back end, che sono legate alle operazioni su richiesta dall'utente (richiesta HTTP specificando l'URI che identifica la risorsa). La richiesta di tutti i messaggi coinvolte il web server REST che riceve la richiesta dall'applicazione web remota, poi il server chiama in causa il database per ottenere l'insieme dei messaggi destinati all'utente, per poi restituirli alla web application. La richiesta di invio di un nuovo messaggio invece chiama in causa tutti i componenti, in quanto il server REST la riceve dalla web application remota, poi si occupa di aggiornare il database, inserendo il nuovo messaggio, per poi passarlo al client real-time, il quale a sua volta lo invierà al server real-time, che lo consegnerà all'utente designato.

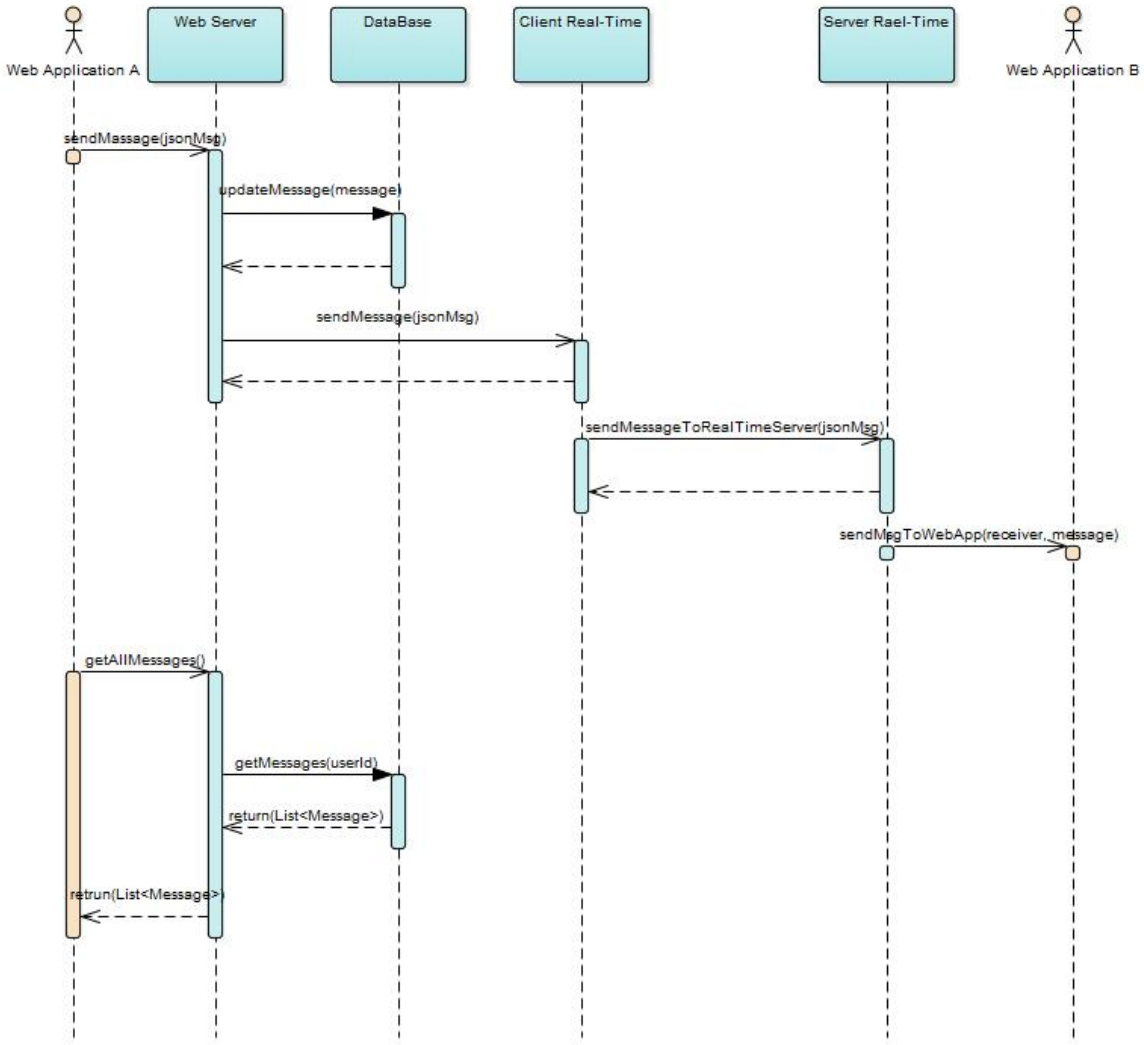

Figura 3.4: La figura rappresenta un diagramma di seguenza che esprime come avviene l'interazione fra i componeni del back end.

3.2 Progettazione della web application

3.2.1 Architettura logica

L'applicazione web è del sistema è stata sviluppata in AngularJS, implementando il pattern MVC, per cui si avrà un componente grafico, la View, e un componente di gestire le informazioni e cambiare dinamicamente quelle visualizzate, il controller, il quale si occuperà anche di fare le richieste HTTP al server, per recuperare i dati da visualizzare. È in questo livello dell'architettura che si inserisce il componente per la comunicazione real-time, che diventerà

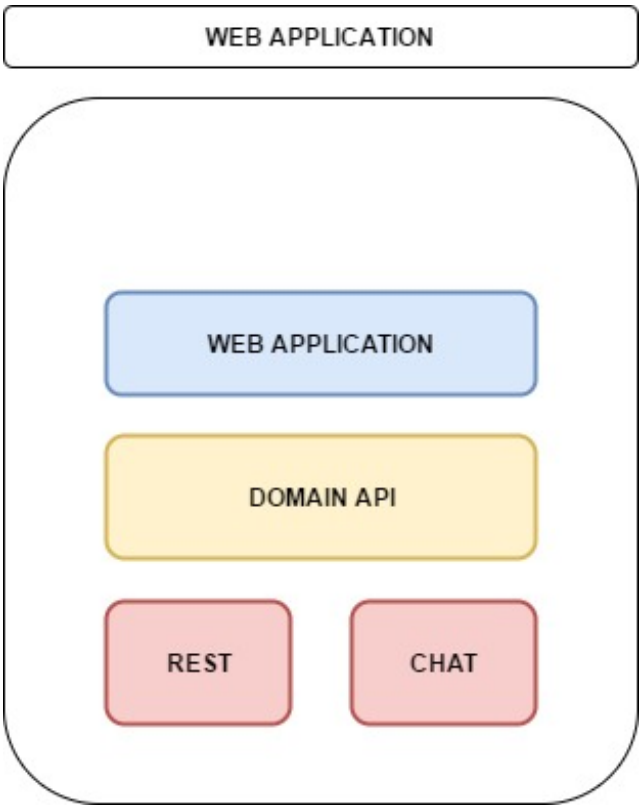

Figura 3.5: La figura rappresenta l'architettura dei livelli logici della web application del sistema già esistente, con inserito il modulo della chat.

un metodo di comunicazione parallelo, ma anche alternativo a REST. Sono due sistemi di comunicazione diversi, con modalità di ottenimento dei dati diverse, REST funziona a polling interrogando il server alla ricerca di nuove informazioni disponibili, mentre il sistema real-time comunica in maniera immediata, con notifiche push, inviate quando un un nuovo dato è disponibile.

3.2.2 Flusso di operazioni

Un sistema può essere descritto in vari modi, ad esempio si può rappresentare il suo comportamento, si può rappresentare la sua struttura, ma c'è un caso in particolare che fornisce informazioni generali, oltre alle suo specifiche per cui è stato definito: di tratta di un diagramma di interazione, che non solo mostra l'interazione fra i componenti del sistema, ma da esso è possibile ricavare informazioni anche sulla struttura e il comportamento del sistema. Perciò racchiude più concetti nella sua rappresentazione, tutti in unico diagramma,

il che lo rende uno strumento molto potente, nonchè nella descrizione dei vari aspetti di un sistema.

Dalla Figura 3.6 si evincono le possibili interazioni fra i componenti della Web Application, una volta che l'utente accede alla pagina di chat, il componente real-time si attiva e invia una richiesta HTTP al server REST per ottenere tutti i messaggi non ancora letti, al ritorno li passa alla web application, che provvede ad aggiornare dinamicamente la View, grazie al data binding bidirezionale. Inoltre si può notare come, una volta ricevuto un messaggio dal Server real-time, il client real-time lo passi alla web application e automaticamente la View risulterà aggiornata (per lo stesso motivo descritto poco fa), infine la web application si occupa di notificare all'utente il nuovo messaggio in arrivo e il mittente.

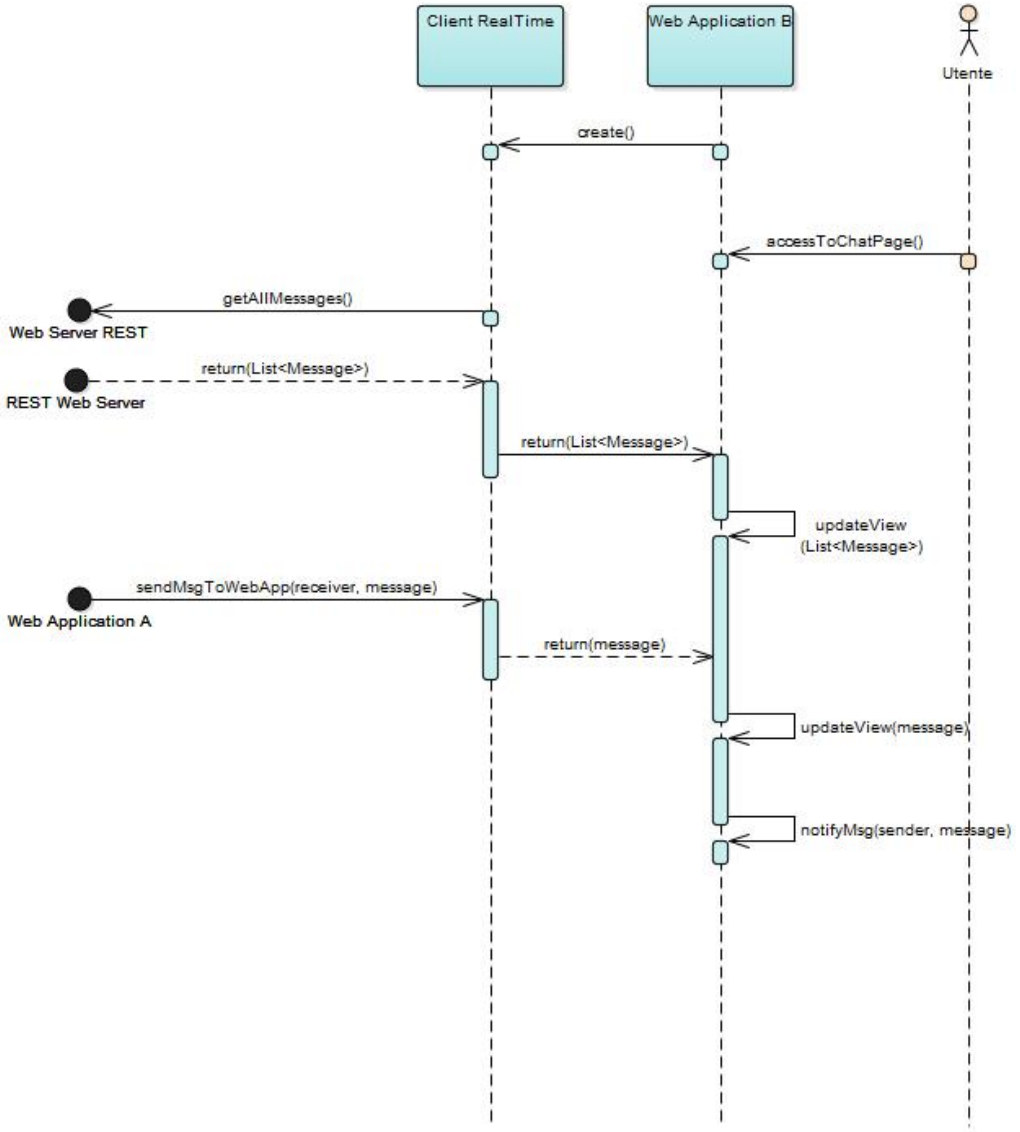

Figura 3.6: La figura mostra il diagramma di sequenza che esprime le interazioni fra i componenti .

Capitolo 4

Implementazione ed evaluation

In questo capitolo verranno trattati tutti i dettagli implementativi del modulo di messaggistica e verranno mostrati i risultati dei test eseguiti nel momento in cui il componente in questione è stato integrato nel sistema.

4.1 Tecnologie utilizzate

In questa fase dello sviluppo sono state fatte delle scelte di implementazione:

- per le librerie con cui implementare il componente real-time si è scelto WebSocket, che ha gli strumenti per fornire il grado di istantaneità richiesto, implementando il meccanismo delle notifiche push;
- per integrare il modulo nel server REST si è scelto di proseguire con la linea di sviluppo del sistema preesistente, ossia di utilizzare le librerie di Restlet;
- per la web application si è scelto di utilizzare AngularJs, per la gestione dinamica della pagina, e Bootstrap per quanto riguarda la disposizione degli elementi all'interno della pagina; anche in questo caso sono state utilizzate le tecnologie in accordo con il resto del sistema.

Oltre alle tecnologie sopra citate occorre specificare che, per quanto riguarda la scelta del database, non è stato utilizzato quello già disponibile nel sistema, ma si è scelto ti utilizzare un oggetto Java che fungesse da database, implementando il pattern Singleton, quindi facendo sì che di quell'oggetto Java ne venga istanziato uno solo e che tutti gli altri oggetti interagiscano con esso, mantenendo quindi i dati consistenti in una ConcurrentHashMap di Java. Infine si vuol far presente che tutti messaggi che viaggiano in rete sono serializzati in una stringa JSON.

4.1.1 WebSocket

Figura 4.1: Logo di WebSocket.

WebSocket[4] è un protocollo che permette di creare una connessione con un host remoto, utilizzando un unico canale di comunicazione bidirezionale full-duplex, consentendo quindi la comunicazione simultanea in entrambi i versi del canale. In questo modo ottiene notevoli miglioramenti in termini di performance, riducendo il carico e la latency della rete, caratteristica che lo rende adatto allo sviluppo di applicazioni real-time, a dispetto di comunicazioni basate su polling, che simulano un canale full-duplex, però tenendo aperte due connessioni, intasando la rete con richieste cicliche. WebSocket è implementato sul protocollo TCP, però appoggiandosi a HTTP per instaurare la connessione tramite un processo di handshake (Figura 4.2), regolando i parametri di comunicazione. Nel citare WebSocket, bisogna far notare che, in base a quanto appena detto, all'interno del modulo prodotto per l'elaborato andrà a ricoprire il ruolo del componente designato all'invio di notifiche push, quindi andrà a gestire tutta la parte real-time del modulo, in quanto è considerato una delle scelte migliori per quanto riguarda l'utilizzo delle notifiche push, fondamentali per il modulo dell'elaborato. Nel back end[1] è utilizzato per l'implementazione del server real-time, che ora prenderà il nome di server websocket, il quale ha il ruolo di smistare e consegnare tutti i messaggi che vengono inviati al sistema. Oltre al server è stato utilizzato anche per il client websocket del server REST, attraverso il quale quest'ultimo riesce ad interfacciarsi e a inviare i messaggi al server websocket, affinchè essi giungano a destinazione.

Mentre lato web application[2], è stato usato per connettere la web application al server non appena viene caricata la pagina della chat, per poi restare in attesa di eventuali messaggi in arrivo dal sistema.

[1] https://www.eclipse.org/jetty/documentation/9.4.x/jetty-websocket-server-api.html
[2] https://github.com/AngularClass/angular-websocket

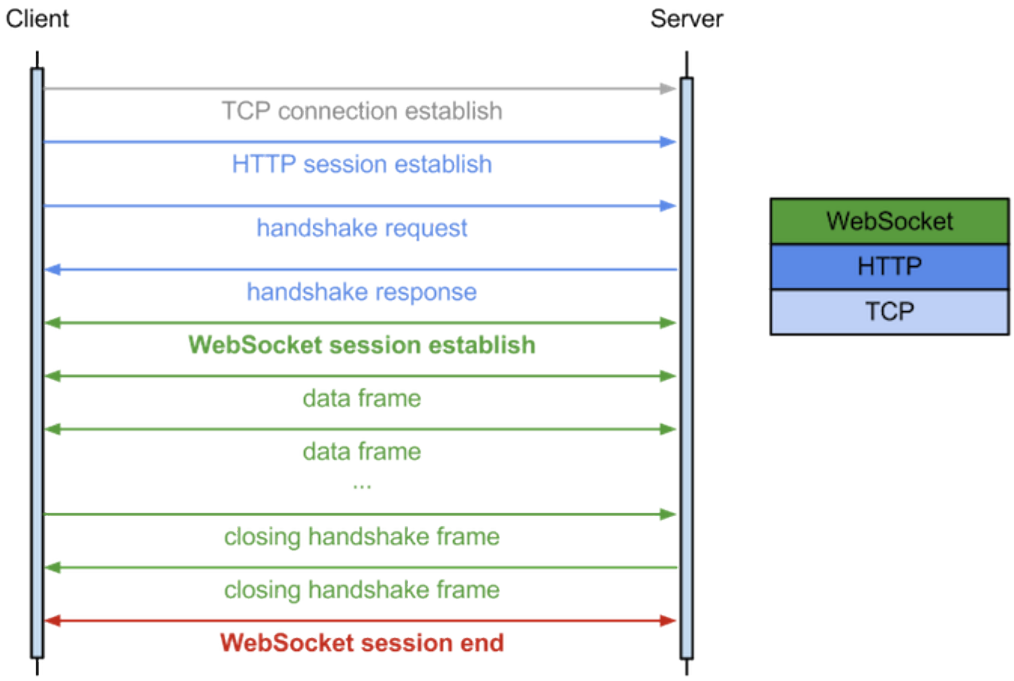

Figura 4.2: Apertura di una connessione tramite handshake e scambio di dati fra client e server.

4.2 Componenti del modulo

In Figura 4.3 è possibile vedere come interagiscono fra loro i componenti in cui è stato suddiviso il modulo al momento dell'implementazione, come comunicano e interagiscono fra di loro al momento dell'invio di un messaggio.

4.2.1 Back end

Il back end è stato suddiviso in sotto-componenti che ora verranno trattati, tra i più degni di nota la risorsa REST e il Server WebSocket, che a loro volta però interagiscono con altri componenti creati appositamente per il corretto funzionamento del sistema.

Implementazione della risorsa REST

È stata implementata come una classe Java, chiamata ChatResouce, che estende la classe di ServerResource di Restlet dalla quale eredita i metodi per effettuare le richieste HTTP. Una volta che le è stato assegnato un URI

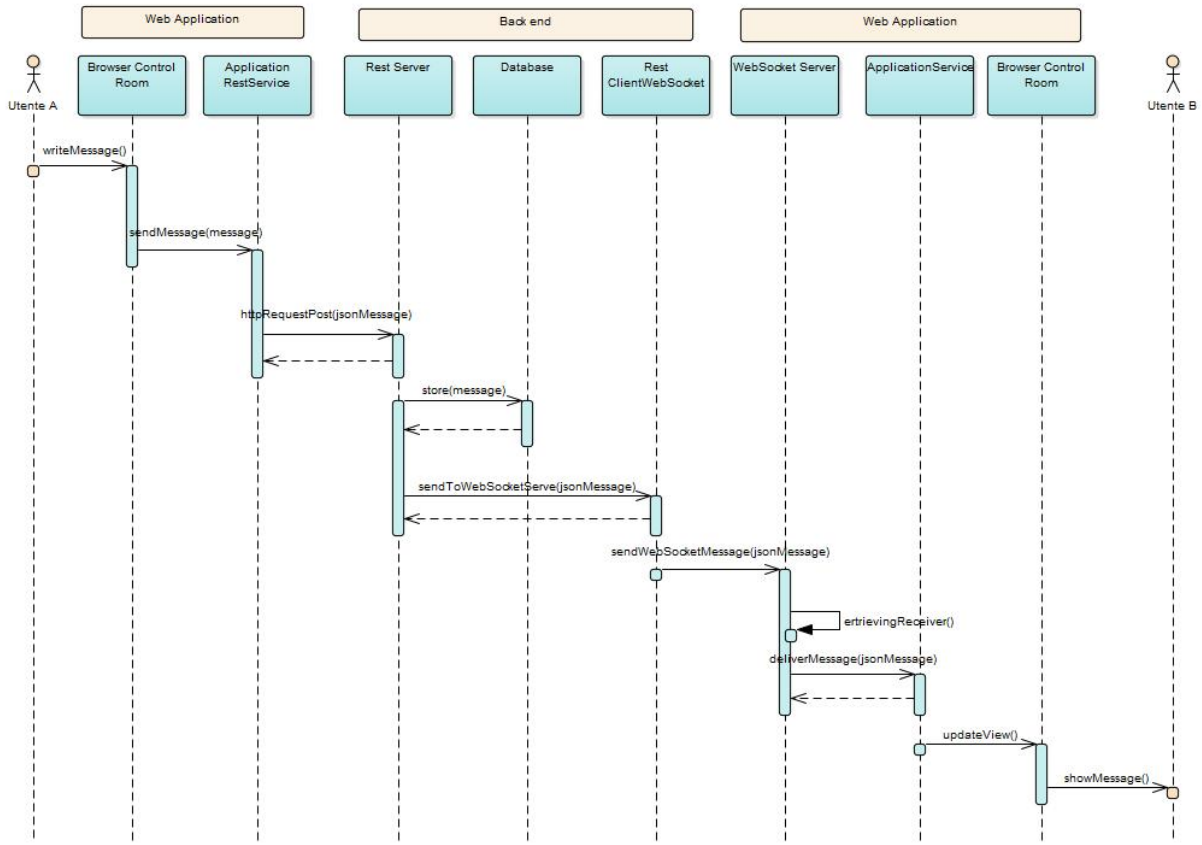

Figura 4.3: Diagramma di sequenza che mostra l'interazione fra i vari componenti del sistema.

univoco, per raggiungibile da remoto, non resta altro che definirli suo comportamento in risposta alle richieste.

La richiesta GET (in Figura 4.4) implementa la funzionalità di recupero di tutti i messaggi per l'utente (il cui ID è stato passato come parametro alla richiesta HTTP) dal database (db), chiamando una funzione di quest'ultimo che prende tutti i valori relativi all'utente specifico della ConcurrentHashMap e li ritorna.

La richesta POST invece non fa altro che mettere nel database il messaggio e inoltrarlo al WebSocket server, affinchè giunga a destinazione.

```
@Get("json")
public List<Message> getMessages(){
   String idUser = getQueryValue("user");
   List<Message> msg = db.getUserMessages(idUser);
   if(msg.isEmpty()){
      return null;
   }
   return msg;
}
```

Figura 4.4: metodo GET della risorsa ChatResource.

```
@Post("json")
public void sendNewMessage(Message msg) throws Exception{
   //Updating database
   db.addMessage(msg);
   //Sending message to WebSocket Server
   ClientSocket.getInstance().sendMessage(new Gson().toJson(msg));
}
```

Figura 4.5: metodo POST della risorsa ChatResource.

Implementazione dei componenti WebSocket

Come già citato sopra, le librerie di Websocket sono state utilittate per implementare due componenti del back end: il ClientSocket e il ServerSocket. Il client è utilizzato dal Server REST per comunicare con il Server WebSocket, esso implementa il pattern Singleton, per fare in modo che si crei una connessione univoca tra i due server. Il comportamento ClientSocket è molto semplice, non fa altro che connettersi al ServerSocket e, a connessione avvenuta, si autentica, in modo che il Server WebSocket sappia che sta comunicando con il server REST e non con un comune client. infine rimane tutto il tempo in attesa che la ChatResoure lo utilizzi per inoltrare un messaggio al ServerSocket.

Il comportamento del ServerSocket è più interessante, è racchiuso tutto in un metodo chiamato handleMessage (Figura 4.6), che viene chiamato quando arriva un nuovo messaggio, i possibili scenari sono 3:

- il messaggio è una richiesta di autenticazione del server REST, in questo caso tiene traccia della sessione aperta fra i due, in modo da poter comunicare in futuro;
- il messaggio è una richiesta di autenticazione di una web application, in questo caso tiene traccia della sessione aperta, in modo da poter

comunicare in futuro;

- un messaggio vero e proprio destinato a un utente, in questo caso allora ricava il destinatario dal messaggio e poi lo invia.

```
// called when a message received
   @OnWebSocketMessage
   public void handleMessage(String message){
       switch (message) {
           case WebSocketUtils.RESTLET_SERVER:
               if (restletSocket == null) {
                   ServerSocket.restletSocket = this;
                   ServerSocket.newSockets.remove(this.sessionId);
               }
               break;
           default:
              if (!message.isEmpty()) {
                  Message msg = new Gson().fromJson(message,
                      Message.class);
                  if(msg.getMessageType() ==
                      MessageUtils.MessageType.CONNECTION_MESSAGE){
                     this.userId = msg.getSenderId();
                     this.userRole = msg.getSenderRole();
                     ServerSocket.newSockets.remove(this.sessionId);
                     ServerSocket.browserSockets.put(this.userId,
                         this);
                }else if(msg.getMessageType() ==
                     MessageUtils.MessageType.DATA_MESSAGE){
                   this.sendMessage(new Integer(msg.getReceiverId()),
                       message);
                }

              }
              break;
       }
   }
```

Figura 4.6: Metodo handleMessage del ServerSocket, chiamato quando è ricevuto un messaggio.

4.2.2 Web application

La web application è stata sviluppata con AngularJS, appoggiandosi a Bootstrap, HTML5 e CSS3 per quanto riguarda la la realizzazione dell'interfaccia utente e a Javascript per gestire le operazioni interne alla web application. La pagina realizzata è responsive, quindi riadattando le dimensioni dello schermo, o cambiando device, la pagina si adatta alle nuove dimensioni in modo da riuscire a visualizzare i contenuti; il compito di visualizzazione e modifica della pagina è stato reso più semplice dal data binding bidirezionale di AngularJS, creando un controller della pagina e utilizzando le opportune direttive nel codice HTML è stato quindi reso dinamico il cambiamento dei dati.
Prima di entrare nel dettaglio è opportuno spiegare cosa siano due componenti fondamentali per lo sviluppo in AngularJS: il controller è il componente che gestisce l'interazione fra il modello dei dati e la View (pattern MVC), mentre un service solitamente implementa la logica dell'applicazione, essi non interagiscono con la view in modo diretto, possono essere utilizzati per condividere funzionalità con altri componenti dell'applicazione, implementando il pattern Singleton.

Implementazione della web application

L'interfaccia grafica è stata creata e organizzata in modo che fosse molto semplice all'uso, è costituita da dei bottoni o gruppi di bottoni per la scelta del destinatario (singolo, per ruolo, broadcast) di un messaggio, una volta fatto ciò si può inserire il testo in una casella di input e alla pressione del tasto "invia" viene richiamata una funzione del controller che utilizza i dati settati dalla view per identificare il destinatario, o il tipo in caso di destinatario multiplo, per poi sfruttare un service appositamente creato per l'invio dei dati al Server REST tramite una richiesta HTTP di tipo POST. Tornando all'interfaccia utente, sotto la parte che si potrebbe definire di "input", sono state posizionate tre tabelle, dedicata ciascuna alla visione di un tipo di messaggio diverso, perciò si possono visualizzare i messaggi ricevuti da un singolo utente, da un ruolo specifico oppure tutti i messaggi arrivati. Questa scelta dei messaggi da visualizzare è stata implementata con un componente di AngularJS chiamato filtro, che ha appunto il compito di filtrare dinamicamente i contenuti degli oggetti di visualizzazione in base alle regole definite dallo sviluppatore. Il service che invia i messaggi è lo stesso che, quando viene caricata la pagina, recupera (richiesta HTTP di tipo GET) dal Server REST tutti i messaggi in attesa di essere consegnati. Infine un altro service è dedicato alla ricezione di messaggi via WebSocket, quando viene caricata la pagina invia la richiesta di connessione, seguita da quella di autenticazione, poi rimane in attesa finchè non viene consegnato un messaggio, a quel punto vengono aggiornate opportunamente

le tabelle dei messaggi ricevuti e viene reso visibile un alert (componente di Bootstrap) che notifica all'utente il nuovo messaggio recapitato.

4.3 Evaluation

L'effettivo funzionamento del sistema è stato testato più volte, accedendo da browser all'applicazione web con il profilo di diversi utenti e, di volta in volta provare tutte le funzionalità, dall'invio di messaggio singolo o multiplo, il ridimensionamento della pagina, la corretta visualizzazione dei messaggi, il modulo di cambio lingua e così via. I requisiti sono stati soddisfatti e il modulo di messaggistica integrato nel sistema funziona in tutti i punti che erano richiesti.

4.3.1 Interfaccia utente

Nella figura 4.7 si può vedere l'interfaccia in cui l'utente visualizza i dati e con la quale può interagire, si vedono in azzurro i tre pulsanti per la scelta del destinatario, e sulla destra la barra in cui verrà visualizzato un volta selezionato. Nella riga sottostante si vede la casella per l'inserimento del testo con alla sua destra il bottone per l'invio del messaggio, mentre la tabella con il pulsante rosso al centro è designata a contenere i messaggi filtrati in base a un singolo utente.

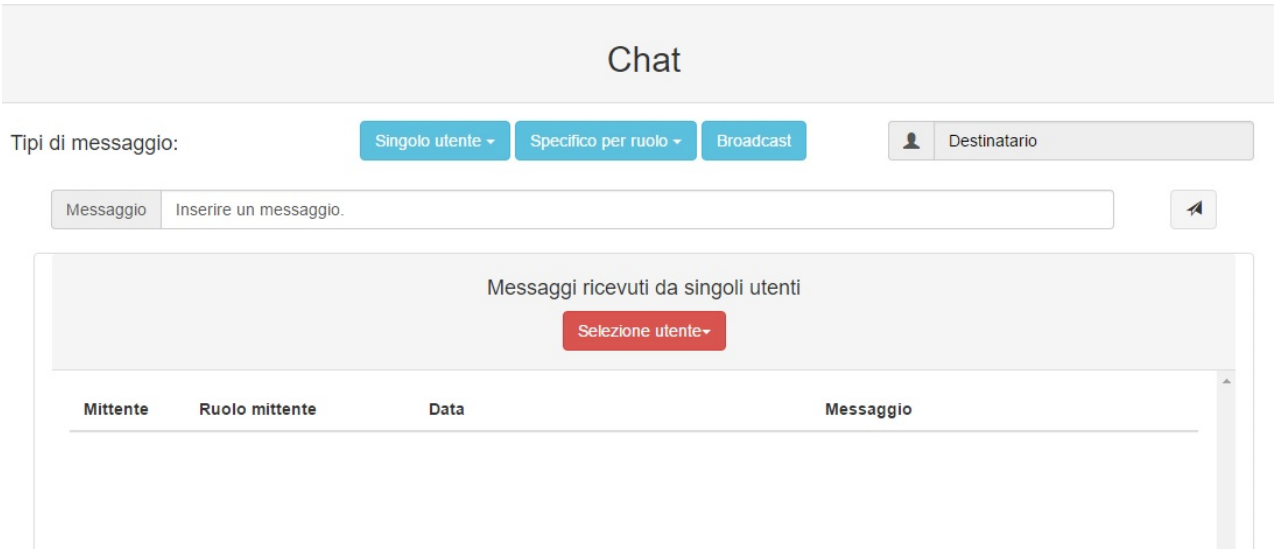

Figura 4.7: Interfaccia utente della web application.

4.3.2 Predisposizione all'invio

Nella Figura 4.8 l'utente ha selezionato un destinatario, che quindi è comparso nella casella a destra, inoltre il messaggio è stato scritto ma non ancora inviato.

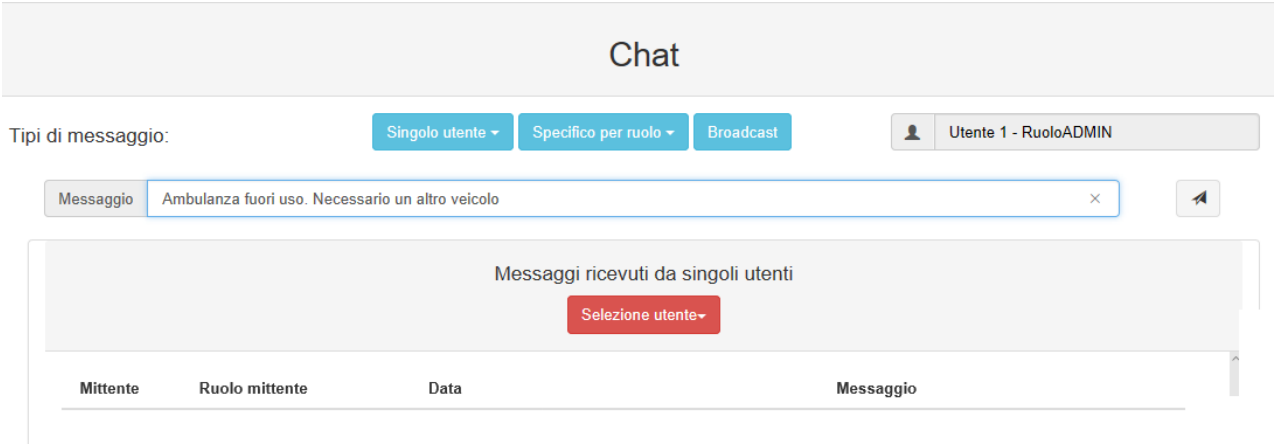

Figura 4.8: Selezione di un utente per inviare un messaggio.

4.3.3 Ricezione di messaggi

Nella Figura 4.9 è visualizzata la la tabella che contiene tutti i messaggi ricevuti, come si può vedere essa viene popolata ogni volta che viene ricevuto un messaggio.

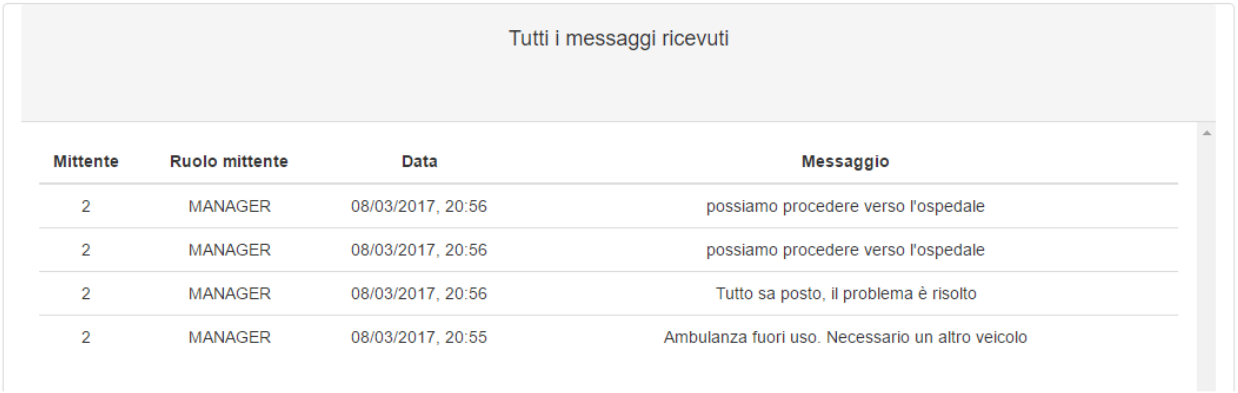

Figura 4.9: Tabella con all'interno tutti i messaggi consegnati all'utente.

4.3.4 Notifica

Nella Figura 4.10 si può vedere un pannello verde in alto nella pagina, è un alert di Bootstrap al cui interno sono settate le informazioni del messaggio ricevuto.

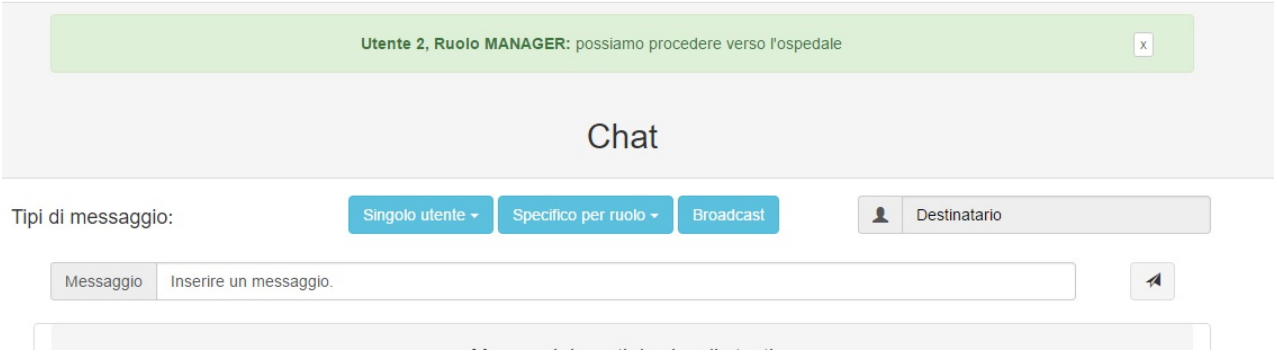

Figura 4.10: Notifica di un nuovo messaggio consegnato.

Conclusioni

Senza dubbio gli obiettivi che si vogliono raggiungere con lo sviluppo del sistema per la gestione delle emergenze, se frutto di coordinazione e scelte oculate possono migliorare notevolmente la speranza di vita di una vittima, l'assistenza, la cura e le tempistiche a mio avviso hanno un ruolo fondamentale in questi tipi di sistemi, perchè basta veramente un po' di disorganizzazione e il tempo a disposizione potrebbe non essere più sufficiente.
Per questo motivo penso che il contributo dato dal modulo di messaggistica real-time arricchisca, per quanto nel suo piccolo, un sistema già ben progettato. In generale sono rimasto molto soddisfatto degli argomenti trattati, è stata un'ottima opportunità per imparare a utilizzare nuove tecnologie, che al di fuori di questo contesto non avrei avuto modo di apprendere così approfonditamente. Il sistema finale è tuttavia aperto ad atri possibili sviluppi, per quanto riguarda il collegare il modulo di messaggistica al database già preesistente, che farebbe si di avere dati persistenti nel tempo, oppure la possibilità di inviare messaggi preconfigurati in caso si avesse la possibilità o il tempo per digitare un messaggio; infine un ulteriore possibile scenario potrebbe essere il refactoring dell'interfaccia della chat, riducendola in dimensioni e integrandola come un componente in un punto dello schermo.

Ringraziamenti

Un ringraziamento speciale va al professor Mirko Viroli, che mi ha dato la possibilità di sviluppare questo elaborato, un'occasione molto interessante che mi ha dato modo di imparare nuove tecnologie e paradigmi di programmazione; ringrazio tanto anche l'ingegnere Simone Grotti che mi ha seguito passo a passo in questo percorso, avendo molta pazienza; voglio ringraziare i miei genitori per avermi supportato fino alla fine, nonostante i vari alti e bassi, infine un grande ringraziamento anche a tutti i miei amici che mi hanno accompagnato in questo lungo viaggio.

Bibliografia

[1] Pagina ufficiale di AngularJS, *https://docs.angularjs.org/guide/*, [Online; in data 7-marzo-2017].

[2] Sito ufficiale di Bootstrap, *http://getbootstrap.com/*, [Online; in data 7-marzo-2017].

[3] A.Giunta, *Sviluppo di un modulo di real-time location-awareness a supporto dei sistemi di soccorso*, Corso di Laurea in Ingegneria e Scienze Informatiche, [Anno 2015-2016].

[4] Guida di Jetty WebSocket, *https://www.eclipse.org/jetty/ documentation/9.0.6.v20130930/*,[Online; in data 7-marzo-2017].

[5] Louvel, *Restlet in Action: Developing RESTful Web APIs in Java*, [Pubblicato nel 2012].

[6] Fielding, *Architectural styles and the design of network-based software architectures*[Pubblicato nel 2000].