Alma Mater Studiorum
Università di Bologna
Campus di Cesena

SCUOLA DI INGEGNERIA E ARCHITETTURA
Corso di Laurea Triennale in Ingegneria e Scienze Informatiche

SVILUPPO DI UNA PUBLISH/SUBSCRIBE APPLICATION PER LA CONDIVISIONE DI PUNTI DI INTERESSE: UNA SOLUZIONE BASATA SU WEBRTC.

Relatore:
Dott. Stefano Ferretti

Presentata da:
Edgal Sanchez Mora

Sessione II
Anno Accademico 2015-2016

Introduzione

Tra i vari servizi offerti da Internet, il World Wide Web è sicuramente il più conosciuto. Oggi, navigando sul Web, possiamo contattare i nostri amici o conoscere persone che condividono i nostri stessi interessi. L'obiettivo di questa tesi è la realizzazione di un'applicazione Web che permette la condivisione di "interessi" legati a un determinato luogo. Permette quindi di realizzare un sistema di condivisione fra utenti, che potranno comunicare in base alla loro posizione e ai propri interessi comuni. Con l'utilizzo dei punti di interesse e della geo-localizzazione è possibile creare un sistema nel quale la gente viene messa in comunicazione in base a interessi in comune,pubblicati dagli stessi. L'applicativo è progettato per utenti che vogliono visitare parchi, laghi, castelli, e che desiderino trovare altre persone con interessi comuni, dalle escursioni al canottaggio. Basandosi su tali interessi, oltre che alla posizione, viene offerta la possibilità di mettersi in comunicazione via WebRTC.

Indice

Elenco delle figure

Capitolo 1

Web

Attualmente, il Web [6] non è più solo uno strumento che permette alle persone di lavorare in contatto tramite l'utilizzo di documenti ipertestuali. La prima "versione" del web, o web 1.0, infatti, era costituita prevalentemente da siti-vetrina. Si parlava quindi di web publishing, in cui l'utente poteva si visualizzare il contenuto, ma non poteva interagire con la pagina. Con la seconda generazione, o Web 2.0, abbiamo assistito all'introduzione di diverse tecnologie fondate sulle basi di dati. Le pagine Web diventano dinamiche e "cambiano" in base alle richieste dell'utente. Dal semplice sito-vetrina, si è passati alla creazione di vere e proprie applicazioni web interattive, come, per esempio portali e-commerce, ma anche Content Management System e piattaforme e-learning. In questa prospettiva, il termine web 2.0, introdotto da Tim O'Reilly nel 2003, non sta tanto nell'evoluzione delle tecnologie, tanto quanto nella crescita della partecipazione e interazione da parte dell'utente. Infatti, gli standard tecnologici come l'architettura client-server, protocolli come HTTP, linguaggi come JavaScript e XHTML, hanno subito evoluzione, ma sono ancora alla base delle applicazioni odierne. Il cambiamento, infatti, risiede nella "rivalutazione" del ruolo dell'utente, che popola il web di propri contenuti. Un altro esempio concreto di web 2.0 sono social network, in cui l'utente condivide informazioni con altri utenti, creando dei veri e propri profili, dove vengono mostrati interessi, hobby, informazioni e molto altro,

liberamente.

The goal of the Web is to serve humanity.

1.1 Applicazioni Web

Di conseguenza, è nata la necessità di creare sistemi interattivi e scalabili che permettessero tali libertà. Un esempio risiede nell'utilizzo di tecnologie, quali HTML5 e Javascript per la realizzazione di applicazioni web e siti web dinamici. Per applicazioni web [4] si intende quella classe di programmi che permettono un ampio utilizzo delle tecnologie basate sul web a cui si accede tramite web browser. Con l'introduzione e l'evoluzione di tecnologie come JavaScript, Ajax,JSON, e Document Object Model, è stato possibile rendere interattive tali applicazioni Web, oltre a definire e a standardizzare il processo di creazione e implementazione di tali applicativi.Tale tipo di applicazione è diventato una possibilità di investimento per qualsiasi compagnia, creando una notevole domanda a livello di mercato. Infatti, qualsiasi azienda che sfrutti il Web necessita di servizi accessibili agli utenti, e che siano appunto, user-friendly.
Uno dei vantaggi principali di tale diffusione risiede nel fatto che le web-application sono cross-platform, possono quindi essere utilizzate indipendentemente dal device che si utilizza,indipendentemente dall'ambiente, purché dotato di browser. Questo ha portato a un incremento esponenziale di utilizzi di web-application da dispositivi mobili, principalmente smartphone e tablet. Inoltre, non vi è la necessità di installarle, o di doverle configurare, rispetto, per esempio, alla loro controparte, le applicazioni desktop o native, che tuttavia, riescono a sfruttare appieno le risorse del dispositivo sulle quali sono istallate.

1.2 Applicazioni native

Il principale limite delle applicazioni native per dispositivi mobili è la "platform fragmentation". Data la varietà di sistemi operativi mobili (Android, iOS, Windows Phone), e, che a loro volta hanno differenti release. Di conseguenza, dal lato dell'utente, coloro che hanno hardware datati, che magari non hanno la possibilità di aggiornare tali sistemi, si ritrovano impossibilitati a usare l'applicazione. Da parte dello sviluppatore, invece, la diversità di ambienti di sviluppo comporta la conoscenza di diversi linguaggi, per esempio Java per Android, o Swift nel caso volessimo realizzare applicazioni native per iOS. Questo significa che lo sviluppatore deve scegliere l'ambiente, sapendo che magari perderà una buona parte di utenti, oppure progettare la stessa applicazione su diversi devices, facendo sempre attenzione a gestire la compatibilità con versioni datate. Per questa serie di motivi, ho scelto di realizzare l'applicativo come web-application, utilizzando HTML5, CSS, e, principalmente, Javascript.

1.3 Web Application Framework

Data il crescente numero di applicazioni web, sono stati introdotti anche dei framework appositi, che facilitino il compito ai programmatori, oltre a definire eventuali pattern da implementare. Molti definiscono interazioni con database, creazione di template HTML gestione della sessione degli utenti. In generale, tutti i framework [15] hanno come principio base quello del riutilizzo del codice. Un esempio di questi è Apache Structs, largamente utilizzata nello sviluppo web,e che ha sua volta utilizza componenti quali Servlet, favorendo l'utilizzo del pattern MVC, discusso in seguito, nella realizzazione di applicativi.

1.4 Struttura applicazioni

Per sua natura, un'applicazione web può essere implementata con diverse architetture e organizzazioni logiche. Una delle strutture classiche [13] è la *Three-tier Architecture*, che prevede la suddivisione dell'applicazione in tre "strati" logici che rappresentano rispettivamente l'interfaccia utente, la logica dell'applicazione, e la gestione della persistenza dei dati.

- **Presentation layer**
- **Business layer**
- **Data layer**

Il modulo "presentation layer" definisce l'interfaccia utente, quindi, generalmente, i documenti HTML che verranno mostrati a coloro che accedono all'applicativo. In pratica il front-end dell'applicazione. La logica e il funzionamento dell'applicazione sono descritte dal "business logic", che ne rappresenta il cuore, tipicamente situtato in un "application server". Infine il "data layer" riconducibile alla gestione della persistenza dei dati, tramite l'utilizzo di database relazionali e non e query per richiamare tali dati.

Tali moduli sono concepiti, generalmente, per le architetture client-server, con una dipendenza fra strati. Lo strato di presentazione è dipendente dal modulo della logica dell'applicazione, che a sua volta dipende dal modulo della gestione dei dati.

Capitolo 2

Tecnologie del Web 2.0

In questo capitolo verranno trattate le tecnologie utilizzate nella realizzazione del progetto, spiegandone il funzionamento generale, rimandando l'utilizzo effettivo nei capitoli successivi, relativi alla progettazione e all'implementazione dell'elaborato.

2.1 Client-Side scripting

Per Client-Side scripting si intende quella classe di programmi eseguiti lato client,ovvero, all'interno del browser, che permettono di mostrare pagine con contenuto dinamico. Generalmente, tali script possono essere inclusi nel documento HTML (o XHTML),oppure sono esterni,e, in tal caso, sono referenziati da essi, richiamati all'occorrenza. I programmi appartenenti a questa classe hanno accesso alle risorse del browser sul quale sono eseguite, anche se limitati in "sandbox", per ovvie ragioni di sicurezza. Ovviamente devono essere scritti in linguaggi supportati dal browser. In questo progetto ho utilizzato principalmente JavaScript per l'implementazione del client, e sarà oggetto di descrizione a parte, in seguito.

2.2 Server-Side Scripting

Classe di programmi scritti per funzionare su web Server. Tramite essi, posso gestire e personalizzare le risposte alle richieste di ogni client da parte del server. Gli script vengono richiamati dal browser dell'utente tramite richiesta, generalmente HTTP. Inoltre,per la realizzazione di tali script, è possibile utilizzare diversi linguaggi di programmazione, fra cui C/C++ e Perl, utilizzando la "Common Gateway Interface", un'interfaccia apposita per la comunicazione fra server e applicativo.

2.3 JavaScript

Linguaggio supportato dalla maggior parte dei browser, come Chrome, Firefox e Opera. Concepito da Netscape nel 1995 [12], inizialmente con il nome "Mochan" e successivamente LiveScript, fino ad arrivare a JavaScript. Si tratta di un linguaggio Object-Oriented debolmente tipizzato, utilizzato prevalentemente per la programmazione lato client, per la modellazione di pagine web dinamiche e largamente utilizzato nell'ambito delle applicazioni web, oltre a fornire strumenti per la validazione di documenti HTML. Si tratta di un linguaggio interpretato, e non compilato, e non è necessario alcun ambiente di sviluppo per la programmazione.

2.3.1 Ereditarietà

Come si evince da [5], JavaScript, a differenza di altri linguaggi Object-Oriented più "classici" come Java, supporta il principio l'"eredità prototipale". In Java, per esempio, si crea la classe, definendone il comportamento tramite metodi. Successivamente si istanzia un oggetto di quella classe che eredita le proprietà e i metodi della classe definita. Vi è inoltre il concetto di ereditarietà, che permette di creare classi "figlie", che erediteranno proprietà e metodi della classe "madre". In questo modo, è possibile organizzare gerarchicamente le classi di un progetto, permettendo così, il riuso del codice,

fondamentale per una buona programmazione. Il JavaScript permette si l'utilizzo dell'ereditarietà classica, tramite costrutti come "inherits" e "upon", simili rispettivamente all'"extends" e la "super" di Java, ma non si limita solo a questo. JavaScript estende il concetto di ereditarietà, rendendolo un linguaggio più dinamico rispetto al Java. Nello specifico, gli oggetti istanziati ereditano direttamente da altri oggetti, eliminando il concetto di classe. Invece di istanziare nuove classi ed interfacce associate, possiamo creare dei prototipi di oggetti, da cui derivare diverse istanze, tramite le è possibile arricchire tale oggetto di metodi e proprietà, e utilizzare i nuovi oggetti come prototipi per altre istanze, anche a runtime.

Ogni funzione, che è un oggetto a se, contiene implicitamente il "prototype", responsabile di fornire dei valori quando un oggetto lo richiede. Con questo meccanismo, è possibile simulare il concetto di costruttore. Aggiungere una proprietà a tale campo, significa rendere disponibile tale attributo al costruttore, e, quindi, anche a tutti gli oggetti che ereditano da esso. Inoltre, anche i prototipi possono essere ridefiniti a runtime. L'immagine sottostante rappresenta, in maniera semplificata, le differenze fra i due meccanismi di ereditarietà:

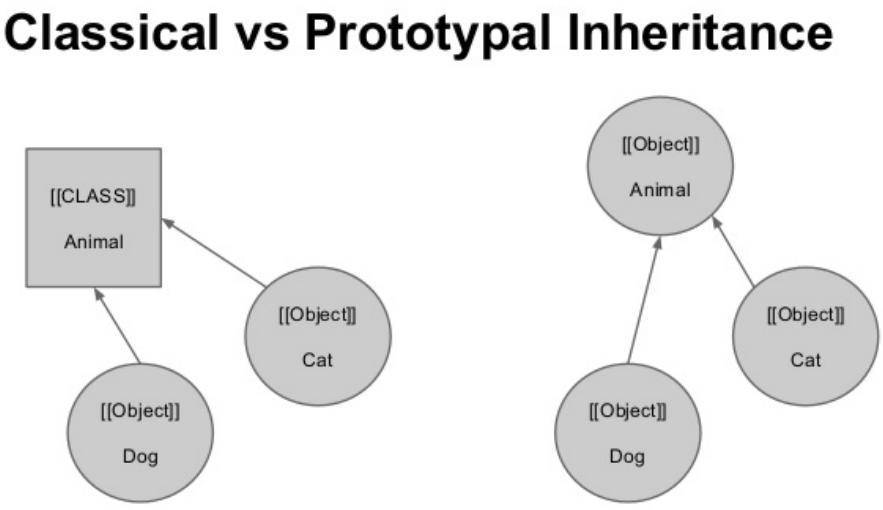

Figura 2.1: Differenza fra ereditarietà classica e prototipale

Concludendo la parte dedicata a tale linguaggio, gli script sono eseguiti direttamente sul browser, alleggerendo il carico di lavoro del server. Utilizzandolo assieme a AJAX, posso richiedere determinate informazioni, senza dover ricaricare la pagina HTML. In questo progetto, ho scelto di utilizzare JavaScript sia lato client,tramite script eseguiti direttamente sul browser, sia lato server, per gestire le richieste HTTP da parte dei vari utenti.

2.3.2 JQuery

Dato che ho utilizzato questa libreria in parte del progetto, occorre citare anche JQuery. Si tratta di una libreria di JavaScript, nata allo scopo di semplificare la manipolazione degli elementi HTML, e ottimizzare la gestione degli eventi.Inoltre, permette l'implementazione di funzionalità AJAX, descritte di seguito.

2.4 AJAX

Più che una tecnologia, è un insieme di tecniche e di metodi di utilizzo di tecnologie in maniera "anticonvenzionale". AJAX [14] sta per "Asynchronous JavaScript and XML", e descrive tecniche di richiesta dati asincrona, ovvero in background, senza che interferiscano con l'applicazione e compromettano l'utilizzo dell'utente. Un esempio pratico si ha con la possibilità di richiedere dati al server, senza dover necessariamente ricaricare la pagina con le informazioni aggiornate.

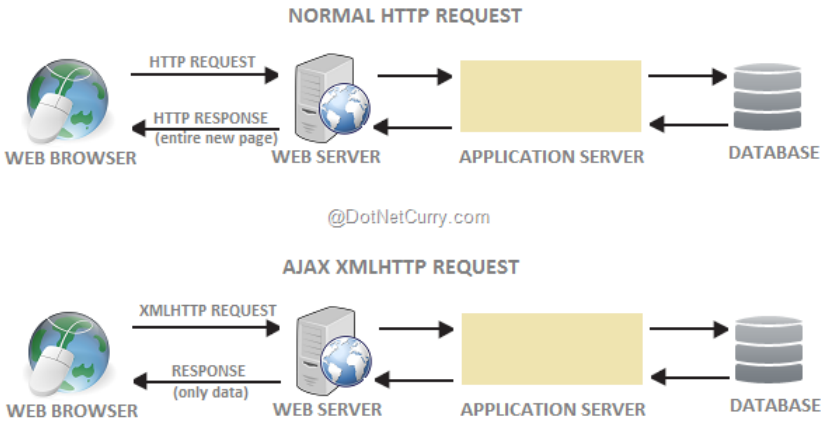

Figura 2.2: Differenza fra chiamata AJAX e normale richiesta HTTP.

Alla base, abbiamo richieste XMLHTTPRequest, ovvero richieste HTTP basate su documenti XML. Le richieste AJAX sono generalmente in formato XML, ma non è il solo tipo di dato supportato. Nel progetto presentato, ho utilizzato AJAX in combinazione con JSON, un formato di rappresentazione dei dati basato su JavaScript, che risulta essere più leggero rispetto a XML. Inoltre, le librerie per il parsing di tale formato, risultano anch'esse più leggere, guadagnandone in risorse utilizzate.

2.5 HTML

HTML (Hypertext Mark-up Language) è un linguaggio utilizzato per definire e formattare pagine o applicazioni web. Si basa su dei tag di formattazione, che definiscono, appunto, il layout del documento. Nonostante preveda l'inserimento di elementi multimediali,quali immagini, video e script, non si tratta di un linguaggio di programmazione. Non esiste la definizione di variabili o funzioni, tramite le quali si possono realizzare programmi.

2.5.1 Storia HTML

Introdotto dal Tim Berners-Lee alla fine degli anni ottanta, parallelamente al protocollo di trasporto HTTP, creato, appunto per il trasferimento di

documenti in tale formato. Questo metalinguaggio ha avuto una forte diffusione agli inizi degli anni novanta, in concomitanza con la diffusione del web a livello commerciale. Nel 2004, un gruppo di lavoro formato da Mozilla, Opera Software e Apple inizia a lavorare a una nuova versione di HTML, a causa del disinteresse (a favore di XHTML) del consorzio W3C. Il 28 ottobre il W3C pubblica le specifiche di HTML5 il 28 Ottobre 2014.

2.5.2 HTML5

Oltre a rendere più rigide le regole di strutturazione del testo in capitoli paragrafi e sezioni, HTML5 ha introdotto diverse novità, fra cui citiamo Canvas, che permette il rendering dinamico di immagini in formato, un sistema alternativo ai cookies, più efficiente in termini di risparmio di banda, e la geo localizzazione, fondamentale vista l'espansione dal punto di vista mobile. Proprio quest'ultima aggiunta, ha permesso la realizzazione di questo progetto.

2.5.3 Geolocation API

Da [7] abbiamo che la geo localizzazione permette, come dice il termine, di ottenere le coordinate della posizione attuale dell'utente in coordinate. Si compone di tre funzioni principali

- *GetCurrentPosition*
- *WatchPosition*
- *ClearWatch*

Con la prima, ricaviamo le coordinate in corrispondenza delle quali il dispositivo dell'utente è connesso. Anche se il metodo di localizzazione è specifico per ogni browser, in generale, nel caso di una soluzione desktop, vengono utilizzate alcune informazioni, come indirizzo IP pubblico e la locazione della stazione del provider di internet.

Figura 2.3: Esempio di utilizzo della funzione getCurrentPosition.

Invece, se si sta utilizza un dispositivo mobile, si tratta di una semplice interrogazione al GPS, per trovare la posizione dell'utente. Con la funzione *watchPoint*, possiamo decidere un intervallo di tempo per richiamare la *getPosition*, in modo tale da tenere traccia del movimento dell'utente per un determinato periodo. Infine la funzione *ClearWatch* serve azzerare le variabili di posizione e, nel caso, disattivare il GPS.

2.5.4 Leaflet

Leaflet è una libreria open-source di JavaScript per la creazione di mappe interattive. Permette un alto livello di personalizzazione, con l'aggiunta di *layers*, marker e popup personalizzabili. Permette, inoltre di gestire eventi, come interazioni del mouse, ed è compatibile con molti dispositivi.

2.6 Tecnologie Server-Side

In questo sezione verranno descritte le tecnologie server-side utilizzate dall'applicativo. L'applicazione è stato costruito sopra Node.js, un framework che offre un certo numero di vantaggi primi tra tutti la sua facilità d'uso e

la sua scalabilità. Verranno introdotti Node.js, il suo motore JavaScript V8, Express, un framework veloce e minimale per la costruzione di servizi web, e infine Passport js, che mette a disposizione diversi metodi di autenticazione.

2.6.1 Node

Node.js ‘e un framework utilizzato per costruire applicazioni di rete altamente scalabili. Nella home page di Node.js, troviamo:

> Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

Node utilizza JavaScript come linguaggio di programmazione, ed è basato su tre principi fondamentali:

- nessuna funzione dovrebbe eseguire I/O diretto; per ricevere informazioni dal disco, dalla rete o da un altro processo, ci deve essere una callback.
- Deve avere API di basso livello.
- Deve supportare i protocolli pi‘u importanti come TCP, HTTP e DNS.

Creato nel 2009 da Ryan Dahl. L'obiettivo era quello di creare un ambiente event-driven per la realizzazione di applicativi altamente interattivi.

2.6.2 Programmazione ad Eventi

Il modello di programmazione è asincrono e basato su eventi. tale stile di programmazione differisce dal "solito" sistema di thread concorrenti, come esposto in [8]. Utilizzando Node, tutte le operazioni prevedono meccanismi di callback da eseguire una volta terminata un'operazione associata. Le operazioni sono avviate in sequenza non bloccante. Questo perché in Node si ha

un Single Event Loop, che orchestra l'avvio di una operazione e l'esecuzione della sua callback. Tutti gli eventi sono elementi di una coda.

2.6.3 V8

Motore JavaScript Open source sviluppato da Google [11]. Scritto in C++, e nato dalla necessità di migliorare le prestazioni dei motori JavaScript, non efficientissimi, che spesso costringevano lo sviluppatore a modificare il codice per evitare operazioni troppo "pesanti". Google sviluppò questo motore puntando sulla velocità, ottenendo un buonissimo risultato. Le chiavi del successo di V8 si basano su tre punti chiave:

- *Fast Property Access*
- *Dynamic Machine Code Generation*
- *Efficient Garbage Collection*

Come già enunciato precedentemente, è un linguaggio dinamico, e le proprietà degli oggetti posso essere aggiunte e rimosse dinamicamente. Di conseguenza ottimizzare JavaScript risulta assai complicato. Rifacendomi all'esempio di Java, che, con il concetto di classe, permette al compilatore di conoscere esattamente il numero di campi e di proprietà di un determinato oggetto, generando codice ottimizzato di conseguenza. In JavaScript, non esistendo il concetto di classe, gli array sono formati da array associativi. Per ovviare a tale problematica, Google ha implementato nel motore V8 il concetto di Hidden Class. L'idea di base è quella di creare queste classi "nascoste" in modo da riuscire a generare codice ottimizzato. Per chiarire, utilizzerò un esempio pratico:

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}
```

Figura 2.4: Codice per le Hidenn class

Quando viene utilizzata una New Point() viene richiamata la prima volta, il motore V8 crea una *hidden class*, inizialmente vuota, supponiamo di chiamarla C0. Al primo assegnamento, viene creata una seconda classe, creato un offset per l'oggetto e la classe corrispondente al dato appena inserito. Il concetto è mostrato in figura:

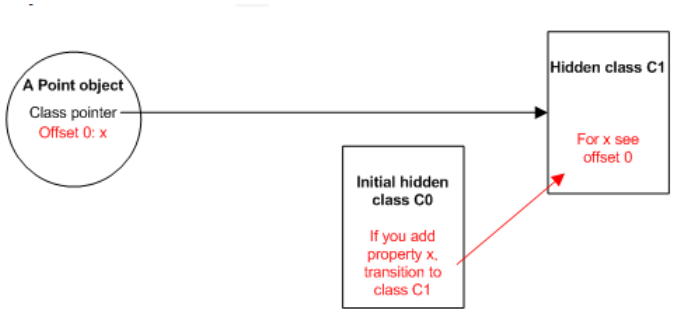

Figura 2.5: Creazione hidden class

Quando viene eseguito il secondo assegnamento this.y = y, viene stanziata una nuova *hidden class*, e definito un nuovo offset, come mostrato nell'immagine seguente:

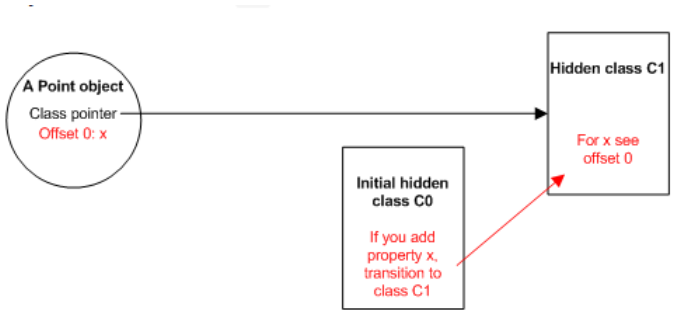

Figura 2.6: creazione seconda hidden class

Il fatto di creare classi nascoste ogni volta che vengono aggiunte proprietà può sembrare particolarmente inefficiente. Tuttavia è possibile il riutilizzo di tali classi.

Il motore progettato da Google, viene compilato in codice macchina quando è eseguito per prima volta, non utilizzando bytecode intermedi. l'*inline caching* è un segmento di codice macchina generato per accedere velocemente a una proprietà e associato una riga di codice specifica. V8 suppone che oggetti inclusi nella stessa riga, facciano parte delle stesse classi anonime. In caso la previsione non sia corretta, V8 dovrà correggere il codice. Un esempio di codice macchina associato a un accesso alla proprietà x di un oggetto p della classe Point, citata sopra, è presente nella seguente immagine.

```
# ebx = the point object
cmp [ebx,<hidden class offset>],<cached hidden class>
jne <inline cache miss>
mov eax,[ebx, <cached x offset>]
```

Figura 2.7: Codice associato all'accesso della proprietà x di un oggetto p della classe Point()

L'ultimo stratagemma adottato da V8 consiste nella gestione del *Garbage Collection*. V8 recupera memoria da oggetti non più necessari tramite un processo di garbage collection. Per assicurare pause brevi e evitare frammentazione di memoria, V8 prevede un ciclo di garbage collection "stop-the-world", quindi:

- Ferma l'esecuzione del programma durante tale ciclo.
- Processa solo parte della memoria heap, per non compromettere l'esecuzione dell'applicazione. V8 divide l'heap in 2 parti; la prima riservato agli oggetti della "young generation", la seconda per la "old generation". In un'applicazione molti oggetti sono usa e getta. Quando un oggetto sopravvive a un ciclo di garbage collection, viene spostato dalla young all'old generation, e i puntatori vengono aggiornati. Dato che la young generation viene processata meno volte, i processi di garbage collection risultano molto brevi.

Npm è il package manager ufficiale per Node.js. E un programma a linea di comando. Attraverso npm è possibile installare facilmente i pacchetti dell'npm registry. L'npm registry 'e il registro ufficiale dei moduli per Node.js. Chiunque vi può pubblicare una libreria. I moduli sono le librerie di Node che possono essere istallate. Sono organizzate in moduli, di più facile mantenimento.

Concludendo con Node, la facilità d'uso per la realizzazione di applicazioni può portare l'utente meno esperto a creare codice poco performante. Per esempio, gestire tutto in una callback, compromettendo la riusabilità e la modularità. Potremo, d'altra parte creare applicazioni con codice incomprensibile per persone diverse da noi. La libertà e la flessibilità architetturale è un limite se non si è molto pratici. Express facilita l'implementazione di applicativi.

2.6.4 Express.js

Express è un framework per applicazioni web Node.js flessibile e leggero per la definizione di *middleware* e l'utilizzo di *routing* che fornisce una serie di funzioni avanzate per le applicazioni web e per dispositivi mobili. Per Routing si intende determinare come un'applicazione risponde a una richiesta client a un endpoint particolare, il quale è un URI (o percorso) e un metodo di richiesta HTTP specifico.

```
// GET method route
app.get('/', function (req, res) {
  res.send('GET request to the homepage');
});

// POST method route
app.post('/', function (req, res) {
  res.send('POST request to the homepage');
});
```

Figura 2.8: Esempio di Routing

Come si può vedere dall'esempio sopra proposto, poso gestire le varie richieste HTTP tramite "app.get()" e "app.post()" Le funzioni middleware sono funzioni con accesso all'oggetto richiesta (req), all'oggetto risposta (res) e alla successiva funzione middleware nel ciclo richiesta-risposta dell'applicazione. In pratica, con Express, posso gestire le richieste e le risposte "in the middle", cioè fra server e client. Quando il server riceve una richiesta HTTP la racchiude all'interno di un oggetto ServerRequest. Questo oggetto, insieme all'oggetto ServerResponse, viene passato al primo middleware che ne può modificare il contenuto, o aggiungere proprietà. Una volta terminata la modifica, il middleware richiamerà il successivo nell'eventuale catena presente. Esistono diversi middleware messi a disposizione da Express; inoltre è possibile implementarne di ogni tipo.

2.6.5 MVC pattern

Express implementa il pattern MVC, nel quale appunto:

- Il model definisce la struttura dei dati, gestisce gli accessi e l'aggiornamento dei dati.
- Il controller, responsabile della gestione delle varie richieste. In Express, il middleware router è il controller, che interpreta le varie richieste HTTP e elabora le risposte.
- La View, rappresentata dalle varie pagine HTML presentate all'utente.

Express, fornisce una *template engine* per lo sviluppo della view dell'applicativo. Significa che è possibile definire pagine HTML statiche all'interno del progetto. I dati verranno modificati a run-time dall'engine.

2.6.6 Passport js

Ultimo componente di Node trattato, si tratta di un middleware che permette di implementare un sistema di autenticazione con username e pas-

sword, ma anche via Facebook,Twitter e Google. Ci sono tre metodi da configurare per poter utilizzare Passport:

- Utilizzando, in combinazione con express, e l'utilizzo di "passport.initialize()" e "passport.sesssion()".
- Configurare Passport con una delle strategie messe a disposizione dallo stesso, e settare le funzioni per la serializzazione degli utenti.
- Specificare nella route dell'applicazione la "passport.autenticate()", in modo da autenticare i vari user.

2.7 Database

2.7.1 MongoDB

Nel progetto, per il *data layer*, ovvero la gestione e il salvataggio dei dati, mi sono affidato a MongoDB. Fa parte della classe dei database noSQL, non basato quindi, su schemi relazionali. Il nome deriva da huMONGOus (gigante in inglese). MongoDb è un database "document-oriented storage", nel quale i dati vengono salvati in documenti, anziché in tabelle con colonne uguali e con campi uniformi, come si evince da [2]. Anche se all'interno delle *collections* è possibile salvare dati non omogenei, con MongoDB si ha la possibilità di definire delle regole di validazione dei documenti in fase di inserimento. I documenti comunque, sono codificati in base a standard, come JSON, o BSON, per esempio. Nel caso di di MongoDB, i *documents* sono salvati in BSON, un formato binario per il JSON. I documenti, a loro volta, sono organizzati in *collections*. Richiamando metodi su tali *collections*, è possibile inserire, modificare, interrogare il database. Per l'applicativo, ho utilizzato anche *Mongoose*, una libreria JavaScript permette la strutturazione di schemi che andranno salvati in MongoDB. In pratica, permette di definire modelli, che riassumono la struttura di una tipologia di dati, e successivamente utilizzati per istanziare oggetti su cui svolgere le classiche operazioni.

2.8 Pub/Sub pattern

Il patter Publish/Subscribe è un pattern che nasce come variante dell' *Observer Pattern*. L'Observer Pattern è un pattern nel quale un oggetto, detto soggetto, a cui è legata una lista di altri oggetti, che riceveranno notifica di un cambiamento di stato del soggetto stesso. Quando il soggetto non è più interessato a notificare un determinato osservatore, lo rimuove dalla lista. Tuttavia, nel mondo di JavaScript, a livello implementativo, l'Observer Pattern viene implementato in una variante: il **Publish/Subscribe Pattern**, come spiegato da David Osmani in [9]. L'obiettivo di tale variante è eliminare la dipendenza che vi è fra il soggetto e gli osservatori. Infatti viene usato un canale (basato su topic o su eventi che si interpone fra gli osservatori, o "subscribers" e il soggetto che ha generato l'evento, o "publisher". Questo permette agli osservatori di poter implementare il proprio gestore di eventi per registrare e gestire la notifica da parte del pusblisher.

Capitolo 3

WebRTC

Capitolo a parte per una parte fondamentale nella realizzazione del progetto. Verrà data una definizione di WebRTC e una breve panoramica sul suo funzionamento, a diversi livelli logici. Dal sito ufficiale di WebRTC abbiamo la seguente definizione.

> "WebRTC is a free, open project that provides browsers and mobile applications with Real-Time Communications (RTC) capabilities via simple APIs. The WebRTC components have been optimized to best serve this purpose."

Integrando tale definizione, la comunicazione avviene in p2p, direttamente fra gli utenti,senza utilizzo di un server (per il flusso di dati). Uno dei vantaggi nell'utilizzo di tale applicazione risiede nel fatto che non è necessario installare applicazioni terze o plug-in. Tutto avviene sul browser, non dovendo utilizzare applicazioni apposite, come per esempio Skype, Google Hangouts, o lo stesso Facebook, attualmente proprietari del mercato delle applicazioni che offrono servizi di video-chiamate in tempo reale. Essendo costruita direttamente sul browser, risulta anche più efficiente, rispetto a tecnologie già esistenti.

3.1 Struttura

Nonostante risulti semplice implementarlo tramite API JavaScript, opera a diversi livelli logici, e la sua architettura risulta alquanto complessa, utilizza diversi protocolli, a vari livelli di rete. Essendo l'architettura di tipo peer-to-peer, e, utilizzando UDP, possono sorgere diverse problematiche, analizzate in seguito. Una panoramica di tale tecnologia è mostrata nell'immagine sottostante.

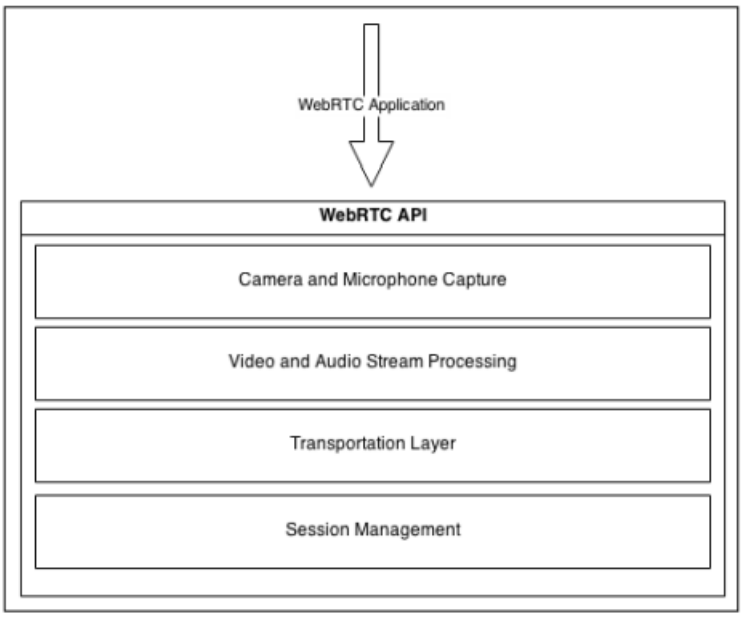

Figura 3.1: Funzionamento WebRTC

Come enunciato da [10] le quattro componenti fondamentali sono:

- **Camera and Microphone Capture**: Il primo passo è la gestione dei dispositivi multimediali. Significa cercare, se presenti, webcam e microfono, e ottenere i permessi di utilizzo da parte dell'utente.

- **Video and Audio Processing**: Rappresenta la gestione dello scambio di stream di dati fra gli utenti. Dato che i dispositivi possono essere

desktop e mobili, con codec [1] e risoluzioni supportate differenti, occorre gestire tale trasmissione. I vari parametri sono discussi in fase di creazione della sessione (discussa in seguito). WebRTC supporta l'utilizzo di vari codec, fra cui Opus, iSAC e VP8.

- **Trasport Layer**: WebRTC gestisce il trasporto dei pacchetti similmente al modo in cui il browser gestisce altri strati di trasporto, come AJAX o nel caso di utilizzo di WebSockets.
- **Session Management**: Livello che rappresenta l'inizializzazione e la gestione della sessione. Denominata *signaling*, verrà discussa successivamente.

3.2 API

A livello di implementazione, abbiamo tre API fondamentali per utilizzare WebRTC:

- MediaDevices
- RTCPeerConnection
- RTCDataChannel

3.2.1 MediaDevices

Con l'introduzione di HTML5, non è più necessario affidarsi a plug-in di terzi, quali Flash, Silverlight, per poter gestire i dispositivi quali microfono e videocamera. Ora è possibile accedere a diversi dispositivi tramite l'utilizzo di MediaDevices. Nello specifico, la funzione "MediaDevices.getUserMedia() permette di ottenere un stream di dati dalla webcam e dal microfono. Per ragioni di sicurezza, è necessario richiedere il permesso di utilizzo di tali dispositivi all'utente. Un esempio di utilizzo di tale funzione è mostrato qui:

[1] codec: Un codec è un programma che permette di codificare/decodificare digitalmente segnali audio/video.

```
var constraints = {
 video : true ,
 audio : true
}
function success(){
 //successo
}
function failure(){
 //qualcosa è andato storto
}
MediaDevices.getUserMedia(costraints,succes,failure);
```

I "costraints" rappresentano i vincoli per i quali viene richiamata la callback associata al buon esito. Possiamo avere vincoli obbligatori e accessori. Se quelli obbligatori non vengono inclusi, verrà richiamata la "failure" callback.

3.2.2 RTCPeerConnection

Questa libreria è utilizzata per gestire il flusso di dati scambiati fra i peer. Tuttavia, prima di poter stabilire una connessione è necessario comunicare informazioni di base, come l'indirizzo IP e i dati relativi alla gestione della sessione. Questa operazione preliminare è detta *signaling*. Un'idea generale è mostrata nella figura:

Figura 3.2: Signaling

Le istanze dell'applicativo si collegano al server, iniziando il processo di signaling, che si occupa di:

- Inizializzazione e controllo della sessione.
- Informazioni relative ai dati da scambiare (codec video e audio, banda disponibile.
- Messaggi di errore.
- Informazioni di rete, come, per esempio,l'IP dei peer in comunicazione, e i numeri di porta.

Queste informazione sono scambiati tramite SDP, un protocollo basato su stringhe e coppie chiave-valore. Tale standard si occupa solamente di descrivere i dati necessari per la gestione della sessione, non definisce un protocollo per il suo invio. La scelta del protocollo di scambio è lasciata al programmatore. Per l'applicazione presentata ho utilizzato le WebSocket. Il processo di signaling non è standardizzato, al fine di garantire compatibilità con tecnologie già esistenti.

3.2.3 ICE,TURN e STUN

ICE, STUN e TURN sono servizi necessari per l'inizializzazione e il mantenimento della connessione. Nella realtà, non è sempre immediato ottenere le informazioni necessarie per contattare un peer. Nello specifico abbiamo che, dato l'utilizzo di Netowrk Address Traslation. L'utilizzo di tale stratagemma dovuto della saturazione degli indirizzi IPv4. Ai dispositivi ai quali viene associata una NAT, avranno, all'interno della rete un unico indirizzo IP pubblico, mentre all'interno della rete su cui la NAT agisce, mantiene un indirizzo privato. Quando un dispositivo manda un pacchetto verso l'esterno, la NAT sovrascrive l'indirizzo privato con quello pubblico. D'altra parte, il NAT mantiene una tabella con gli indirizzi privati, in modo tale da instradare i pacchetti in entrata verso i giusti dispositivi. Invece, un host esterno

non può iniziare una connessione con un dispositivo dietro una NAT, dato che non vi è corrispondenza nella tabella. Per ovviare a questo problema si può utilizzare ICE, un framework che ha l'obiettivo di instaurare connessione dirette fra peer, aggirando eventuali NAT frapposte tra gli host. ICE tenta di trovare il percorso migliore per connettere gli host, utilizzando due servizi.

- Session Traversal Unit for NAT (STUN): Protocolle che permette di individuare eventuali NAT. Utilizzando un server STUN, forniti in generale da Mozilla e Google. Il server STUN, quando riceve una richiesta da un host (collegato a una NAT), risponde con l'indirizzo pubblico di quell'host (con indirizzo si intende IP e porta). Questo permette al peer di conoscere il suo indirizzo, necessario per stabilire la connessione diretta. STUN non funziona con le NAT simmetriche. Tale tipologia di NAT, associa un indirizzo pubblico per ogni richiesta in uscita da un host interno. Tale stratagemma garantisce in buon livello di isolamento della rete, interna, ma impedisce la connessione peer-to-peer.

- Nel caso il servizio STUN non fosse efficace, è possibile utilizzare un altro protocollo, chiamato Traversal Using Relays around NAT (TURN). In pratica vengono utilizzati TURN-server, che hanno indirizzi pubblici, per costruire una rete di relay, per reindirizzare e inoltrare il flusso audio/video ad un altro server TURN, che, a sua volta, lo inoltra all'host destinatario.

In figura il caso in cui il servizio STUN fallisse, e utilizzo, quindi, del protocollo TURN.

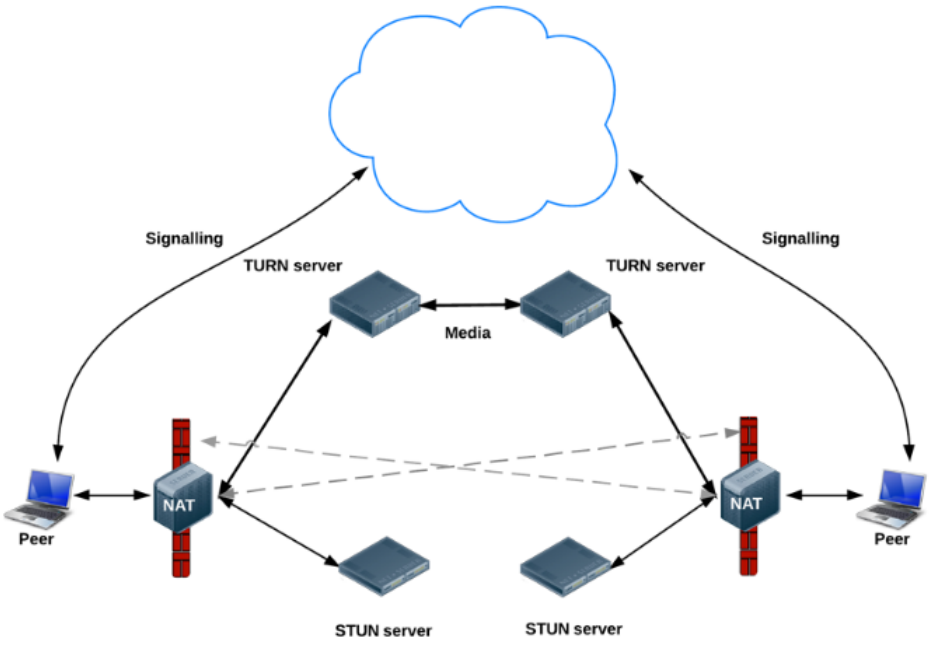

Figura 3.3: Funzionamento di TURN, nel caso del fallimento di STUN

Il protocollo TURN è da considerarsi l'"ultima spiaggia", dato l'elevato consumo di banda. In generale, per la maggior parte delle applicazioni, STUN è sufficiente.

3.2.4 RTCDataChannel

Libreria che rappresenta logicamente il flusso di dati bidirezionale fra i peer. Associato a un'istanza di RTCPeerConnection, mentre ogni connessione può avere svariati flussi associati.

```
var pc = new RTCPeerConnection();
var dc = pc.createDataChannel("my channel");
dc.onmessage = function (event) {
  console.log("received: " + event.data);
};
dc.onopen = function () {
```

```
  console.log("datachannel open");
};
dc.onclose = function () {
  console.log("datachannel close");
};
```

Le potenzialità di RTCDataChannel non si limitano solo allo scambio di stream audio/video. Tramite tale libreria possiamo garantire un trasferimento di dati diretto e a bassa latenza. Quindi le possibilità sono molte, fra le quali gaming, o applicazioni dektop remote.

3.2.5 Breve accenno alla Sicurezza

Come spiegato nel dettaglio in [3] WebRTC utilizza diversi meccanismi per la sicurezza, fra i quali:

- Il primo livello di sicurezza è intrinseco a WebRTC. Non vi è la necessità di installare plug-in o applicazioni terze.
- WebRTC richiede che l'utente autorizzi esplicitamente l'uso della sua webcam e microfono, evitando accessi non autorizzati.
- WebRTC utilizza DTLS, una "variante" di TLS specifica pertraffico UDP.
- Utilizzo di SRTP, criptando il flusso audio/video, e utilizzando meccanismi di autenticazione al fine di minimizzare rischi di intercettazioni.

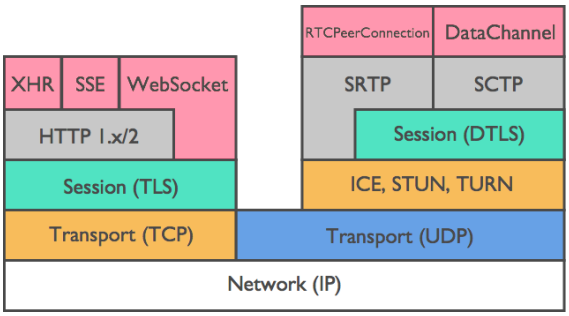

Figura 3.4: Panoramica sulla pila protocollare di WebRTC.

Capitolo 4

Analisi e Progettazione

L'obiettivo di questo capito è descrivere il problema che il progetto mira a risolvere. Descrivere il contesto in cui l'applicazione opera è fondamentale per una buona analisi dei requisiti e un'efficace progettazione.

4.1 Analisi dei requisiti

Come scritto nell'introduzione, l'applicazione ha come obiettivo "costruire" un prototipo di un applicativo che permettere a utenti di condividere interessi e passioni in determinati luoghi di interesse e, sulla base di tali interessi, offrire una comunicazione mediate video-chiamate in tempo reale.

Di seguito verranno analizzati i vari casi d'uso e verrà fornita un'analisi e progettazione dell'elaborato. La parte di implementazione effettiva sarà delegata a capitolo successivo.

4.1.1 Diagramma dei casi D'uso

Con il diagramma dei casi d'uso è possibile descrivere le funzionalità offerte dal software. Si tratta della rappresentazione di scenari nella quale viene descritta una sequenza di eventi e attori. Nel nostro caso la sequenza viene iniziata da un attore, nel nostro caso l'utente. Le entità utilizzate sono rappresentate da stickman.

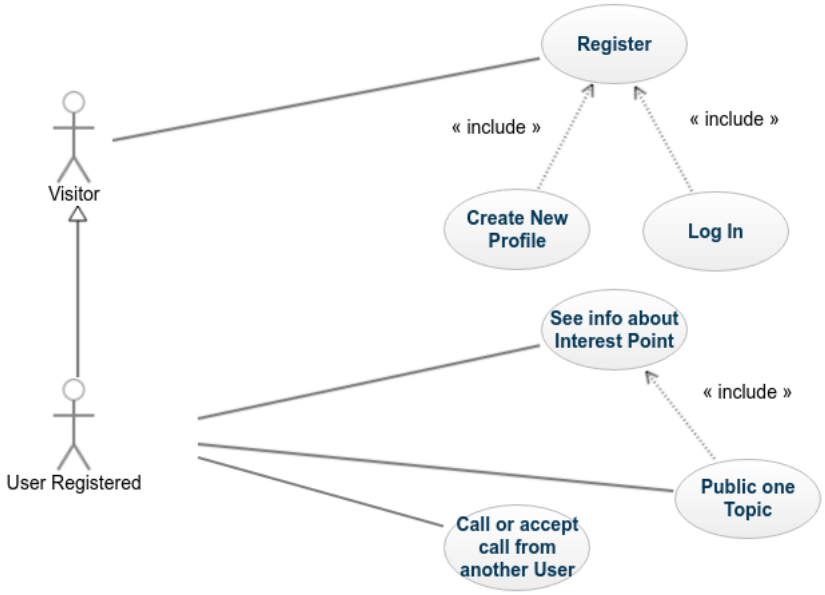

Figura 4.1: Diagrammi dei casi d'uso (solo per l'utente)

Nella figura 4.1 viene mostrato un diagramma semplificato dei casi d'uso del progetto. L'utente, viene differenziato in base alla registrazione all'interno dell'applicazione.

- Il visitatore potrà registrarsi inserendo username, email e password. Nel caso l'user si fosse già registrato, o tenti di utilizzare un username già presente nel sistema, verrà avvertito. In tal caso dovrà riprovare, con nome differente. L'applicazione permette, tramite Passport, la registrazione o autenticazione tramite alcuni social, tra i quali, per esempio, Facebook.

- Un utente non registrato non può compiere alcuna operazione, tranne quella di iscriversi.
- Una volta autenticato, all'utente verrà mostrata una mappa centrata sulla sua posizione attuale. In tale mappa, sono presenti dei marker, differenziati per il tipo del luogo di interesse.
- A quel punto, l'utente selezionerà uno dei punti di interessi presenti.
- Una volta selezionato, l'applicazione permette all'utente di pubblicare un topic relativo al punto di interesse selezionato.
- All'interno del topic, l'utente potrà inserire i propri interessi, e qualche riga di accompagnamento.
- Una volta pubblicato, verranno mostrati sulla mappa i topic di altre persone che ne hanno pubblicati associati allo stesso punto di interesse. Notare che tale operazione è possibile se e solo se si selezionato un punto di interesse fra quelli proposti.
- Selezionando uno dei topic, si accedono alle informazioni relative al topic, compreso l'utente che lo ha pubblicato. Una volta inserito il topic, un alert confermerà il buon fine dell'operazione. Il sistema inoltre, notificherà agli altri utenti che hanno pubblicato topic relativi al punto di interesse selezionato l'inserimento appena avvenuto.
- A questo punto, è possibile chiamare via WebRTC coloro che hanno pubblicato i topic visualizzati.

4.1.2 Diagramma di sequenza

Un Sequence Diagram viene disegnato per descrivere cosa accade durante un processo e mette in risalto le attività del programma. In questo caso, nel disegnare tale diagramma, tengo in considerazione solo il processo principale di visualizzazione dei punti di interesse e, successivamente, della pubblicazione di un topic.

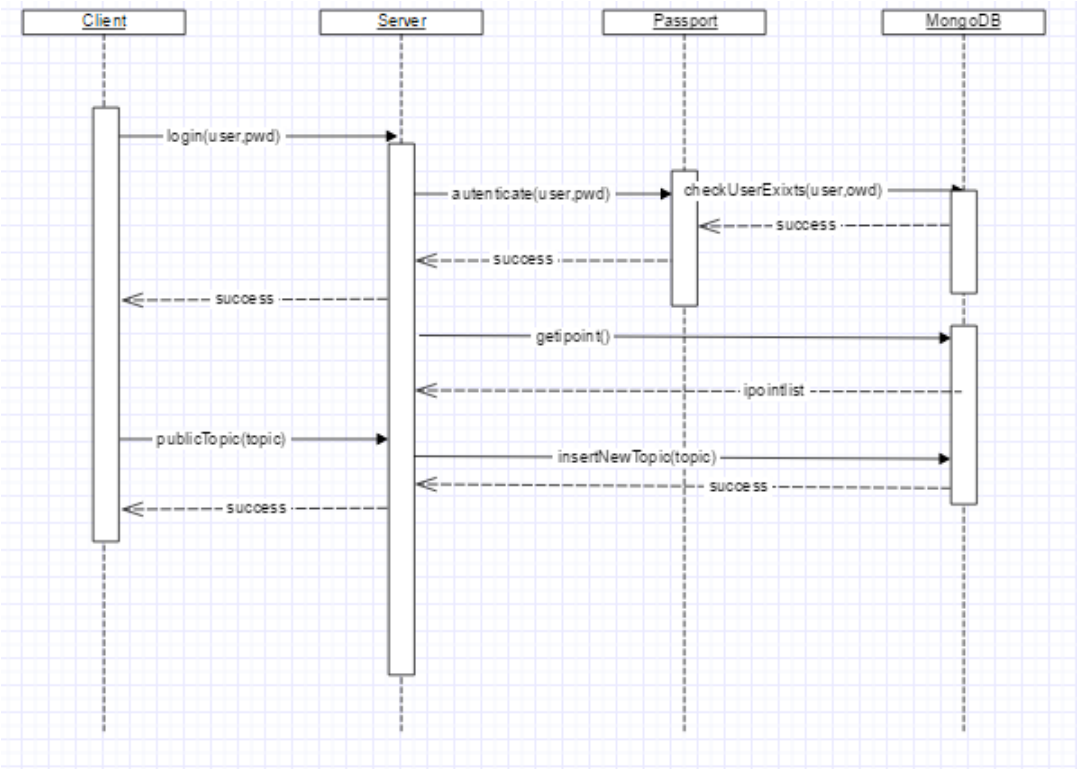

Figura 4.2: Diagramma di sequenza

4.1.3 Database

Il livello dati di dell'applicazione descrive la logica con cui i dati sono memorizzati, e di conseguenza come vengono gestiti dal sistema. Il modello più comune per descrivere tale logica è lo schema E/R (entity/relationship), utilizzato anche in questo caso, come si può notare in figura. In questo contesto

non verranno descritti gli attributi, evidenziando quindi solo i collegamenti fra le entità.

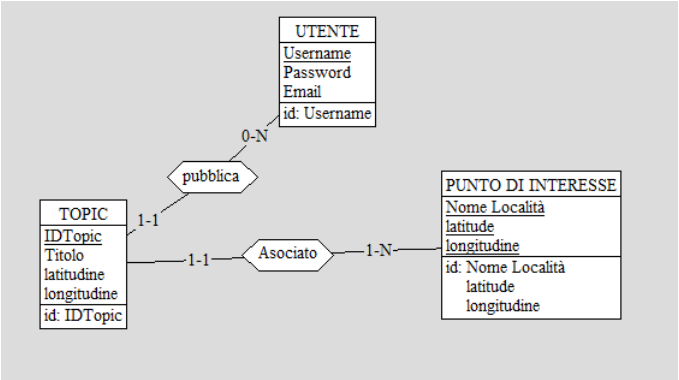

Figura 4.3: E/R per la rappresentazione dei dati

Descrizioni delle entità:

- **Utente**: Rappresenta gli utenti iscritti all'applicazione. Gli attributi principali sono i classici username e password e email. Come chiave primaria abbiamo l'username, univoco. Un utente può pubblicare un topic, scegliendo un punto di interesse, inserendone un titolo e un interesse disponibile da una lista. Un utente può pubblicare quanti topic desidera.

- **Punto di interesse**: Utilizzato per memorizzare i punti di interessi visibili dagli utenti, cioè mostrati dalla mappa. Come chiave primaria hanno il nome e la posizione, espressa in coordinate. In un determinato punto posso essere pubblicati diversi topic.

- **Topic**: Rappresenta i topic pubblicati dagli utenti per la condivisione degli interessi. Ha come chiave primaria un ID univoco. Ogni topic pubblicato è associato ad un solo punto di interesse, ed è pubblicato da un solo utente.

Capitolo 5

Implementazione

In questo capitolo, verranno affrontate le parti salienti dello sviluppo dell'applicativo. L'attenzione sarà concentrata sulla pagina principale, ovvero quella contenente la mappa. Possiamo infatti definire tale applicazione come una Single Page App (SAP). Una SPA e un'applicazione web che non ricarica la pagina durante il suo utilizzo e che puo comunicare in maniera asincrona con il server.

5.1 Struttura dell'applicativo

La struttura dell'applicazione è del tipo client-server. Tramite l'utilizzo di Express, quindi, definendo una "route", possiamo utilizzare il pattern MVC, individuando quindi:

- la View è l'insieme di pagine HTML che vengono mostrare all'utente.
- il Model rappresenta la logica con la quale i dati sono rappresentati, quindi definizione di utenti, topic e punti di interesse.
- infine, il Controller, che si occuperà di gestire le varie richieste HTTP dal client, prelevare il dati dal model, e restituirli al client.

Un'idea della struttura può essere data dalla figura sottostante:

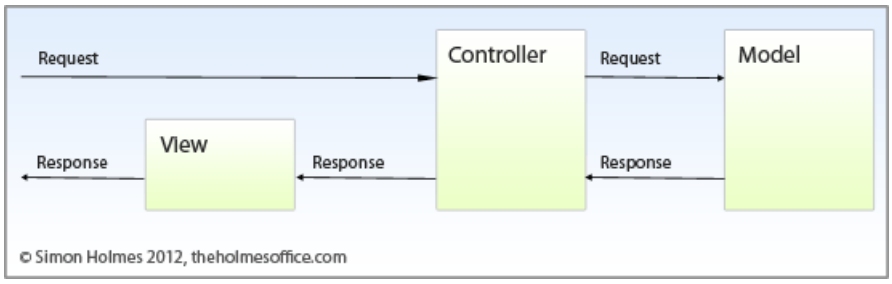

Figura 5.1: MVC utilizzando Express js

5.2 Server

Per poter descrivere le varie funzionalità dell'applicativo, è necessario suddividere i componenti in "moduli", descrivendone le funzionalità e indicandone le caratteristiche. Abbiamo quindi il server che, per chiarezza, è descritto suddiviso in varie componenti con un ruolo specifico.

Figura 5.2: Schema semplificato del server

5.3 Autenticazione

L'applicazione si apre con la richiesta di registrazione da parte dell'utente. Come già enunciato nel capitolo precedente, il meccanismo di registrazione/autenticazione è gestito da Passport js.

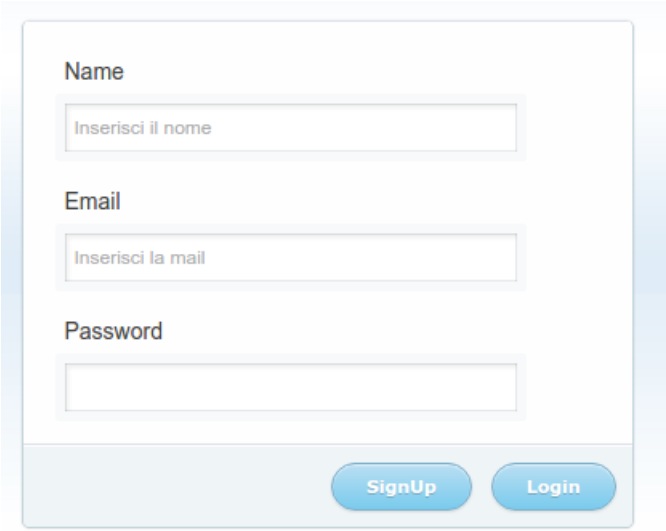

Figura 5.3: Autenticazione

L'autenticazione avviene richiamando la **passport.autenticate()**, che ha in ingresso come parametro una *strategy*. Il codice per la definizione di una strategia è rappresentato nella figura sottostante, dove è definito solo per il caso di registrazione.

```
var LocalStrategy    = require('passport-local').Strategy;
passport.use('signup', new LocalStrategy({
   usernameField : 'email',
   passReqToCallback : true
},
function(req, email, password, done) {

   process.nextTick(function() {

       if (!req.user) {
           User.findOne({ 'user.email' :  email }, function(err, user) {
             if (err){ return done(err);}
               if (user) {
                   return done(null, false, req.flash('signuperror', 'User already exists'));
               } else {
                   var newUser            = new User();
 newUser.user.username    = req.body.username;
                   newUser.user.email    = email;
                   newUser.user.password = newUser.generateHash(password);
 newUser.user.name  = ''
 newUser.user.address = ''
                   newUser.save(function(err) {
                       if (err)
                           throw err;
                       return done(null, newUser);
                   });
               }
         });

```

Figura 5.4: Definizione della strategy di registrazione utente

Di default, se l'autenticazione fallisce, non è prevista nessuna callback di errore, ma verrebbe semplicemente restituito un errore 401, "Unauthorized Status". Nel caso opposto, viene imposto come utente autenticato della sessione l'user della request. Esistono diverse strategie incluse di default nel modulo, incluse quelle necessarie all'autenticazione via Facebook e Twitter. Inoltre, in alcuni casi è possibile salvare le informazioni degli utenti.

5.4 Logica dell'Applicazione

5.4.1 Routing

Nello specifico, viene utilizzata una tecnica, definita routing, tramite la quale gestiamo le varie richieste HTTP. Per Routing si intende determinare come un'applicazione risponde a una richiesta client a un endpoint particolare. La route ha la seguente struttura:

```
app.METHOD(PATH, HANDLER);
```

Dove app è un'istanza di express, METHOD rappresenta il tipo di richiesta. METHOD è il metodo di richiesta HTTP. PATH è un percorso sul server. HANDLER è la funzione eseguita quando si trova una corrispondenza per la route.

Un esempio lo possiamo vedere nel codice sottostante, il cui obiettivo è gestire una richiesta di tipo POST, rispondendo con una stringa.

```
app.post('/', function (req, res) {
  res.send('Got a POST request');
});
```

Una volta effettuata l'autenticazione, al client verranno mostrati i vari punti di interesse, contenuti nel database, che verranno richiamati tramite query. Le due entità fondamentali sono:

- Punti di interesse
- Topic

Utilizzando mongoose, posso rappresentare tali entità nei seguenti modi.

```
var mongoose = require('mongoose');
var bcrypt   = require('bcrypt-nodejs');
var userSchema = mongoose.Schema({
    user              : {
```

```
        username      :String,
        email         : String,
        password      : String,
        name          : String,
        address       : String,
    }
});

userSchema.methods.updateUser = function(request, response){
        this.user.name = request.body.name;
        this.user.address = request.body.address;
         this.user.save();
        response.redirect('/user');
};
```

Inoltre, al client viene data la possibilità di inserire dei topic. La struttura di tali topic è del seguente tipo:

```
var mongoose = require('mongoose');
var GeoJSON = require('mongoose-geojson-schema');
var topicSchema = mongoose.Schema({

  publisher     : { type: String, ref: 'User' },
  title         :String,
  interest      : String,
  iPointAssociated  :{type: String, ref:'IPoint'},
  loc :  { type: {type:String}, coordinates: [Number]}
});
topicSchema.index({loc: '2dsphere'});
module.exports = mongoose.model('Topic', topicSchema);
```

Gli attributi "publisher" e "iPointAssociated" sono attributi che fanno riferimento alle altre due entità, rispettivamente al punto di interesse associato

a quel topic e all'utente che lo ha pubblicato. Infine, l'indice "2dsphere" è indicato per definire la posizione.

5.5 Gestione persistenza dei dati

5.5.1 Query

Le query sono messe a disposizione dello stesso MongoDB. La sintassi utilizzata diverge dai classici database relazionali, data appunto la natura "NoSQL" di MongoDB. Il metodo principale per la ricerca all'interno di MongoDB è la **db.collection.find()**, tramite la quale si interrogano le varie collezioni di documenti.

```
var myCursor = db.collection.find({Username:"eddy"});
```

Tramite il costrutto next posso ricercare all'interno del database in base alla posizione. Nel dettaglio, inserendo come parametri la latitudine, longitudine e una distanza massima posso ricercare elementi che siamo vicini alla posizione descritta da tali coordinate entro il raggio definito dalla distanza. La query di ricerca dei topic "vicini" utilizzata nel progetto è del tipo:

```
Topic.find({
      "loc": {
        "$near": {
          "$geometry": {
            "type": "Point",
            "coordinates": coordinates,
          },
          "$maxDistance": 1000
        }
      }
    }
```

La distanza indicata è in metri, le coordinate sono salvate in un array (Prima la latitudine, poi la longitudine).

5.6 Signaling

Il processo di signaling è già stato trattato a livello teorico nei capitoli precedenti. In questo contesto, sarà analizzato a livello di codice. Riassumendo brevemente il processo di signaling è sostanzialmente lo scambio di informazioni necessarie per la comunicazione diretta fra i peer, quali, per esempio, indirizzo IP e porta. A livello di codice, è necessario creare un sistema che permetta lo scambio di dati necessari per la sessione, gestire l'apertura e la chiusura della connessione, trovare gli "ICE candidates".

Per la comunicazione utilizziamo sempre il modulo Socket.IO, modulo di Node js che permette una comunicazione full-duplex fra client e server, basata su TCP. Data la sua natura asincrona, fino alla diffusione di Node js, il suo utilizzo è risultato molto limitato.

```
wss.on("connection", function (connection) {
    connection.on("message", function (message) {
        var data;
        try {
            data = JSON.parse(message);
        } catch (e) {
            console.log("Error parsing JSON");
            data = {};
        }
}
```

Il codice appena mostrato ha come scopo quello di gestire la ricezione di un messaggio lato server. Come formato di dati viene utilizzato il JSON, dato che permette la rappresentazione di dati strutturati in maniera comprensibile. Per poter trovare l'indirizzo IP e porta del peer da contattare, nel progetto presentato vengono utilizzati server di Google dagli indirizzi IP pubblici, come mostrato dal codice sottostante.

```
var configuration = {
    "iceServers": [{ "url": "stun:stun.1.google.com:19302"
```

```
}]
  };
```

Per quanto riguarda la sessione, vengono utilizzati i meccanismi di domanda e offerta. Il peer che vuole iniziare la connessione manda un'"offer" al server di signaling, che si occuperà di mandare l'offerta all'user da contrattare, delegando la gestione della connessione ai peer oggetto della connessione. Stesso concetto per l'"answer", che rappresenta la risposta dell'host destinatario.

```
  case "offer":
                console.log("Sending offer to", data.name);
                if (conn != null) {
                    connection.otherName = data.name;
                    sendTo(conn, {
                        type: "offer",
                        offer: data.offer,
                        name: connection.name
                    });
                }
                break;
   case "answer":
                var conn = users[data.name];
                if (conn != null) {
                    connection.otherName = data.name;
                    sendTo(conn, {
                        type: "answer",
                        answer: data.answer
                    });
                }
```

5.7 Publish/Subscribe

Con MongoDB abbiamo la possibilità di simulare un meccanismo di publish/subscribe, anche se non è previsto nativamente. Tale stratagemma è possibile utilizzando le *Capped Collections*, collezioni di documenti memorizzate in maniera circolare e di dimensione fissata. Il fatto che si memorizzata circolarmente, significa che, una volta terminato lo spazio allocato per tale collezione, a fronte di nuovi inserimenti, vengono eliminati i documenti più "vecchi", senza comandi espliciti. Generalmente, con questo tipo di collezioni, vengono utilizzati i *Tailable Cursor*. Dal manuale di riferimento abbiamo che:

> "By default, MongoDB will automatically close a cursor when the client has exhausted all results in the cursor. However, for capped collections you may use a Tailable Cursor that remains open after the client exhausts the results in the initial cursor. "

Si possono considerare equivalenti al comando "Tail -f" in ambienti Linux o Unix-like, dato che, anche dopo che i client esauriscono documenti in una collezione, il cursore continuerà a referenziare i documenti. Utilizzando tale tipo di cursore, avremo l'impressione che il cursore sia in ascolto di dati. Questo processo ricorda il long-polling con HTTP, nel quale si inverte il flusso di dati, essendo il server che mantiene la comunicazione aperta e invia dati al determinarsi di un evento. Questa soluzione risulta più pratica rispetto a interrogare periodicamente il database.

5.8 Struttura del Client

Per quanto riguarda il client, non abbiamo una struttura modulare come per il server. Il front-end dell'applicazione si limita a un insieme do pagine HTML che vengono visualizzate da parte dell'applicativo. In figura viene mostrata la pagina principale visualizzata dall'utente dopo che l'autenticazione è andata a buon fine:

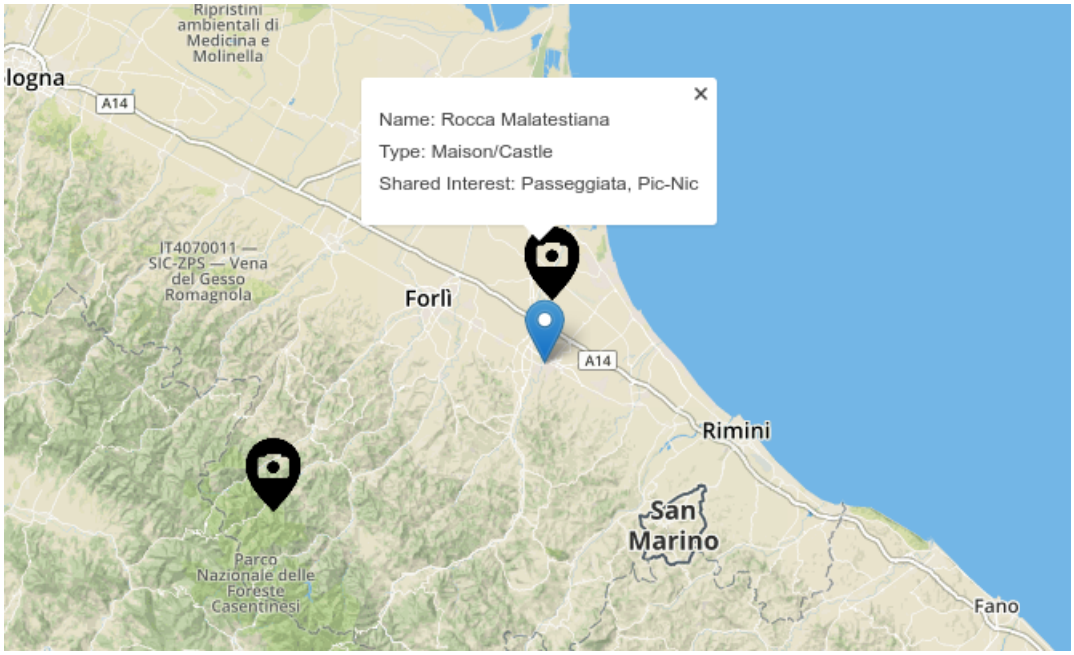

Figura 5.5: Screenshot relativo ai punti di interesse

Nella mappa, realizzata con Leaflet, è mostrata la posizione dell'utente, tramite le API messe a disposizione da HTML5. I marker neri, invece sono relativi ai punti di interesse quale, per esempio *"Rocca Malatestiana"*. Cliccando con il marker in corrispondenza di tali punti di interesse, viene visualizzato un popup contenente le informazioni relativa al punto di interesse, quali nome, tipo di località e gli interessi condivisi recentemente da altri utenti.

Una volta selezionato il punto di interesse è possibile pubblicare un topic, contenente, un titolo, eventualmente un testo, e una lista di interessi fra cui scegliere, quali, per esempio, *escursione* o *bicicletta*. La figura sottostante rappresenta il form che permette tale inserimento.

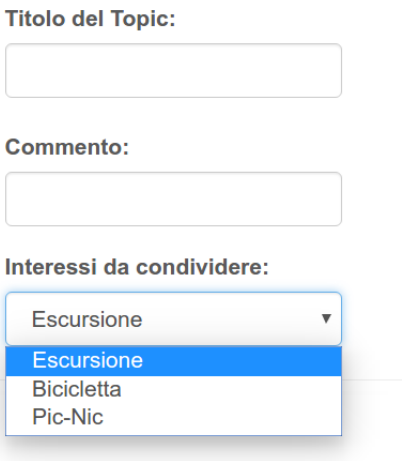

Figura 5.6: From di pubblicazione topic

Una volta pubblicato il topic, sulla mappa appariranno i topic degli altri utenti, come mostrato in figura 5.7

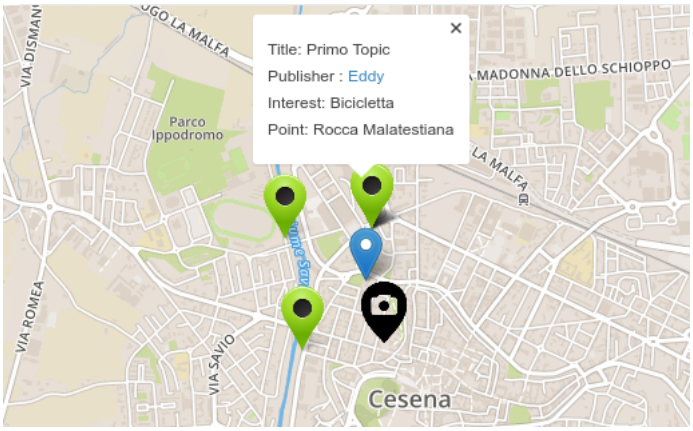

Figura 5.7: Vista sui Topic

Inoltre è possibile chiamare un utente che ha pubblicato un topic, selezionandolo dal popup.

Conclusioni

Nel corso di questa dissertazione di laurea ho descritto il funzionamento di un prototipo di un'applicazione per la condivisione di interessi, basata su geo-localizzazione e WebRTC, con una componente Publish/Subscribe. All'inizio, è stata fornita una panoramica del problema nel quale il progetto lavora, e, successivamente,sono stati illustrati prima i vari componenti utilizzati nella realizzazione di tale applicativo, spiegandone prima il funzionamento generale e, i metodi tramite i quali vengono utilizzati all'interno dell'applicazione poi.
Viene inoltre fornita una panoramica dettagliata sulla fasi di analisi e di progettazione, individuando i possibili casi d'uso degli utenti che intendono utilizzare tale applicazione, e descrivendo il funzionamento dell'applicativo a livello logico. L'ultima sezione è dedicata all'implementazione, con esempi di codice e di utilizzo dei vari elementi descritti nei capitoli precedenti.

L'applicazione ha raggiunto l'obiettivo preposto, tuttavia non è completa e sicuramente vi è la possibilità di completarla e di aggiungere nuove funzionalità, ma la sua modularità semplifica di certo tale compito.

Per quanto riguarda gli sviluppi futuri, le possibilità sono infinite, dall'aggiungere servizi per l'utente, quali, solo per citarne alcuni, la possibilità di creare gruppi di interesse comune o l'inserimento nella mappa anche di servizi quale ristorazione,noleggio e molto altro, nonché di integrare quelli esistenti. Sicuramente tale tipo di applicazioni in contesti mobili apre a moltissime opportunità, soprattutto dal punto di vista della condivisione di determinati interessi.

Bibliografia

[1] Tim Berners-Lee. Long live the web. 2010.

[2] Kristina Chodorow. *MongoDB: The Definitive Guide*. O'Reilly Media.

[3] NTT Communications. A study of webrtc security, 2011.

[4] Cosmobile. A cosa servono le applicazioni web, 2014.

[5] Douglas Crockford. Prototypal inheritance in javascript, 2008.

[6] Paolo Davoli. Didattica 2.0: metodologie e tecnologie web 2.0, 2012.

[7] Mozilla Develepers Newtork. Geolocation api, 2015.

[8] Trevor Norris. Understanding the node.js event loop, 2015.

[9] Addy Osmani. Understanding the publish/subscribe pattern for greater javascript scalability, 2011.

[10] Dan Ristic. *Learning WebRTC*. PACKT publishing.

[11] Seth Thompson. V8 developer documentation, 2012.

[12] Wikipedia. Javascript, 2002.

[13] Wikipedia. 3-tier architecture, 2006.

[14] Wikipedia. Ajax, 2010.

[15] Wikipedia. Web application framework, 2013.

Ringraziamenti

In primis vorrei ringraziare il prof. Stefano Ferretti per la fiducia accordatami accettando il ruolo di Relatore. Un doveroso ringraziamento va ovviamente alla mia famiglia, senza la quale non avrei mai neppur cominciato questa carriera: è per merito loro se sono arrivato fino a qui. Tutti i miei amici hanno avuto un peso (più o meno determinante) nel conseguimento di questo risultato, che è punto d'arrivo e contemporaneamente di ripartenza per la mia vita. E grazie a tutti!