Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

Progettazione ed implementazione di un database per le gestione della mappa della connettività urbana utilizzando tecnologie NoSQL

Relatore:
Chiar.mo Prof.
Marco Di Felice

Presentata da:
Marin Kola

Sessione II
Anno Accademico 2014/2015

Alla mia famiglia, amici e colleghi ...

Indice

Elenco delle figure

Introduzione

Lo sviluppo tecnologico ha causato una vera e propria esplosione di dati. Questo fenomeno risulta in continua crescita e sembra mantenersi anche per il futuro. I dati vengono generati in maniera massiva da sorgenti come: sensori RFID che raccolgono i dati del traffico, sensori per la raccolta di informazioni sul meteo, post sui social network, video e immagini digitali, dati GPS raccolti attraverso dispositivi mobili, registrazione online delle transazioni di acquisto, e da qualsiasi altra fonte che può produrre informazioni di nostro interesse. Tutti questi dati, chiamati anche Big Data, combinati con sofisticate analisi di business attraverso le tecnologie appropriate, danno la possibilità alle imprese di prendere decisioni strategiche, velocemente ed efficacemente, in base ai comportamenti dei clienti e alle condizioni del mercato.

I Big Data offrono vantaggi importanti ma riportano alcune problematiche. Infatti, nella maggior parte dei casi, questi dati non sono strutturati, quindi non sono facilmente riconducibili a modelli di dati basati sui RDBMS tradizionali. Questo perché cambiare in corsa una determinata tabella, magari introducendo nuovi attributi, è problematico e, spesso, sconsigliato proprio per evitare problemi di corruzione al DBMS. Questa limitazione rende i DBMS relazionali poco efficaci alle trasformazioni dei Big Data.

La nascita e il recente sviluppo dei DBMS NoSQL si sono rivelati di grande importanza, poiché hanno permesso le gestione efficace di grandi quantità di dati, grazie a database flessibili e soprattutto scalabili che garantiscono velocità di elaborazione di terabyte di dati, anche non strutturati. Esistono diverse tipologie di DBMS NoSQL, ciascuna delle quali presenta caratteristiche differenti. La scelta su quale DBMS utilizzare va fatta accuratamente del progettista software in base alle caratteristiche del sistema da gestire.

Nella tesi viene presentata un introduzione sui Big Data e le tecnologie utilizzate per la loro gestione, descrivendo nel dettaglio il DBMS MongoDB, il quale è stato anche utilizzato come supporto di base di dati per un'applicazione mobile che scannerizza le reti wifi e, interfacciato con il Framework Node.JS per rappresentare i dati.

La tesi ha la seguente struttura:

- Nel primo capitolo viene introdotto il tema dei Big Data, descrivendo le caratteristiche che caratterizzano tali dati, il loro utilizzo, la provenienza e le opportunità che possono apportare
- Nel secondo capitolo vengono introdotte le tecnologie NoSQL utilizzate per i Big Data, descrivendo le tipologie esistenti. Infine viene fatto un confronto tra queste nuove tecnologie e i DBMS relazionali

- Nel terzo capitolo viene descritto nel dettaglio il database MongoDB. In particolare vengono rappresentate le caratteristiche fondamentali e la shell
- Nel quarto capitolo viene descritta la progettazione e l'implementazione del database come supporto applicazione per scansione reti Wifi attraverso il DBMS MongoDB e la rappresentazione dei dati tramite Node.JS

Capitolo 1

Introduzione ai Big Data

Il termine "Big" Data[27][29], sicuramente abusato nelle moderne strategie di mercato, indica grandi aggregazioni di dati che non possono essere processati e analizzati con i tradizionali processi e strumenti di analisi. I Big Data aprono le porte verso nuovi modi di percepire il mondo e portano vantaggi enormi in termini di processi decisionali. I Big Data possono essere ricavati da una molteplicità di fonti, e ogni giorno sempre più applicazioni vengono create per trarne valore e arricchire la vita personale e professionale degli utenti.

Figura 1.1: Big Data

I nostri pensieri, sentimenti, opinioni politiche e religiose lasciano traccia nei social media a cui partecipiamo, nelle ricerche che facciamo nei motori di ricerca, nei post su facebook, nei tweet, cosi come i nostri stili di vita lasciano traccia nei record dei nostri acquisti, i nostri movimenti lasciano traccia nelle traiettorie disegnate dai nostri dispositivi mobili e dai sistemi di navigazione delle nostre auto. Anche le nostre relazioni sociali lasciano traccia nella rete dei nostri contatti telefonici, delle mail e nei link di amicizia del nostro social network preferito.
Da un lato, alcuni fattori, quali la crescita dei dati scientifici, hanno fortemente contribuito all'accelerazione della produzione di questi dati. Dall'altro, questa crescita esponenziale è il risultato di alcuni mutamenti sociali ed economici estremamente positivi avvenuti nella nostra società. Si consideri, per esempio, la rapida diffusione dei dispositivi mobili, ricchi di contenuti multimediali e compatibili con i sistemi GPS, e dei social network, che hanno consentito a miliardi di persone in tutto il mondo di tenersi in contatto in modo digitale.
Questo grande volume di dati può essere utilizzato in vari settori e non solo in campo economico-finanziario, per esempio, possono essere utilizzati per rilevare le frodi prima che i risarcimenti vengano corrisposti. Attraverso il loro studio è possibile decodificare il DNA umano in poco tempo, prevedere il comportamento umano, sventare attacchi terroristici, prevenire le malattie e anticipare eventi naturali.

1.1 Caratteristiche principali dei Big Data

Il termine Big Data non descrive solo la dimensione dei dati, ma permette di identificare spunti a nuovi ed emergenti tipi di dati e contenuti per rispondere alle domande che prima erano considerati al di là della nostra portata. I Big Data sono disponibili in grandi volumi, sono presenti con formati non strutturati e caratteristiche eterogenee e spesso sono prodotti con estrema velocità. Volume, Varietà e Velocità[27][08] sono i fattori che identificano questi dati.

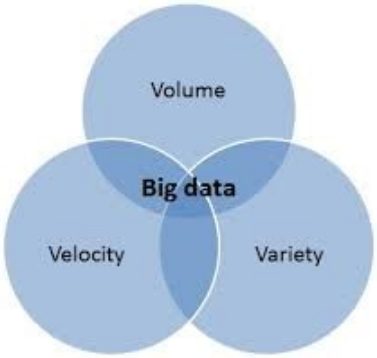

Figura 1.2: Volume, Velocità e Varietà

- **Volume**
  Uno degli aspetti che caratterizzano i Big Data è la loro quantità, come dice il nome stesso. Basti pensare che il 90% dei dati di tutto il mondo è stato generato negli ultimi anni. Le organizzazioni oggi sono sopraffatte con volumi di dati, siccome è facile accumulare terabyte, ma anche petabyte di informazioni di tutti i tipi, alcuni dei quali hanno bisogno di essere organizzati, verificati ed analizzati. È possibile immagazzinare i dati all'interno di un RDBMS tradizionale, sostenendo però costi molto elevati sia per i dispositivi di archiviazione, sia per ottenere la capacità di calcolo necessaria per elaborare i dati. Nondimeno, gli investimenti potrebbero rivelarsi non giustificabili dai risultati ottenuti in termini di performance. È per questo motivo che grandi aziende come Google e Facebook adottano strumenti diversi dagli RDBMS tradizionali per l'archiviazione e l'analisi delle grandissime quantità di dati di cui dispongono.

- **Varietà**
  La diversità dei formati e l'assenza di una struttura sono la seconda caratteristica dei Big Data. In prima istanza si possono avere dati generati da diverse fonti, interne o esterne. Questi possono avere diversi formati (testo, video, immagini, audio ecc.) riconducibili a tre categorie: dati strutturati, semi-strutturati e non strutturati. I dati strutturati sono quei dati che rispettano regole predefinite quali, tipo di contenuto, lunghezza, formato. Questa tipologia di dati è di semplice archiviazione, interpretazione e categorizzazione, tipicamente nei database relazionali. I dati semi-strutturati, invece, non sono conformi al modello di dati di un tipico database ma possono essere divisi in

records più o meno strutturati, infine, i dati non strutturati non seguono in nessun modo gli schemi di un tradizionale database. Per il salvataggio di dati semi-strutturati e non strutturati, la scelta ricade spesso sui database NoSQL, che forniscono i meccanismi adatti a organizzare i dati ma non impongono una rigidità dello schema consentendo di adattarsi alla variabilità dei dati, cosa che non avviene per i database relazionali.

- **Velocità**
  La velocità è il terzo fattore che identifica i Big Data, che definisce la rapidità con cui i dati si generano, si raccolgono, si aggirano e si elaborano. Per le aziende la produttività di dati a grande velocità, impone di sfruttarli con altrettante rapidità, attingendo da essi le informazioni utili per il business e minimizzando i tempi di elaborazione. L'esigenza di ottenere tempi rapidi di risposta ha favorito lo sviluppo di database non relazionali. La definizione di Big Data è stata successivamente arricchita con i concetti di Veridicità e Valore. La prima si riferisce alla correttezza e l'affidabilità dei dati, invece, la seconda si indica la capacità dei dati di portare un reale beneficio nel processo di analisi.

1.2 Big Data come fonte di valore

I fattori[27][07] capaci di dare valore ai Big Data sono:

- **Supporto alle decisioni**: le decisioni prese con il supporto di strumenti analitici si stanno affermando sempre di più con il passare del tempo. Fino ad ora queste tecniche erano diffuse solo nelle grandi organizzazioni, anche per il costo elevato dei sistemi e degli strumenti informatici per l'elaborazione, l'archiviazione e l'analisi dei dati. Le attuali evoluzioni tecnologiche, come il diffondersi di strumenti di analisi open source, rendono molto meno costoso e più flessibile l'adozione di queste tecniche anche da parte delle organizzazioni medio-piccole.

- **Innovazioni**: la possibilità tramite la raccolta, l'elaborazione e l'analisi dei dati consentono alle organizzazioni di innovare i prodotti, i servizi, i processi e i modelli di business per cogliere le sfide di un ambiente in continuo cambiamento.

- **Migliorare le prestazioni**: tutte le organizzazioni, sia pubbliche che private, analizzando ed elaborando accuratamente i dati e le informazioni disponibili possono migliorare i loro processi e le loro prestazioni. Una attenta elaborazione dei dati relativa alle vendite può incrementare in modo significativo la capacità di segmentare l'offerta e di personalizzarla sulle specifiche esigenze dei clienti. Analizzare i dati sul venduto può dare origine ad una più precisa previsione dell'andamento delle vendite future con evidenti vantaggi nella gestione logistica. Nel settore pubblico, il processo di digitalizzazione in atto nell'area della salute, può consentire oltre ad una accurata analisi dell'efficienza ed efficacia delle strutture sanitarie, una disponibilità di informazioni cliniche che contribuisce ad aumentare la qualità e la personalizzazione dei

trattamenti sanitari. Sempre nel settore pubblico, un'analisi delle tendenze e dei dati relativi al mercato del lavoro può aiutare in modo decisivo la definizione di politiche attive di supporto ai non-occupati e l'impostazione di adeguati processi formativi.

- **Segmentazione dei Clienti**: i Big Data consentono alle organizzazioni di creare segmentazioni altamente specifiche e di adattare i prodotti e i servizi alle esigenze dei consumatori.

Solo attraverso soluzioni tecnologiche nuove è possibile gestire questa grande complessità e quantità di dati. È stata sviluppata e adattata un'ampia varietà di tecniche e tecnologie per manipolare, analizzare e visualizzare i Big Data. L'insieme di tecniche e tecnologie attingono da settori diversi, quali la statistica, l'informatica, la matematica e l'economia: questo significa che un'azienda, la quale intende trarre valore dai Big Data, dovrebbe adottare un approccio flessibile e multidisciplinare.

1.3 Limiti dei Big Data

I Big Data non racchiudono in se solo caratteristiche positive e grandi opportunità: essi infatti presentano alcuni aspetti critici[27] che potrebbero invalidare i vantaggi.

1.3.1 La qualità

Il processo che sta alla base della qualità dei dati è che se si basano le analisi e, di conseguenza, i nostri processi decisionali su dati di scarsa qualità, allora anche il risultato sarà di scarsa qualità. Nel mondo dei Big Data, si possono distinguere tre tipologie di dati, nei confronti dei quali vi sono problematiche differenti riguardo alla qualità:

- **Dati Provenienti dai sistemi operazionali**: sono i casi in cui i sistemi operazionali producono una vasta quantità di dati. Questi sistemi riguardano, per esempio, il mondo della finanza o le grandi distribuzioni. In questi casi i problemi di qualità sono conosciuti e vi sono a disposizione molti strumenti per il controllo e la pulizia dei dati. Tali strumenti possono essere utilizzati anche quando il volume dei dati è molto elevato

- **Dati provenienti da sensori, RFID e strumenti scientifici**: questi dati vengono generati automaticamente da macchine, quindi non sono soggetti a errori di immissione, però possono presentare problemi di qualità dovuti a difetti dei sensori o negli strumenti di misura

- **Dati provenienti dal Web**: nel caso dei dati provenienti, per esempio, dai social network, essi si presentano in un formato semi-strutturato: i metadati, che in genere costituiscono la porzione strutturata, sono più affidabili e completi, mentre il testo è spesso soggetto a errori e imprecisioni. Questi errori si possono trovare, per esempio, nei commenti, tweet o post, contenenti errori di battitura, errori grammaticali, ma anche abbreviazioni e modi di dire

La sfida che i Big Data pongono è dunque legata alla rilevazione del legame che essi hanno rispetto allo scopo dell'analisi. Il Problema è reso più semplice quando i dati sono etichettati, poiché dei tag, che sono metadati, è possibile estrapolare l'ambito di pertinenza, però, notizie o documenti presenti nel web non sempre contengono affermazioni e dati veritieri, anzi, possono discostarsi completamente dalla realtà. Un caso evidente è Wikipedia, dove gli articoli sono redatti dagli utenti che vi accedono, i quali potrebbero scrivere informazioni non veritiere.
È importante però non farsi fuorviare dai problemi di qualità dei Big Data, in particolare quelli provenienti dal web, poiché i processi di analisi che li utilizzano non è detto che richiedano l'esattezza e la precisione. Infatti analisi di sentiment, relativi a un'azienda, a un prodotto o a un personaggio politico all'interno di un grande volume di dati, non sono sicuramente influenzate da qualche valore anomalo o da qualche commento, post o tweet non pertinente. Piuttosto, bisogna tenere in considerazione che la pulizia dei dati potrebbe portare all'eliminazione di dati potenzialmente utili. Se per esempio ricerchiamo anomalie nell'utilizzo di carte di credito, è altamente sconsigliato eliminare dall'insieme delle transazioni i cosiddetti outliers, ovvero valori estremi. Analisi di diversa natura possono avere differenti esigenze di qualità, per questo motivo è consigliabile procedere con operazioni graduali di pulizia dei dati, guidate dagli obbiettivi e dalla tipologia dell'analisi.

1.3.2 Privacy e proprietà dei dati

L'analisi dei Big Data, pur offrendo delle enormi potenzialità nello sviluppo di nuovi servizi e soluzioni, pone sicuramente delle problematiche rilevanti sotto il profilo della tutela della riservatezza. Nel Web, il fatto che esistano molti dati e che siano accessibili a tutti non significa che sia giusto utilizzarli, però è noto a tutti che chi pubblica informazioni sulla propria vita, le proprie abitudini e i propri pensieri, una volta sul web siano visibili e utilizzabili da tutti. Dai social network è possibile estrarre informazioni sensibili, per esempio sull'orientamento politico e religioso che potrebbero essere utilizzati in modo appropriato. Un'altra casistica molto importante è a localizzazione geografica degli individui e dei loro spostamenti, infatti è molto facile lasciare traccia dei propri spostamenti attraverso smartphone dotati di GPS, utilizzo di sistemi elettronici a pagamento, pubblicazione di una foto con il tag geografico su facebook.

1.4 Tecnologie utilizzate per i Big Data

A causa delle grandi dimensioni dei dati e della varietà degli stessi, non è possibile gestire i Big Data attraverso i tradizionali DBMS relazionali, i quali non garantiscono velocità di analisi e archiviazione di grandi quantità di dati. I DBMS relazionali, inoltre, gestiscono in modo efficiente informazioni che sono fortemente strutturate, ma all'interno dei Big Data possono esserci dati non strutturati, come immagini e vide digitali, dati GPS e tanti altri che rientrano nella categoria di dati che non sono gestibili attraverso i sistemi per la gestione di basi di dati relazionali.
A supporto dei Big Data, si sono sviluppate tecnologie specifiche[27] in grado di andare oltre questi limiti; esse fanno uso di sistemi scalabili, che rispondono in maniera veloce e precisa, e che riescono a gestire grandi quantità di dati in modo

tale da estrapolare informazioni preziose. Queste tecnologie sono conosciute come Tecnologie NoSQL.

1.4.1 Tecnologie per l'acquisizione dei dati

L'acquisizione di queste grandi quantità di dati può avvenire attraverso differenti mezzi, che si possono suddividere in quattro categorie:

- *API di chi fornisce i dati.* Queste API sono protocolli utilizzati come interfaccia che permettono al programmatore di richiamarne le funzionalità per comunicare con la fonte dati. In questa casistica rientrano sia dati provenienti da fonti operazionali, si dati provenienti dal Web. Per quanto riguarda i dati provenienti dal web, in particolare dai social network, possiamo citare le Graph API di Facebook, che rappresenta una semplice API http-based che permette di accedere al grafo sociale di Facebook ed esaminare tutti i contenuti pubblici (o accessibili tramite amicizia) che rispondono ai criteri di ricerca desiderati

- Anche i motori di ricerca permettono di accedere ai contenuti attraverso le API. Per esempio Yahoo!, ha un linguaggio SQLite, per eseguire interrogazioni al proprio motore

- *Importazione dei dati mediante strumenti ETL.* Questi strumenti possono essere sfruttati per caricare i big data a partire dalle fonti da cui essi provengono. Molti di questi strumenti sono già attrezzati per connettersi a determinati sistemi di storage, semplificandone l'accesso

- *Software di Web scraping.* Rappresentano l'estrazione automatica di dati da pagine web, ad esempio mediante parser HTML. Il Web scraping in assenza di API pubblice è l'unico modo per estrarre dati, anche se non sempre permesso (es. Google Scholar)

- *Lettura di stream di dati.* Tecnologie per acquisire flussi di dati, come ad esempio le piattaforme CEP, che consentono di catturare eventi, anche ad alta frequenza, in modo efficiente

1.4.2 Tecnologie per lo storage e l'organizzazione

Quando si parla di big data, l'aspetto principale da tenere in considerazione è la gestione di grandi quantità di dati, che possono essere non strutturati o semi-strutturati.
Esistono diverse soluzioni per i big data, ma un ruolo importante per lo sviluppo e la diffusione di questi dati è svolto dalle tecnologie NoSQL e Hadoop.
Hadoop[27] nasce come progetto per l'analisi distribuita di grandi insiemi di dati attraverso un semplice modello di programmazione. Consente la distribuzione dei dati su più server, evitando costi computazionali e di storage, per l'immagazzinamento e l'analisi dei big data. Questa tecnologia è diventata il punto di riferimento per le aziende che intendono gestire i big data.

Essi è composto dai seguenti componenti:

1. *Hadoop common*: uno strato di software comune che fornisce funzioni di supporto agli altri moduli
2. *HDFS*: il file system distribuito che fornisce un'elevata capacità di accesso ai dati
3. *YARN*: sistema di scheduling e gestione delle risorse del cluster
4. *MapReduce*: sistema di parallel processing di grandi quantità di dati

Come abbiamo già introdotto, le tecnologie NoSQL permettono di immagazzinare i dati e consentono flessibilità nel partizionamento. Questi sistemi fanno parte di questa grande famiglia non perché sono simili tra di loro, ma perché si discostano dai sistemi tradizionali basati sul modello relazionale. Infatti, come vedremo in seguito, queste tecnologie presentano caratteristiche molto diverse tra loro.

1.4.3 Tecnologie per l'analisi

L'utilizzo di sistemi per l'analisi dei Big Data permette alle aziende di fare misurazioni efficaci in modo da favorire i processi previsionali e di assunzione delle decisioni. Per questa ragione, molte aziende hanno finanziato lo sviluppo di software per la gestione e l'analisi di questi dati.

Le tecnologie più utilizzate per l'analisi dei Big Data sono:

- *Mahout*: Fornisce gli strumenti per trovare automaticamente i pattern significativi nei grandi insiemi di dati. Mahout mira a rendere più facile e veloce la trasformazione dei Big Data in grandi informazioni
- *Pig*: è una piattaforma per l'analisi di grandi insiemi di dati che si compone di un linguaggio testuale chiamato Pig Latin, tale linguaggio ha le seguenti proprietà chiave: facilità di programmazione; opportunità di ottimizzazione ed estendibilità
- *R*: è un software gratuito per il calcolo statistico e grafico. Compila e gira su una vasta gamma di piattaforme UNIX, Windows e MacOS

Capitolo 2

Tecnologie NoSQL

Con il termine NoSQL si fa riferimento alla nuova famiglia di database, sviluppatasi negli ultimi anni, che si discostano dai database relazionali.

Per far fronte alla necessità di gestire quantità di dati in continua crescita e a velocità sempre più elevate, questi sistemi di memorizzazione sono più flessibili, utilizzano modelli di dati meno complessi e consentono di aumentare le prestazioni di gestione e interrogazione dei dati.
Questi nuovi sistemi presentano particolarità[27] molto utili:

- *Non relazionali*: non è presente una definizione di uno schema fisso come nel caso dei sistemi relazionali
- *Open source*: caratteristica che è alla base del movimento, volta a promuovere l'utilizzo e lo sviluppo degli approcci NoSQL
- *Distribuiti*: capacità di distribuire e replicare i dati su sistemi differenti, in modo totalmente trasparente all'utilizzatore, consentendo di aumentare la possibilità di avere sempre disponibili i dati
- *Scalabili orizzontalmente*: sono focalizzati su una scalabilità orizzontale, e non verticale come i database relazionali

2.1 Confronto RDBMS e NoSQL

Mentre la caratteristica principale dei sistemi relazionali è quella di mantenere una forte consistenza fra i dati, i database NoSQL garantiscono alti livelli di disponibilità dei dati ed un'elevata velocità di recupero, a sfavore della consistenza delle informazioni.
I database relazionali sono caratterizzati dalle proprietà ACID, mentre per i database NoSQL si passa a parlare di proprietà BASE.

2.1.1 Proprietà ACID

Figura 2.1: Atomicità, Consistenza, Isolamento e Durabilità

L'acronimo ACID[01] indica le 4 proprietà che il DBMS deve garantire che valgano per ogni transazione:

- **Atomicity** (Atomicità): l'esecuzione della transazione deve essere totale o nulla, di conseguenza, o si riescono a eseguire tutte le operazioni della transazione o questa viene distrutta
- **Consistency** (Coerenza): quando inizia una transazione il database si trova in uno stato coerente e, quando la transazione termina, il database deve essere in un altro stato coerente, ovvero non deve violare eventuali vincoli di integrità
- **Isolation** (Isolamento): ogni transazione deve essere eseguita in modo isolato e indipendente dalle altre transazioni, e l'eventuale fallimento di una transazione non deve interferire con le altre transazioni
- **Durability** (Durabilità/persistenza): al commit della transazione, le modifiche apportate dalla stessa non dovranno essere perse in nessun modo

2.1.2 Proprietà BASE

Le proprietà BASE[23] sono state introdotte da Eric Browser, autore anche del Teorema di **CAP**:

- **C**onsistency (Coerenza): Tutti i client vedono gli stessi dati nello stesso momento, ovvero dopo una modifica tutti i nodi del sistema distribuito rispecchiano la modifica
- **A**vailability (Disponibilità): Ad una richiesta, il sistema è sempre in grado di dare una risposta

- **P**artition Tollerance (Tolleranza al Partizionamento): il sistema continua a funzionare nonostante siano presenti interruzioni di comunicazione tra due punti del sistema

Secondo il teorema, avere tutte e tre queste caratteristiche è impossibile, ma si può scegliere quale delle due avere a sfavore della terza.

Nel caso degli RDBMS tradizionali, si rinuncia alla Particion Tollerance a favore di Coerenza e Disponibilità.

Le proprietà Base rinunciano alla consistenza per garantire disponibilità dell'informazione e scalabilità. L'acronimo indica le caratteristiche che un sistema deve avere sottostando al teorema del CAP:

- **B**asically **A**vailable: deve garantire la disponibilità delle informazioni
- **S**oft State: non deve garantire la consistenza in qualsiasi istante
- **E**ventually Consistency: non si danno garanzie sul quando i dati raggiungeranno uno stato consistente, poiché il sistema deve essere sempre pronto a rispondere

2.1.3 Scalabilità

La scalabilità[25][27] è la capacità di un sistema di accrescere o decrescere le proprie prestazioni a seconda della necessità.
Un database NoSQL afferma la scalabilità orizzontale[23], cosa che un RDBMS tradizionale non può garantire poiché, per contenere sempre più informazioni, ha la necessità di scalare verticalmente.

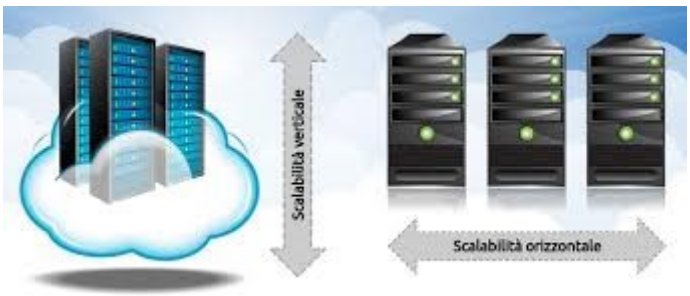

Figura 2.2: Scalabilità verticale e orizzontale

La scalabilità orizzontale è una caratteristica molto importante quando si parla di Big Data, poiché consente di distribuire i dati e le operazioni su macchine differenti al fine di parallelizzare le operazioni e ottenere un quantitativo di dati da elaborare per singolo server. Nonostante sia possibili aumentare le prestazioni anche tramite scalabilità verticale, quella orizzontale permette di ridurre i costi, utilizzando macchine diverse, e di superare i limiti hardware dati dal quantitativo massimo di core e memoria installabili su un singolo mainframe.

2.1.4 Standardizzazione

Quando si confrontano database relazionali e database NoSQL, un aspetto molto evidente è che i primi appartengono a uno standard, infatti i risultati di richieste effettuate tramite opportune query possono essere visualizzati in ambienti standard. Per quanto riguarda l'approccio NoSQL, invece, non si ha una vera e propria standardizzazione[23], quindi ogni implementazione ha un'interfaccia utente propria.

2.1.5 Popolarità

Con la diffusione dei Big Data, l'interesse per i database NoSQL ha una sostanziale crescita, in quanto i database relazionali presentano limiti importanti a cui bisogna rispondere con soluzioni non sempre adatte.

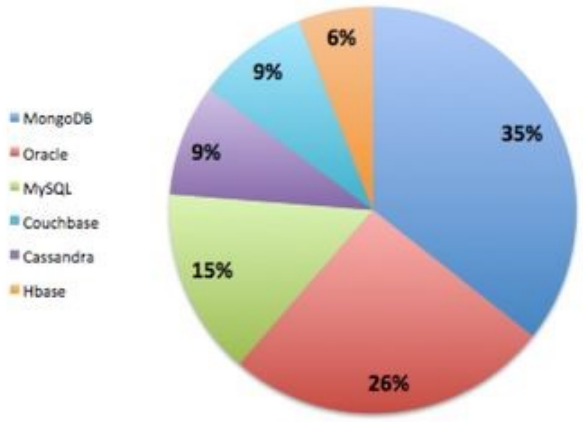

Figura 2.3: Popolarità database

Attraverso un'analisi effettuata da TechRepublic[30] misurando il volume delle menzioni sui social media, articoli di notizie e altre fonti, indica che la diffusione dei database NoSQL è in sostanziale crescita e, come si può notare dal grafico, il database MongoDB, appartenente alla famiglia NoSQL, risulta essere più popolare di alcuni dei database relazionali più utilizzati e dominanti nel mercato dei DBMS.

2.2 Tipologie di Database NoSQL

I database NoSQL sono classificati in base al tipo di modello che utilizzano per la memorizzazione dei dati, in particolare possono essere individuati quattro grandi famiglie:

- Key-Values Stores
- Column-Oriented Stores
- Document-Oriented Stores
- Graph Stores

2.2.1 Key-Values Stores

La caratteristica principale di questa tipologia di database è, sicuramente, la semplicità della struttura dati, in grado di contenere un insieme di coppie chiave-valore. In generale la chiave è un valore univoco con il quale è possibile identificare e ricercare i dati nel database.

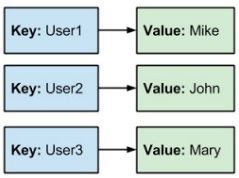

Figura 2.4: Key-Values Stores

I database Key-Values[27] sono basati sul concetto di array associativi, consentendone le stesse operazioni:

- *Add*: aggiungere un elemento all'array
- *Remove*: rimuovere un elemento dall'array
- *Modify*: cambiare il valore associato a una certa chiave
- *Find*: ricercare un valore nell'array tramite chiave

In particolare, l'utilizzatore vede la base di dati come una grossa hash table contenente oggetti di vario tipo, accessibili tramite chiave primaria. Questa soluzione, pur presentando un modello di dati molto semplice, permette di costituire un sistema scalabile orizzontalmente, cioè capace di distribuire le coppie chiave-valore in maniera appropriata sui vari nodi che lo costituiscono.

- **Redis**
  Redis[06] è un database basato sul modello chiave-valore, open surce e scritto in C. Questo database implementa inserimenti, cancellazioni e operazioni di ricerca, inoltre consente di associare alle chiavi liste e insiemi, supportando operazioni su di essi.

2.2.2 Column-Oriented Stores

I database che appartengono a questa categoria[23][22] presentano un'organizzazione dei dati per colonna piuttosto che per riga. Le colonne sono raggruppate in column-family. Una column family può contenere un numero virtualmente illimitato di colonne che possono essere create in fase di esecuzione o definizione dello schema

Figura 2.5: Column-Oriented stores

Questo modello può essere visto come un'evoluzione del key-value store. I dati, infatti, vengono organizzati ancora come una hash table ma, utilizzando generalmente due o più livelli di indicizzazione. Il nome Column family deriva dal fatto che la chiave più esterna, row key, mappa a sua volta una tupla costituita da altre colonne. In un certo senso le tabelle crescono orizzontalmente piuttosto che verticalmente; si pensi ad esempio ad una column family "anagrafica", la quale raggruppa le colonne nome, cognome, età e sesso.
I punti di forza consistono nell'evitare di sprecare spazio quando un determinato valore non esiste per una certa colonna e di consentire spesso un elevato grado di compressione dei dati. Infatti, i database colonnari, grazie all'uso efficiente dello spazio, sono favoriti rispetto ai database con organizzazione per riga dove anche i valori nulli utilizzano spazio. Questo modello è molto flessibile poiché ciascuna riga può contenere valori di una o più colonne.

- **Cassandra**
  Cassandra è un database[23] di tipo Column-Oriented sviluppato per gestire grandi quantità di dati dislocati in diversi server, fornendo un servizio orientato alla disponibilità. Cassandra fornisce una struttura di memorizzazione chiave-valore, con consistenza eventuale. Alle chiavi corrispondono column family. Le colonne possono essere aggiunte ad una column family in qualsiasi momento. Inoltre l'aggiunta delle colonne avviene specificando le chiavi, cosi, differenti chiavi possono avere differenti numeri di colonne in una data column family.

2.2.3 Document-Oriented Stores

La categoria dei Document-Oriented[27] raggruppa i database che supportano la ricerca, oltre che sulle chiavi, anche sui valori legati ad esse. Tali ricerche sono ammesse perché i valori sono rappresentati da strutture in formati pensati appositamente, ovvero XML o JSON. Rappresenta quindi una sofisticazione dei Key-Value Stores poiché, forniscono una struttura alle coppie chiave-valore.

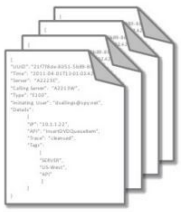

Figura 2.6: Document-Oriented Stores

L'utilizzo di questo tipo di database è piuttosto conveniente in situazioni in cui i dati hanno una struttura dinamica o presentano un gran numero di campi opzionali. Tra le caratteristiche dei Big Data abbiamo descritto anche la variabilità, cioè la presenza di strutture molto diverse fra loro o addirittura la mancanza di una struttura; Tale caratteristica è gestita molto bene dai database Document –Oriented che, si dimostrano quindi particolarmente adatti all'immagazzinamento di tipologie di dati complessi ed eterogeni senza penalizzare le performance.

- **MongoDB**
  MongoDB è un database[23] Document-Oriented con elevate prestazioni in lettura e in scrittura e risulta facilmente scalabile. Combina le caratteristiche dei Key-Values Stores, semplici, veloci e scalabili, con le caratteristiche dei database relazionali, grazie al suo potente linguaggio di interrogazione

2.2.4 Graph Stores

Le precedenti tipologie di database focalizzano l'attenzione sulla scalabilità dei dati e sulla flessibilità delle informazioni, senza tenere in considerazione la connessione tra i dati. Da questa nuova esigenza nasce l'idea di database orientati ai grafi[23]. Questa tipologia di database utilizza nodi e archi per rappresentare e archiviare le informazioni, non richiede le onerose operazioni di JOIN e permette di scalare facilmente grandi quantità di dati.

I modelli di riferimento per l'implementazione di basi di dati con questo schema sono due: il Property Graph model e il Resource Description Graph (RDF). Il secondo è il modello di riferimento del web semantico e i database che lo utilizzano sono anche noti come Triple Store, Quad Store o RDF Store.

- **Neo4J**
  Sviluppato totalmente in Java, Neo4J[22] è un database a grafo open source. È in grado di gestire dati strettamente interconnessi, ambito in cui altri database appartenenti alla famiglia NoSQL falliscono. Il beneficio di usare un database a grafo è la velocità di attraversare nodi e relazioni per trovare dati rilevanti. Spesso questi database vengono usati nei Social Network.

Capitolo 3

MongoDB

Figura 3.1: Logo MongoDB

MongoDB rientra tra i DBMS più utilizzati della famiglia NoSQL e addirittura più utilizzato di alcuni degli RDBMS relazionali. Le caratteristiche che lo rendono cosi apprezzato sono[10]:

- Innanzitutto il fatto che è document-oriented ed è schemaless, ovvero dinamicamente tipato per una facile evoluzione dello schema, questo indica che il campo, con il relativo datatype, non deve essere definito, poiché può variare
- L'assenza di JOIN che consente operazioni di lettura e scrittura molto veloci
- Ha un ricco linguaggio di interrogazione, che consente agli sviluppatori di creare rapidamente potenti funzionalità con i dati
- È altamente scalabile grazie all'automatic sharding, ovvero i dati vengono partizionati automaticamente sui server
- È molto consistente, in modo che le applicazioni possano vedere immediatamente gli aggiornamenti avvenuti sui dati;

MongoDB, come già detto, è un database orientato ai documenti, ognuno dei quali è memorizzato nel formato JSON. Un documento è un insieme di chiavi alle quali sono associati dei valori. I valori possono essere semplici tipi di dati come ad esempio stringhe, numeri o date, ma possono anche essere valori complessi come altri documenti, array e perfino array di documenti[03]

Un esempio di documento può essere il seguente:

```
{
   _id:  Object(“221432674273654385843”),
   nominativo:    {nome: ”mario”, cognome: ”Rossi”},
   PIVA: ”000439043234”,
   promozioni: [”promo A”, ”promo B”]
}
```

Nel documento rappresentato ha particolare importanza il campo ′ id ′, attraverso il quale si ha la possibilità di identificare in modo univoco il documento, come nei database relazionali attraverso la chiave primaria. Nel caso in cui, questo campo, non venga inserito manualmente, viene generato automaticamente dal sistema con un ObjectID, cioè un oggetto in formato Binary JSON.

Un ObjectID è un oggetto con dimensioni pari a 12 byte composto da:

- 4 byte contenenti i secondi dal 1 Gennaio 1970 fino al istante della creazione del documento
- 3 byte che identificano la macchina utilizzata
- 2 byte che identificano il processo di creazione
- 3 byte di contatore che si incrementa ad ogni inserimento

In caso di inserimento consecutivo, attraverso questo campo, si ha la possibilità di memorizzare i record in ordine di inserimento.

Ugualmente alle tabelle dei database relazionali, i documenti vengono raggruppati in collezioni, con la differenza che, in MongoDB, i documenti possono avere campi diversi, per tipo e valore, perché le collezioni non hanno vincoli strutturali. Inoltre, si ha la possibilità di modellare i dati che cambiano nel tempo in base alle proprie esigenze poiché, nei sistemi NoSQL non esiste una struttura prestabilita, come invece avviene in quelli relazionali.

3.1 Persistenza e Velocità

MongoDB è stato sviluppato per raggiungere un compromesso tra velocità in scrittura e persistenza.

La velocità in scrittura rappresenta le operazioni che il sistema può effettuare in un determinato periodo di tempo, tali operazioni possono riguardare l'inserimento di un documento, l'aggiornamento o la cancellazione; la persistenza, invece, garantisce la permanenza delle operazioni all'interno del sistema.

MongoDB da la possibilità di scegliere tra due tipologie di scrittura: *fre-and-forget* o *safe-mode*. La prima permette di inviare le operazioni tramite un socket TCP senza sapere il risultato dell'operazione, invece tramite safe-mode è possibile ricevere una risposta dal sistema dove viene indicato se l'operazione è andata a buon fine.

3.2 Indicizzazione

Un indice[03] è una struttura dati che consente la ricerca rapida delle informazioni relative al campo su cui è costruito. Qualsiasi campo in MongoDB può essere indicizzato. In MongoDB sono supportati tutti i tipi di indici utilizzati anche dai database relazionali come gli indici su chiave singola, indici composti da più chiavi e indici geospaziali. Proprietà molto importante di MongoDB è che si può indicizzare qualsiasi campo di un documento, indipendentemente dal tipo di dato, rendendo possibile l'indicizzazione di campi complessi quali array e documenti. Analogamente ai principali database relazionali, anche in MongoDB è possibile imporre vincoli di unicità sui campi indicizzati, con la possibilità di eliminare i duplicati presenti in fase di creazione. Normalmente, quando viene imposto un indice su un campo, qualora quest'ultimo non fosse presente in un documento lo si considera indicizzato con il valore NULL. Questa situazione potrebbe causare problemi se non sono presenti indici in numerosi documenti, quindi, per rimediare è possibile specificare l'attributo sparse dell'indice, in modo da non considerare i valori non presenti. Un indice molto importante è il **TTL** (**Time To Live**), il quale permette di rimuovere automaticamente i documenti dopo un certo periodo di tempo. La scadenza dei documenti è utile per alcuni tipi di informazioni che devono esistere nel database per un periodo limitato di tempo.

3.3 Replicazione

La replicazione[24] è la distribuzione e la manutenzione del database su più macchine, con lo scopo di prevenire guasti nell'ambiente in cui il database viene utilizzato, pertanto fornisce una garanzia contro problemi relativi alla rete e al server.

MongoDB fornisce due tipologie di repliche: *master-slave* e *replica set*. Per entrambi, un singolo nodo primario riceve tutte le scritture e, in seguito, tutti i nodi secondari leggono e applicano le operazioni in modo asincrono. I due metodi di replicazione utilizzano lo stesso meccanismo di replica ma hanno rilevanti differenze.

- **Master-Slave**
  Il metodo di replica master slave è facile da configurare e ha il vantaggio di supportare qualsiasi numero di nodi secondari, ma ha lo svantaggio di avere un meccanismo di failover completamente manuale. Infatti, se il nodo primario fallisce deve essere un amministratore di sistema a dover scegliere un nodo secondario e attivarlo come nodo primario. Questo metodo non è più utilizzato, in quanto il metodo replica set è diventato lo standard di fatto, nonché la soluzione raccomandata.

- **Replica Set**
  Il replica set è costituito da un gruppo di nodi in cui, tramite una votazione interna al set, uno di questi viene eletto nodo primario, mentre gli altri vengono classificati secondari. Il nodo primario riceve tutte le operazioni di scrittura dai client, e successivamente effettua la scrittura ed esegue la replica dell'operazione sui server secondari in maniera asincrona. Questo metodo garantisce il failover automatico, infatti se per qualsiasi motivo il nodo primario

non è raggiungibile, i nodi secondari eleggono un primario, facendo sì che il sistema diventi nuovamente disponibile con tutti i dati. Quando il precedente nodo primario ritorna attivo, viene inserito nel sistema come secondario. Opzionalmente, è possibile configurare un nodo arbitro, esclusivamente con il compito di stabilire il nuovo nodo primario. Il vantaggio di usare replica set è che, finché ci sarà un nodo attivo, i dati saranno sempre disponibili. Per quanto riguarda le letture, per aumentare la velocità e non sovraccaricare il nodo primario, è possibile leggere anche da un nodo secondario, che però deve essere specificato, essendo che le replicazioni sono asincrone e non è detto che tutti i nodi siano aggiornati.

3.4 Scalabilità: lo Sharding

MongoDB per distribuire i dati e per effettuare operazioni su un cluster di machine permette di usare lo sharding[03], ovvero un meccanismo di scalabilità orizzontale che divide i dati su server differenti, chiamati shard. Ogni shard contiene una parte di dati in un database indipendente.

Lo sharding è di particolare importanza anche per il recupero dei dati, infatti ogni shard si troverà a operare su un numero più ristretto di documenti e, potrà concludere le interrogazioni nel minor tempo possibile. In particolare, MongoDB, supporta l'auot-sharding, attraverso il quale si evitano alcuni problemi amministrativi del carico di lavoro dovuto allo sharding manuale. Infatti, i dati vengono distribuiti e bilanciati in modo automatico sui diversi shard. Dato che risulta essere conveniente che l'applicazione sia all'oscuro di che shard contenga i dati su cui sta operando, viene attivato un processo chiamato mongos che è una sorta di router ed è a conoscenza della suddivisione dei dati, cosi l'applicazione può connettersi ad esso e può operare nel solito modo.

La distribuzione dei dati viene effettuata tramite una chiave chiamata shard key, cioè un campo di un documento su cui effettuare la partizione per range di valori. La scelta della chiave di shard è di grande importanza e determina sia l'equità della distribuzione, sia le prestazioni del sistema. Una suddivisione bilanciata in modo adeguato permette di ottenere un carico di lavoro simile per tutto il sistema, diminuendo così la probabilità di avere code di attesa. Però, non è sempre possibile ottenere una distribuzione perfettamente bilanciata dei dati e, per evitare questa situazione, l'elemento che ha il compito di mantenere il più possibile equilibrata la distribuzione dei dati è il balancer. Tale processo agisce sui chunk, ovvero piccole porzioni di dati di dimensione pari a 64 MB. Il Balancer controlla periodicamente che non ci sia una differenza sostanziale tra lo shard più piccolo e lo shard più grande e, nel caso ci fosse, viene avviato un processo di migrazione dei dati dallo shard sovrappopolato a quello meno popolato.

3.5 La Shell di MongoDB

Il server del database MongoDB[13] viene attivato attraverso l'eseguibile chiamato mongod. La shell di comando, invece, viene attivato tramite l'eseguibile mongo, che si connette a uno specifico processo mongod e, se nessun database viene specificato all'avvio, viene selezionato un database di default chiamato test.

La shell[17][18] è basata sul linguaggio Javascript ed uno strumento che consente di amministrare e manipolare i dati del database. In questi sistemi, database e collezioni vengono creati solo quando viene inserito un documento.

3.5.1 Inserimento di un documento

Per inserire un documento in una collezione viene utilizzato il metodo insert. Un esempio può essere il seguente:

```
db.utenti.insert({nome: ''mario''})
```

In questa operazione, al documento, viene associato automaticamente una chiave chiamata $'\ \_id\ '$ che, rappresenta l'identificatore globale nel sistema.

Dopo l'inserimento, la struttura dati viene convertita in BSON (Binary JSON), ovvero in formato binario che memorizza qualsiasi documento come stringa di byte. Successivamente il database lo riceve e controlla la validità del campo $'\ \_id\ '$ e la dimensione, che può essere al massimo 4 Mb.

3.5.2 Lettura di un documento

Una volta inseriti i documenti all'interno della collezione è possibile controllare la loro esistenza con il metodo find, che ritorna tutti i documenti presenti in una collezione oppure il metodo findOne, che ritorna un solo documento dalla collezione.

```
Db.utenti.find({nome: ''mario''})
```

Con questa operazione si esegue una query selector e, il documento restituito è il seguente:

```
{
   _id: ObjectID(''4bf9bec60bac34542343892349''),
   nome: ''mario''
}
```

3.5.3 Eliminazione di un documento

Per eliminare uno o più documenti da una collezione viene utilizzato il metodo remove. Questo metodo se viene utilizzato senza alcun parametro elimina tutti i documenti di una collezione. Se invece si vuole rimuovere un determinato documento da una collezione, bisogna specificarne i parametri che identificano tale documento, come nell'esempio:

```
db.utenti.remove({nome: ''mario''})
```

Con questo esempio, viene eliminato il documento con il valore ′ mario ′ associato alla chiave ′ nome ′. Se invece si vuole eliminare una collezione assieme ai suoi indici, è necessario utilizzare il metodo drop, come nel seguente esempio:

```
db.utenti.drop()
```

3.5.4 Aggiornamento di un documento

MongoDB offre due metodi per aggiornare un documento: sostituendo il documento; oppure utilizzando operatori di aggiornamento per modificare uno specifico campo all'interno di un documento, ovvero attraverso un aggiornamento mirato.

Per sostituire completamente un documento è necessario prima ricavare il documento dal database, modificarlo e rimpiazzare il documento modificato. Ad esempio supponiamo di avere il seguente documento:

```
{
   _id:  Object(''221432674273654385843''),
   nominativo:    {nome: ''mario", cognome: ''Rossi''},
   PIVA: ''000439043234'',
   promozioni: [''promo A'', ''promo B'']
}
```

Si vuole modificare il valore associato al campo email sostituendolo con un nuovo valore.

```
Var document  = db.utenti.find({_id:
                  ObjectID(''4bf9bec60bac34542343892349'')})
documento.email = ''marcoverdi@gmail.com''
db.utenti.update({_id: ObjectID(''4bf9bec60bac34542343892349''},
                     documento)
```

Utilizzando invece il metodo dell'aggiornamento mirato, al metodo update vengono passati due argomenti: Nel primo viene specificato quale documento aggiornare e nel secondo viene definito come il documento deve essere aggiornato. Ad esempio:

```
db.utenti.update({_id: ObjectID(''4bf9bec60bac34542343892349'')},
      {$set: {email: ''marcoverdi@gmail.com''}})
```

L'approccio relativo all'aggiornamento mirato di solito consente migliori performance, in quanto si elimina il tempo di andata e ritorno del documento, pertanto questo metodo utilizza meno tempo per serializzare e trasmettere i dati.

3.6 Conclusione

MongoDB[19] è un progetto open source scritto prevalentemente in C++.

È distribuito come release stabile oppure in versione beta. È disponibile per Windows, Linux, Mac OS X e Solaris. Infine, è interessante notare come MongoDB sia il database NoSQL più richiesto nelle offerte di lavoro e, per il momento è il quinto database più popolare nella classifica di db.engines.com.

Capitolo 4

Progettazione e Realizzazione con MongoDB e NodeJS

In questo capitolo viene illustrato la realizzazione di un'applicazione web che permette di mostrare la connettività urbana attraverso le mappe di Google.
Le informazioni necessarie per mostrare la connettività, sono inviate attraverso chiamate http effettuate da un'applicazione mobile, nella quale si raccolgono dati riguardanti reti wifi e dati cellulare. I dati ricevuti sono in formato JSON nel quale viene indicato lo scenario che rappresenta l'area in cui è stata effettuata la scansione, le coordinate geografiche, dati relativi al cellulare attraverso il quale è stata effettuata la scansione e infine un array di elementi che riguardano le reti wifi, dove, per ognuno, viene indicato la potenza della rete, la sicurezza adottata e il canale utilizzato per trasmettere le informazioni. Una volta ricevute, queste informazioni vengono salvate nel database MongoDB e, successivamente, vengono estrapolati i dati per essere infine elaborati e rappresentati nell'applicativo. L'applicazione prevede due modalità di visualizzazione, la prima permette di visualizzare le informazioni riguardanti le reti wifi, invece la seconda permette di visualizzare le informazioni in base ai dati cellulare.

Figura 4.1: Modalità di visualizzazione

4.1 Funzionalità Wifi mode

All'avvio dell'applicazione viene mostrato un modale per dare all'utente la possibilità di effettuare l'autenticazione e, successivamente, procedere ad usufruire del servizio. Questo modale contiene i campi per inserire l'username e la password. Una volta confermata l'autenticazione, in caso di successo, si potrà procedere ad utilizzare l'applicazione.

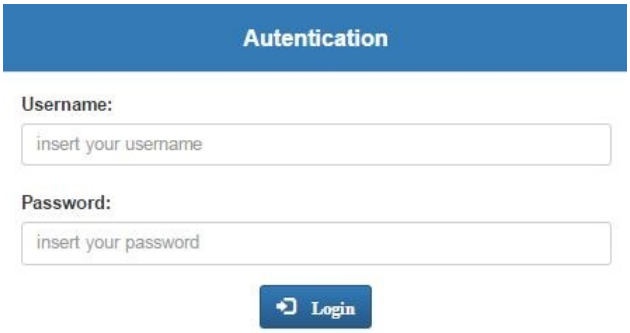

Figura 4.2: Autenticazione

Scanned Points

Una volta effettuata l'autenticazione e al clic del pulsante ′ scanned points ′, viene effettuata una chiamata http al server per ricevere le informazioni contenute nel database, successivamente viene mostrata la mappa, posizionando il centro in base allo scenario scelto, con i markers che indicano i punti in cui sono state effettuate le scansioni delle reti wifi. Inoltre, viene gestito l'evento di click su ogni marker in modo da visualizzare le informazioni delle reti wifi. Infatti, se si effettua il clic su un marker, viene mostrato un modale indicando tutte le reti wifi trovate in quel determinata posizione geografica con le informazioni sulla rete, ovvero la potenza, il canale e la sicurezza adottata.

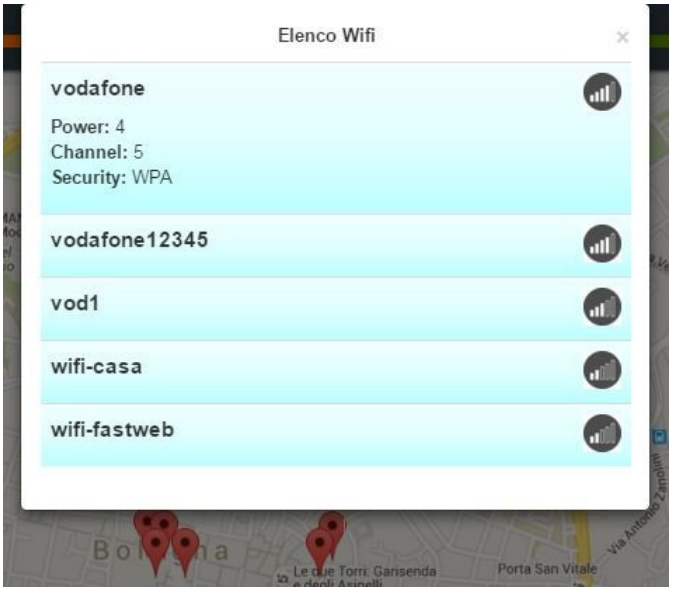

Figura 4.3: Lista reti wifi

Heatmap

Oltre a visualizzare i dati nella mappa attraverso i markers, viene utilizzata un heatmap, ovvero una mappa di calore che è una rappresentazione grafica dei dati dove i singoli punti sono rappresentati da colori. Attraverso questa mappa, cliccando sui relativi bottoni, si ha la possibilità di scegliere quale rappresentazione utilizzare, ovvero in base a quali caratteristiche colorare i punti nella mappa:

- *Potenza wifi.* Con questa rappresentazione ogni punto nella mappa viene colorato in base alla potenza media delle reti wifi scansionate in quel punto, ovvero se la potenza media è bassa, avrà un colore che tende al rosso, se invece la potenza media risulta elevata, avrà un colore che tende al verde

- *Wifi libere.* Attraverso questa rappresentazione ogni punto nella mappa viene colorato in base alla presenza di almeno una rete wifi che non adotta alcun tipo di sicurezza. In particolare se in un punto è presente almeno una rete wifi libera, tale punto verrà colorato di verde, altrimenti rosso

- *Numero reti.* Infine attraverso quest'ultima rappresentazione, ogni punto nella heatmap viene colorato in base al numero di reti wifi scansionate in quella posizione e, se il numero di reti wifi risultasse basso, avrà un colore che tenderà al rosso, in caso contrario verde

Scenario

Per rappresentare la mappa in una determinata posizione, viene data la possibilità all'utente di scegliere lo scenario, ovvero l'area in cui si vuole rappresentare i dati. Questa funzionalità, è resa disponibile dalla presenza di un menu a tendina contenente tutti scenari disponibili e, al click di ogni opzione del menu, la mappa viene centrata nella posizione geografica che rappresenta lo scenario identificato dall'elemento cliccato, e successivamente vengono mostrati i punti. Fino a quando non viene modificato lo scenario, la mappa resterà centrata in quella posizione anche quando si passa a modalità dati cellulare.

Grafici

Oltre a rappresentare i dati attraverso una mappa, sono stati implementati due grafici per indicare i canali utilizzati e la sicurezza adottata. Il primo riguarda la frequenza del canale in un determinato scenario, ed è rappresentato attraverso un istogramma, dove nell'asse delle Y è presente la frequenza dei canali e nell'asse delle X sono presenti i canali possibili, ovvero 15.

Per quanto riguarda il secondo grafico, viene effettuata una riproduzione attraverso un diagramma circolare, costituito dividendo il cerchio in spicchi, dove ognuno dei quali ha una dimensione basata sulla frequenza di una tipologia di sicurezza adottata in un determinato scenario.

(a) Grafico frequenza canali (b) Grafico frequenza sicurezze

Figura 4.4: Grafici frequenze

4.2 Funzionalità Cellular mode

Nell'applicativo, oltre a visualizzare i dati che riguardano le reti wifi, è possibile passare a modalità ′ cellulare ′. Questa modalità, permette di visualizzare le mappe in base ai dati cellulare.

Scanned Points

Quando si passa a modalità ′ cellulare ′ viene mostrata la mappa centrata in base allo scenario, con i relativi marker. Ad ogni marker vengono associati le informazioni relativi al cellulare e, al momento di click su ogni marker viene mostrato un modale che rappresenta i dati del cellulare. In particolare viene indicato la potenza di trasmissione, il tipo di connessione, velocità download e velocità di upload.

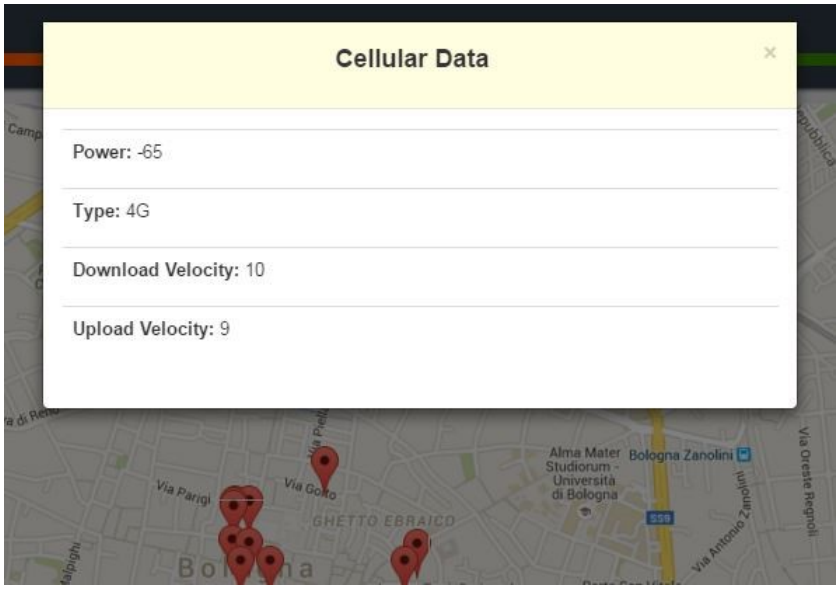

Figura 4.5: Dati cellulare

Heatmap

Anche par la modalità ′ cellulare ′, oltre a visualizzare la mappa con i markers, viene data la possibilità di visualizzare le informazioni attraverso una mappa di calore. In particolare, le informazioni vengono visualizza in base alle seguenti caratteristiche:

- *Potenza connessione.* Attraverso questa rappresentazione, i punti nella mappa vengono colorati in base alla potenza della connessione. In particolare se la potenza è bassa, avrà un colore che tende al rosso, altrimenti il colore tenderà al verde
- *Tipo connessione.* In questo caso, per ogni punto viene verificato il tipo di connessione. Se esiste una connettività 3G il punto sarà rosso, invece, in caso di connettività 4G il punto sarà di colore verde
- *Velocità download.* La velocità di download presenta un range da 1 a 10, che rappresentano i megabit per secondo. Valori bassi avranno un colore che tende al rosso altrimenti verde
- *Velocità di trasmissione.* Come per il download, la velocità di trasmissione ha un range da 1 a 10 megabit per secondo. Se la velocità è bassa il punto avrà un colore tendente al rosso, in caso contrario verde

Grafici

In questa modalità viene rappresentato un grafico circolare che rappresenta le tipologie di connettività e la loro frequenza in base allo scenario scelto. In particolare viene mostrata le connessione 3G e 4G indicando al centro del grafico la frequenza di ognuna di esse.

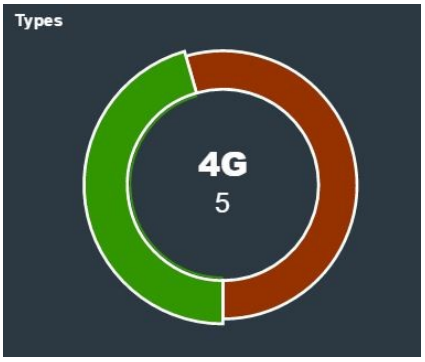

Figura 4.6: Grafico frequenza connettività

4.3 Tecnologie utilizzate

MongoDB

MongoDB, come abbiamo già introdotto, è un database NoSQL orientato ai documenti. In questo progetto è stato utilizzato come supporto per l'immagazzinamento di dati riguardanti le scansioni della connettività. Il database contiene una collection denominata ′ wifiData ′ e, al suo interno sono contenuti i documenti in formato JSON. Per installare[18] MongoDB si è scaricato l'ultima versione dalla pagina ufficiale di MongoDB in base al sistema operativo utilizzato. Successivamente è stata effettuata l'installazione automatica da riga di comando di windows utilizzando msiexec.exe, indicandone la locazione. Dopo l'installazione si ha la possibilità di avviare il server mongod e, successivamente, accedere al client di MongoDB attraverso il comando mongo. Inoltre, per accedere al client è stato utilizzato uno strumento open source chiamato Robomongo, descritto in seguito.

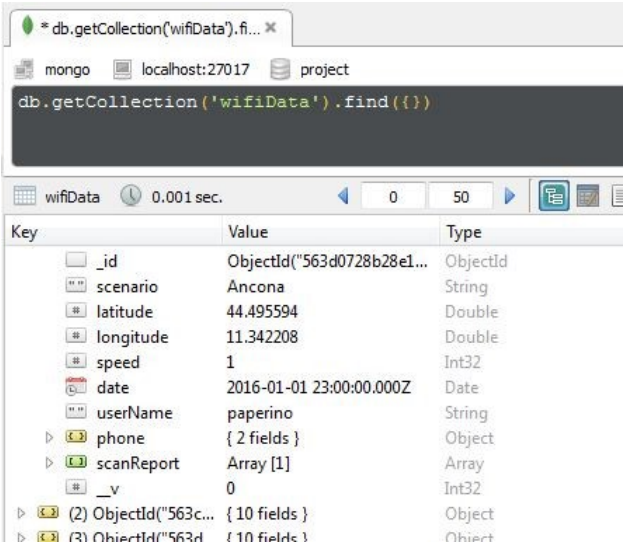

Figura 4.7: Interfaccia Robomongo

- **Robomongo**
  Robomongo è uno strumento [28] open source che permette di gestire lo shell di MongoDB attraverso un'interfaccia. Robomongo incorpora lo stesso motore JavaScript che alimenta lo shell mongo di MongoDB 2.2, consentendo di effettuare interrogazione al database e mostrando i risultati attraverso diverse modalità di visualizzazione, inoltre offre evidenziazioni della sintassi, completamento automatico e altre funzionalità che permettono di interagire con il database con molta semplicità.

Node.JS


Figura 4.8: Logo Node JS

Node.JS[12] è un Framework utilizzato per lo sviluppo di applicazioni server-side utilizzando il linguaggio JavaScript normalmente utilizzato per la realizzazione di codice client-side. Il sistema si basa su JavaScript Engine V8, che è il runtime di Google utilizzato anche da Chrome ed è disponibile per le maggiori piattaforme.

Caratteristica molto interessante di Node.JS è la possibilità di sfruttare la programmazione ad eventi e non il classico modello basato su processi o thread concorrenti. Il programma rimane in attesa di specifici eventi ai quali il sistema reagisce in determinate maniere.

Ogni evento quindi risulta asincrono a differenza dei pattern di programmazione più comuni in cui un evento accade ad un altro solo dopo che esso è stato terminato. Grazie a questo comportamento, durante l'attesa di un certo evento, il runtime può gestire altre azioni garantendo una maggiore efficienza dell'applicazione.

I vantaggi di Node.JS sono molteplici, quello più evidente è la possibilità di realizzare applicazioni server-side e client-side utilizzando lo stesso linguaggio, consentendo così ad un'unica persona, o unico team, di occuparsi sia degli aspetti della presentazione, sia della logica dell'applicazione. Ovviamente questo dipende dal contesto specifico del progetto. Considerando però che uno dei punti di forza di Node.JS è la scalabilità del software, può essere utilizzato sia per piccoli progetti, sia per applicazioni enterprise.

NodeJS dà la possibilità di aggiungere nuove funzionalità attraverso l'integrazione di moduli, dove ognuno dei quali aggiunge una classe o singoli metodi utilizzabili all'interno del codice.

I moduli utilizzati nel progetto sono Express e MongoJS. Una volta installati attraverso il gestore dei pacchetti interni di Node (NPM), per importarli, è stato utilizzato il metodo require come segue:

```
var express = require("express");
var mongojs = require("mongojs");
```

- **Express**
  Express[09] è un server framework Node.JS che fornisce un robusto set di funzionalità per sviluppare applicazioni web, facilitando lo sviluppo di applicazioni basate su Node. Di seguito alcune delle caratteristiche principali del quadro Express:
  - Permette di creare middleware per rispondere alle richieste http
  - Definisce una tabella di routing che viene utilizzata per eseguire un'azione basata sul metodo http e url
  - Permette di creare pagine HTML dinamiche

- **MongoJS**
  MongJS[20] è un modulo Node.JS che consente di accedere a MongoDB utilizzando un'API che è estremamente simile alla shell JavaScript di MongoDB.

Infine è importante sottolineare che per gestire la logica dell'applicazione è stato utilizzato Node.JS piuttosto che altri linguaggi, come PHP, perché oltre che a utilizzare un unico linguaggio sia per il front-end che per il back-end, si interfaccia molto bene con il database MongoDB, sia grazie ai suoi moduli, come MongoJS, ma anche perché utilizza la stessa struttura JSON.

MorrisJS

MorrisJS[21] è un libreria che permette di rappresentare dati attraverso grafici. È una semplice API per disegnare grafici a linee, a barra, ad area e circolari. I requisiti necessari per l'utilizzo di questa tecnologia sono:

- JQuery almeno alla versione 1.7, framework JavaScript che descriveremo nel prossimo paragrafo
- Raphael.js[26] con versione maggiore o uguale alla 2.0, che è una piccola libreria JavaScript con l'obbiettivo di fornire un adattatore per rendere il disegno vettoriale facile con compatibilità cross-browser

JQuery

JQuery[15] è una libreria Javascript per applicazioni web. Nasce con l'obbiettivo di semplificare la selezione, la manipolazione, la gestione degli eventi e l'animazione di elementi DOM in pagine HTML, nonché implementare funzionalità AJAX.

Figura 4.9: Logo JQuery e AJAX

AJAX[02] permette di ricavare dati dal server attraverso chiamate asincrone e caricarli in background senza interferire con il comportamento della pagina. Permette, quindi, l'aggiornamento dinamico della pagina senza dover effettuare il ricaricamento dell'url.
I Parametri di base di questa tecnologia sono:

1. *url*: l'indirizzo al quale inviare la chiamata
2. *success*: funzione da lanciare in caso di successo e accetta come parametri i dati inviati dal server
3. *error*: funzione da lanciare in caso di errore della chiamata

Google maps API

Le API di Google Maps[11] forniscono tutta una serie di strumenti per manipolare le mappe del globo terrestre messe a disposizione da Google, con la possibilità di integrare nuovi servizi per arricchire le funzionalità di base offerte da Google Maps.

HTML

L'HyperText Markup Language (HTML)[14], tradotto come linguaggio di descrizione per ipertesti, è un linguaggio usato per controllare la struttura dei documenti ipertestuali disponibili nel World Wide Web. Tutti i siti web sono scritti o passano attraverso il formato HTML, codice che viene letto (interpretato) dal web browser, il quale, a seguito dell'elaborazione, genera la visualizzazione della pagina desiderata sullo schermo del computer dopo la preventiva richiesta al web server da parte dell'utente. L'HTML non è un linguaggio di programmazione, ma un linguaggio di markup, ossia descrive le modalità di impaginazione, formattazione o visualizzazione grafica (layout) del contenuto, testuale e non, di una pagina web ma, non supporta nessuna gestione di istruzioni, di variabili, di sequenze o di strutture di comando. Punto HTML (.html) o punto HTM (.htm) è l'estensione comune dei documenti HTML

CSS

Il CSS (Cascading Style Sheets)[05] è un linguaggio usato per definire la rappresentazione di documenti HTML, XHTML e XML. Le regole per comporre il CSS sono contenute in un insieme di direttive (Recommendations) emanate a partire dal 1996 dal W3C. L'introduzione del CSS si è resa necessaria per separare i contenuti dalla formattazione e permettere una programmazione più chiara e facile da utilizzare, sia per gli autori delle pagine HTML che per gli utenti.

Twitter Bootstrap

Bootstrap[04] è una raccolta di strumenti che offre la possibilità di costruire pagine HTML coerenti e funzionali. Per utilizzare Bootstrap, basta includere nei nostri progetti gli asset messi a disposizione in modo da utilizzarne gli stili e le funzionalità presenti. Molto importante da aggiungere è il fatto che permette di rendere il risultato finale completamente responsive, ovvero che la nostra applicazione si adatta alle dimensioni dello schermo ed al dispositivo che lo visionerà.

Figura 4.10: Logo Bootstrap

4.4 Implementazione

4.4.1 Server

La logica dell'applicazione viene gestita attraverso Node.JS. Una volta installato correttamente sulla macchina, attraverso il suo gestore dei pacchetti interni NPM, si è potuto installare i moduli messi a disposizione, in modo da creare un server semplice e veloce.
Il framework Express fornisce un numero di astrazioni di concetti e applicazioni server. Inizialmente, è stata utilizzata la sua capacità di servire risorse statiche, come nelle seguenti righe di codice presenti nel file server.js:

```
var express = require( ''express'' );
var app = express();
app.use(express.static( __dirname + ''/public'' ) );
app.listen( 3000 );
```

Per avviare il server si esegue node *server.js* da riga di comando e, successivamente, si può vedere la propria pagina su *http://localhost:3000*.

Per avere una comunicazione tra client e server, sono state implementate diverse funzioni che effettuano richieste http, consentendo di inviare e ricevere i dati in modo asincrono. Attraverso l'utilizzo di Express, è stato definito il percorso sul server dove si vuole accettare le richieste e un callback. Il callback riceve un request object dove sono presenti i dati ricevuti dal client, e un response object per definire i dati che ritornano al client.

Per interagire con MongoDB è stato utilizzato il framework MongJS, attraverso il quale si è potuto interrogare il database come nella shell mongo. Inizialmente viene effettuata la connessione al server MongoDB attraverso le seguenti righe di codice:

```
var databaseUrl = ''localhost:27017/wifiViewer'';
var collections = [''wifiData''];
var mongojs = require(''mongojs'');
var db = mongojs(databaseUrl, collections);
```

La variabile *databaseUrl* può contenere l'host del server con la porta e il nome del database a cui vogliamo connetterci. In questo caso si fa riferimento al server locale. La variabile *collections* è un array che contiene l'insieme di dati che la nostra applicazione utilizza.
Il database MongoDB contiene un'unica collection chiamata wifiData e, al suo interno sono presenti i documenti con le informazioni ricevute dall'applicazione mobile.

In particolare, i documenti memorizzati hanno la seguente struttura:

```
{
   scenario: K,
   userName: G,
   latitude: A,
   longitude: B,
   speed: C,
   time: D,
   scanReport: [
            {ssid: C1, bssid: X1,   power: D1,
             channel: E1, security: F1 },
            {ssid: C2,  bssid; X2, power: D2,
             channel: E2, security: F2 }
            ...
      ],
   cellInfo: {type: 3G/4G, power: X, variance: Y,
              cellid: Y, cqi: Z, ssnr: U},
   testInfo: {throughput_downlink :X1,
              throughput_uplink: X2, delay: X3,
      connection_delay: X4, pdr: X6}
}
```

Partendo dal primo campo del documento, lo scenario, rappresenta l'area geografica di interesse. Successivamente sono presenti le coordinate geografiche che indicano il punto in cui è stata fatta la scansione.

- *scanReport*, contiene un array di elementi che rappresentano le reti wifi scansionate. In particolare, ogni elemento contiene il nome della rete wifi, un identificatore univoco, la potenza della rete, il canale attraverso il quale trasmette le informazioni, e infine la sicurezza adottata
- *cellInfo* contiene le informazioni del dispositivo utilizzato, nelle quali è presente il tipo di connessione, e la potenza
- *testInfo* contiene le informazioni sulla velocità di download e upload, tempo di trasmissione, tempo per aprire la connessione TCP e percentuale di pacchetti ricevuti correttamente

Una volta effettuata la connessione, si è potuto eseguire le query per manipolare i dati nel database, come nell'esempio:

```
db.wifiData.find({scenario: "Bologna}, function(err, docs) {
   if(!err){
   console.log(docs)
   }
});
```

In questo esempio vengono ricavati dalla collection wifiData i documenti che hanno come scenario Bologna. Si può notare che la Query sia praticamente uguale a quella corrispondente nella console di MongoDB. In aggiunta alla Query, passiamo una funzione di callback per gestire i risultati, ovvero viene controllata la presenza di errori e, se non sono presenti, viene stampato il risultato sulla console di Node.

Node.js implementa un paradigma su eventi concorrenti e quindi la maggior parte delle funzioni sono dei callback, questo permette all'applicazione di non bloccarsi e di essere performante.

I due moduli che sono stati inclusi, sono stati adoperati per implementare le funzioni che stanno alla base del server. In particolare sono stati utilizzati per realizzare le seguenti funzionalità in cui si trasmettono e ricevono i dati interagendo con il database:

- Inserimento di documenti nel database attraverso una richiesta http, di tipo POST:

```
app.post('/insert', function (req, res){

   db.wifiData.insert(req.body);
   res.json("data have been received");

});
```

  Attraverso queste righe di codice, si rimane in ascolto di una richiesta http e, una volta ricevuta, si effettua la Query a MongoDB per l'inserimento dei documenti e si restituisce una risposta nella quale si conferma la ricezione delle informazioni.

- Restituzione al client del centro della mappa in base allo scenario ricevuto:

```
app.post('/center', function (req, res){
   db.wifiData.find({scenario:req.body.scenario}).sort({
   latitude:-1}).limit(1, function(err, docs) {
   ...
   });
});
```

  In questa funzione, una volta ricevuta la richiesta, si effettua l'interrogazione al database, ricavando la longitudine e latitudine massima e minima, in modo da effettuare la media delle latitudini e longitudini, restituendo al client il centro della mappa in base allo scenario richiesto.

- Restituzione degli scenari disponibili al client attraverso una richiesta http, di tipo GET:

```
app.get('/scenario', function (req, res){
   db.wifiData.distinct('scenario', {}, function(err, list) {
      if(!err) {
            res.json(list);
      }
   });
});
```

  Attraverso questa richiesta vengono ricavati da MongoDB tutti i scenari disponibili utilizzando il distinct in modo da evitare le ripetizioni, successivamente vengono inviati in risposta al client.

- Richiesta da parte del client di tutti i documenti che contengono lo scenario indicato nella request del callback:

```
app.post('/wifi', function (req, res){
   db.wifiData.find({scenario: req.body.nome}, function(err,
   docs){
      if(!err){
         res.json(docs);
      }
   });
});
```

  Attraverso questa richiesta, vengono ricavati tutti i documenti che contengono lo scenario indicato e restituiti al client in risposta al callback.

4.4.2 Client

Nel client le informazioni vengono mostrate attraverso una pagina html e, grazie all'utilizzo del framework JQuery la selezione degli elementi del DOM avviene in modo semplice e intuitivo. La pagina html viene aggiornata dinamicamente senza un esplicito ricaricamento da parte dell'utente, questo avviene grazio all'utilizzo della tecnologia AJAX, ovvero i dati sono richiesti al server, attraverso chiamate asincrone, e caricati in background senza interferire con il comportamento della pagina.

Le richieste AJAX effettuate al server hanno la seguente struttura:

```
$.ajax({
   url: "http://localhost:3000/wifi",
   contentType: 'application/json',
   type: "POST",
   data: JSON.stringify(data),
   dataType: "json",
   success: function(data) {
         ...
   },
   error: function(jqXHR, textStatus, errorThrown) {
         ...
   }
});
```

Attraverso questa chiamata vengono inviati i dati presenti nel campo data e nel parametro della funzione del success sono restituiti i dati dal server. Nel caso in cui la chiamata non è andata a buon fine, viene eseguita la funzione presente nel campo error, dove i parametri rappresentano l'errore restituito.

All'avvio dell'applicazione e, ogni volta che si cambia lo scenario, vengono effettuate delle chiamate AJAX al server per ricevere i documenti contenenti lo scenario specificato e salvarli globalmente per essere successivamente fatti visualizzare attraverso la mappa e grafici.

Mappa

Le informazioni ricavate dal server, vengono mostrate nella pagina attraverso l'utilizzo di mappe di Google e, per inizializzare la mappa vengono eseguite le seguenti righe di codice:

```
var mapProp = {
   center: new google.maps.LatLng(center.lat, center.lng),
   zoom: 15,
   mapTypeId: google.maps.MapTypeId.ROADMAP
   };
mapNormal = new google.maps.Map(document.getElementById("mappa"),
            mapProp);
```

Dove vengono indicate le proprietà della mappa, ovvero il centro, lo zoom e il tipo di mappa da far visualizzare. Infine, si seleziona l'elemento della pagina html dove viene visualizzata la mappa. La mappa visualizzata può rappresentare i punti in due modalità: attraverso Markers, ovvero delle icone messe a disposizione dalle API di Google map; oppure attraverso punti colorati.

Grafici

Oltre a rappresentare i dati attraverso una mappa, vengono rappresentati due grafici in modalità wifi e un grafico in modalità cellulare. I primi mostrano la frequenza dei canali e la frequenza delle sicurezze adottate in un determinato scenario; Il grafico presente in modalità cellulare, invece, indica la frequenza della connettività 3G e 4G. In particolare, i canali utilizzati vengono mostrati attraverso un istogramma, invece la frequenza di ogni sicurezza e la frequenza delle tipologie di connessione vengono mostrate attraverso una grafico circolare.

- **Grafico canali**
  Per realizzare questo grafico, vengono verificati tutti i canali dei documenti appartenenti allo scenario scelto e, per ogni canale identificato viene incrementata un variabile contatrice associata al canale. Una volta verificata la frequenza di ogni canale, viene rappresentato il grafico attraverso la tecnologia MorrisJS. Come nel seguente esempio:

```
Morris.Bar({
   element: 'channelGraphic',
   data: [
         { y: '1', a: c1},
         { y: '2', a: c2},
         { y: '3', a: c3},
            ...
        ],
   xkey: 'y',
   ykeys: 'a',
   labels: 'Channel',
   resize: true
});
```

Morris.bar indica che il Grafico che si sta creando è un grafico a barre, invece tra gli elementi interni, element indica l'id dell'elemento html a cui associare il grafico, data rappresenta i dati sui quali si basa il grafico, xkey e ykeys indicano le assi, infine viene indicata una label e il fatto che il grafico è adattabile al elemento html che lo contiene.

- **Grafico sicurezza**
  Per realizzare questo grafico viene verificata la frequenza di ogni sicurezza adottata da ogni rete wifi presente nei documenti appartenenti allo scenario indicato. Una volta ricavate le informazioni, viene realizzato il grafico nel seguente metodo:

```
Morris.Donut({
      element: 'securityGraphic',
      data: [
            {label: "WPA", value: WPA},
            {label: "WPA1", value: WPA1},
            {label: "WPA2", value: WPA2},
            {label: "WEP", value: WEP},
            {label: "None", value: None}
         ],
      resize: true
      });
```

  Dove Morris.Donut permette di creare un grafico circolare in base ai dati passati, indicando l'elemento html al quale associare il grafico.

- **Grafico Connettività**
  Questo grafico indica la connettività del cellulare, ovvero 3G o 4G. Inizialmente viene verificata la frequenza della connettività analizzando ogni documento appartenente allo scenario indicato in modo da verificare la frequenza della connessione 3G e 4G.

```
Morris.Donut({
      element: 'typeGraphic',
      colors: ["rgba(150, 50, 0)", "rgba(50, 150, 0)"],
      data: [
            {label: "3G", value: g3},
            {label: "4G", value: g4}
         ],
      resize: true
});
```

  Questo grafico è diviso in due spicchi, uno per la connettività 3G e l'altro per la connettività 4G. In base alla frequenza di ogni canale viene adattata la grandezza di ogni spicchio.

Grafica

Per definire la formattazione della pagina html è stato usato un foglio di stile CSS esterno accedendo cosi al DOM delle pagine html attraverso i selettori. In questo modo si è separato nettamente il contenuto ed il formato di presentazione.

Oltre a CSS, per la parte front-end, è stato utilizzato Twitter Bootstrap, un framework CSS che ha consentito di iniziare a sviluppare un front-end partendo già da una base solida. Inoltre, grazie all'utilizzo di bootstrap è stata costruito la struttura della pagina in modo che il risultato finale sia responsive, ovvero la pagina si adatta in base alle dimensioni dello schermo ed al dispositivo che lo visionerà.

Capitolo 5

Conclusioni

Nella tesi viene introdotto il fenomeno dei Big Data mostrandone le caratteristiche principali e i benefici che possono apportare. La produzione di questi dati ha una crescita continua e si prevede che, nei prossimi anni, aumenti esponenzialmente.

La necessità di gestire queste grandi quantità di dati ha portato alla nascita del movimento NoSQL, ovvero tecnologie in grado di gestire i Big Data con grandi performance grazie alla loro scalabilità orizzontale, che si differenziano dai sistemi tradizionali, i quali risultano poco adatti a causa dei costi da sostenere e non garantiscono grandi performance. Di queste nuove tecnologie, dopo aver trattato gli aspetti principali che hanno portato al loro sviluppo e alle caratteristiche che li accomunano, viene presentata una panoramica delle tipologie di database NoSQL più rappresentativi, introducendo un DBMS per ogni tipologia.

In seguito, si è preso in esame MongoDB, database NoSQL orientato ai documenti, e descritto le principali caratteristiche, il modello di dati basato su documenti JSON, il linguaggio di interrogazione basato su JavaScript, inoltre si è proseguito a descrivere il funzionamento e la sintassi delle operazioni di inserimento, cancellazione e aggiornamento.

MongoDB è adatto quando si necessita di Query dinamiche, si preferisce lavorare con gli indici ed è veloce quando si lavora con grandi quantità di dati. Grazie a queste caratteristiche che lo rendono molto performante, è stato utilizzato come supporto di base di dati per un applicazione mobile che raccoglie informazioni sulla connettività urbana e, successivamente, interfacciandosi con Node.JS vengono estrapolati i dati e mostrati attraverso le mappe di Google. Infine nell'elaborato viene descritto l'applicazione presentando le funzionalità consentite e le tecnologie utilizzate.

I dati presenti nel database per il momento sono limitati ma si prevede che in futuro sarà necessario gestire elevate quantità di dati e, siccome MongoDB è orientato alla scalabilità orizzontale, si studierà la possibilità di distribuire i dati attraverso lo sharding. In particolare, si realizzerà un cluster di database MongoDB in modo da aumentare le capacità di storage e le prestazioni in termini di risposta.

Bibliografia

[01] ACID
https://it.wikipedia.org/wiki/ACID

[02] AJAX(Asincronous JavaScript and XML)
https://it.wikipedia.org/wiki/AJAX

[03] Bianchi Manuel, *MongoDB Analisi e prototipazione su applicazioni di Social Business Intelligence*, 2013

[04] Bootstrap
http://getbootstrap.com/2.3.2/

[05] Cascading style sheets (CSS)
https://it.wikipedia.org/wiki/CSS

[06] Classificazione database NoSQL
http://nosql-database.org/

[07] Daprà Alberto, *Big Data: una opportunità di sviluppo, crescita e innovazione*, Statistica & Società/Anno 1, N. 1/Strumenti. Consultabile al sito:
http://new.sis-statistica.org/

[08] Dedagroup Highlights (ICT NETWORK) e ECOS, *BIG DATA: riconoscerli, gestirli, analizzarli*. Luglio 2012

[09] ExpressJS
http://expressjs.com/

[10] Ficetola Francesco, *Information Engineer and IT Specialist, NO-SQL e introduzione a MongoDB*, 14 Marzo 2012

[11] Google map API
https://developers.google.com/maps/

[12] Guida a Node.JS
http://www.hostingtalk.it/guide/guida-a-nodejs/

[13] Karl Seguin, *The Little MongoDB Book*

[14] HyperText Markup Language (HTML)
https://it.wikipedia.org/wiki/HTML

[15] JQuery framework JavaScript
https://it.wikipedia.org/wiki/JQuery

[16] Mayer-Schönberger V., Cukier K. N., Merlini R., *Big data. Una rivoluzione che trasformerà il nostro modo di vivere e già minaccia la nostra libertà*. Garzanti Libri, 2013

[17] Merelli Luca, *Analisi delle performance dei database non relazionali il caso di studio di MongoDB*, 2012

[18] Mongodb
http://www.mongodb.org/

[19] MongoDB
http://www.fulviogabana.it/Data/introduzione-a-mongodb

[20] MongoJS
http://www.srirangan.net/

[21] MorrisJS
http://morrisjs.github.io/morris.js/

[22] NoSQL
http://www.w3resource.com/mongodb/nosql.php

[23] NoSQL Database
http://www.researchgate.net/publication/259625092_NoSQL_Databases

[24] NoSQL replicazione
http://www.html.it/pag/54548/alta-affidabilita-la-replicazione/

[25] NoSQL scalabilità
https://it.wikipedia.org/wiki/Scalabilità

[26] RaphaelJS
http://raphaeljs.com/

[27] Rezzani Alessandro, *Big data. Architettura, tecnologie e metodi per l'utilizzo di grandi basi di dati*, Apogeo 2013

[28] Robomongo
http://robomongo.org/

[29] Rome Francesco, *Analisi del fenomeno dei Big Data e comparazione di DBMS NoSQL a loro supporto*, 2014

[30] TechRepublic Popolarità Database NoSQL
http://www.techrepublic.com/